Efficient Large-Scale Multi-Drone Delivery Using Transit Networks

Shushman Choudhury, Kiril Solovey, Mykel J. Kochenderfer, and Marco Pavone

***Abstract*— We consider the problem of controlling a large fleet of drones to deliver packages simultaneously across broad urban areas. To conserve their limited flight range, drones can seamlessly hop between and ride on top of public transit vehicles (e.g., buses and trams). We design a novel comprehensive algorithmic framework that strives to minimize the maximum time to complete any delivery. We address the multifaceted complexity of the problem through a two-layer approach. First, the upper layer assigns drones to package delivery sequences with a provably near-optimal polynomial-time task allocation algorithm. Then, the lower layer executes the allocation by periodically routing the fleet over the transit network while employing efficient bounded-suboptimal multi-agent pathfinding techniques tailored to our setting. We present extensive experiments supporting the efficiency of our approach on settings with up to 200 drones, 5000 packages, and large transit networks of up to 8000 stops in San Francisco and the Washington DC area. Our results show that the framework can compute solutions within a few seconds (up to 2 minutes for the largest settings) on commodity hardware, and that drones travel up to 450% of their flight range by using public transit.**

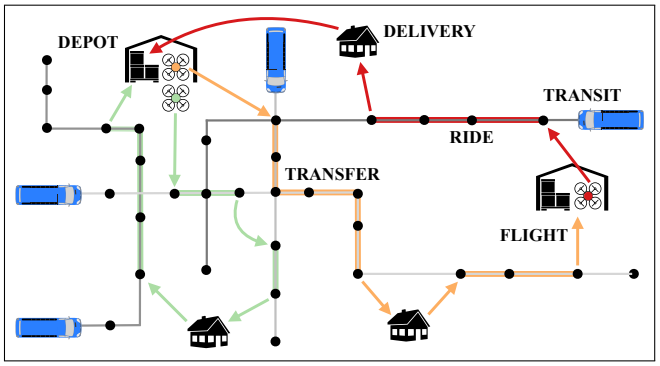
I. Introduction

Rapidly growing e-commerce demands have greatly strained dense urban communities by increasing delivery truck traffic and slowing operations, and impacting travel times for public and private vehicles [1, 2]. Further congestion is being induced by newer services (e.g., Instacart and Uber Eats) relying on ride-sharing vehicles. There is a clear need to redesign the current method of package distribution in cities [3]. The agility and aerial reach of drones, the flexibility and ease of establishing drone networks, and recent advances in drone capabilities make them highly promising for modern logistics networks [4]. However, drones have limited travel range and carrying capacity [5, 6]. On the other hand, ground-based transit networks have less flexibility but far greater coverage and throughput. By combining the strengths of both, we can achieve significant commercial benefits and social impact (reducing ground congestion; delivering medicine and essentials).

We address the problem of operating a large number of drones to deliver multiple packages simultaneously in an area. The drones can use one or more vehicles in a public-transit network as modes of transportation, thereby saving their limited battery energy stored on-board and increasing their effective travel range. We are required to decide which deliveries each drone should make and in what order, which modes of transit to use, and for what duration.

Our approach must contend with the multiple significant challenges of our problem. It must plan over large time-dependent transit networks, while accounting for energy

The authors are with Stanford University, CA, USA: S.C. is with the Department of Computer Science and K.S., M.K., and M.P. are with the Department of Aeronautics and Astronautics.



constraints that limit the drones' flight ranges. It must avoid inter-drone conflicts, such as where more than one drone attempts to board the same vehicle at the same time, or when the maximum carrying capacity of a vehicle is exceeded. We seek not just feasible multi-agent plans but high-quality solutions in terms of a cumulative objective over all drones, the makespan, i.e., the maximum individual delivery time for any drone. Additionally, our approach must also solve the task allocation problem of determining which drones deliver which packages, and from which distribution centers.

A. Related work

Some individual aspects of our problem have been studied. The single-agent setting of controlling an autonomous drone to use multiple modes of transit en route to its destination was investigated in [7]. Recent work has considered pairing a drone with a delivery truck, which does not exploit public transit [8–10]. The multi-agent issues of task allocation and inter-agent conflicts also not addressed either.

Our problem is closely related to routing a fleet of autonomous vehicles providing mobility-on-demand services [11–13]. Specifically, the task is to compute routes for the vehicles (both customer-carrying and empty) so that travel demand is fulfilled and operational cost is minimized. In particular, recent works [14, 15] study the combination of such service with public transit, where passengers can use several modes of transportation in the same trip. However, such works abstract away inter-agent constraints or dynamics and are not suited for autonomous pathfinding. The task-allocation setting we consider in our problem can be viewed as an instance of the vehicle routing problem [16–18], variants of which are typically solved by mixed integer linear programming (MILP) formulations that scale poorly, or by heuristics that do not provide optimality guarantees.

We must contend with the challenges of planning for multiple agents. Accordingly, the second layer of our approach is a multi-agent path finding (MAPF) problem [19, 20]. Since the drones are on the same team, we have a centralized or cooperative pathfinding setting [21]. The MAPF

problem is NP-hard to solve optimally [22]. A number of efficient solvers have been developed that work well in practice [23]. The MAPF formulation and algorithms have been extended to several relevant scenarios such as lifelong pickup-and-delivery [24] and joint task assignment and pathfinding [25, 26], though for different task settings and constraints than ours. Also, a MAPF formulation was applied for UAV traffic management in cities [27]. However, none of the approaches considered pathfinding over large time-dependent transit networks. We use models, algorithms and techniques from transportation planning [28–30].

B. Statement of contributions

We present a comprehensive algorithmic framework for large-scale multi-drone delivery in synergy with a ground transit network. Our approach strives to minimize the maximum time to complete any delivery. We decompose the highly challenging problem and solve it stage-wise with a two-layer approach. First, the upper layer assigns drones to package-delivery sequences with a task allocation algorithm. Then, the lower layer executes the allocation by periodically routing the fleet over the transit network.

Algorithmically, we develop a new delivery sequence allocation method for the upper layer that obtains a provably near-optimal solution in polynomial runtime. For the lower layer, we extend techniques for multi-agent path finding that account for time-dependent public transit networks and agent energy constraints to perform multi-drone routing.

Experimentally, we present extensive results supporting the efficiency of our approach on settings with up to 200 drones, 5000 packages, and transit networks of up to 8000 stops in San Francisco and the Washington DC area. We demonstrate that our framework can compute solutions within a few seconds (up to 2 minutes for the largest settings) on our commodity hardware, and that in our problem scenarios, drones travel up to 450% of their flight range by using transit.

The following is the paper structure. We present an overall description of the two-layer approach in Section II, and then elaborate on each layer in Sections III and IV. We present experimental results on simulations in Section V, and conclude the paper with Section VI. We will also refer to the appendices, which have additional details, illustrations, results, and discussions.

II. Methodology

We provide a high-level description of our formulation and approach to illustrate the various interacting components.

A. Problem Formulation

We are operating a centralized homogeneous fleet of m drones within 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 large time interval (e.g., a day), a batch of delivery requests for k different *packages*, denoted $V_P := \{p_1, \ldots, p_k\} \subset \mathbb{R}^2$, is received (we assume that $k \gg m$). Each p_i has a corresponding location. We assume that any package can be dispatched from any depot; our framework exploits this property to optimize the solution quality in terms of *makespan*, i.e., the maximum execution time for any delivery. In Section III, we mention how our approach 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 operating in the area (e.g., a public bus network), which remain unaffected by the drones' actions. Our problem is to route drones to deliver all packages while minimizing makespan. A drone route consists of its current location and the sequence of depot and package locations to visit with a combination of flying and riding on transit. A central constraint is the drones' limited energy, which we characterize as a maximum flight distance. A feasible solution must satisfy inter-drone constraints such as collision avoidance and transit vehicle capacity limits.

Finally, we make some assumptions for our setting: a drone carries one package at a time, which is reasonable given state-of-the-art drone payloads [6]; drones are recharged upon visiting a depot in negligible time (e.g., a battery replacement); depots have unlimited drone capacity; the transit network is deterministic with respect to locations and vehicle travel times (we mention uncertainty in Section VI). We do account for the time-varying nature of the transit.

B. Approach overview

In principle, we could frame the entire problem as a mixed integer linear program (MILP). However, for real-world problems (hundreds of drones; thousands of packages; large transit networks), even state-of-the-art MILP approaches are unlikely to scale. Moreover, even a simpler problem that ignores the interaction constraints is an instance of the (notoriously challenging) multi-depot vehicle routing problem [17]. Thus, we decouple the problem into two distinct subproblems that we solve stage-wise in layers.

The upper layer performs *task allocation* to determine which packages are delivered by which drone and in what order. It takes as input the known depot and package locations, and an estimate of the drone travel time between every pair of locations. It then solves a threefold allocation to minimize delivery makespan and assigns to each package (i) the dispatch depot and (ii) the delivery drone, and to each drone (iii) the order of package deliveries. To this end, we develop an efficient polynomial-time task-allocation algorithm that achieves a near-optimal makespan.

The lower layer performs *route planning* for the drone fleet to execute the allocated delivery tasks. It generates detailed routes of drone locations in time and space and the transit vehicles used, while accounting for the time-varying transit network. It also ensures that (i) simultaneous transit boarding by multiple drones is avoided, (ii) no transit vehicle exceeds its drone-carrying capacity, and (iii) drone (battery) energy constraints are respected. We efficiently handle individual and inter-drone constraints by framing the routing problem as an extension of multi-agent path finding (MAPF) to transit networks. We adapt a scalable, bounded sub-optimal variant of a highly effective MAPF solver called Conflict-Based Search (CBS) [31] to solve the one-delivery-per-drone problem. Finally, we obtain routes for the sequence

of deliveries in a receding-horizon fashion by replanning for the next task once a drone completes its current one.

Decomposition-based stage-wise optimization approaches typically have an approximation gap compared to solutions of the full problem. In our case, this gap manifests in the surrogate cost estimate we use for the drone's travel time between locations in the task-allocation layer (instead of jointly solving for allocation and multi-agent routing over transit networks, which is not feasible at scale). The better the surrogate, the more *coupled* the layers are, i.e., the better is the solution of the first stage for the second one. Such surrogates have a tradeoff between efficiency and approximation quality.

An easy-to-compute travel time surrogate, for instance, is the drone's direct flight time between two locations (ignoring the transit network). However, that can be of very poor quality when the drone requires transit to reach an out-of-range target. We use a surrogate that actually accounts for the transit network, at the expense of some modest preprocessing. We defer details to Appendix III, but the basic idea is to precompute the pairwise shortest travel times between locations spread around the city, over a representative snapshot of the transit network.

III. Task Assignment and Package Allocation

We now discuss how we leverage our problem's structure to design a new algorithm called MergeSplitTours for the task-allocation layer, which guarantees a near-optimal solution in polynomial time. The goal of this layer is to (i) distribute the set of packages V_P among m agents, (ii) assign each package destination $p \in V_P$ to a depot $d \in V_D$, and (iii) assign drones to a sequence of depot pickups and package deliveries. The objective is to **minimize the maximum travel time among all agents** over all three of the above components.

Our problem can be cast as a special version of the m traveling salesman problem [32], which we call the m *minimal visiting paths problem* (m-MVP). We seek a set of m paths such that the makespan, i.e., the maximum travel time for any path, is minimized. We only need *paths* that start and end at (the same or different) depots, not tours. Our formulation is the *asymmetric* variant, for a *directed* underlying graph, which is NP-hard even for $m = 1$. Moreover, the current best polynomial-time approximation [33] yields the fairly large approximation factor $O(\log n / \log\log n)$, for a graph with n vertices. An additional challenge is the inability to assume the triangle inequality on our objective of travel times.

A key element of m-MVP is the *allocation graph* $G_A = (V_A, E_A)$, with vertex set $V_A = V_D \cup V_P$. 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. As we flagged in Section II-B, any dispatch constraints are modeled by excluding edges from the corresponding depot. We are now ready for the full definition of m-MVP:

Definition 1. Given the allocation graph G_A, the m minimal visiting paths problem (m-MVP) consists of finding m-paths $P_1^*, \ldots, P_m^*$ 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 depot $d' \in V_D$, (2) exactly one path visits each package $p \in V_P$, and (3) the maximum travel time of any of the paths is minimized.

Algorithm 1: MergeSplitTours(G_A)

Solve MCT(G_A) to get t tours $\mathbb{T} := \{T_1, \ldots, T_t\}$;
while $|\mathbb{T}| > 1$ **do**
  Pick distinct tours $T, T' \in \mathbb{T}$ and depots $d \in T, d' \in T'$ that minimize $c_{dd'} + c_{d'd}$;
  Merge T, T' by adding $(d, d'), (d', d)$ edges ;
Split final tour T into m sub-paths $P_1, \ldots, P_m$, where LENGTH(P_i) is proportional to LENGTH(T)/m for each i (similar to [34]);
Extend each P_i to ensure it begins and ends at a depot;
return $P_1, \ldots, P_m$;

TABLE I: An integer programming formulation of the MCT problem.

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

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

subject to

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

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

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

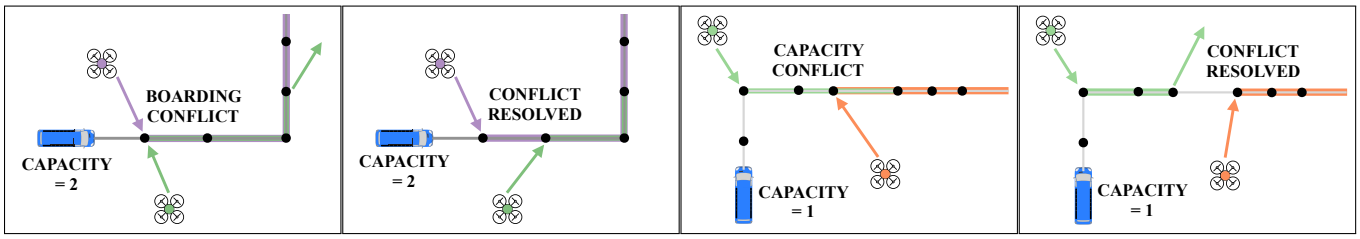
$$\sum_{v\in N_+(d)} x_{vd} - \sum_{v\in N_-(d)} x_{dv} = 0, \quad \forall d \in V_D. \tag{5}$$

where $N_+(v), N_-(v)$ denote the in and out going neighbors of $v \in V_A$.

Let OPT be the optimal makespan, i.e., OPT := $\max_{i\in[m]}$ LENGTH(P_i^*), where LENGTH($\cdot$) denotes the total travel time along a given path or tour. We make three 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.

A. Algorithm Overview

We present our MergeSplitTours algorithm for solving m-MVP (Algorithm 1); see a detailed description in Appendix I. A key step is generating an initial set of tours $\mathbb{T}$ by solving the minimal-connecting tours (MCT) problem (see Table I), which attempts to connect packages to depots within tours to minimize the total edge weight in eq. (1). The constraint in eq. (4) is that each package is connected to precisely one incoming and one outgoing edge from and to depots respectively. The final constraint in eq. (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. The solution to MCT is the assignment $\{x_{uv}\}_{(u,v)\in\mathbb{E}}$, i.e., which edges of G_A are used and how many times. This assignment implicitly represents the desired collection of aforementioned



(a) (b) (c) (d)

Fig. 1: 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.

tours $T_1, \ldots, T_t$; see Appendix I.

B. Theoretical Guarantees

All proofs from this secion are in Appendix I. The following theorem states that MERGESPLITTOURS is correct and that its makespan is close to optimal.

Theorem 1. *Suppose G_A is strongly connected and the subgraph $G_A(V_D)$ induced by the vertices V_D is a directed clique. Let $P_1, \ldots, P_m$ be the output of* MERGESPLITTOURS. *Then, every package $p \in V_P$ is contained in exactly one path P_i, and every P_j starts and ends at a depot. Moreover, it holds that* $\max_{i\in[m]} \text{LENGTH}(P_i) \leqslant \text{OPT} + \alpha + \beta$,

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

The key idea is that the total cost of the tours induced by the solution to MCT cannot exceed the total length of $\{P_1^*, \ldots, P_m^*\}$. The MCT solution is then adapted to m paths with an additional overhead of $\alpha + \beta$ per path. When $m \ll |V_P|$ (typically the case), α, β are small compared to OPT, making the bound tight. For instance, in our randomly-generated scenarios in Section V-A, for $m = 5, k = 200$, the approximation ratio $\max_{i\in[m]} \text{LENGTH}(P_i)/\text{OPT} = 1.09$, and for $m = 10, k = 500$, the factor is 1.06.

The main computational bottleneck of the algorithm is MCT, while the other components can clearly be implemented polynomially in the input size. However, it suffices to solve a relaxed version of MCT to obtain the same integral solution.

Lemma 1. *The optimal solution to the fractional relaxation of* MCT, *in which $x_{uv} \in [0, 1]$ for all $u \in V_P \vee v \in V_P$, and $x_{uv} \in \mathbb{R}_+$ otherwise, yields the integer optimal solution.*

The lemma follows from casting MCT as the minimum-cost circulation problem, for which the constraint matrix is totally unimodular [35]. Therefore, MERGESPLITTOURS can be implemented in polynomial time.

IV. MULTI-AGENT PATH FINDING

For each drone $i \in [m]$, the allocation layer yields a sequence of delivery tasks $d_1 p_1 \ldots p_l d_l$. Each delivery sequence has one or more subsequences of dpd'. The route-planning layer treats each dpd' subsequence *as an individual drone task*, i.e., leaving with the package from depot d, carrying it to package location p and returning to the (same or different) depot d', without exceeding the energy capacity. We seek an efficient and scalable method to obtain high-quality (with respect to travel time) feasible paths, while using transit options to extend range, for m different drone dpd' tasks simultaneously. The full set of delivery sequences can be satisfied by replanning when a drone finishes its current task and begins a new one; we discuss and compare two replanning strategies in Appendix IV. 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 sub-optimal algorithm.

A. MAPF with Transit Networks (MAPF-TN)

The problem of Multi-Agent Path Finding with Transit Networks (MAPF-TN) is the extension of standard MAPF to where agents can use one or more modes of transit in addition to moving. The incorporation of transit networks introduces additional challenges and underlying structure.

The **input** to MAPF-TN is the set of m tasks $(d_i, p_i, d_i')_{i=1:m}$ and the directed operation graph $G_O = (V_O, E_O)$. In Section III, the allocation graph G_A only considered depots and packages, and edges between them. Here, G_O also includes transit vertices, $V_{TN} = \bigcup_{\tau\in\mathcal{T}} R_\tau$, where $\mathcal{T}$ is the set of trips, and each trip $R_\tau = \{(s_1, t_1) \ldots\}$ is a sequence of time-stamped stop locations (a given stop location may appear as several different nodes with distinct time-stamps). Similarly, we also use time-expanded [28] versions of V_D and V_P.

The edges are defined as follows: An edge $e = (u \to v) \in E$ is a *transit edge* if $u, v \in V_{TN}$ and are consecutive stops on the same trip R_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 distance between them for a suitable metric. 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 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.

We now define the three attributes for E_O. For time-constrained edges, $T(e) = v.t - u.t$ is the difference between corresponding 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 bounded by the capacity of the vehicle, while for flight edges, $C(e) = \infty$. Here, we assume that time-unconstrained flight in open space can be accommodated (thoroughly examined in [27]).

We now describe the remaining relevant details of the MAPF-TN problem. 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. In addition, the drone should 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 reduced speed to reach the transit just in time. Thus, the constraint $\bar{N}$ is 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** $\Pi = \bigcup_{i=1:m} \pi_i$ is a set of m individually feasible paths that does not violate any of the following two *shared constraints* (see Figure 1): (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.

B. Conflict-Based Search for MAPF-TN

To tackle MAPF-TN, we modify the Conflict-Based Search (CBS) algorithm [31]. The multi-agent level of CBS 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 (i.e., violate 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. In MAPF-TN, conflicts arise from boarding and capacity constraints. CBS obtains optimal multi-agent solutions without having to run (potentially significantly expensive) multi-agent searches. However, its performance can degrade heavily with many conflicts in which constraints are violated. Figure 1 illustrates the generation and resolution of conflicts in our MAPF-TN problem.

For scalability, we use a bounded sub-optimal variant of CBS called *Enhanced CBS* (ECBS), which achieves orders of magnitude speedups over CBS [36]. ECBS uses bounded sub-optimal *Focal Search* [37] at both levels, instead of best-first A* [38]. Focal search allows using an inadmissible heuristic that prioritizes efficiency. We now describe a crucial modification to ECBS required for MAPF-TN.

Focal Weight-constrained Search: Unlike typical MAPF, the low-level graph search in MAPF-TN has a path-wide constraint (traversal distance) *in addition to the objective function* of traversal time. For the shortest path problems on graphs, adding a path-wide constraint makes it NP-hard [39]. Several algorithms for constrained search [40, 41] require an explicit enumeration of the edges. We extend the A* for MultiConstraint Shortest Path (A*-MCSP) algorithm [42] (suitable for our implicit graph) to focal search (called Focal-MCSP). Focal-MCSP uses admissible heuristics on both objective and constraint and maintains only non-dominated paths to intermediate nodes. This extensive book-keeping requires a careful implementation for efficiency.

TABLE II: The mean computation time for MERGESPLITTOURS in seconds, over 100 different trials for each setting. 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. The Out-of-Memory cases are due to the linear programming step of `MCT` and could be resolved in practice with a larger machine or a distributed implementation.

k	$\ell = 2$	$\ell = 5$	$\ell = 10$	$\ell = 20$	$\ell = 30$
50	0.006	0.022	0.074	0.326	0.981
100	0.016	0.070	0.268	1.274	2.823
200	0.053	0.272	1.171	4.323	11.09
500	0.311	1.731	6.532	31.15	83.31
1000	1.483	6.811	31.08	OutOfMem	OutOfMem
5000	38.05	OutOfMem	OutOfMem	OutOfMem	OutOfMem

Focal-MCSP inherits the properties of A*-MCSP and Focal Search; therefore, it yields a bounded-suboptimal feasible path to the target node. Accordingly, **ECBS with Focal-MCSP yields a bounded sub-optimal solution to MAPF-TN**. The result follows from the analysis of ECBS [36]. Also, note that a dpd' path requires a bounded sub-optimal path from d to p and another from p to d', such that their concatenation is feasible. Since this is even more complicated, in practice, we run Focal-MCSP twice (from d to p and p to d') with half the energy constraint each time and concatenate the two paths, thus guaranteeing feasibility. In Appendix II-B we discuss other required modifications to standard MAPF and important speedup techniques that nonetheless retain the bounded sub-optimality of Enhanced CBS for our MAPF-TN formulation.

V. EXPERIMENTS AND RESULTS

We implemented our approach using the Julia language and tested it on a machine with a 6-core 3.7 GHz 16 GiB RAM CPU [1]. For very large combinatorial optimization problems, solution quality and algorithm efficiency are of interest. We have already shown that the upper and lower layers are near-optimal and bounded-suboptimal respectively in terms of solution quality, i.e., makespan. Therefore, for evaluation we focus on their efficiency and scalability to large real-world settings. We do not attempt to baseline against a MILP approach for the full joint problem; we estimate that a typical setting of interest to us will have $\mathcal{O}(10^7)$ variables in a MILP formulation, besides exponential 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 data [43] for each network. We considered only the bus network (by far the most extensive), but our formulation can accommodate multiple modes. We defined a geographical bounding box in each case, of area $150\,\mathrm{km}^2$ for SFMTA and $400\,\mathrm{km}^2$ for WMATA (illustrated in Appendix IV), within which depots and package locations were randomly generated. For the transit network, we considered all bus trips that operate within the bounding box. The *size* of the time-expanded

[1] The code for our work is available at `https://github.com/sisl/MultiAgentAllocationTransit.jl`

TABLE III: (All times are in seconds) An extensive analysis of the MAPF-TN layer, on 20 trials for each $\{\ell, m\}$ setting, with different randomly generated depots and delivery locations for each trial. The integer carrying capacity of any transit edge $C(e)$ was randomly chosen from $\{3, 4, 5\}$ (single and double-buses). The sub-optimality factor for ECBS was 1.1. For settings with $m/\ell = 10$, a number of trials timed out (over 180 s) and were discarded.

	San Francisco ($\lvert V_{TN}\rvert = 4192$; Area 150 km^2)				**Washington DC** ($\lvert V_{TN}\rvert = 7608$; Area 400 km^2)			
{Depots, Agents} $\{\ell, m\}$	{Median, Avg} Plan Time	{Avg, Max} Range Ext.	{Avg, Max} Transit Used	Avg Soln. Makespan	{Median, Avg} Plan Time	{Avg, Max} Range Ext.	{Avg, Max} Transit Used	Avg Soln. Makespan
$\{5, 10\}$	$\{0.53, 1.07\}$	$\{1.5, 3.0\}$	$\{2.9, 6\}$	2464.8	$\{4.73, 7.17\}$	$\{2.0, 3.5\}$	$\{3.3, 8\}$	5006.5
$\{5, 20\}$	$\{1.11, 2.79\}$	$\{1.6, 2.5\}$	$\{3.1, 6\}$	2555.5	$\{11.5, 12.4\}$	$\{2.3, 3.8\}$	$\{4.5, 7\}$	5819.2
$\{5, 50\}$	$\{3.18, 8.29\}$	$\{1.7, 2.4\}$	$\{4.5, 6\}$	3485.8	$\{50.5, 57.6\}$	$\{2.9, 3.6\}$	$\{5.1, 7\}$	6861.7
$\{10, 20\}$	$\{0.64, 0.68\}$	$\{1.2, 1.8\}$	$\{2.3, 4\}$	2082.7	$\{7.31, 8.78\}$	$\{2.1, 3.2\}$	$\{3.7, 7\}$	5195.6
$\{10, 50\}$	$\{1.73, 2.96\}$	$\{1.6, 3.6\}$	$\{3.2, 5\}$	2671.1	$\{15.6, 19.8\}$	$\{2.5, 4.5\}$	$\{3.9, 6\}$	6316.7
$\{10, 100\}$	$\{2.09, 4.17\}$	$\{1.4, 1.8\}$	$\{3.7, 7\}$	3084.5	$\{35.9, 39.7\}$	$\{2.5, 3.7\}$	$\{4.9, 8\}$	6889.8
$\{20, 50\}$	$\{0.19, 0.26\}$	$\{0.9, 1.2\}$	$\{0.8, 4\}$	1054.8	$\{8.14, 11.2\}$	$\{1.9, 4.1\}$	$\{3.6, 7\}$	5086.9
$\{20, 100\}$	$\{0.41, 0.66\}$	$\{1.1, 1.3\}$	$\{1.4, 4\}$	1457.6	$\{17.5, 19.2\}$	$\{2.2, 3.7\}$	$\{4.1, 6\}$	5400.9
$\{20, 200\}$	$\{0.73, 1.70\}$	$\{1.1, 2.3\}$	$\{2.2, 5\}$	1824.4	$\{22.9, 26.2\}$	$\{2.2, 2.9\}$	$\{4.4, 6\}$	6050.1

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 between queries, but the full graph G_O can be dense). The drone's flight range constraint is set (conservatively) to 7 km and average speed to 25 kph, based on the DJI Mavic 2 specifications [6]. In this section, we evaluate the performance of the two primary components — the task allocation and multi-agent path finding layers. In Appendix IV we compare two replanning strategies for when a drone finishes its current delivery, and compare two surrogate travel time estimates for layer coupling.

A. Task Allocation

The scale of the allocation problem is determined by the number of depots and packages, i.e., $\ell + k$. The runtimes for MERGESPLITTOURS with varying ℓ, k over SFMTA are displayed in Table II. The roughly-quadratic increase in runtimes along a specific row or column demonstrate that our provably near-optimal MERGESPLITTOURS algorithm is indeed polynomial in the size of the input. Even for up to 5000 deliveries, the absolute runtimes are quite reasonable. We do not compare with naive MILP even for allocation, as the number of variables would exceed $\ell \cdot k$, in addition to the expensive subtour elimination constraints [44].

B. MAPF with Transit Networks (MAPF-TN)

Solving multi-agent path finding optimally is NP-hard [22]. Previous research has benchmarked CBS variants and shown that Enhanced CBS is most effective [36, 45]. Therefore, we focus on extensively evaluating our own approach rather than redundant baselining. Table III quantifies several aspects of the MAPF-TN layer with varying numbers of depots (ℓ) and agents (m), the two most tunable parameters. Before each trial, we run the allocation layer and collect m dpd' tasks, one for each agent. We then run the MAPF-TN solver on this set of tasks to compute a solution.

We discuss broad observations here and provide a detailed analysis in Appendix IV. The results are very promising; our approach scales to large numbers of agents (200) and large transit networks (nearly 8000 vertices); the highest average makespan for the true delivery time is less than an hour (3485.8 s) for SFMTA and 2 hours (6889.8 s) for WMATA; drones are using up to 8 transit options per route to extend their range by up to 4.5x. 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 (compare $\{5, 20\}$ 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 (compare $\{10, 50\}$ to $\{20, 50\}$). Plan times are much higher for WMATA due to a larger area and a larger and less uniformly distributed bus network, leading to higher single-agent search times and more multi-agent conflicts. Trials taking more than 3 minutes were discarded; two pathological cases with SFMATA and WMATA (each with $\{l = 10, m = 100\}$) took nearly 4 and 8 minutes, due to 30 and 10 conflicts respectively. In any case, a deployed system would have better compute and parallelized implementations. Finally, note that the running times reported here are actually pessimistic, because we consider cases where drones are released simultaneously from the depots, which increases conflicts. However, a gradual release by executing the MAPF solver over a longer horizon (as we discuss in Appendix IV-B), results in fewer conflicts, allowing us to cope with an even larger drone fleet.

VI. CONCLUSION AND FUTURE WORK

We designed a comprehensive algorithmic framework for solving the highly challenging problem of multi-drone package delivery with routing over transit networks. Our two-layer approach is efficient and highly scalable to large problem settings and obtains high-quality solutions that satisfy the many system constraints. We ran extensive simulations with two real-world transit networks that demonstrated the widespread applicability of our framework and how using ground transit judiciously allows drones to significantly extend their range.

A key future direction is to perform case studies that estimate the operational cost of our framework, evaluate its impact on road congestion, and considers potential externalities like noise pollution and disparate impact on urban communities. Another direction is to extend our model to overcome its limitations: delays and uncertainty in the travel pattern of transit vehicles [46] and delivery time windows [47]; jointly routing ground vehicles and drones; optimizing for the placements of depots, whose locations are currently randomly generated and given as input.

ACKNOWLEDGMENTS

This work was supported in part by NSF, Award Number: 1830554, the Toyota Research Institute (TRI), and the Ford Motor Company.The authors thank Sarah Laaminach, Nicolas Lanzetti, Mauro Salazar, and Gioele Zardini for fruitful discussions on transit networks.

REFERENCES

[1] E. Humes, "Online Shopping Was Supposed to Keep People Out of Traffic. It Only Made Things Worse," 2018, accessed: August 30, 2019. [Online]. Available: http://bit.ly/2HCkAmQ

[2] J. Holguin-Veras, J. A. Leal, I. Sanchez-Diaz, M. Browne, and J. Wojtowicz, "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*, 2018.

[3] N. Kafle, B. Zou, and J. Lin, "Design and Modeling of a Crowdsource-Enabled System for Urban Parcel Relay and Delivery," *Transportation Research Part B: Methodological*, vol. 99, pp. 62 – 82, 2017.

[4] M. Joerss, F. Neuhaus, and J. Schroder, "How Customer Demands are Reshaping Last-Mile Delivery," 2016, accessed: August 30, 2019. [Online]. Available: https://mck.co/2NIRdmE

[5] A. W. Sudbury and E. B. Hutchinson, "A Cost Analysis of Amazon Prime Air (Drone Delivery)," *Journal for Economic Educators*, vol. 16, no. 1, pp. 1–12, 2016.

[6] DJI, "DJI Mavic 2 Specifications Sheet." [Online]. Available: http://bit.ly/2mfCAvz

[7] S. Choudhury, J. Knickerbocker, and M. Kochenderfer, "Dynamic Real-time Multimodal Routing with Hierarchical Hybrid Planning," in *IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019.

[8] C. C. Murray and A. G. Chu, "The Flying Sidekick Traveling Salesman Problem: Optimization of Drone-Assisted Parcel Delivery," *Transportation Research Part C: Emerging Technologies*, vol. 54, pp. 86 – 109, 2015.

[9] N. Agatz, P. Bouman, and M. Schmidt, "Optimization Approaches for the Traveling Salesman Problem with Drone," *Transportation Science*, vol. 52, no. 4, pp. 965–981, 2018.

[10] S. M. Ferrandez, T. Harbison, T. Weber, R. Sturges, and R. Rich, "Optimization of a Truck-Drone in Tandem Delivery Network using k-means and Genetic Algorithm," *Journal of Industrial Engineering and Management*, vol. 9, no. 2, pp. 374–388, 2016.

[11] K. Solovey, M. Salazar, and M. Pavone, "Scalable and Congestion-Aware Routing for Autonomous Mobility-On-Demand via Frank-Wolfe Optimization," in *Proceedings of Robotics: Science and Systems*, 2019.

[12] R. Iglesias, F. Rossi, R. Zhang, and M. Pavone, "A BCMP network approach to modeling and controlling autonomous mobility-on-demand systems," *I. J. Robotics Res.*, vol. 38, no. 2-3, 2019.

[13] A. Wallar, M. V. D. Zee, J. Alonso-Mora, and D. Rus, "Vehicle rebalancing for mobility-on-demand systems with ride-sharing," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018, pp. 4539–4546.

[14] M. Salazar, F. Rossi, M. Schiffer, C. H. Onder, and M. Pavone, "On the interaction between autonomous mobility-on-demand and public transportation systems," in *International Conference on Intelligent Transportation Systems*, 2018, pp. 2262–2269.

[15] J. Zgraggen, M. Tsao, M. Salazar, M. Schiffer, and M. Pavone, "A Model Predictive Control Scheme for Intermodal Autonomous Mobility-on-Demand," in *IEEE International Conference on Intelligent Transportation Systems*, 2019.

[16] J. Caceres-Cruz, P. Arias, D. Guimarans, D. Riera, and A. A. Juan, "Rich Vehicle Routing Problem: Survey," *ACM Comput. Surv.*, vol. 47, no. 2, pp. 32:1–32:28, 2014.

[17] A. Otto, N. Agatz, J. Campbell, B. Golden, and E. Pesch, "Optimization Approaches for Civil Applications of Unmanned Aerial Vehicles (uavs) or Aerial Drones: A Survey," *Networks*, vol. 72, no. 4, pp. 411–458, 2018.

[18] P. Toth and D. Vigo, *Vehicle Routing – Problems, Methods, and Applications*, 2nd ed., K. Scheinberg, Ed. SIAM, 2014.

[19] M. Erdmann and T. Lozano-Perez, "On Multiple Moving Objects," *Algorithmica*, vol. 2, no. 1-4, p. 477, 1987.

[20] J. Yu and S. M. LaValle, "Optimal Multirobot Path Planning on Graphs: Complete Algorithms and Effective Heuristics," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.

[21] D. Silver, "Cooperative Pathfinding," in *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. AAAI Press, 2005, pp. 117–122.

[22] J. Yu and S. M. LaValle, "Structure and Intractability of Optimal Multi-robot Path Planning on Graphs," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2013.

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

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

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

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

[27] F. Ho, A. Salta, R. Geraldes, A. Goncalves, M. Cavazza, and H. Prendinger, "Multi-agent Path Finding for Uav Traffic Management," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 131–139.

[28] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis, "Efficient Models for Timetable Information in Public Transportation Systems," *Journal of Experimental Algorithmics (JEA)*, vol. 12, pp. 2–4, 2008.

[29] D. Delling, P. Sanders, D. Schultes, and D. Wagner, "Engineering Route Planning Algorithms," in *Algorithmics of Large and Complex Networks*. Springer-Verlag, 2009, pp. 117–139.

[30] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck, "Route Planning in Transportation Networks," in *Algorithm Engineering*. Springer, 2016, pp. 19–80.

[31] G. Sharon, R. Stern, A. Felner, and N. Sturtevant, "Conflict-based Search for Optimal Multi-Agent Path Finding," in *AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

[32] T. Bektas, "The Multiple Traveling Salesman Problem: an Overview of Formulations and Solution Procedures," *Omega*, vol. 34, no. 3, pp. 209 – 219, 2006.

[33] A. Asadpour, M. X. Goemans, A. Madry, S. O. Gharan, and A. Saberi, "An $O(\log n/\log\log n)$-Approximation Algorithm for the Asymmetric Traveling Salesman Problem," *Operations Research*, vol. 65, no. 4, pp. 1043–1061, 2017.

[34] G. N. Frederickson, M. S. Hecht, and C. E. Kim, "Approximation Algorithms for some Routing Problems," in *17th Annual Symposium on Foundations of Computer Science*, 1976, pp. 216–227.

[35] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Pearson, 1993.

[36] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal Variants of the Conflict-based Search Algorithm for the Multi-agent Pathfinding Problem," in *Symposium on Combinatorial Search*, 2014.

[37] J. Pearl and J. H. Kim, "Studies in Semi-Admissible Heuristics," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 4, pp. 392–399, 1982.

[38] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 2, no. 4, pp. 100–107, 1968.

[39] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. WH Freeman & Co., 1990.

[40] I. Dumitrescu and N. Boland, "Improved Preprocessing, Labeling and Scaling Algorithms for the Weight-Constrained Shortest Path Problem," *Networks: An International Journal*, vol. 42, no. 3, pp. 135–153, 2003.

[41] W. M. Carlyle, J. O. Royset, and R. Kevin Wood, "Lagrangian Relaxation and Enumeration for Solving Constrained Shortest-Path Problems," *Networks: An International Journal*, vol. 52, no. 4, pp. 256–270, 2008.

[42] Y. Li, J. Harms, and R. Holte, "Fast Exact Multiconstraint Shortest Path Algorithms," in *IEEE International Conference on Communications*. IEEE, 2007, pp. 123–130.

[43] "General Transit Feed Specification." [Online]. Available: https://developers.google.com/transit/gtfs/
[44] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer Programming Formulation of Traveling Salesman Problems," *Journal of the ACM (JACM)*, vol. 7, no. 4, pp. 326–329, 1960.
[45] L. Cohen, T. Uras, T. S. Kumar, H. Xu, N. Ayanian, and S. Koenig, "Improved Solvers for Bounded-Suboptimal Multi-Agent Path Finding." in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2016, pp. 3067–3074.
[46] M. Müller-Hannemann, F. Schulz, D. Wagner, and C. Zaroliagis, "Timetable Information: Models and Algorithms," in *Algorithmic Methods for Railway Optimization*. Springer, 2007, pp. 67–90.
[47] M. M. Solomon, "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints," *Operations research*, vol. 35, no. 2, pp. 254–265, 1987.
[48] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.
[49] H.-T. Kung, F. Luccio, and F. P. Preparata, "On Finding the Maxima of a Set of Vectors," *Journal of the ACM (JACM)*, vol. 22, no. 4, pp. 469–476, 1975.
[50] J. H. Halton, "On the Efficiency of certain Quasi-Random Sequences of Points in Evaluating Multi-Dimensional Integrals," *Numerische Mathematik*, vol. 2, no. 1, pp. 84–90, 1960.

APPENDIX I
TASK ALLOCATION: ADDITIONAL DETAILS AND PROOFS

We present a full and extended version of the MERGESPLITTOURS algorithm (Algorithm 2) for the task allocation layer. Figure 2 illustrates the behaviour of MCT, which provides an approximate solution for the m-MVP problem (Definition 1). The algorithm consists of three main steps:

Step 1 (lines 1): Generate a collection of $t \geqslant 1$ tours $T_1, \ldots, T_t$, for some $t \geqslant 1$, such that every package $p \in V_P$ is covered by exactly one tour, and the total distance of the tours is minimized. This step is achieved by solving the minimal-connecting tours (MCT) problem (see Table I). 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. This assignment implicitly represents the desired collection of tours $T_1, \ldots, T_t$, as described above. The reason behind why such an assignment breaks into a collection of tours is discussed in Lemma 2 below.

Step 2 (lines 2-10): The $T_1, \ldots, T_t$ tours are merged in an iterative fashion, until a single tour T is generated. We first identify $t \geqslant 1$ connected depot sets $\mathbb{D} = \{D_1, \ldots, D_t\}$, which are induced by the MCT solution (line 2). That is, every D_i consists of all the depots that belong to one specific tour T_i encoded by $\boldsymbol{x}$. We then perform a merging routine which merges the tours and consequently merges the connected depot sets. This routing iterates over all combinations of $D, D' \in \mathcal{D}, d \in D, d' \in D'$ (lines 5-8), and chooses $(d, d'), (d', d)$, such that $c_{dd'} + c_{d'd}$ is minimized. Then $\boldsymbol{x}$ and $\mathbb{D}$ are updated accordingly (lines 9, 10). For a given $\mathbb{D}$ and $d \in V_D$, the notation $\mathbb{D}(d)$ represents the depot component $D \in \mathbb{D}$ that d belongs to.

Step 3 (lines 6-14): The tour T is partitioned into m paths $\{P_1, \ldots, P_m\}$ such that the length of every path is proportional to the length of T divided by m. Additionally, every path P_i starts and ends in a depot, but not necessarily the same one. This step is reminiscent to an algorithm presented in [34] for m-TSP in undirected graphs.

Algorithm 2: MERGESPLITTOURS-FULL

Input: Allocation graph $G_A = (V_A, E_A)$, with $V_A = V_D \cup V_P$, number of agents $m \geqslant 1$;
Output: Paths $\{P_1, \ldots, P_m\}$, such that every package is visited exactly once;
$\boldsymbol{x} := \{x_{uv}\}_{(u,v)\in E_A} \leftarrow \text{MCT}(G_A, V_P)$;
$\mathbb{D} := \{D_1 \ldots, D_t\} \leftarrow \text{CONNECTEDDEPOTS}(G_A, \boldsymbol{x})$;
while $|\mathbb{D}| > 1$ **do**
  $c_{\min} \leftarrow \infty, d_{\min} \leftarrow \emptyset, d'_{\min} \leftarrow \emptyset$;
  for $D, D' \in \mathbb{D}, D \neq D', d \in D, d' \in D'$ **do**
    if $c_{dd'} + c_{d'd} < c_{\min}$ **then**
      $c_{\min} \leftarrow c_{dd'} + c_{d'd}$, $d_{\min} \leftarrow d, d'_{\min} \leftarrow d'$;
  $x_{d_{\min} d'_{\min}} \leftarrow 1, x_{d'_{\min} d_{\min}} \leftarrow 1$;
  $\mathbb{D} \leftarrow \mathbb{D} \setminus \{\mathbb{D}(d_{\min}), \mathbb{D}(d'_{\min})\} \cup \{\mathbb{D}(d_{\min}) \cup \mathbb{D}(d'_{\min})\}$;
$T := (d_1, p_1, \ldots, p_{\ell-1}, d_\ell) \leftarrow \text{GETTOUR}(G_A, \boldsymbol{x})$;
$i \leftarrow 1$; $j \leftarrow 1$;
for $i = 0$ to m **do**
  $P_i \leftarrow \emptyset, L_i \leftarrow 0$;
  while $L_i \leqslant \text{LENGTH}(T)/m$ **and** $\ell < t$ **do**
    $L_i \leftarrow L_i + c_{d_j p_j} + c_{p_j d_{j+1}}$;
    $P_i \leftarrow P_i \cup \{(d_j, p_j), (p_j, d_{j+1})\}$;
    $j \leftarrow j + 1$;
return $\{P_1, \ldots, P_m\}$;

A. Completeness and optimality

In preparation to the proof Theorem 1, we have the following lemma, which states that MCT produces a collection of pairwise-disjoint tours. Henceforth we assume that that G_A is strongly connected and that $G_A(V_D)$ is a directed clique.

Lemma 2. *Let $\boldsymbol{x}$ be the output of $\text{MCT}(G_A, V_P)$. Then there exists a collection of vertex-disjoint tours $T_1, \ldots, T_{m'}$, such that for every $(u, v) \in E_A$ such that $x_{uv} > 0$, there exists T_i in which (u, v) appears exactly x_{uv} times.*

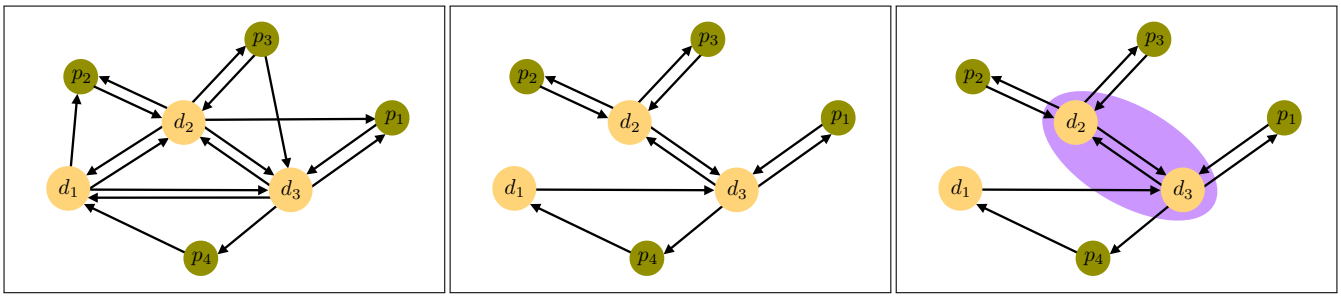
Proof. By definition of MCT, for every $p \in V_p$ there exists precisely one incoming edge (d, p) and one outgoing edge (p, d') such that $x_{dp} = x_{pd'} = 1$. Also, note that by Equation (5), the in-degree and out-degree of every $d \in V_D$ are equal to each other. Thus, an Eulerian tour can be formed, which traverses every edge (u, v) exactly x_{uv} times. □

We are ready for the main proof.

Proof of Theorem 1. First, note that after every iteration of the "while" loop, the updated assignment $\boldsymbol{x}$ still represents a collection of tours. Second, 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.

Next, let OPT be the optimal solution to m-MVP. That is, there exists m paths $\{P_1^*, \ldots, P_m^*\}$ which represent the solution to m-MVP, and for every $i \in [m]$, $|P_i^*| \leqslant \text{OPT}$. Observe that

$$\sum_{i=1}^{m} |P_i| \leqslant m \cdot \max_{i\in[m]} |P_i^*| = m \cdot \text{OPT},$$



(a) (b) (c)

Fig. 2: Three key steps in the MERGESPLITTOURS algorithm for delivery sequence allocation: (a) The allocation graph is defined for the depots $d_{1:3}$ and packages $p_{1:4}$. (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.

where $\{P_1, \ldots, P_m\}$ is the result of MERGESPLITTOURS. Next, by definition of α, we have that $|T| \leqslant m \cdot \text{OPT} + m\alpha$. Lastly, by definition of β we have that

$$T_i \leqslant |T|/m + \beta \leqslant \text{OPT} + \alpha + \beta. \qquad \square$$

B. Computational complexity

We conclude this section with an analysis of the computational complexity of MERGESPLITTOURS. Recall that we identified the main bottleneck of MERGESPLITTOURS to be the solution computation for MCT. We proceed to prove Lemma 1, which states such a solution to MCT can be obtained via a linear relaxation.

Proof. The main observation to make is that MCT can be transformed into a minimum-cost circulation (MCC) problem. If all edge capacities are integral, the linear relaxation of MCC enjoys a totally unimodular constraint matrix form [35]. Hence, the linear relaxation will necessarily have an integer optimal solution, which will be a fortiori an optimal solution to the original MCF problem.

In the current representation of MCT, eq. (4) does not correspond to a circulation problem. However, we can introduce a small modification to G_A which would allow us to recast it as circulation. Consider the graph $G' = (V', E')$ such that

$$\begin{aligned} V' :=& V_D \cup V_P \cup \{p' | p \in V_P\}, \\ E' :=& \{(d, p) | d \in V_D, p \in V_P, (d, p) \in E\} \\ & \cup \{(p', d) | d \in V_D, p \in V_P, (p, d) \in E\} \\ & \cup \{(p, p') | p \in V_P\} \end{aligned}$$

and we require that for every such (p, p'), $x_{pp'} = 1$. $\square$

APPENDIX II
MAPF-TN: ADDITIONAL DETAILS

We now elaborate on two aspects of multi-agent pathfinding with transit networks (MAPF-TN) that we alluded to in Section IV. First, we discuss how we extend the notion of conflict handling in Conflict-Based Search to the capacity conflicts of MAPF-TN, where more than one agent can use a transit edge. Second, we discuss two important speedup techniques that improve the empirical performance of our MAPF-TN solver, without sacrificing bounded suboptimality.

A. Capacity Conflicts in (E)CBS

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 multi-agent level search tree of Conflict-Based Search (CBS) and any of its modified variants. In MAPF-TN, however, transit edges in general have capacity $C(e) \geqslant 1$. Consider a solution generated during a run of Enhanced CBS that has assigned to some transit edge $p > C(e) > 1$ drones. In order to guarantee bounded sub-optimality 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 being restricted from using the transit edge in question.

As we pointed out in Section V-B, conflict resolution is a significant bottleneck for solving large MAPF-TN instances. In our experiments, we generated all constraint subsets of a capacity conflict, however, pathological scenarios may arise where this significantly degrades performance in practice (our ultimate yardstick). Whether there exists a principled way to analyse constraint set enumeration and suboptimality and how this can be efficiently implemented in practice are both important questions for future research.

B. Speedup Techniques

The NP-hardness of multi-agent path finding [22] and the additional computational challenges of MAPF-TN (path energy constraint; large and dense graphs) make empirical performance paramount, given our real-world scenarios and emphasis on scalability. We now discuss some speedup techniques that improve the efficiency of the low-level search while maintaining its bounded sub-optimality (which in turn ensures bounded sub-optimality of the overall solution, as per Enhanced CBS). Certainly, these techniques are not exhaustive; there is an entire body of work in transportation planning devoted to speeding up algorithm running times [29]. We devised and implemented two simple methods.

1) Preprocessing Public Transit Networks: Focal-MCSP can become a bottleneck when it has to be run multiple times (at least twice for each agent's current dpd' task and more in case of conflicts). Its performance depends significantly on the availability and quality of admissible heuristics, i.e., heuristics that *underestimate* the cost to the goal, for the

objective (elapsed time) and constraint (distance traversed, a surrogate for the energy expended). The public-transit network for a given area is usually known in advance and follows a pre-determined timetable. We can analyze and preprocess such a network to obtain admissible heuristics. These can then be used for multiple instances of MAPF-TN throughout the day, while searching for paths to a specific package delivery location.

For the objective function, i.e., the elapsed time, a lower bound is typically the time to fly directly to the goal, without deviating and waiting to board public transit (of course, taking such a route in practice is usually infeasible due to the distance constraint). Therefore, we define the heuristic simply as:

$$h_T(v, vg) = \frac{\|vg - v\|}{\sigma} \tag{6}$$

where σ is the average drone speed, vg is the goal node and v is the node being expanded. The above heuristic will be admissible, i.e., be a lower bound on elapsed time if the average drone speed is higher than average transit speed. This assumption is typically true for the transit vehicles we consider, given that they are required to wait at stops for people to get on. A more data-driven estimate can be obtained by analyzing actual flight times, but that is out of the scope of this work.

For the constraint function, i.e., the distance traversed, we use a heuristic based on extensive network preprocessing. For a given transit network in the area of operation, we consider the minimal time window such that every instance of a transit vehicle trip in that network can start and finish (as per the timetable). We then create the so-called *trip metagraph*, whose set of vertices is $V_D \cup V_P \cup V_{\mathcal{T}}$, where, from our earlier notation, V_D and V_P are the sets of depot and package vertices respectively. Each vertex $v_\tau \in V_{\mathcal{T}}$ represents a single transit vehicle trip R_τ, and encodes its sequence of time-stamped stops (we will discuss what this means in practice shortly). The trip metagraph is complete, i.e., there is an edge between every pair of vertices.

We now define the cost of energy expenditure, i.e., distance traversed for each edge $e = (u \rightarrow v)$ in the trip metagraph. If $u, v \in V_{\mathcal{T}}$ correspond to trips R_τ and $R_{\tau'}$ respectively,

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

where, as before, $v.t$ refers to the time-stamp of the stop v for that particular trip. The edge cost here is thus *the shortest distance between stops that can be traversed by the drone in the difference between time stamps*. If $u, v \in V_D \cup V_P$, we simply set $N(e) = \|v - u\|$, the direct flight distance between the locations. For all other edges, i.e., where one of u or v corresponds to a trip R_τ and the other to a depot or package location (in either direction), we set

$$N(e) = \min_{u \in R_\tau} \|v - u\|$$

and in such cases, the cost for edge $(v \rightarrow u)$ is equal to that of $(u \rightarrow v)$. This concludes the assignment of edge costs.

Given the complete specification of the edge cost function, we now run Floyd-Warshall's algorithm [48] on the trip metagraph to get a cost matrix $\bar{N}_{\mathcal{T}}$, where $\bar{N}_{\mathcal{T}}(u, v)$ is the cost of the shortest-path from v_τ to $v_{\tau'}$ on the trip metagraph. Intuitively, this cost matrix encodes the *least 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 is shorter.

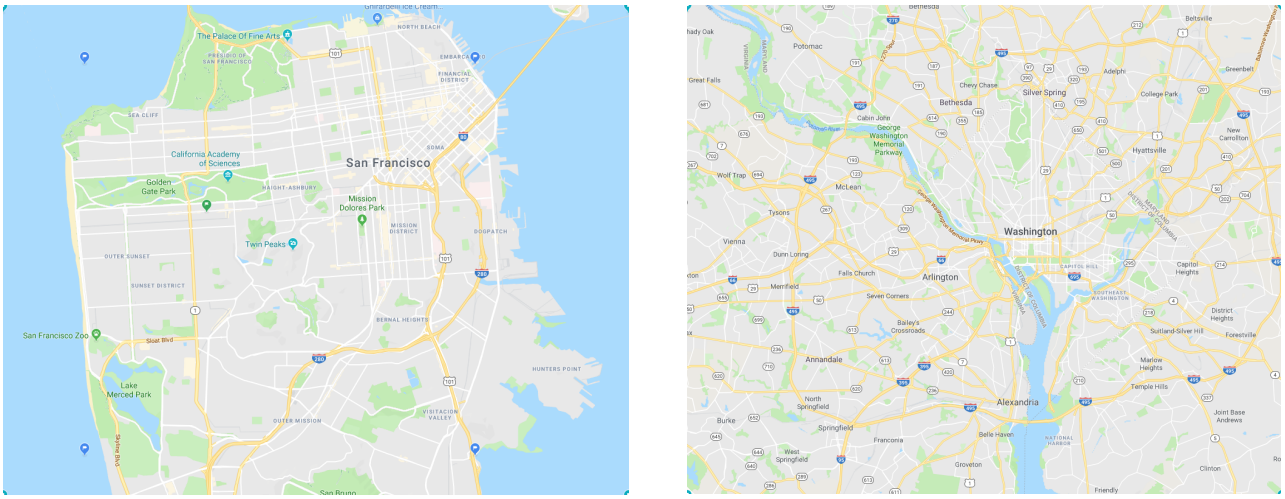
We can now define the goal-directed heuristic function h_N for the distance traversed. Let the goal node for a query to Focal-MCSP be $vg \in V_D \uplus V_P$. We want the heuristic value for the operation graph node $v \in V_O \equiv (V_D \uplus V_P \cup V_{TN})$ that is expanded during Focal-MCSP. If $v \in V_D \uplus V_P$ is a depot or package, we set $h_N(v, vg) = \bar{N}_{\mathcal{T}}(v, vg)$. Otherwise, $v \in V_{TN}$ is a transit vertex. Recall that each transit vertex is a stop that is associated with a corresponding transit trip. Let the trip associated with $v \in V_{TN}$ be R_τ. We then set $h_N(v, vg) = \bar{N}_{\mathcal{T}}(v_\tau, vg)$, where $v_\tau \in V_{\mathcal{T}}$ is the trip metagraph vertex corresponding to the trip R_τ. The heuristic h_N as defined above is admissible, i.e. 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 several instances of MAPF-TN throughout a day, with traffic delays and other disruptions to the timetable. However, the handling of dynamic networks and timetable delays is a separate subfield of research in transportation planning [29, 30] and out of the scope of this work. We make the reasonable assumption (made often in transit planning work) that travel times between locations do not vary greatly throughout the day, and we ignore the effect of delays and disruptions to the pre-determined timetable while using our heuristics.

2) Pruning Focal-MCSP search space: As we mentioned in Section IV-A, the edges of the operation graph are not explicitly enumerated but rather implicitly encoded and generated just-in-time during the node expansion stage of Focal-MCSP. An implicit edge set makes the Focal-MCSP search highly memory-efficient by only having to store the vertices of the operation graph. This memory-efficiency comes at the cost of computation time as the outgoing edges of a vertex must be computed during the search. A careful observation of the transit vertices allows us to prune the set of out-neighbors of a vertex expanded during Focal-MCSP, *while still guaranteeing bounded sub-optimality*.

Let $u \in V_O$ be an operation graph vertex that is expanded during Focal-MCSP. Consider all the transit vertices of a transit trip R_τ (if $u \in V_{TN}$ is itself a transit vertex, then consider a trip different from the trip that u lies on). Those transit vertices are candidate out-neighbors for the expanded node u (candidate target vertices of a *time-constrained flight edge* emanating from u and making a connection to the trip R_τ). It may appear that all trip stops in R_τ that the drone can reach in time, i.e., for which $\sigma \times (v.t - u.t) \geqslant \|v - u\|$ (a required condition, as we mentioned in Section IV-A) should be added as out-neighbors.

However, while considering connections to a trip R_τ, we actually need to only add the transit vertices on R_τ that are



(a) (b)

Fig. 3: The geographical bounding boxes for (a) San Francisco (roughly $150\,\mathrm{km}^2$) and the (b) Washington DC Metropolitan area (roughly $400\,\mathrm{km}^2$).

non-dominated in terms of the tuple of time difference and flight distance $(v.t - u.t, \|v - u\|)$. Doing so will continue to ensure bounded sub-optimality of Focal-MCSP. We formalize this observation through a lemma.

Lemma 3. *Let $u \in V_O$ be an operation graph vertex expanded during Focal-MCSP. While considering time-constrained flight connections to trip R_τ, let v_1 and v_2 be two consecutive transit vertices on trip R_τ such that $(v_1.t - u.t, \|v_1 - u\|) \preceq (v_2.t - u.t, \|v_2 - u\|)$. Then, pruning v_2 as an out-neighbor has no effect on the solution of Focal-MCSP.*

The following proof relies heavily on the analysis of A*-MCSP (see, e.g., [42, Section V]), upon which Focal-MCSP is based.

Proof. We assume that both v_1 and v_2 are physically reachable by the drone, i.e., $\sigma \times (v.t - u.t) \geqslant \|v - u\|$ for $v = v_1, v_2$ (otherwise they would be discarded anyway).

Note that $(v_1 \to v_2)$ is a transit edge, so the flight distance $N(v_1 \to v_2) = 0$ by definition. The Focal-MCSP algorithm tracks the *objective* (traversal time) and *constraint* (flight distance) values of partial paths to nodes. It discards a partial path that is dominated by another on both metrics. Consider the only two possible partial paths to v_2 from u, $u \to v_2$ and $u \to v_1 \to v_2$.

Let the weight constraint accumulated on the path thus far to the expanded node 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 traversal distance weight at v_2 for $u \to v_2$ is $Wu + N(u \to v_2) = Wu + \|v_2 - u\|$. On the other hand, 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 the original assumption. Focal-MCSP will always discard the partial path $u \to v_2$ and instead prefer the alternative, $u \to v_1 \to v_2$. Therefore, pruning v_2 as an out-neighbor will have no effect on the solution of Focal-MCSP, which thus continues to be bounded sub-optimal. □

The intuition is that if a transit connection is useful to make, then a stop that is both earlier and closer in distance than another will always be preferred. The above result was for *consecutive* vertices on a transit trip; we can extend it to the full sequence of vertices on the trip R_τ by induction.

We use Kung's algorithm [49] to find the non-dominated elements of the set of transit trip vertices. For two criteria functions, as in our case, Kung's algorithm yields a solution in $\mathcal{O}(n \log n)$ time, where n is the size of the set and the bottleneck is due to sorting the set as per one of the criteria. In our specific case, since the transit trip vertices are already sorted in increasing order of the traversal time criterion, we can add out-neighbors for a transit trip in $\mathcal{O}(n)$ time, which is as fast as we could have done anyway.

Appendix III
Surrogate Travel Time Estimate

At the end of Section II, we mentioned the role of the surrogate estimate for travel time between two depots/packages used by MergeSplitTours for the task allocation. We also briefly discussed the actual surrogate estimate we use in our approach. We now provide some more details about how the estimate is actually computed in a preprocessing step and then used during runtime. In Appendix IV, we quantitatively compare the surrogate estimate that we use to the direct flight time between two locations, in terms of the computation time and solution quality of the MAPF-TN layer.

Consider the given geographical area of operation, encoded as a bounding box of coordinates (Figure 3 illustrates both areas). During preprocessing, we generate a representative set of locations across the area. To ensure good coverage, we use a quasi-random low dispersion sampling scheme [50] to compute the locations. This set of locations induces a Voronoi decomposition [48] of the geographical area where the locations are the sites. Every point in the bounding box is associated with the nearest element (by the appropriate distance metric) in the set of locations. We then choose a representative time window of transit for the area. Between

every pair of locations in the set, we compute and store the travel time using the transit network (with the same Focal-MCSP parameters we use for MAPF-TN).

During runtime, at the task allocation layer, we need the estimated travel time between two depot/package locations $v, v' \in V_D \cup V_P$. Each of v and v' has a corresponding nearest representative location (the site of its Voronoi cell). We then look up the precomputed travel time estimate between the corresponding sites and use that value in MERGESPLITTOURS. The implicit assumption is that the travel time between the representative sites is the dominating factor compared to 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 representative location is the same, we use the direct flight time between v and v', i.e., $\|v' - v\|/\sigma$. The assumption here is that v and v' are more likely to share a cell if they are close together, and in that case, the drone is more likely to be able to fly directly between them anyway.

The number of representative locations for a given area is an engineering parameter. For our results, we use 100 points in San Francisco and 150 points in Washington DC. For the quasi-random sampling scheme we use, the higher the number of sampled points, the lower the dispersion, i.e., the better the coverage of the area, and typically, the better is the quality of the surrogate estimate. Domain knowledge about the transit network and travel time distribution in a given urban area may yield a higher quality surrogate than our domain-agnostic approach.

APPENDIX IV
FURTHER RESULTS

We now elaborate on three additional aspects of our results, as we alluded to in Section V. First, we provide a more extensive analysis of the behavior of our layer for multi-agent path finding with transit networks (MAPF-TN). Second, we compare two different replanning strategies to solve for a sequence of drone delivery tasks. Third, we quantitatively compare the effect of two different surrogate travel time estimates.

A. Further Insights of MAPF-TN Results

We will now supplement our discussion in Section V-B on prominent observations of the behavior of the MAPF-TN layer, based on the numbers in Table III. With regards to scalability, recall that each low-level search is actually *two* Focal-MCSP searches (from $d \to p$ and $p \to d'$) that are concatenated, so the effective number of agents (from a typical MAPF perspective) is actually $2m$ and not m. This observation only serves to strengthen our scalability claim. Since our MAPF-TN solver is built upon Conflict-Based Search, the key factor affecting plan time is the generation and resolution of conflicts, which we have discussed in detail already. We also discussed how the number of depots and the ratio of depots to agents affects the likelihood of conflicts. Depots or warehouses are highly expensive to construct in practice. Thus, in a given area, the placement of depots (that we generate randomly for our benchmarks) can have a significant impact on computation time and scalability; indeed, that is a key question for future work.

TABLE IV: (All times are in seconds) A comparison of replanning strategies for a subset of the $\{l, m\}$ scenarios from Table III for the San Francisco network. We run 20 different trials for each setting and depict the average values in each case.

$\{l, m\}$	Replan-1		Replan-m	
	Replan Time	Soln. Mksp.	Replan Time	Soln. Mksp.
$\{5, 10\}$	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

The order of magnitude higher runtimes for Washington DC is worth commenting on a bit more. Note that we are using the same drone parameters and transit capacity settings for Washington DC, which has an area nearly three times that of SF, and a transit network nearly twice as big. Consequently, the need for using transit to satisfy deliveries is greatly increased (notice how the average transit usage is reliably higher than for SF). Additionally, the bus network for Washington DC is more sparse in the outskirts and suburban areas. Thus, the bus network becomes more of a bottleneck than for San Francisco, leading to more conflicts. Even when there are no conflicts, the average Focal-MCSP search times increase because more of the larger transit graph is being explored by the search algorithm.

With regards to solution quality (makespan), we briefly commented on the real-world significance that even for a large metropolitan area of $400\,\mathrm{km}^2$, the longest delivery in a set of m tasks is under 2 hours. We used a representative transit window that is largely replicated throughout the rest of the day; therefore, for a given business day of, say, 12 hours, we can expect any drone to make *at least* 6 deliveries (and typically many more).

B. Replanning Strategies

We have previously discussed how our MAPF-TN solver based on Enhanced Conflict-Based Search (ECBS) computes paths for a single dpd' task for each drone. However, drones will typically be assigned to a sequence of deliveries by the task allocation layer. Rather than 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 computation time is negligible compared to the actual solution execution time (compare the 'Plan Time' and 'Makespan' columns in Table II); therefore, a receding horizon strategy appears to be quite reasonable.

Two natural replanning strategies emerge in such a context: replanning *only* for the finished drone, while maintaining the paths of all the other drones, which we call Replan-1, and replanning for all drones, from each of their current states, which we call Replan-m. In terms of the tradeoff between computation time and solution quality, these two approaches are at the opposite ends of a spectrum. The Replan-m strategy will be optimal among replanning strategies, while being the most computationally expensive as it recomputes m paths; on the other hand, Replan-1 requires only the computation of a single path with the remaining $m - 1$ paths imposing boarding and capacity constraints.

TABLE V: We compare our MAPF-TN results from Table III (Average 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 clarity of viewing, we split out the results by city/network into two separate tables. **The values for the Preprocessed sub-table are copied over from Table III.**

San Francisco

$\{l, m\}$	Preprocessed		Direct Flight	
	Plan Time	Soln. Mksp.	Plan Time	Soln. Mksp.
$\{5, 10\}$	1.07	2464.8	2.34	2763.6
$\{5, 20\}$	2.79	2555.5	5.94	3014.6
$\{5, 50\}$	8.29	3485.8	8.04	3454.1
$\{10, 20\}$	0.68	2082.7	0.61	1832.7
$\{10, 50\}$	2.96	2671.1	1.96	2408.7
$\{10, 100\}$	4.17	3084.5	4.23	2774.1
$\{20, 50\}$	0.26	1054.8	0.23	1396.9
$\{20, 100\}$	0.66	1457.6	0.43	1115.5
$\{20, 200\}$	1.7	1824.4	1.49	1387.9

Washington DC

$\{l, m\}$	Preprocessed		Direct Flight	
	Plan Time	Soln. Mksp.	Plan Time	Soln. Mksp.
$\{5, 10\}$	7.17	5006.5	11.4	4836.6
$\{5, 20\}$	12.4	5819.2	23.0	5711.6
$\{5, 50\}$	57.6	6861.7	56.8	6002.6
$\{10, 20\}$	8.78	5195.6	14.4	5204.2
$\{10, 50\}$	19.8	6316.7	45.9	5684.3
$\{10, 100\}$	39.7	6889.8	53.9	6594.7
$\{20, 50\}$	11.2	5086.9	11.4	4694.7
$\{20, 100\}$	19.2	5400.9	28.1	5668.9
$\{20, 200\}$	26.2	6050.1	88.5	5419.2

To evaluate the two replanning strategies, we use the same setup that we did for evaluating MAPF-TN in Section V. For each MAPF-TN solution (one path for each drone), we consider the drone that finishes first among the m drones (since we use a continuous time representation, ties are highly unlikely in practice). In the case of Replan-1, we run Focal-MCSP for the drone with the various constraints induced by the remaining paths of the other agents. We update the (m-agent) solution with the new path (updating makespan if need be). In the case of Replan-m, we run Enhanced CBS for the m agents with their current states (at that time) as their initial state; this yields another (m-agent) solution.

In Table IV, we compare the average makespan and computation times of the m-agent solutions resulting from the two strategies. We use a representative subset of the {Depots, Agents} scenarios that we used in Table III; few depots with a lower agent/depot ratio ($\{5, 10\}$); few depots with a higher ratio ($\{5, 20\}$); and similarly for many depots ($\{20, 50\}$ and $\{20, 100\}$). It is clear that Replan-1 achieves similar quality solutions as Replan-m does, at fairly lower computational cost. This motivates our decision to use Replan-1 in practice.

In principle, we can design scenarios where Replan-1 has a much greater solution quality gap against Replan-m than what we see in Table IV. However, the Replan-1 strategy is sub-optimal only when (i) the $(m-1)$ unfinished drone paths actually conflict with the new Focal-MCSP path of drone i, that has just finished, and (ii) resolving the conflict(s) would have prioritized the path of drone i over the others. In practice, it is not very likely that both of these conditions will hold together, especially when there are many depots and some drones can fly directly to their next target; in our trials for $l = 20$, the sub-optimality condition for Replan-1 never holds, which is why the makespans for those two rows are exactly the same for both strategies.

C. Comparison of Surrogate Estimates

We now compare the effect of two different surrogate travel time estimates — the approximate travel time between representative locations in the city using the transit (as described in Appendix III) and the direct flight time between two locations, ignoring the transit. For the results in Table III, recall that we ran MAPF-TN on the first dpd' task for each drone obtained from the result of MERGESPLITTOURS; for those results, MERGESPLITTOURS used the preprocessed surrogate for the allocation graph edge costs. As a comparison, we rerun the exact same scenarios as in Table III, but this time, we use the direct flight time (ignoring the transit) as the edge cost for MERGESPLITTOURS. We compare the two primary performance factors, plan time and solution makespan, for both surrogates in Table V.

The results of the comparison are intuitive in some respects, and counter-intuitive in others. We expect the flight time surrogate to be a poor estimate in scenarios where transit is likely to be used, because the allocation does not account for the effect of using transit on the solution quality. Accordingly, we do observe a large difference in plan time between Preprocessed and Direct Flight for the low-depot and high agent-to-depot ratio settings in San Francisco and all of the settings in Washington DC. In both cases, the $\{5, 50\}$ setting has similar plan times, probably because conflict resolution is the predominant factor regardless of the allocation.

On the other hand, the absence of a consistent and significant difference in the solution quality (makespan) between the two surrogates is somewhat surprising. In fact, for most settings of Washington DC, the solutions with the direct flight time surrogate actually have lower makespan than those for the preprocessed surrogate. One hypothesis is that the set of representative samples is insufficiently dense, and increasing the number of samples would improve the resulting solution makespan. This specific aspect of our framework is worth investigating in further work.