Hierarchical Large Scale Multirobot Path (Re)Planning

Lishuo Pan, Kevin Hsu, and Nora Ayanian

***Abstract*— We consider a large-scale multi-robot path planning problem in a cluttered environment. Our approach achieves real-time replanning by dividing the workspace into cells and utilizing a hierarchical planner. Specifically, we propose novel multi-commodity flow-based high-level planners that route robots through cells with reduced congestion, along with an anytime low-level planner that computes collision-free paths for robots within each cell in parallel. A highlight of our method is a significant improvement in computation time. Specifically, we show empirical results of a 500-times speedup in computation time compared to the baseline multi-agent pathfinding approach on the environments we study. We account for the robot's embodiment and support non-stop execution with continuous replanning. We demonstrate the real-time performance of our algorithm with up to 142 robots in simulation, and a representative 32 physical Crazyflie nano-quadrotor experiment.**

I. Introduction

Large fleets of robots, such as those used in warehouse operations [1], disaster response [2], and delivery [3], demand coordination solutions that adjust in real time to changing goals. In this work, we present a real-time lifelong hierarchical method for navigating a large team of robots to independent goals in a large, cluttered environment that guarantees collision avoidance. By lifelong, we mean robots can enter and exit the space, and can receive another goal at any time, as they would in a warehouse or delivery problem. Our approach partitions the space into disjoint cells, allowing planning algorithms to run concurrently in parallel within each cell. A novel high-level planner routes robots through the partition, while a low-level anytime multi-agent pathfinding (MAPF) algorithm navigates robots to local goals within each cell in parallel. The real-time property holds as long as there are not too many cells or robots in a workspace; the limits for real-time operation are empirical and problem-specific, however, we demonstrate real-time performance for 142 robots in simulation with a 25-cell partition.

We are primarily interested in unmanned aerial vehicles (UAVs) operating in 3D space, such as in city-scale on-demand UAV package delivery; however, our approach applies to robots operating in 2D as well. We present two approaches for high-level planning depending on the problem's requirements: 1) an egocentric greedy approach that always operates in real-time and 2) a novel high-level planner that routes robots through the partition using multi-commodity flow (MCF) [4]. There are tradeoffs between these two approaches. The egocentric greedy planner operates in real-time regardless of the number of cells; however, it has no mechanism for distributing robots, thus it can result in congestion and longer low-level planning times within some cells. On the other hand, the MCF-based approach eases cell congestion by regulating the flow of robots into each cell while ensuring bounded-suboptimal inter-cell routing; thus, it can be useful in environments such as urban UAV package delivery, where different types of cells (e.g., residential vs. highway) may have different limits on the influx of robots. The MCF-based planners can operate in real-time under certain conditions, thus allowing for lifelong replanning while reducing congestion, which leads to faster, real-time low-level planning within each cell.

This work was supported by NSF grants 1837779, 2317145, 2311967, 2330942. All authors are with the Department of Computer Science, Brown University, Providence, RI 02912, USA. Email: {lishuo_pan, kevin_hsu, nora_ayanian}@brown.edu

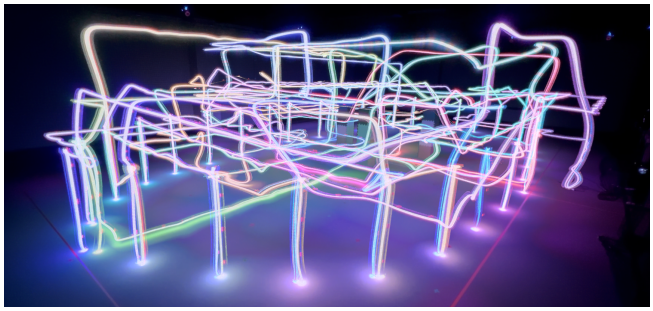


Fig. 1: Long exposure of 32 quadrotors navigating a cluttered environment.

The low-level planner prohibits collisions while respecting the robots' geometric shapes. A novel cell-crossing protocol allows robots to transition between cells without stopping in midair. Combined with the MCF-based planner, this allows real-time computation and safe, non-stop execution of multi-robot plans. The contributions of this work are:

- a hierarchical framework for large-scale multi-robot real-time coordination that significantly reduces computation time compared to the baseline MAPF solver, while resulting in a moderately suboptimal solution; and
- novel multi-commodity flow-based high-level planners, MCF/OD and one-shot MCF, that reduce congestion by regulating the influx of robots to each cell.

We demonstrate the algorithm in simulation with up to 142 robots and in physical robot experiments with 32 nano-quadrotors in cluttered environments, shown in Fig. 1.

II. Related Work

Centralized approaches to multi-robot planning [5], [6] face substantial computational challenges due to their theoretical hardness [7], prohibiting real-time replanning for many-robot systems. Decentralized online approaches, on the other hand, can result in deadlocks, livelocks, congestion,

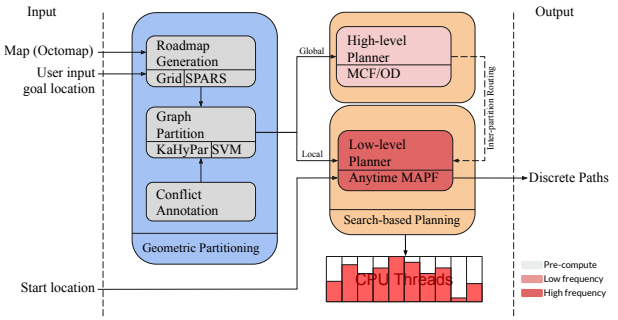


Fig. 2: The user inputs a map, start, and goal locations. Our approach generates a geometric partition, distributes robots among cells (hierarchical planner), and coordinates them within each cell in parallel.

collision, and reduced efficiency. RLSS [8] generates trajectories based on a single-robot planner without a deadlock-free guarantee. Furthermore, control barrier functions [9], or reactive control synthesis methods, are prone to deadlocks. Distributed MPC [10] results in deadlocks when narrow corridors are present. In cluttered environments, a buffered Voronoi cell-based algorithm [11] also suffers from potential deadlocks, and an algorithm using relative safe flight corridor [12] leads to collisions. In the present work, we aim to address these problems and facilitate real-time MAPF at the discrete planning phase.

Search-based MAPF solvers generate deadlock- and collision-free paths, but the complexity of optimal solutions scale exponentially with the number of agents [7]. Bounded-suboptimal algorithms have been proposed to overcome this complexity, but poor scalability still prevents their application to real-time coordination for large teams. To address this, partition-based MAPF [13], [14] divides the workspace into cells, reducing the complexity in each cell. However, inter-cell routing is either single-agent based [13] or the solution of a multi-commodity flow problem constrained on single-robot shortest paths [14] without congestion-awareness, leading to hard instances within regions. Instead, we propose a novel congestion-aware inter-cell routing algorithm to distribute robots while maintaining bounded-suboptimality. Furthermore, the cell-crossing method in [14] is unsuitable for aerial vehicles due to energy expenditure while stationed in hover for the cell-crossing channel to clear. Our novel cell-crossing protocol addresses this drawback.

III. Preliminaries

A. Multi-agent Pathfinding (MAPF) on Euclidean Graph

Consider an undirected graph $\mathcal{G}=(V,E)$ embedded in a Euclidean space, where each vertex $v\in V$ corresponds to a position in the free space $\mathcal{F}$ and each edge $(u,v)\in E$ denotes a path in $\mathcal{F}$ connecting vertices u and v. For N agents, we additionally require the existence of vertices v_g^i and v_s^i, corresponding to the goal and start positions, $\mathbf{g}^i$ and $\mathbf{s}^i\in\mathbb{R}^3$ of robot r^i (superscript i represents robot index), respectfully. At each time step k, an agent can either move to a neighbor vertex $\left(u_k^i, u_{k+1}^i\right)\in E$ or stay at its current vertex $u_{k+1}^i = u_k^i$, where $u_k^i\in V$ is the occupied vertex in the i-th agent's path at time step k. To respect vertex conflict constraints, no two agents can occupy the same vertex simultaneously, i.e., $\forall k, i\neq j: u_k^i \neq u_k^j$. To respect edge conflict constraints, no two agents can traverse the same edge in the opposite direction concurrently, i.e., $\forall k, i\neq j: u_k^i \neq u_{k+1}^j \vee u_{k+1}^i \neq u_k^j$. The objective is to find conflict-free paths $\mathcal{P}^i=\left[u_0^i,\cdots,u_{T-1}^i\right]$, where $u_0^i=v_s^i$ and $u_{T-1}^i=v_g^i$ for all agents, and minimize cost, e.g., the sum over the time steps required to reach the goals of all agents or the makespan T.

B. MAPF for Embodied Agents and Conflict Annotation

Many works address MAPF for embodied agents [15]–[17]. We adopt multi-agent pathfinding with generalized conflicts (MAPF/C), due to its flexibility with different shapes. Generalized conflicts, different from typical MAPF conflicts, include the extra conflicts caused by the robot embodiment [5]. To account for the downwash effect between robots [18], we denote $\mathcal{R}_{\mathcal{R}}(\mathbf{p})$ as the convex set of points representing a robot at position $\mathbf{p}$, i.e., a robot-robot collision model. We follow the conflict annotation in [5] to annotate the graph with generalized conflicts, defined as:

$$\begin{aligned}\mathrm{conVV}(v) &= \{u\in V \mid \\ &\qquad \mathcal{R}_{\mathcal{R}}(\mathrm{pos}(u))\cap\mathcal{R}_{\mathcal{R}}(\mathrm{pos}(v))\neq\emptyset\}\\ \mathrm{conEE}(e) &= \{d\in E \mid \mathcal{R}^*_{\mathcal{R}}(d)\cap\mathcal{R}^*_{\mathcal{R}}(e)\neq\emptyset\}\\ \mathrm{conEV}(e) &= \{u\in V \mid \mathcal{R}_{\mathcal{R}}(\mathrm{pos}(u))\cap\mathcal{R}^*_{\mathcal{R}}(e)\neq\emptyset\},\end{aligned}$$

where $\mathrm{pos}(u)\in\mathbb{R}^3$ returns the position of vertex u. $\mathcal{R}^*_{\mathcal{R}}(e)$ is the *swept* collision model representing the set of points swept by the robot when traversing edge e.

IV. Problem Formulation

Consider a time-varying number of homogeneous non-point robots in workspace $\mathcal{W}$, which is partitioned into a union of disjoint convex polytopic cells. Robots must reach specified individual goal positions, which change over time, while avoiding collisions with robots and obstacles and obeying maximum cell influx limits $\boldsymbol{\theta}$ (influx refers to the number of robots entering a workspace cell). A motivating scenario is a multi-UAV package delivery system, where the number of UAVs entering certain types of airspaces must be limited, and UAVs exit or enter the workspace to charge or redeploy. The problem above requires solving a path replanning problem that obeys cell influx limits while handling new goal positions and a varying number of robots. While that is the general problem we aim to solve, the present work addresses a critical subproblem: the underlying planner that is repeatedly called to safely and efficiently route the robots through $\mathcal{W}$.

Now, consider N (fixed) homogeneous non-point robots operating in the partitioned workspace. $\mathcal{W}$ contains obstacles defined as unions of convex polytopes $\mathcal{O}_1,\cdots,\mathcal{O}_{N_{obs}}$. We use a robot-environment collision model $\mathcal{R}_{\mathcal{E}}(\mathbf{p})$ that is distinct from $\mathcal{R}_{\mathcal{R}}(\mathbf{p})$. The free space is $\mathcal{F}=\mathcal{W}\backslash(\bigcup_h\mathcal{O}_h)\ominus\mathcal{R}_{\mathcal{E}}(\mathbf{0})$, where $\ominus$ is Minkowski difference. For each robot r^i at initial position $\mathbf{s}^i$, our algorithm finds a path to its goal position $\mathbf{g}^i$ such that there are no collisions (e.g., between robots or between robots and $\bigcup_h\mathcal{O}_h$) and the total number of robots that enter each cell is less than its user-defined influx θ_m (influx limits can vary by cell).

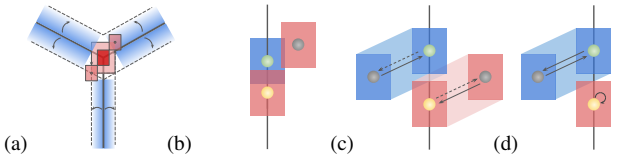


Fig. 3: (a) inter-robot collision configuration $\mathcal{C}_{col}$ and buffered hyperplanes. (b)-(d) vertex-vertex, edge-edge, and edge-vertex generalized conflicts across cells.

To efficiently address the above problem, our framework has three components: geometric partitioning, high-level planner, and low-level planner, as depicted in Fig. 2. Geometric partitioning divides the workspace into disjoint convex cells, where the plan can be computed in parallel. A centralized high-level planner regulates the congestion for each cell while guaranteeing inter-cell routing quality. An anytime low-level planner plans collision-free paths for robots within each cell. Initial planning includes pre-computation (the geometric partitioning), and replanning only involves high- and low-level planning. Once the initial plan is established, our algorithm runs in real-time for replanning.

V. Geometric Partitioning

The geometric partitioning of a bounded workspace consists of three steps: 1) roadmap generation, 2) graph partitioning and spatial linear separation, and 3) local goal generation.

A. Roadmap Generation

A roadmap is an undirected graph, introduced in Sec.III-A, satisfying three properties: 1) connectivity-preserving, i.e., if a path between two points in $\mathcal{F}$ exists, there should be a path in the roadmap as well; 2) optimality-preserving, i.e., the shortest path between two points in $\mathcal{F}$ can be well approximated by a path in the roadmap; and 3) sparse, i.e., have a small number of vertices and edges. In our experiments, we use a 6-connected grid graph, however, it can be generated by other methods, such as SPARS [19].

B. Graph Partitioning and Spatial Linear Separation

Our method partitions the workspace into disjoint convex cells and solves a MAPF instance in each cell. Partitioning has two benefits: 1) fewer robots for each subgraph, and 2) decomposed instances can be solved in parallel. While the user can adopt any generic workspace partitioning approach, we propose a method that generates Q convex polytopes. First, we use graph partitioning (KaHyPar [20]) to group the roadmap into Q *balanced* (similar vertex numbers) subgraphs $\mathcal{G}_m{=}(V_m, E_m)$, for $m{=}1, \ldots, Q$. *Balanced* subgraphs lead to cells of similar volume and an even spread of robots.

We further enforce each cell, containing a subgraph, to be a convex polytope. Cell convexity prevents robots from penetrating into neighboring cells before exiting their current cells. We use soft-margin support vector machines (SVM) [21] to compute a hyperplane H_{ml} between vertices of $\mathcal{G}_m$ and $\mathcal{G}_l$, and reassign misclassified vertices. Here, the subscript l denotes the index of the subgraphs with vertices connecting to vertices of $\mathcal{G}_m$. The resulting set of hyperplanes forms the m-th cell, denoted as P_m.

C. Local Goal Generation

For navigating out of a cell, we generate candidate goal states on the faces between adjacent cells. For each cell P_m, we uniformly sample random local goals on the hyperplane H_{ml} and add them as shared vertices to both $\mathcal{G}_m$ and $\mathcal{G}_l$. Despite being shared vertices, local goals generated by partition P_m have in-edges from P_m and out-edges to P_l to avoid collision during cell transit. Thus, this part of the graph is directed. To enable parallel computation, there must be no communication between cells. Thus, the cell roadmap $\mathcal{G}_m$ is modified such that the planned paths are collision-free when crossing cells without information exchange between cells. The following properties should be satisfied:

P1: Robots avoid collision when stationary at vertices (local-goal or non-local-goal vertices) of different cells (generalized vertex-vertex conflict across cells), i.e., $\forall i, j, m \neq l : v_m^i \notin \mathrm{conVV}(v_l^j)$, where the superscript in v_m^i refers to the vertex index and the subscript refers to the cell index.

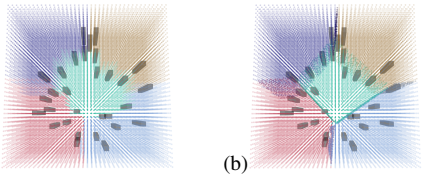
P2: Robots avoid collision when traversing edges between different cells (generalized edge-edge conflict across cells), i.e., $\forall i, k, m \neq l : e_m^i := (v_m^i, v_m^j) \notin \mathrm{conEE}(e_l^k)$, here the superscript in e_m^i refers to the edge index.

P3: Robots avoid collision when one robot is stationary at a vertex while the other robot is traversing an edge of a different cell (generalized edge-vertex conflict across cells), i.e., $\forall i, k, m{\neq}l : e_m^i := (v_m^i, v_m^j), v_l^k \notin \mathrm{conEV}(e_m^i)$.

We depict violations of P1-P3 in Fig. 3b-3d. To prevent conflicts between stationary robots at local-goal and non-local-goal vertices across cells, we buffer the separating hyperplane by the inter-robot collision configuration, $\mathcal{C}_{col}$ (c.f. Fig. 3a), which is computed as $\mathcal{C}_{col}(\mathbf{p}) = \mathcal{R}_{\mathcal{R}}(\mathbf{p}) \oplus \mathcal{R}_{\mathcal{R}}(\mathbf{0})$, where $\oplus$ is the Minkowski sum. The buffering is achieved by modifying the offset $\mathcal{H}_a^{'} = \mathcal{H}_a + \max_{\mathbf{y} \in \mathcal{C}_{col}(\mathbf{0})} \mathcal{H}_{\mathbf{n}} \cdot \mathbf{y}$ of the hyperplane, where $\mathcal{H}_a$ and $\mathcal{H}_{\mathbf{n}}$ are the offset and the normal vector of the hyperplane. Given the buffered hyperplanes, all non-local-goal vertices within the buffered region are removed, preventing collisions between stationary robots at local-goal and non-local-goal vertices. The local goals sampled on the hyperplane are confined within the blue region in Fig. 3a to avoid vertex-vertex conflicts between stationary robots at local goals of different separating hyperplanes. The buffering satisfies P1 between local-goal and non-local-goal vertices across cells and P1, P2, and P3 between non-local-goal vertices across cells. To satisfy P1 between local-goal vertices across cells, we uniformly randomly sample points on the hyperplane and reject those that violate P1. For P2 and P3 between local-goal and non-local-goal vertices, we add the connection between the sampled local-goal vertex to valid (no collision with the environment) non-local-goal vertices within a radius on both sides of the hyperplane. The local goal is removed if any connected edge violates P2 or P3 (see Fig. 3c, 3d). Otherwise, the edges are added to the cell roadmaps. Figure 4 depicts an exemplar partition.

VI. Multi-commodity Flow with Optimal Detour

Our hierarchical approach relies on high-level planning to 1) regulate cell congestion and 2) preserve the bounded-



(a) (b)

Fig. 4: Geometric partitioning with Q=5, before spatial linear separation (a), and after (b). Each cell is a convex polytope in the workspace.

suboptimality of inter-cell routing solutions. Thus, our high-level planner simplifies MAPF instances within cells and leads to real-time replanning. We abstract the partition as a directed graph $\mathcal{G}_p=(V_p, E_p)$, where the vertices (nodes) represent cells and edges connect neighboring cells that share at least one face. Edges are weighted according to Euclidean distance between the cells' centers of mass. The high-level planner finds an inter-cell routing $\mathcal{U}^i = \left[P_s^i, \cdots, P_g^i\right]$ for each robot r^i, where P_s^i and P_g^i are its start and goal cell, satisfying: 1) the influx of cell m is under a user-defined value θ_m, and 2) $cost(\mathcal{U}^i) \leq w_{mcf} \cdot cost(\mathcal{U}^{i,*})$. Here, $w_{mcf} \geq 1$ is a scalar representing the suboptimality bound for the routing solutions. $\mathcal{U}^{i,*}$ is the optimal routing of robot r^i. The influx of a node represents the number of robots entering the cell; we define influx formally in the following section.

A. High-level Planning Formulation

The SOTA partition-based MAPF solvers [13], [14] suffer from cell congestion, leading to hard instances in certain cells, causing computational bottlenecks. To address this, we formulate the inter-cell routing as a variant of the MCF problem and propose multi-commodity flow with optimal detour (MCF/OD), which optimally distributes robots among cells such that the number of robots entering any intermediate cell m (not the start or goal cells of the robot teams) is under a user-defined value θ_m, if a solution exists.

Specifically, robots sharing the same start cell P_s and goal cell P_g are one commodity c_{sg}. The commodities set $C = \{c_1, \cdots, c_O\}$ includes all commodities given the robot positions. Solving the MCF problem results in optimal flows $\{y^*_{sgml}\}$, that is the number of robots in commodity $c_{sg} \in C$ traversing along edge $e_{ml} \in E_p$. In our minimal influx MCF formulation, the optimal flow solutions lead to minimized intermediate cell influx and the most dispersed routing. We define the cell influx of P_l as the total number of entering robots, that is, $\sum_{c_{sg} \in C, e_{ml} \in E_p} y_{sgml}$. We formulate the following integer linear program (ILP):

$$\underset{\{y_{sgml}\}}{\arg\min} \quad \alpha \cdot \sum_{c_{sg}} \mathcal{L}_{sg} + \beta \cdot \mathcal{L}_{in} \tag{1a}$$

$$\text{s.t.} \sum_{e^{ml}, e^{ln} \in E_p} y_{sgml} - y_{sgln} = \begin{cases} |c_{sg}|, & l = g \\ -|c_{sg}|, & l = s \\ 0, & o.w. \end{cases}, \forall c_{sg} \in C \tag{1b}$$

$$\mathcal{L}_{sg} \geq y_{sgml}, \ \forall e_{ml} \in E_p, \forall c_{sg} \in C \tag{1c}$$

$$\mathcal{L}_{in} \geq \mathbf{I}_{v^i}^\top \mathbf{y}, \ \forall v^i \in V_p \tag{1d}$$

$$y_{sgml} \in [0, |c_{sg}|], \ \forall e_{ml} \in SP_{sg}, \forall c_{sg} \in C \tag{1e}$$

$$y_{sgml} = 0, \ \forall e_{ml} \notin SP_{sg}, \forall c_{sg} \in C. \tag{1f}$$

Algorithm 1: MCF/OD

Input: a multi-commodity flow instance.
Root.conflict_counts[o] = 1, for $o = 1, \cdots, O$
Root.shortest_paths = solve shortest path for each commodity in the $\mathcal{G}_p$
Root.cost = sum of the largest cost among all paths in each commodity
Root.solution = solve MCF with root shortest path constraints
insert Root to OPEN
Visited[Root.conflict_counts] = True
while *OPEN not empty* **do**
  $P \leftarrow$ node from OPEN with the lowest cost
  $CS \leftarrow congestionDetection(\mathcal{G}_p, P.solution, \boldsymbol{\theta})$
  if *CS is empty* **then**
    return *P.solution*
  $C \leftarrow getAllConflict\,(P, CS)$.
  for *commodity* $c_o \in C$ **do**
    $A \leftarrow$ new node.
    A.conflict_counts $\leftarrow$ P.conflict_counts + $\mathbf{I}_o$
    if *Visited[A.conflict_counts]* **then**
      Continue
    A.shortest_paths $\leftarrow$ P.shortest_paths
    $p_k \leftarrow generateKShortestPath(c_o, \mathcal{G}_p,$ A.conflict_counts[o])
    Update A.shortest_paths[o] with p_k
    if $p_k \leq w_{mcf} \cdot p_{min}$ **then**
      Update A.cost
      Update A.solution by solving MCF with updated shortest paths
      if $A.cost \leq \infty$ **then**
        Insert A to OPEN

Here, $\mathcal{L}_{sg}$ represents the maximum flow for commodity $c_{sg} \in C$, defined by (1c). $\mathcal{L}_{in}$ represents the maximum influx among all the cells, defined by (1d), where $\mathbf{I}_{v^i} = [I_{y_1}, \cdots, I_{y_F}]$, where $y_f \in \{y_{sgml}\}, I_{y_f} \in \{0, 1\}$ is an indicator function that returns 1 when y_f has positive flow to an intermediate vertex $v^i \in V_p$. $\mathbf{y}$ is the vector of all flows. By minimizing both, the objective function penalizes the maximum congestion among all cells and disperses the flows related to one commodity. We weight $\mathcal{L}_{sg}$ and $\mathcal{L}_{in}$ with coefficients α and β, respectively, and set $\beta \gg \alpha$ to prioritize minimizing $\mathcal{L}_{in}$. (1b) is the set of constraints for MCF formulation. The set of constraints (1e, 1f) enforces the flows y_{sgml} onto the shortest paths between the start and goal cells SP_{sg} guaranteeing the solution optimality.

Despite the minimized influx to the intermediate cells, the minimal influx MCF formulation leads to congestion in certain cells due to tight constraints on the shortest paths (robots' shortest paths may intersect at certain cells). To detour the robots optimally, ensuring that the number of robots that enter any intermediate cell m remains below its influx limit θ_m and no unnecessary detour is introduced, we present a complete and optimal solver, MCF/OD, in Algo. 1. It maintains a conflict tree and resolves congestion iteratively. The function $congestionDetection(\mathcal{G}_p, P.solution, \boldsymbol{\theta})$ computes the influx for each cell in $\mathcal{G}_p$, given the flow solution $P.solution$ and returns the set of cells with their influx larger than the corresponding limit. The function $getAllConflict(P)$ returns the set of all commodities passing congested cells.

Theorem 1. *MCF/OD is complete on a locally finite graph.*

Proof. The cost of a conflict tree node equals the sum of the costs of the longest routing (without cycles) in all commodities. For each expansion, k-th shortest paths will

be added to the commodity, which means the cost of the conflict tree is monotonically non-decreasing. For each pair of costs $X < Y$, the search will expand all nodes with cost X before it expands the node with cost Y. As the graph is locally finite, there are a finite number of routing with the same cost for each commodity. Thus, expanding nodes with cost X requires a finite number of iterations.

To include an arbitrary combination of $\hat{Z}$ unique edges of all commodities, the minimal cost of the conflict tree node is Z. Z is finite, as the worst-case scenario is to include all the cell routing within the suboptimality bound. Since we are considering a graph with well-defined edge weights and a finite number of commodities, the worst cost is finite. For a finite cost Z, because the conflict tree node cost is monotonically non-decreasing and only a finite number of nodes with the same cost exists, we can find arbitrary combinations of $\hat{Z}$ unique edges in finite expansions. Thus, if a solution exists by including a combination of $\hat{Z}$ unique edges in the MCF, the algorithm can find it within finite expansions. If all the unique edges have been added to the MCF solver and the optimization cannot find the solution that satisfies the user-defined influx limit, the problem is identified as unsolvable. □

Theorem 2. *MCF/OD is optimal. If a solution is found, it will have the lowest possible cost, i.e., the sum of the costs of the longest routing in all commodities will be minimized if a solution is found.*

Proof. MCF/OD is a best-first search. In each expansion, the k-th shortest path for the selected commodity is inserted. Thus, the cost of a descendant node is monotonically non-decreasing. Therefore, if a solution is found, it is the optimal solution w.r.t. the cost. □

MCF/OD can find the optimal detouring solution. However, the complexity is high due to solving an ILP in each expansion. To tackle many commodities in a large partition, we propose another efficient detour algorithm, one-shot MCF, which solves MCF once. One-shot MCF augments the shortest paths in (1e), (1f) to include all the w_{mcf} bounded-suboptimal paths for each commodity. We employ the k-th shortest path routing algorithm to find all the candidate paths. The proposed One-shot MCF is complete as it includes all bounded-suboptimal paths for each commodity. While it does not optimize for the routing length for all commodities, it optimizes for minimum influx. Intuitively, MCF/OD adds bounded-suboptimal paths iteratively to relax the constraints and terminates once all cell influx limits are satisfied. On the other hand, One-shot MCF adds all bounded-suboptimal paths at once and optimizes for the minimum influx, so it could result in unnecessary detour.

In each high-level planning iteration, we run both MCF/OD and One-shot MCF in parallel. If MCF/OD times out, we use the solution generated from One-shot MCF. The high-level replanning happens every δ_h time interval.

VII. Low-level Planner

Within each cell, the low-level planner, or the cell planner, computes collision-free paths that navigate robots to their local goals in an anytime fashion. The cell planner can be divided into three steps: 1) local goal assignment, 2) anytime MAPF/C generates discrete paths, and 3) cell-crossing protocol for non-stop transiting between cells.

A. Local Goal Assignment

As the first step, local goal assignment aims to route robots to the closest local goals while spreading out robots optimally by solving the following ILP:

$$\arg\min_{\mathbf{A}} \quad \sum_{ij} A_{ij} \cdot D_{ij} + \alpha \sum_{j} u_j + \beta U \tag{2a}$$

$$\text{s.t.} \quad \sum_{j} A_{ij} = 1, \ \forall i \tag{2b}$$

$$U \geq u_j \geq \sum_{i} A_{ij} - 1, \ \forall j, \tag{2c}$$

where $A_{ij} \in \{0, 1\}$ indicates if robot r^i is assigned to the j-th local goal, denoted as lg^j. D_{ij} is the Euclidean distance between r^i and lg^j. Auxiliary variables u_j in the objective function minimize the number of robots queueing at local goal lg^j, prioritizing filling less congested local goals first. Auxiliary variable U in the objective function minimizes the maximum number of robots waiting in queue among all the local goals. This leads to evenly routing robots to different local goals to reduce congestion. A local goal is occupied if assigned with at least one robot. Thus, the number of robots waiting in queue for a local goal lg^j is $(\sum_i A_{ij} - 1)$.

B. Anytime MAPF/C

We adopt the SOTA anytime MAPF method, namely LNS [22], which iteratively improves the solution quality until a solution is needed, to facilitate real-time replanning. We use ECBS [23] as the initial planner as it provides a bounded-suboptimal solution and prioritized planning with SIPP [24] to rapidly replan for a subset of robots. We extend both ECBS and SIPP using MAPF/C to account for the robot embodiment.

For priority planning with SIPP, we propose the following SIPP with generalized conflicts algorithm.

1) SIPP with generalized conflicts: SIPP compresses the time dimension into sparse safety intervals to significantly reduce the search space. The SIPP configuration augments the position with its safety intervals. In the resulting configuration space, A^* finds the shortest path for a robot. Planned robots are considered moving obstacles and modify the safety interval of the traversed states. We propose SIPP with generalized conflicts (SIPP/C) with the following modifications and a different *getSuccessors(s)* algorithm, where line $11-14$ (the highlighted part) differs from the original algorithm. In SIPP/C, the collision intervals, the complements of safety intervals, are added to vertices and edges as follows:

SIPP/C vertex conflict: for a robot at the vertex u_k in a planned path at time step k, we add the collision interval

Algorithm 2: getSuccessors(s) for SIPP/C

```
successors = ∅
for each action e in E(s) do
    cfg = configuration of e applied to s
    e_time = time to execute action e
    start_t = time(s) + e_time
    end_t = interval(s).end + e_time
    for each safe interval i in cfg do
        if i.start > end_t || i.end < start_t then
            continue
        t = earliest arrival time at cfg during interval i with no collisions
        if interval(e).end ≥ t − e_time && interval(e).start ≤
        interval(s).end && i.start ≤ interval(e).start + e_time ≤ i.end
        then
            t = max(interval(e).start + m_time, t)
        else
            continue
        s' = state of configuration cfg with interval i and time t
        insert s' into successors
return successors
```

$[k, k]$ to vertices and edges that intersect with the robot-robot collision model at $pos(u_k)$, i.e., $\mathcal{R}_{\mathcal{R}}(pos(v)) \cap \mathcal{R}_{\mathcal{R}}(pos(u_k))$ and $\mathcal{R}^*_{\mathcal{R}}(e) \cap \mathcal{R}_{\mathcal{R}}(pos(u_k))$, $\forall v, e$. Note here, we use the *swept* model $\mathcal{R}^*_{\mathcal{R}}(e)$ when the robot traverses an edge.

SIPP/C edge conflict: for a robot traversing an edge $e_k := (u_k, u_{k+1})$ at time step k, we add the collision interval $[k, k]$ to vertices and edges that intersect with $\mathcal{R}^*_{\mathcal{R}}(e_k)$, i.e., $\mathcal{R}_{\mathcal{R}}(pos(v)) \cap \mathcal{R}^*_{\mathcal{R}}(e_k)$ and $\mathcal{R}^*_{\mathcal{R}}(e) \cap \mathcal{R}^*_{\mathcal{R}}(e_k)$, $\forall v, e$.

In Algo. 2, $E(s)$ is the action space at state s. For a discrete path $\mathcal{P}^i$, we assign a time $t_k = k\Delta t$ to each discrete time step and obtain the path f^i. Δt is a user-defined value to satisfy the robot's dynamic constraints. The low-level replanning happens repeatedly with a δ_l time interval.

C. Cell-crossing Protocol

A robot would idle at a local goal if the path in its next cell is not yet computed. We propose a cell-crossing protocol that results in non-stop execution, even when traversing between cells. We buffer the hyperplane H_{ml} by a distance d_e towards cell P_m. All robots, currently in P_m and exiting to P_l, compute paths for P_l within the buffer. Buffering is achieved by changing the hyperplane offset to $\mathcal{H}'_a = \mathcal{H}_a - \mathcal{H}_{\mathbf{n}} \cdot d_e$. By enforcing the buffer distance $d_e \geq \delta_l \cdot V_{max}$, where V_{max} is the maximum robot speed, the robot is guaranteed to have a plan in its next cell computed at least once before leaving its current cell. A robot entering the buffer zone will then have a plan to exit its current cell and transition through its next cell. The robot, in the buffer zone, fixes its plan of the current cell to lock the local goal and expected arrival time to its next cell. Thus, when computing the plan for the next cell, the robot's expected start time and position will be predetermined and independent of cell planning order. The robot computes a plan for its next cell, then concatenates the fixed plan of its current cell and the plan in its next cell to form a complete transition plan. Fig. 5 depicts our cell-crossing protocol.

VIII. Results and Discussion

We now demonstrate the system in experiments on simulated and physical robots. For large-scale simulated robot

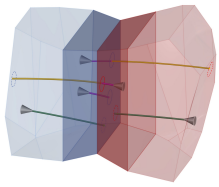


Fig. 5: Robots (cones) fix their paths within the buffer zone and start replanning after crossing the hyperplane.

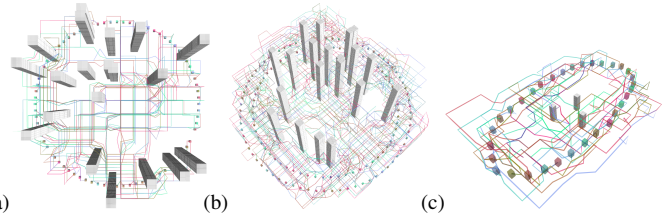
experiments, we create a confined 3D space with random obstacles uniformly generated on a disk of radius 10m. To validate the algorithm's scalability, we scale up the number of robots and the corresponding workspace size to maintain the robot density in different experiment instances. The "Circle74" workspace is $20\text{m}\times20\text{m}\times8\text{m}$. We generate 74 robots whose start states form a circle with a 10m radius at the height of 1m and are centered at the x-y plane's origin, depicted in Fig. 6a. In "Circle142" we scale x and y dimensions by $\sqrt{N}$ to $27.7\text{m}\times27.7\text{m}\times8\text{m}$. The robots start in concentric circles with 13.85m and 11.85m radii at 1m high, centered at the x-y plane's origin, shown in Fig. 6b. In "Demo32" we model the occupancy map of our cluttered lab environment. It is $12.55\text{m}\times7.63\text{m}\times2.8\text{m}$. We run 32 Crazyflies with initial x-y positions uniformly on an ellipse at 1m high, shown in Fig. 6c. The goal states are the antipodal points on the circle (or ellipse). We construct the roadmap using a 6-connected grid graph with an edge length of 1.6m for large-scale simulation and 0.7m for the lab environment.

For simplicity, we set the same influx limit θ_m=20 for all cells in high-level planning and a suboptimal bound w_{mcf}=2 for both MCF/OD and One-shot MCF algorithms. We set the high-level planning time interval δ_{h}=5s for "Circle74", δ_{h}= 3s for "Circle142", and δ_{h}=10s for "Demo32". We set the low-level planning time interval δ_{l}=1s. For the LNS planner with random neighborhood selection, we use ECBS as the initial planner and prioritized planning with SIPP as the iterative planner, both planners we extend with MAPF/C. To account for downwash, we use an axis-aligned bounding box to represent the robot-robot collision model $\mathcal{R}_{\mathcal{R}}(\mathbf{0})$. In simulations, i.e., "Circle74" and "Circle142", we use the axis-aligned bounding box from $[-0.12\text{m}, -0.12\text{m}, -0.3\text{m}]^\top$ to $[0.12\text{m}, 0.12\text{m}, 0.3\text{m}]^\top$. Since "Demo32" is denser, we use the bounding box from $[-0.12\text{m}, -0.12\text{m}, -0.2\text{m}]^\top$ to $[0.12\text{m}, 0.12\text{m}, 0.2\text{m}]^\top$. We use the same shape representation for the robot-environment collision model $\mathcal{R}_{\mathcal{E}}(\mathbf{0})$. All experiments run on an Intel i7-11800H CPU computer.

Fig. 6 depicts typical solutions of the proposed algorithm. We summarize quantitative results in Table I, where MCF refers to MCF/OD and One-shot MCF running in parallel, as described in Sec. VI-A. Note that, as $|V|$ and $|E|$ suggest, the number of vertices and edges increases after partitioning as we add local goals and corresponding edges. With a small computational overhead $\bar{t}_{\text{high}}$, the proposed high-level planner effectively reduces the congestion among cells compared to both greedy and partitionless baseline approaches, by inspecting $\bar{N}_{\max}$. Here $\bar{t}_{\text{high}}$ is the average high-level planning time and $\bar{N}_{\max}$ is the averaged maximum number of robots in a cell throughout the whole execution. By increasing

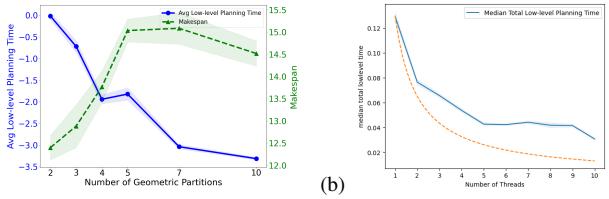
Instance	Q	Roadmap		High-level				Low-level			
		$\|V\|$	$\|E\|$	Method	$\bar{t}_{\text{high}}$(s)	$\bar{t}^{\max}_{\text{high}}$(s)	$\bar{N}_{\max}$	Method	$\bar{T}$	$\bar{t}_{\text{low}}$(s)	$\bar{t}^{\max}_{\text{low}}$(s)
Demo32	8	1059	5890	MCF	0.06	0.06	18.0	ECBS(3.0)+PP	26.88	0.03 ± 0.01	0.13 ± 0.06
Circle74	1	1110	6728	-	-	-	29.0	ECBS(2.0)	**29.00**	12.31 ± 0.17	-
Circle74	10	1515	8760	Greedy	-	-	42.4	ECBS(2.0)+PP	46.85	0.04 ± 0.03	0.40 ± 0.42
Circle74	10	1515	8760	MCF	0.01	0.04	**23.9**	ECBS(2.0)+PP	42.85	$\mathbf{0.02 \pm 0.00}$	$\mathbf{0.09 \pm 0.02}$
Circle142	1	2100	13220	-	-	-	32.0	ECBS(4.5)	**36.00**	16.32 ± 0.28	-
Circle142	25	2954	16383	Greedy	-	-	73.6	ECBS(4.5)+PP	121.60	1.69 ± 0.82	37.58 ± 23.07
Circle142	25	2954	16383	MCF	0.13	0.35	**27.1**	ECBS(4.5)+PP	98.35	$\mathbf{0.03 \pm 0.01}$	$\mathbf{0.28 \pm 0.14}$

TABLE I: Influence of different parameters on different instances. The statistics are averaged across 20 trials.



(a) (b) (c)

Fig. 6: Representative runs of our algorithm with (a) 74 robots, (b) 142 robots, and (c) 32 robots in cluttered environments.



(a) (b)

Fig. 7: 95% Confidence intervals of (a) log-median of low-level planning time and Makespan of our algorithm (b) median of total low-level planning time. The statistics are averaged across 20 trials.

the number of cells Q, the algorithm significantly reduces MAPF computation time. Specifically, for the average low-level planning time $\bar{t}_{\text{low}}$, in instance "Circle74", the proposed algorithm runs 616-times faster than the baseline method and 544-times faster in instance "Circle142". Both the average low-level replanning time $\bar{t}_{\text{low}}$ and the averaged maximum low-level replanning time $\bar{t}^{\max}_{\text{low}}$ are within the real-time regime for all instances. Since we use an anytime algorithm, we only record its initial planning time in Table I, which can be directly compared to the baseline. As we would expect, while running in real-time, the algorithm yields suboptimal solutions compared to the baseline MAPF, according to the average makespan $\bar{T}$. Because the partitioning invalidates the global optimality of MAPF algorithms, and the high-level planner detours robots among the partition, the planned paths become longer.

A. *Effectiveness of Partition in Low-level Planning*

Fig. 7a shows a quantitative evaluation of the low-level planning time and its solution makespan against the number of cells in "Circle74". We report the median of low-level planning time for its robustness. The logarithm of low-level planning time decreases significantly as the number of cells increases. Thus, by dividing the workspace and parallelizing computation, the proposed algorithm significantly reduces the low-level planning time. The makespan increases at the beginning then plateaus. This phenomenon is expected. As the number of cells increases, at the beginning, the high-level planner becomes more effective in detouring the robots to reduce congestion. At a certain point, no further detour is required as congestion regulation is satisfied. Additionally, the partition breaks the global optimality of the MAPF planner, leading to a degeneracy in solution quality.

Our algorithm runs MAPF in parallel for all cells; in Fig. 7b, we investigate the utilization of multi-threading. The orange dashed curve represents the theoretical lower bound of the total low-level planning time (assuming no overhead is introduced when allocating the computation to different threads). We observe that our method's computation time follows the same trend, while the gap between the theoretical lower bound and experimental computation time increases as we increase the number of threads, indicating an increase in parallelization overhead in more cells.

B. *Effectiveness of High-level Planning*

To demonstrate the effectiveness of our MCF-based high-level planner, we compare its cell congestion to an egocentric greedy approach. The greedy planner outputs a single-robot-based shortest inter-cell routing without considering other robots' routing, and may lead to congestion in certain cells. Additionally, we compare the MCF-based approach with the baseline by imposing the partition onto the workspace.

In Table I, we report the average high-level computation time $\bar{t}_{\text{high}}$ and the averaged maximum number of robots in a cell throughout the whole execution $\bar{N}_{\max}$. The complexity of a MAPF instance scales poorly with the number of robots. Thus, $\bar{N}_{\max}$ is an insightful indicator of the computational hardness of a MAPF instance. The computation time for greedy high-level planning is instant, while the MCF-based methods have additional overhead that increases with the number of cells, since they are centralized. For all instances in this paper, MCF-based methods provide real-time solutions.

The MCF-based methods reduce the congestion compared to greedy and baseline methods, according to $N_{\max}$. Although the advantage to the baseline is minor in the current experiment setup, the MCF-based approach can further reduce the congestion by relaxing the suboptimality bound, i.e., increase w_{mcf}. For a fair comparison of low-level planning time, both greedy and MCF-based utilize the same number of threads in computation. Compared to the greedy approach, the proposed MCF-based inter-cell routing brings cell planning computation time to real time while maintaining its solution quality, i.e., makespan.

C. *Scalability of the Proposed Algorithm*

To validate the scalability of our algorithm, we run simulations on increasing numbers of robots (c.f., Table. I). Note that our algorithm achieves real-time performance in all instances as we scale up, according to the averaged maximum high-level and low-level replanning time $\bar{t}^{\max}_{\text{high}}$ and $\bar{t}^{\max}_{\text{low}}$. Due to our hierarchical approach, it can further scale to larger

teams with more cells and CPU threads. The computation bottleneck is the centralized high-level planning in larger scale problems, which we aim to address in future work.

D. Physical Robots

Fig. 1 shows a representative experiment with 32 physical Crazyflies in a cluttered environment (video link: `https://youtu.be/ftdWVpLkErs`). We use a VICON motion capture system to localize and CrazySwarm [25] to control the robots. In the experiment, robots are uniformly located on an ellipse, with their antipodal goals. Three column obstacles are placed within the ellipse. The experiment demonstrates that the proposed algorithm distributes robots effectively through the workspace and achieves real-time replanning.

IX. Conclusion and Future Work

We have introduced a hierarchical path planning algorithm for large-scale coordination tasks. Despite yielding a suboptimal solution, our algorithm significantly reduces the computation time and suits on-demand applications such as drone delivery. The framework achieves real-time operation by dividing the workspace into disjoint cells; within each, an anytime MAPF planner computes collision-free paths in parallel. Our high-level planner regulates congestion while guaranteeing routing quality. Additionally, our algorithm considers the robot embodiment, and we run experiments with the downwash-aware collision model of a quadrotor. We also devise a cell-crossing protocol, which guarantees non-stop execution even when transiting between cells.

The proposed algorithm is designed for lifelong replanning. In our experiments, goals are chosen from a predetermined set, however, it can be extended to a lifelong replanning as we request new goals online, and the algorithm runs conflict annotation to update the conflicts.

While the MCF-based high-level planner operates in real-time in our experiments with up to 142 robots in a 25-cell partition, the limits of its real-time operation are not well defined and depend on the number and density of robots and cells in the space. Future work will explore distributed MCF [26], [27] to increase scalability of the system with real-time, distributed operation regardless of the number of robots and cells. Furthermore, we aim to solve real time large scale motion planning, which respects the robot's kinodynamic constraints and plans in continuous space and time.

References

[1] L. Wawrla, O. Maghazei, and T. Netland, "Applications of drones in warehouse operations," *ETH Zurich, D-MTEC, Whitepaper*, 2019.

[2] A. van Wynsberghe and T. Comes, "Drones in humanitarian contexts, robot ethics, and the human–robot interaction," *Ethics and Information Technology*, vol. 22, pp. 43–53, 2020.

[3] J. E. Scott and C. H. Scott, "Drone delivery models for healthcare," in *Hawaii Int Conf Syst Sci*, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:27433971

[4] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: theory, algorithms, and applications*. USA: Prentice-Hall, Inc., 1993.

[5] W. Hönig, J. A. Preiss, T. S. Kumar, G. S. Sukhatme, and N. Ayanian, "Trajectory planning for quadrotor swarms," *IEEE Trans Robot*, vol. 34, no. 4, pp. 856–869, 2018.

[6] J. Yu and D. Rus, "An effective algorithmic framework for near optimal multi-robot path planning," *Robotics Research: Volume 1*, pp. 495–511, 2018.

[7] J. Yu and S. LaValle, "Structure and intractability of optimal multi-robot path planning on graphs," in *Proc AAAI Conf Artif Intell*, vol. 27, no. 1, 2013, pp. 1443–1449.

[8] B. Şenbaşlar, W. Hönig, and N. Ayanian, "RLSS: real-time, decentralized, cooperative, networkless multi-robot trajectory planning using linear spatial separations," *Autonomous Robots*, vol. 47, no. 7, pp. 921–946, 2023.

[9] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems," *IEEE Transactions on Robotics*, vol. 33, no. 3, pp. 661–674, 2017.

[10] C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Online trajectory generation with distributed model predictive control for multi-robot motion planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 604–611, 2020.

[11] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, "Fast, online collision avoidance for dynamic vehicles using buffered voronoi cells," *IEEE Robot Automat Lett*, vol. 2, no. 2, pp. 1047–1054, 2017.

[12] J. Park and H. J. Kim, "Online trajectory planning for multiple quadrotors in dynamic environments using relative safe flight corridor," *IEEE Robot. and Automat. Lett.*, vol. 6, no. 2, pp. 659–666, 2020.

[13] H. Zhang, M. Yao, Z. Liu, J. Li, L. Terr, S.-H. Chan, T. S. Kumar, and S. Koenig, "A hierarchical approach to multi-agent path finding," in *Proc Int Symp Combin Search*, vol. 12, no. 1, 2021, pp. 209–211.

[14] C. Leet, J. Li, and S. Koenig, "Shard systems: Scalable, robust and persistent multi-agent path finding with performance guarantees," in *Proc AAAI Conf Artif Intell*, vol. 36, no. 9, 2022, pp. 9386–9395.

[15] J. Li, P. Surynek, A. Felner, H. Ma, T. S. Kumar, and S. Koenig, "Multi-agent path finding for large agents," in *Proc AAAI Conf Artif Intell*, vol. 33, no. 01, 2019, pp. 7627–7634.

[16] T. T. Walker, N. R. Sturtevant, and A. Felner, "Extended increasing cost tree search for non-unit cost domains." in *IJCAI*, 2018, pp. 534–540.

[17] K. Yakovlev and A. Andreychuk, "Any-angle pathfinding for multiple agents based on SIPP algorithm," in *Proc Int Conf Automated Planning Scheduling*, vol. 27, 2017, pp. 586–594.

[18] D. Yeo, E. Shrestha, D. A. Paley, and E. M. Atkins, "An empirical model of rotorcrafy uav downwash for disturbance localization and avoidance," in *AIAA Atmos Flight Mech Conf*, 2015, p. 1685.

[19] A. Dobson and K. E. Bekris, "Sparse roadmap spanners for asymptotically near-optimal motion planning," *Int J Robot Res*, vol. 33, no. 1, pp. 18–47, 2014.

[20] L. Gottesbüren, M. Hamann, S. Schlag, and D. Wagner, "Advanced flow-based multilevel hypergraph partitioning," *arXiv preprint arXiv:2003.12110*, 2020.

[21] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, pp. 273–297, 1995.

[22] J. Li, Z. Chen, D. Harabor, P. Stuckey, and S. Koenig, "Anytime multi-agent path finding via large neighborhood search," in *Proc Int Joint Conf Artif Intell*, 2021.

[23] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Proc Int Symp Combin Search*, vol. 5, no. 1, 2014, pp. 19–27.

[24] M. Phillips and M. Likhachev, "SIPP: Safe interval path planning for dynamic environments," in *IEEE Int Conf Robot Automat*. IEEE, 2011, pp. 5628–5635.

[25] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *IEEE Int Conf Robot Automat*. IEEE, 2017, pp. 3299–3304.

[26] B. Awerbuch and R. Khandekar, "Greedy distributed optimization of multi-commodity flows," in *Proc ACM Symp Princ Distrib Comput*, 2007, pp. 274–283.

[27] B. Awerbuch, R. Khandekar, and S. Rao, "Distributed algorithms for multicommodity flow problems via approximate steepest descent framework," *ACM Trans. on Algorithms*, vol. 9, no. 1, pp. 1–14, 2012.