

Congestion Mitigation Path Planning for Large-Scale Multi-Agent Navigation in Dense Environments

Takuro Kato¹, Keisuke Okumura^{2,3}, Yoko Sasaki², Naoya Yokomachi¹

Abstract—In high-density environments where numerous autonomous agents move simultaneously in a distributed manner, streamlining global flows to mitigate local congestion is crucial to maintain overall navigation efficiency. This paper introduces a novel path-planning problem, *congestion mitigation path planning (CMPP)*, which embeds congestion directly into the cost function, defined by the usage of incoming edges along agents’ paths. CMPP assigns a flow-based multiplicative penalty to each vertex of a sparse graph, which grows steeply where frequently-traversed paths intersect, capturing the intuition that congestion intensifies where many agents enter the same area from different directions. Minimizing the total cost yields a set of coarse-level, time-independent routes that autonomous agents can follow while applying their own local collision avoidance. We formulate the problem and develop two solvers: (i) an exact *mixed-integer nonlinear programming* solver for small instances, and (ii) a scalable two-layer search algorithm, *A-CMTS*, which quickly finds suboptimal solutions for large-scale instances and iteratively refines them toward the optimum. Empirical studies show that augmenting state-of-the-art collision-avoidance planners with CMPP significantly reduces local congestion and enhances system throughput in both discrete- and continuous-space scenarios. These results indicate that CMPP improves the performance of multi-agent systems in real-world applications such as logistics and autonomous-vehicle operations.

Index Terms—Multi-agent systems, multi-robot systems, path planning, optimization

I. INTRODUCTION

MODERN industrial environments often involve scenarios where agents move autonomously toward their destinations. For example, in logistics warehouses [1], human operators pick items in densely stocked conditions, adapting to inventory changes and order modifications. In restaurants, human workers and mobile robots collaborate on tasks such as wayfinding and meal serving. Other scenarios, including airport surface operations [2], railway systems [3], and traffic control at autonomous intersections [4], highlight the applicability of autonomous and distributed agent systems.

Such systems benefit from the increase in the number of agents as they concurrently process tasks. However, without

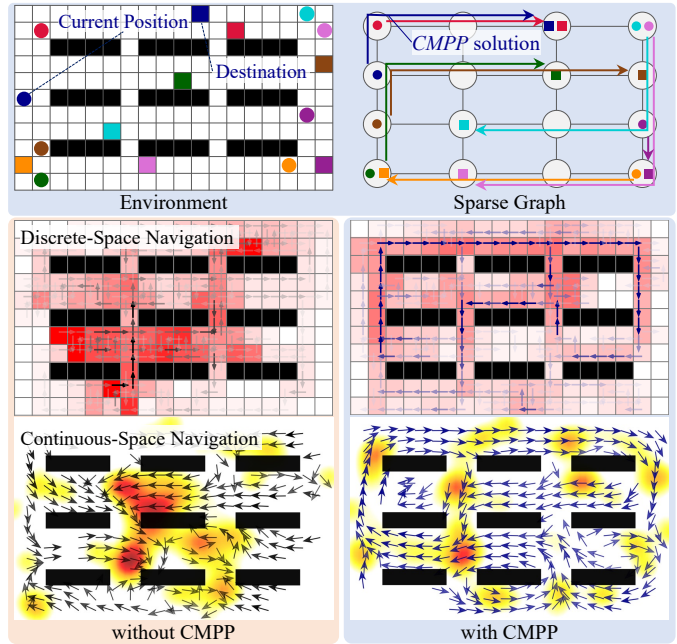


Fig. 1. Overview of CMPP. **Top:** Environment with agents’ current positions (circles) and destinations (squares), and the sparse graph used in CMPP. Multiple agents may share the same vertex in the graph. **Middle/Bottom:** Simulation snapshots in a discrete (middle) and a continuous (bottom) space with 60 agents; heatmaps indicate spatial occupancy, with higher intensity penalizing stuck agents. CMPP streamlines agent movements and mitigates congestion. Demonstration videos are provided in the supplementary material.

attention to coordination tactics, *congestion* can degrade system performance as agents block each other’s movement. In the worst case, the entire system can get into a deadlock situation. This poses the non-trivial challenge of mitigating congestion in an environment where agents operate in a (semi-)decentralized manner with minimal safety mechanisms such as collision avoidance.

Congestion mitigation has been explored across various domains. In freeway traffic management, strategies such as variable speed limits [5] have been proposed. These methods target structured, lane-based networks, limiting their applicability to diverse environments. Other work dynamically adjusts traversal costs between spatial regions [6], but relies on pre-trained neural networks, which restrict adaptability. A different line of research focuses specifically on multi-agent path finding (MAPF) [7], optimizing congestion-aware guide paths at the grid level [8].

Inspired by these works, we formulate the *congestion mitigation path planning (CMPP)* problem, which works effectively across diverse high-density navigation scenarios in both

Manuscript received: March 30, 2025; Revised June 30, 2025; Accepted July 24, 2025. This paper was recommended for publication by Editor M. Ani Hsieh upon evaluation of the Associate Editor and Reviewers’ comments.

¹TICO-AIST Cooperative Research Laboratory for Advanced Logistics, National Institute of Advanced Industrial Science and Technology (AIST), 1-1-1, Umezono, Tsukuba, Japan {takuro.katou, naoya.yokomachi}@aist.go.jp

²Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology (AIST), 2-4-7 Aomi, Koto-ku, Tokyo, Japan {okumura.k, y-sasaki}@aist.go.jp

³Department of Computer Science and Technology, University of Cambridge, Cambridge, CB3 0FD, UK

continuous and discrete spaces, without requiring pre-training. We assume agents autonomously handle collision avoidance, making fine-grained path planning impractical. Thus, we abstract the environment as a sparse graph, whose vertices represent key locations such as intersections or corridor entrances, to provide coarse-level route guidance, as shown in the top row of Fig. 1. On this graph, we propose a cost function defined by the usage of incoming edges along agent paths. In our design, the cost at each vertex increases multiplicatively when frequently-traversed paths intersect, reflecting the intuition that congestion is severe in areas that many agents enter from different directions. Minimizing the total cost yields a set of time-independent paths that naturally streamline agent flows and mitigate congestion, as demonstrated in the middle and bottom rows of Fig. 1.

CMPP is related to MAPF, which assigns agents collision-free, time-dependent paths on a graph, typically minimizing total travel time. While MAPF is a popular abstraction for warehouse automation [9], real-world deployments introduce time inconsistencies due to communication delays, motion inaccuracies, and human intervention, making MAPF plans difficult to execute accurately. Instead, CMPP abstracts path planning by identifying congestion-resistant routes suitable for online adjustment by autonomous agents.

Our contributions are (i) defining CMPP itself, (ii) proposing practical CMPP solvers, and (iii) demonstrating CMPP’s applicability to discrete and continuous multi-agent navigation. Specifically, (i) we formulate CMPP as an optimization problem that minimizes overall congestion and promotes efficient agent flow. This is based on quantifying the degree of congestion by measuring the overlap of agent paths. (ii) We introduce two efficient solvers: an exact solver based on *mixed-integer nonlinear programming (MINLP)* for small-scale instances, and a scalable two-layer search algorithm, *anytime congestion mitigation tree search (A-CMTS)*, which quickly finds suboptimal solutions, refines them toward optimality. As an application, (iii) we demonstrate CMPP’s practical effectiveness in both continuous and discrete domains: integrating CMPP with the continuous-space collision-avoidance algorithm ORCA [10] raises the success rate of navigating 400 agents from 83.9 % to 99.0 %, while coupling CMPP with the discrete planner PIBT [11] in a lifelong-MAPF [12] warehouse scenario yields up to a 58 % throughput gain for 1,500 agents.

II. PRELIMINARIES

A. Assumptions

Map Abstraction: We assume the environment is modeled as a sparse directed graph $G = (V, E)$, where each edge is bidirectional (if $(u, v) \in E$, then $(v, u) \in E$). The sparse graph has vertices at key locations (e.g., intersections and corridors) instead of a fine-grained discretization of the entire space as shown in Fig. 1. Each vertex $v \in V$ represents a localized area, aggregating multiple nearby agents into a single node, simplifying the computation.

Path: Let $\pi := \{\pi_1, \dots, \pi_n\}$ denote the set of agent paths on G , where each π_i is an ordered sequence of vertices

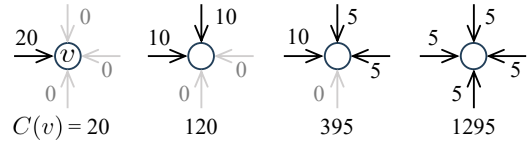


Fig. 2. Example of changes in $C(v)$ when flow f_e occurs on multiple edges in $\delta^-(v)$. The cost increases rapidly as the inflow routes become more dispersed.

representing the path for agent i . As CMPP does not explicitly control movement timing, paths are defined as sequences of vertices without specifying the arrival times.

B. Problem Formulation

A *CMPP instance* consists of a symmetric directed graph $G = (V, E)$, a set of agents $A = \{1, \dots, n\}$, and start-goal vertex pairs $\mathcal{S} = (s_1, \dots, s_n)$ and $\mathcal{G} = (g_1, \dots, g_n)$, where each s_i and g_i belongs to V .

Let $\pi_i = (\pi_i[1], \dots, \pi_i[L_i])$ be the path of agent i , where L_i is the path length. A set of paths $\pi = \{\pi_1, \dots, \pi_n\}$ is a *solution* to the CMPP instance if each agent $i \in A$ satisfies the following constraints:

- *Start and goal:* $\pi_i[1] = s_i$ and $\pi_i[L_i] = g_i$,
- *Adjacency:* Each step follows an edge, i.e., $(\pi_i[k], \pi_i[k+1]) \in E$ for all valid $k \in \mathbb{N}^+$,
- *Simple path:* No vertex is revisited, i.e., $\pi_i[p] \neq \pi_i[q]$ for all $p \neq q$.

Our objective is to find a solution π that minimizes *total congestion cost* on G , as defined in the following subsection.

C. Cost Function Design

Congestion occurs when many agents enter the same vertex from different directions and cross or wait. To quantify it, we first count the agents traversing each directed edge $e = (u, v) \in E$ by defining the *flow* f_e as:

$$f_e := |\{i \in A \mid \exists k \in \mathbb{N}^+, (\pi_i[k], \pi_i[k+1]) = (u, v)\}|. \quad (1)$$

To penalize multi-directional merges, we define the *congestion degree* $C(v)$ at a vertex v as:

$$C(v) := \left(\prod_{e \in \delta^-(v)} (f_e + 1) \right) - 1, \quad (2)$$

where $\delta^-(v)$ is the set of directed edges entering v . The multiplicative term in $C(v)$ increases sharply as flows arrive from multiple directions, while the -1 term ensures $C(v) = 0$ when v is unused. As Fig. 2 illustrates, 20 agents on one incoming edge yield $C(v) = 20$, whereas distributing the same 20 agents over four edges raises $C(v)$ to 1295.

CMPP aims to find a solution π that minimizes the *total congestion cost* $\sum_{v \in V} C(v)$. Optimizing this objective encourages agents either to align in the same direction or to traverse well-separated areas, thereby suppressing cross-traffic bottlenecks.

Note: Longer detours increase the number of traversed edges, thereby increasing the total congestion. Therefore, reducing detours is implicitly encouraged.

III. RELATED WORK

Online Collision Avoidance Planners: CMPP assumes that autonomous agents perform local collision avoidance. Several online collision avoidance planners have been proposed in robotics, including PIBT [11] and collision shielding [13] for discrete-space scenarios, as well as ORCA [10] and buffered Voronoi cells [14] for continuous-space settings. These methods focus on local path adjustments and operate independently of higher-level global route planning.

MAPF: MAPF typically discretizes the environment into a fine grid and plans time-dependent, collision-free paths that minimize metrics such as total path length. A well-known optimal solver is conflict based search (CBS) [15], while numerous suboptimal solvers trade optimality for computational speed and scalability [16], [17]. Variants such as lifelong-MAPF [12] assign new goals to agents upon reaching their current destinations. Unlike MAPF, CMPP shifts focus from strict collision avoidance and path-length minimization to streamlining agent flow and mitigating congestion by planning time-independent paths on a sparse graph.

Traffic Management and Guide Heuristics: Various approaches have been explored for congestion mitigation. For example, in freeway traffic management, variable speed limits employing model predictive control have been studied [5]. These methods assume structured, lane-based networks and do not generalize well to environments where agents move with greater freedom, such as warehouse situations. For autonomous robot systems, methods that use pre-trained neural networks to dynamically adjust traversal costs between spatial regions have been proposed to guide agents toward congestion-avoiding routes [6]. However, reliance on pre-training restricts adaptability to changes in graph topology or agent scale. Several MAPF extensions have also been studied, including priority paths (highways) [18], offline-optimized guidance graphs [19], uniform space utilization [20], and delay predictions based on fleet histories [21]. Another notable method integrates *traffic flow optimization (TFO)* to compute congestion-avoiding guide paths for MAPF planners [8]. These approaches remain specialized for discrete-space MAPF scenarios.



Unlike methods that rely on pre-training or offline-optimization, CMPP employs a sparse graph formulation that enables efficient online computation. Moreover, CMPP is compatible with both discrete-space and continuous-space online collision avoidance planners, providing flexibility across diverse navigation scenarios.

IV. MATHEMATICAL PROGRAMMING APPROACH

A straightforward approach to solving CMPP is to formulate it as a *mixed-integer nonlinear programming (MINLP)* problem. In this formulation, we define the following decision and auxiliary variables:

- $z_{i,e} \in \{0, 1\}$ (decision): $z_{i,e} = 1$ if agent $i \in A$ traverses the directed edge $e \in E$, and 0 otherwise.
- $f_e \in \mathbb{Z}^{0+}$ (auxiliary): flow on edge e .
- $C(v) \in \mathbb{Z}^{0+}$ (auxiliary): congestion degree at vertex v .

TABLE I
STRESS TEST TO FIND OPTIMAL SOLUTIONS WITH MINLP

Map	$ A $	Time (s)
3×3 Grid 	10	1.2
	20	12.3
	30	39.6
10×10 Grid 	100	Timeout
	200	Timeout
	300	N/A

The objective and constraints are given as follows:

$$\min \sum_{v \in V} C(v)$$

$$\text{s.t. } f_e = \sum_{i \in A} z_{i,e} \quad \forall e \in E, \quad (3a)$$

$$C(v) = \prod_{e \in \delta^-(v)} (f_e + 1) - 1 \quad \forall v \in V, \quad (3b)$$

$$\sum_{e \in \delta^+(s_i)(\delta^-(g_i))} z_{i,e} = 1 \quad \forall i \in A, \quad (3c)$$

$$\sum_{e \in \delta^-(v)} z_{i,e} = \sum_{e \in \delta^+(v)} z_{i,e} \quad \forall v \in V \setminus \{s_i, g_i\}, \forall i \in A, \quad (3d)$$

$$z_{i,(u,v)} + z_{i,(v,u)} \leq 1 \quad \forall i \in A, \forall (u,v) \in E. \quad (3e)$$

Constraints (3a) and (3b) implement Eq. (1) and Eq. (2), respectively. Once the binary variables $z_{i,e}$ are determined, the values of f_e and $C(v)$ are uniquely defined by these constraints. Constraint (3c) enforces the start and goal conditions of CMPP, where $\delta^+(v)$ denotes the set of outgoing edges from vertex v . Constraint (3d) ensures adherence to the adjacency constraint, while constraint (3e) guarantees compliance with the simple path constraint.

Discussion: Although this approach efficiently finds exact solutions for small-scale instances, its computational cost becomes prohibitively high for larger problems. Table I reports a stress test in which SCIP[22] was allowed 60s per instance. Each instance is created by randomizing start and goal vertices. “Timeout” indicates no optimal solution within the limit, whereas “N/A” means no feasible solution was obtained. On the 3×3 Grid, the solver finds an optimal solution, but the runtime sharply increases as the number of agents grows. On the 10×10 Grid, the solver fails to compute an exact solution within the runtime limit for instances with $|A| = 100$ or $|A| = 200$ agents, although it still returns a feasible solution. For $|A| = 300$, however, it fails to return even a feasible solution within the same time limit. These results indicate that the MINLP-based method has scalability limitations. In the next section, we propose a search-based approach to overcome this limitation.

V. ANYTIME CONGESTION MITIGATION TREE SEARCH

We present *anytime congestion mitigation tree search (A-CMTS)*, which employs a two-layer structure similar to CBS [15]. A-CMTS is designed to efficiently find suboptimal solutions even for large problem instances. At the high level, the algorithm identifies vertices with high congestion and the

Algorithm 1 High-level Search of A-CMTS

Input: a CMPP instance $\{G, A, S, \mathcal{G}\}$, suboptimal factor $\omega \in \mathbb{R}_{\geq 1}$
Output: a solution π_{Best}

- 1: **procedure** HIGH-LEVEL
- 2: $R \leftarrow \text{NewNode}()$ ▷ R : root node
- 3: $R.C^+ \leftarrow \emptyset, R.C^- \leftarrow \emptyset$ ▷ Forced / forbidden constraints
- 4: $R.\pi \leftarrow \text{Prioritize Planning (PP) using LOW-LEVEL}()$
- 5: $R.\text{cost} \leftarrow \text{total congestion cost in } R.\pi$
- 6: $\text{Open} \leftarrow \text{PriorityQueue}()$ ▷ Ordered by cost (lowest first)
- 7: $\text{Open.push}(R)$
- 8: $\pi_{\text{Best}} \leftarrow R.\pi, UB \leftarrow R.\text{cost}$ ▷ UB : cost upper bound
- 9: **while** $\text{Open} \neq \emptyset \wedge \neg \text{interrupt}()$ **do**
- 10: $N \leftarrow \text{Open.pop}()$ ▷ Node with lowest cost
- 11: **if** $N.\text{cost} < UB$ **then**
- 12: $\pi_{\text{Best}} \leftarrow N.\pi, UB \leftarrow N.\text{cost}$
- 13: $N.\text{LB} \leftarrow \text{estimate lower bound of } N \text{ satisfying Eq. (4)}$
- 14: **if** $UB \leq \omega \cdot N.\text{LB}$ **then**
- 15: **continue** ▷ Branch cut
- 16: $v^* \leftarrow \underset{v \in \Gamma(N)}{\text{argmax}} C(v)$ ▷ $\Gamma(N)$: defined as Eq. (5)
- 17: **if** $v^* = \text{Null}$ **then**
- 18: **continue** ▷ No suitable vertex remains
- 19: $a^* \leftarrow \text{select an agent that visits } v^* \text{ satisfying Eq. (6)}$
- 20: $e^* \leftarrow \text{the edge to } v^* \text{ used by agent } a^*$
- 21: $(P, Q) \leftarrow \text{EXPANDNODE}(N, a^*, e^*, v^*)$ ▷ Algorithm 2
- 22: $\text{Open.push}(P), \text{Open.push}(Q)$
- 23: **return** π_{Best}

Algorithm 2 High-level Node Expansion of A-CMTS

Input: node N , agent a , edge e , vertex v
Output: a pair of child nodes (P, Q)

- 1: **procedure** EXPANDNODE
- 2: $P \leftarrow \text{duplicate } N$ ▷ Child with forced constraint
- 3: $P.C^+ \leftarrow P.C^+ \cup \{(a, e)\}$
- 4: $Q \leftarrow \text{duplicate } N$ ▷ Child with forbidden constraint
- 5: $Q.C^- \leftarrow Q.C^- \cup \{(a, e)\}$
- 6: $Q.\pi_a \leftarrow \text{replan the path of } a \text{ using LOW-LEVEL}()$
- 7: */* Local search for agents sharing v */*
- 8: **for each** agent a' such that $a' \neq a$ and $v \in Q.\pi_{a'}$ **do**
- 9: $Q.\pi_{a'} \leftarrow \text{replan the path of } a' \text{ using LOW-LEVEL}()$
- 10: $Q.\text{cost} \leftarrow \text{total congestion cost in } Q.\pi$
- 11: **return** (P, Q)

agents visiting them, and expands search nodes with different constraints (forced or forbidden) to partition the solution space. At the low level, each agent’s path is replanned to satisfy all constraints imposed by the high level.

A. High-Level Search

Each search node in A-CMTS contains: (i) forced edges C^+ , (ii) forbidden edges C^- , (iii) a set of paths π , (iv) total congestion cost, and (v) a lower bound LB on the solution. Algorithm 1 outlines the high-level procedure of A-CMTS.

The root node R starts with empty constraints, and initial paths $R.\pi$ are computed by prioritized planning (PP) [23] via the low-level search described in Sec. V-C (lines 2–4). The solution cost for $R.\pi$ is computed using Eq. (2) (line 5). Node R is pushed into priority queue Open , ordered by ascending cost (lines 6–7). Best-known solution π_{Best} and upper bound

UB are initialized using R (line 8). Throughout the search, UB is updated whenever a lower-cost solution is found.

In the main loop (lines 9–22), A-CMTS employs branch-and-bound strategy to find the optimal solution. First, the node N with the lowest cost is extracted from Open (line 10). If the cost of N is lower than current upper bound UB , π_{Best} and UB are updated (lines 11–12). Next, a lower bound $N.\text{LB}$ is computed (line 13). Specifically, $N.\text{LB}$ satisfies:

$$\text{LB} \leq \min \left\{ \sum_{v \in \pi} C(v) \mid \pi \text{ satisfies } N.C^+, N.C^- \right\}, \quad (4)$$

meaning that $N.\text{LB}$ guarantees a lower bound on the total congestion cost for any feasible solution derived from node N . Node N is pruned (lines 14–15) if $UB \leq \omega \cdot N.\text{LB}$, where $\omega \in \mathbb{R}_{\geq 1}$ controls the accuracy-speed trade-off: $\omega = 1.0$ ensures optimality, while $\omega > 1.0$ permits bounded suboptimality (see Sec. V-D for details).

If node N is not pruned, A-CMTS expands it. First, the vertex v^* with the highest congestion is selected from the set $\Gamma(N)$, which is defined as:

$$\Gamma(N) := \{v \mid \exists a \in A, \exists e \in (\delta^-(v) \cap N.\pi_a) \text{ such that } (a, e) \notin N.C^+\}. \quad (5)$$

In other words, $\Gamma(N)$ contains vertices where at least one agent’s path can still be modified (line 16). If no suitable vertex remains ($v^* = \text{Null}$), node N is not expanded (lines 17–18). Otherwise, among the agents whose paths pass through v^* , an agent a^* is selected according to:

$$a^* \in \{a \in A \mid \exists e^* \in \delta^-(v^*) \cap N.\pi_a, (a, e^*) \notin N.C^+\}, \quad (6)$$

as the one whose rerouting is expected to effectively reduce the total congestion cost (lines 19–20).

Algorithm 2 described in the next subsection imposes new constraints on agent a^* and generates two child nodes, which are pushed into Open (lines 21–22). The search continues until Open is empty or a stopping criterion (e.g., runtime limit) is met (line 9).

Note: In this study, the priorities for PP at the root node (line 5) are assigned based on agent indices. The lower bound $N.\text{LB}$ (line 13) is estimated as:

$$N.\text{LB} = \text{Length}(\pi_{\min}) + \Delta_C,$$

where π_{\min} is the shortest feasible path satisfying all constraints in $N.C^+$ and $N.C^-$, and Δ_C represents the additional congestion cost induced by the forced edges in $N.C^+$, as these edges cannot be modified in subsequent replanning. Improving the accuracy of this estimate while maintaining computational efficiency remains an open research direction.

B. High-Level Node Expansion

Algorithm 2 expands a node N by adding constraints related to a specific edge e used by agent a to enter vertex v . Two child nodes are created: (i) node P , where agent a is forced to use edge e (lines 2–3). (ii) node Q , where agent a is forbidden from using edge e (lines 4–5). These constraints partition the solution space into two distinct regions.

Child node P inherits paths directly from node N , as forcing edge e requires no immediate path replanning. Forbidding edge e in node Q necessitates replanning agent a 's path (line 6). Since replanning agent a may affect congestion at vertex v , the paths of all other agents through v are also replanned (lines 8–9). This replanning does not affect the optimality guarantee, but the pilot studies observed a convergence speedup with tighter upper bounds. Finally, the total congestion cost of node Q is updated (line 10), and both child nodes P and Q are returned (line 11).

C. Low-Level Search

Given an agent a and its associated constraints $C_a^+ \subseteq C^+$ and $C_a^- \subseteq C^-$, the low-level planner returns a path π_a that minimizes additional congestion at vertices while satisfying all constraints:

$$\begin{aligned} \min \quad & \sum_{(u,v) \in \pi_a} \Delta C(v) \\ \text{s.t.} \quad & \pi_a[1] = s_a, \pi_a[L_a] = g_a, \\ & (u,v) \notin C_a^- \quad \forall (u,v) \in \pi_a, (u,v) \in \pi_a \quad \forall (u,v) \in C_a^+ \end{aligned}$$

Here, $\Delta C(v)$ is the incremental congestion at vertex v when edge (u,v) is used by agent a , given the other agents' paths.

Note: This optimization must determine the order in which forced edges are visited, making it analogous to the traveling salesperson problem [24] and generally computationally intractable for real-time use. Therefore, we adopt a simple approximation for efficiency: (i) Sort the forced edges in C_a^+ by their Euclidean distance from the agent's current vertex s_a . (ii) Run Dijkstra's algorithm to compute the path from s_a to the start of the first forced edge, minimizing $\sum \Delta C(v)$, then traverse that edge. Repeat this sequentially for each forced edge, computing the congestion-minimizing path from the end of the previously traversed edge. (iii) Concatenate the resulting segments to obtain a solution π_a .

D. Property

Theorem: For any suboptimality factor $\omega \geq 1.0$, A-CMTS returns a solution whose cost is at most $\omega \cdot c^*$, where c^* is the optimal solution cost. When $\omega = 1.0$, A-CMTS returns an optimal solution.

Proof sketch: Assume, for contradiction, that the returned solution cost exceeds $\omega \cdot c^*$. By Eq. (4) and pruning condition in A-CMTS, this would imply that a node containing the optimal solution is either incorrectly pruned or never explored, leading to a contradiction. Thus, the returned solution's cost is at most $\omega \cdot c^*$. When $\omega = 1.0$, optimality follows.

E. Lifelong Variant

In applications such as lifelong-MAPF or *multi-agent pickup and delivery* [25], where agents are continuously assigned new destinations upon reaching their current goals, A-CMTS must repeatedly solve CMPP instances. To facilitate this, we initialize the root node of each A-CMTS computation with the solution paths computed in the previous calculation, denoted by π_{Prev} . In other words, π_{Prev} is reused as the solution

path of the root node, allowing the algorithm to converge to stable solutions efficiently with minimal disruption between iterations.

VI. EXPERIMENTS

We evaluate CMPP through three sets of experiments. First, we compare two complementary solvers: *MINLP*, an exact method for small instances, and *A-CMTS*, a bounded-suboptimal algorithm for large instances. Second, we demonstrate CMPP's practical effectiveness in continuous-space scenarios by integrating A-CMTS with the online collision-avoidance planner *ORCA*. Third, we validate our approach in discrete, lifelong-MAPF scenarios by coupling A-CMTS with the grid-based planner *PIBT*. All experiments are conducted on a MacBook Pro (Apple M3 Max, 128 GB RAM). MINLP computations are performed using SCIP, while A-CMTS is implemented in C++. For ORCA and PIBT, we use publicly available implementations provided by the original authors.¹ The videos are available in the supplementary material.

A. Performance Evaluation of CMPP Solvers





Setup: We evaluate the runtime, solution quality, and scalability of MINLP and A-CMTS under two suboptimality settings ($\omega = 1.0$ and $\omega = 1.3$). Table II includes the sparse graph used for benchmarking. For each test setting, 50 problem instances are generated by randomly assigning start and goal locations to the agents. Each solver is given a 1 min runtime limit per instance.

Results: Table II summarizes the results. *Success* denotes the percentage of 50 runs yielding feasible solutions within 1 min. *Obj.* and *Time* indicate the average objective values and computation times (s), respectively. For A-CMTS, *Initial solution* shows the values from the PP-based initial solution, while *Final solution* gives the values at termination. *Impr.* represents the average cost improvement (%) from the initial to final solution. While MINLP efficiently finds optimal solutions on small-scale instances, its success rate declines for larger instances. On larger maps, MINLP often returns suboptimal solutions or fails to find feasible solutions within the runtime limit. In contrast, A-CMTS consistently achieves a 100% success rate across all tests, benefiting from its two-level structure: a fast PP-based initial solution followed by iterative refinement. On small-scale maps, the initial solutions computed by A-CMTS can be up to 43% costlier than those from MINLP. However, iterative refinements significantly reduce this gap to within 0 – 17.8%. On larger-scale maps, A-CMTS not only produces superior initial solutions compared to MINLP within 0.1 s but also further improves them, ultimately obtaining solutions up to 11.1% better upon termination.

Impact of Suboptimality Factor: Table II also shows that A-CMTS with $\omega = 1.3$ sometimes terminates earlier than with $\omega = 1.0$, while still providing comparable or even better solution quality. A higher ω value enables more aggressive pruning in the high-level search, which enables faster progress through

¹We use the RVO2 library (<https://gamma.cs.unc.edu/RVO2/>) for ORCA, and the PIBT component from the author's LaCAM3 repository (<https://github.com/kei18/lacam3>).

TABLE II
 PERFORMANCE OF MINLP AND A-CMTS FOR SOLVING CMPP

Map	$ V $	$ A $	MINLP			A-CMTS								
			Success (%)	Obj.	Time (s)	Success	Initial solution		Final solution					
							Obj.	Time	$\omega = 1.0$		$\omega = 1.3$			
									Obj.	Impr. (%)	Time	Obj.	Impr.	Time
<i>3×3 Grid</i> 	9	10	100	28.0	0.8	100	31.2	0.0	28.0	10.1	32.2	28.2	9.4	18.1
		15	100	47.7	2.8	100	57.9	0.0	48.5	16.2	60.0	48.4	16.4	22.3
		20	100	70.4	19.0	100	90.7	0.0	74.4	18.0	60.0	72.8	19.7	50.3
		25	100	95.4	48.7	100	131.0	0.0	107.2	18.1	60.0	102.0	22.1	58.3
		30	100	123.8	58.1	100	177.0	0.0	145.7	17.7	60.0	143.4	19.0	60.0
<i>Connected</i> 	12	20	100	176.1	0.2	100	200.7	0.0	179.5	10.6	60.0	179.7	10.5	15.2
		40	100	569.7	0.4	100	694.9	0.0	596.7	14.1	60.0	590.8	15.0	45.0
		60	100	1,191.7	0.6	100	1,556.4	0.0	1,294.8	16.8	60.0	1,273.1	18.2	60.0
		80	100	2,018.7	0.8	100	2,728.7	0.0	2,248.6	17.6	60.0	2,212.7	18.9	60.0
		100	100	3,048.6	1.1	100	4,217.0	0.0	3,590.2	14.9	60.0	3,561.8	15.5	60.0
<i>10×10 Grid</i> 	100	50	100	1,106.6	60.0	100	623.0	0.0	578.8	7.1	60.0	579.3	7.0	60.0
		100	96	6,928.9	60.0	100	1,926.6	0.0	1,774.1	7.9	60.0	1,774.1	7.9	60.0
		150	76	24,232.2	60.0	100	4,009.2	0.0	3,720.4	7.2	60.0	3,720.3	7.2	60.0
		200	66	66,493.7	60.0	100	6,831.8	0.0	6,403.9	6.3	60.0	6,403.9	6.3	60.0
		250	60	130,480.6	60.0	100	10,375.4	0.0	9,743.9	6.1	60.0	9,743.9	6.1	60.0
<i>lak303d</i> 	265	100	32	64,068.4	60.0	100	13,394.2	0.0	12,500.4	6.7	60.0	12,500.4	6.7	60.0
		200	2	529,492.0	60.0	100	55,518.8	0.0	50,992.8	8.2	60.0	50,992.8	8.2	60.0
		300	0	N/A	N/A	100	135,029.9	0.1	122,897.7	9.0	60.0	122,897.7	9.0	60.0
		400	0	N/A	N/A	100	260,542.0	0.1	236,333.3	9.3	60.0	236,333.3	9.3	60.0
		500	0	N/A	N/A	100	450,215.5	0.1	400,370.1	11.1	60.0	400,370.1	11.1	60.0

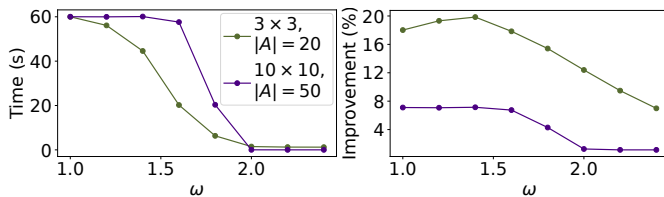
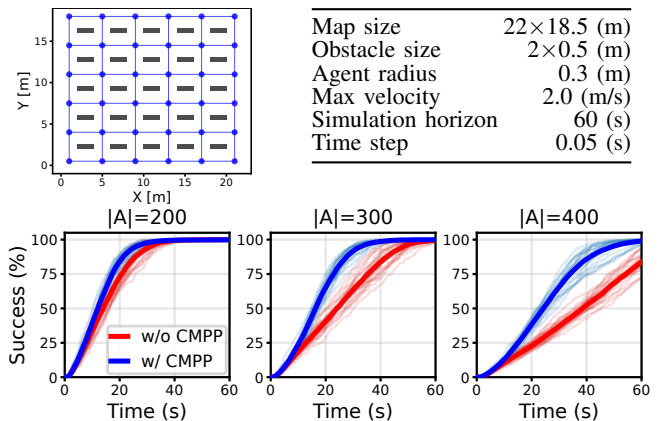

 Fig. 3. Effect of suboptimality factor ω on runtime (s) and improvement (%) from initial to final solution of A-CMTS.

 TABLE III
 PERFORMANCE OF A-CMTS FOR LARGE-SCALE INSTANCES

Grid	$ A $	Init.		Final	
		Time (s)	Improvement (%)	Explored nodes	
10×10 ($ V =100$)	2,000	0.3	2.3	7,441.4	
	6,000	2.6	1.7	2,573.5	
	10,000	7.3	0.8	1,441.5	
50×50 ($ V =2,500$)	2,000	4.8	0.9	954.5	
	6,000	19.4	0.2	355.8	
	10,000	37.5	0.1	210.9	

the search space within the limited time frame. Figure 3 shows the relationship between ω , computation time, and solution quality. These results highlight a flexible trade-off between solution optimality and computational efficiency, making A-CMTS practical for a variety of real-world applications.

Scalability of A-CMTS: Table III presents the scalability of A-CMTS ($\omega = 1.0$) for the number of agents $|A|$ and vertices $|V|$. *Init.* indicates the initial solution. When $|V| = 100$, A-CMTS rapidly computes initial solutions for instances with up to 10,000 agents, achieving cost improvements ranging from 0.8% to 2.3% through iterative refinement. Given the scale of the problems (2,000 to 10,000 agents), improvements of a few percentage points represent significant performance


 Fig. 4. Continuous-space guidance experiment. **Top:** Environment, sparse graph, and simulation parameters. **Bottom:** Success rate for $|A| = 200, 300, 400$ with vanilla ORCA (red) and CMPP-guided ORCA (blue). Thin curves show individual results from 25 trials; bold curves are their mean.

gains for the overall system. For $|V| = 2,500$, although initial solution times increase with the number of agents, the refinement step still achieves consistent, albeit modest, cost improvements. These results confirm that A-CMTS scales effectively to large-scale problems.

B. Multi-Agent Guidance Simulation in Continuous Space

We evaluate CMPP in a continuous-space scenario, where agents perform online collision avoidance using ORCA [10], one of the representative collision-avoidance planners.

Setup: The map at the top of Fig. 4 shows the environment. Starts $\mathcal{S}^* = (s_1^*, \dots, s_n^*)$ and goals $\mathcal{G}^* = (g_1^*, \dots, g_n^*)$ are randomly sampled. A sparse graph $G = (V, E)$ is obtained by placing vertices on a $4 \text{ m} \times 3.5 \text{ m}$ lattice and connecting 4-neighboring vertices. Each s_i^* and g_i^* is mapped to its nearest

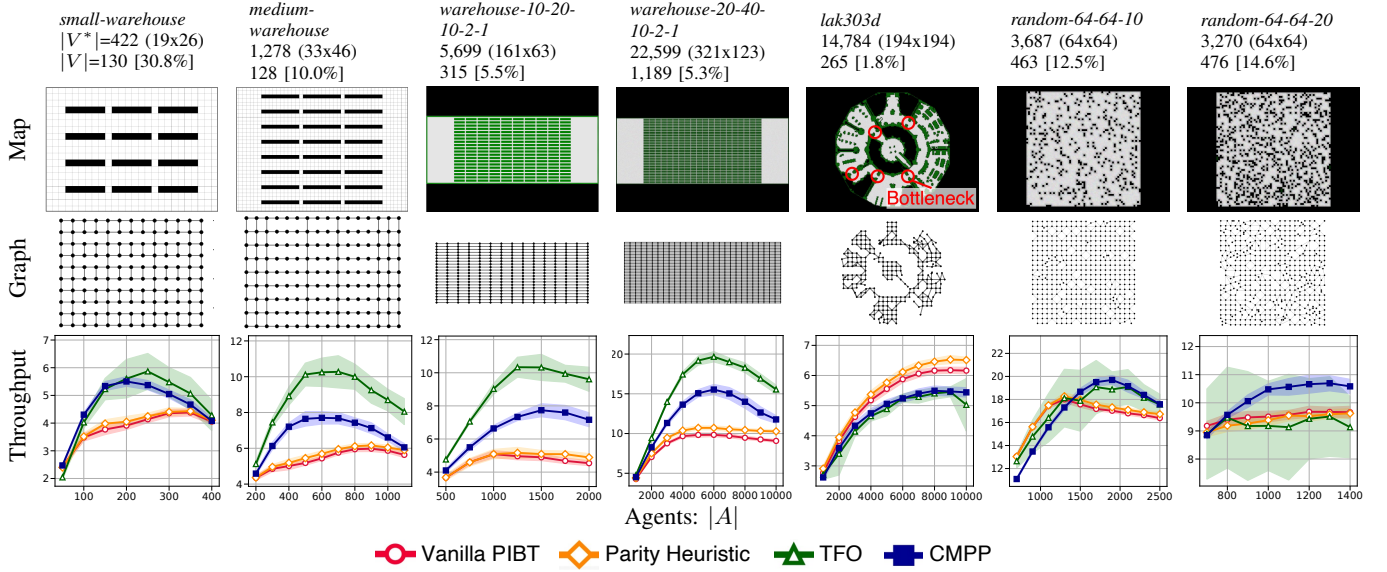


Fig. 5. Results of applying CMPP to lifelong-MAPF setting. Numbers above each map indicate the map size $|V^*|$, the sparse graph size $|V|$, and the reduction ratio $|V|/|V^*|$ (%). *Graph* shows the graph used as input to CMPP, and *Throughput* is the total number of arrivals normalized by the simulation duration (500 steps). Each plotted line represents the average of 25 random scenarios for each setting, with the shaded regions indicating standard deviations.

vertex, yielding \mathcal{S}, \mathcal{G} on G . A-CMTS ($\omega = 1.3$) is then run for 10s on $(G, \mathcal{S}, \mathcal{G})$ to generate CMPP paths π .

Integration of CMPP with ORCA: For each agent i , we form a waypoint queue $\mathcal{Q}_i = (\pi_i[2], \dots, \pi_i[L_i - 1], g_i^*)$, i.e., the CMPP path without its first and last element, and with the exact goal appended. ORCA steers the agent toward the current head waypoint $\mathcal{Q}_i^{\text{front}}$. When the agent is sufficiently close to $\mathcal{Q}_i^{\text{front}}$ (within a threshold r), the front waypoint is popped from \mathcal{Q}_i and the next becomes active. We set $r = 5.0$ m in this experiment.

Results: The bottom row of Fig. 4 plots the *Success Rate*, which represents the percentage of agents that have reached their goals at each time step. With $|A| = 200$, both vanilla ORCA and CMPP-guided ORCA reach 100% success within 40s. At $|A| = 300$, agent congestion emerges under vanilla ORCA, delaying convergence to nearly 60s, whereas CMPP guidance maintains a 100% success rate after 40s. For $|A| = 400$, the mean success rate at 60s improves from 83.9% with vanilla ORCA to 99.0% with CMPP guidance, a gain of 15.1 percentage points.

C. Multi-Agent Guidance Simulation in Discrete Space

We validate the effectiveness of CMPP in a discrete-space lifelong-MAPF scenario, where each agent receives a new destination immediately upon reaching its current goal. Local collision avoidance is handled online with PIBT [11]. This experimental setup closely resembles the robotics competition *The League of Robot Runners* [26] sponsored by Amazon Robotics, where PIBT serves as the default planner. The evaluation is conducted in the seven environments in Fig. 5. The first two are from [27], while the remaining five are from the MAPF benchmark [7].

Sparse-Graph Abstraction for CMPP: A sparse graph $G = (V, E)$ (middle row of Fig. 5) is constructed from the

original 4-connected grid $G^* = (V^*, E^*)$ by sampling vertices at fixed intervals. A surjective map $f : V^* \rightarrow V$ assigns each grid cell to its nearest sparse vertex, while an injective map $g : V \rightarrow V^*$ links each sparse vertex back to a representative grid cell.

Integration of CMPP with PIBT: A-CMTS computes CMPP solutions $\pi_i = (\pi_i[1], \dots, \pi_i[L_i])$ for each agent i on graph G . Each agent's next waypoint on the grid G^* is determined from this path: if $L_i \geq 3$, the agent is directed to the representative cell of $\pi_i[2]$ via the injective map $g : V \rightarrow V^*$; otherwise, it proceeds directly to its goal g_i . PIBT then plans one-step, collision-free moves toward these waypoints. Since agents advance one grid cell at each time step, their positions on the sparse graph are updated before the next CMPP iteration using the surjective map $f : V^* \rightarrow V$. The visited vertex is then removed from each agent's CMPP path, and the remaining path is reused as the initial solution for the subsequent A-CMTS computation.

Setup: Agents are placed randomly on traversable cells and assigned random goals. Each simulation runs for 500 steps and is repeated 25 times under different random scenarios. A-CMTS with $\omega = 1.3$ solves CMPP at each step within a 1s runtime limit. We compare the following baseline strategies against CMPP-guided PIBT: (i) **Vanilla PIBT:** each agent moves directly toward its destination; (ii) **Parity-Heuristic:** a simple guide heuristic for PIBT, in which an agent prefers moving up (down) when its current x -coordinate is odd (even), and right (left) when its y -coordinate is odd (even), thereby promoting left-hand-rule behavior; (iii) **TFO:** a state-of-the-art lifelong-MAPF solver (*GP-R100* variant) [8].

Results: The bottom row of Fig. 5 presents the results. Parity-Heuristic improves vanilla PIBT but offers only marginal throughput gains. TFO optimizes guidance paths at the grid level, achieving strong performance on warehouse

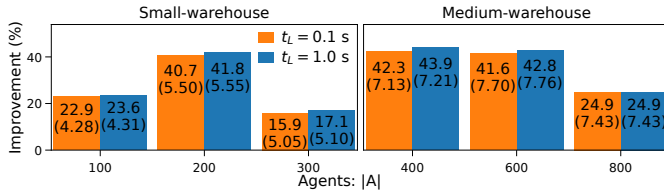


Fig. 6. Throughput improvements of CMPP-guided PIBT relative to vanilla PIBT for varying A-CMTS runtime t_L (s). Bars: improvement (%); raw throughput in parentheses.

maps, but reducing throughput on *lak303d* and *random-64-64-20*. CMPP consistently improves PIBT throughput across both warehouse and random maps, raising throughput by 58.1% for *warehouse-10-20-10-2-1* ($|A|=1,500$) and by 15.7% for *random-64-64-10* ($|A|=1,900$). On *lak303d*, unavoidable single-lane bottlenecks prevent rerouting; thus, the longer detours imposed by TFO and CMPP slightly reduce throughput compared to vanilla PIBT. These results confirm that CMPP boosts guidance efficiency in dense environments whenever alternative routes are available.

Impact of Runtime Limit: Figure 6 shows the results of varying the runtime limit, denoted as t_L , for A-CMTS. To observe the effect of t_L clearly, we selected test patterns where A-CMTS can still perform a meaningful number of search expansions under $t_L = 0.1$. Increasing t_L to 1.0 allows A-CMTS to explore more nodes, improve the solution quality, and achieve higher throughput.

VII. CONCLUSION

In this paper, we introduced a novel path-planning problem, CMPP, which mitigates congestion for distributed autonomous agents by embedding a flow-based multiplicative congestion penalty directly into the cost function. To solve CMPP, we proposed an MINLP-based approach for small instances and A-CMTS for large-scale problems. Experiments showed that while the MINLP solver quickly found exact solutions on small instances, it failed to produce feasible ones for large instances. In contrast, A-CMTS produced scalable suboptimal solutions within limited computation time. Applied experiments confirmed that CMPP reduces agent congestion and improves overall navigation efficiency in both continuous- and discrete-space scenarios.

CMPP is applicable to a wide range of autonomous systems that follow coarse-level route guidance while handling local collision avoidance in a decentralized manner, including traffic management for aircraft and UAVs. In practical deployments, challenges may arise in scenarios involving agents with unpredictable behaviors; thus, ensuring robustness in such environments remains an open issue. Future work includes integrating CMPP with related optimization problems, such as product placement in logistics warehouses and dynamic task allocation.

REFERENCES

[1] R. Koster, T. Le-Duc, and K. J. Roodbergen, "Design and control of warehouse order picking: A literature review," *Eur. J. Oper. Res.*, vol. 182, no. 2, pp. 481–501, 2007.

[2] R. Morris, C. S. Pășăreanu, K. S. Luckow, W. A. Malik, H. Ma, T. K. S. Kumar, and S. Koenig, "Planning, scheduling and monitoring for airport surface operations," in *Proc. AAAI Workshop*, 2016, pp. –.

[3] Z. Chen, J. Li, D. Harabor, and P. J. Stuckey, "Scalable rail planning and replanning with soft deadlines," *arXiv preprint*, 2023.

[4] K. M. Dresner and P. Stone, "A multiagent approach to autonomous intersection management," *J. Artif. Intell. Res.*, vol. 31, no. 1, pp. 591–656, 2008.

[5] A. Hegyi, B. Schutter, and J. Hellendoorn, "Optimal coordination of variable speed limits to suppress shock waves," *IEEE Trans. Intell. Transp. Syst.*, vol. 6, no. 1, pp. 102–112, 2005.

[6] J. Chung, C. Rehbuhn, C. Yates, G. Hollinger, and K. Tumer, "A multiagent framework for learning dynamic traffic management strategies," *Auton. Robots*, vol. 43, no. 6, pp. 1375–1391, 2019.

[7] R. Stern, N. R. Sturtevant, A. Felner, S. Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. S. Kumar, E. Boyarski, and R. Bartak, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proc. Symp. Combin. Search*, 2019, pp. 151–158.

[8] Z. Chen, D. Harabor, J. Li, and P. J. Stuckey, "Traffic flow optimisation for lifelong multi-agent path finding," in *Proc. AAAI Conf. Artif. Intell.*, 2024, pp. 20674–20682.

[9] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Mag.*, vol. 29, no. 1, p. 9, 2008.

[10] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," *Robot. Res.*, vol. 70, pp. 3–19, 2011.

[11] K. Okumura, M. Machida, X. Défago, and Y. Tamura, "Priority inheritance with backtracking for iterative multi-agent path finding," *Artif. Intell.*, vol. 310, p. 103752, 2022.

[12] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 11272–11281.

[13] R. Veerapaneni, Q. Wang, K. Ren, A. Jakobsson, J. Li, and M. Likhachev, "Improving learnt local MAPF policies with heuristic search," in *Proc. Int. Conf. Automated Plan. Scheduling*, 2024, pp. 597–606.

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

[15] G. Sharon, R. Stern, A. Felner, and N. Sturtevant, "Conflict-based search," in *Proc. AAAI Conf. Artif. Intell.*, 2012, pp. 563–569.

[16] J. Li, W. Ruml, and S. Koenig, "EECBS: A bounded-suboptimal search for multi-agent path finding," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 12353–12362.

[17] K. Okumura, "LaCAM: Search-based algorithm for quick multi-agent pathfinding," in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 11655–11662.

[18] L. Cohen, T. Uras, T. K. S. Kumar, H. Xu, N. Ayanian, and S. Koenig, "Improved solvers for bounded-suboptimal multi-agent path finding," in *Proc. Int. Joint Conf. Artif. Intell.*, 2016, pp. 3067–3074.

[19] Y. Zhang, H. Jiang, V. Bhatt, S. Nikolaidis, and J. Li, "Guidance graph optimization for lifelong multi-agent path finding," in *Proc. Int. Joint Conf. Artif. Intell.*, 2024, pp. 311–320.

[20] S. D. Han and J. Yu, "Optimizing space utilization for more effective multi-robot path planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2022, pp. 10709–10715.

[21] G. Yu and M. T. Wolf, "Congestion prediction for large fleets of mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2023, pp. 7642–7649.

[22] T. Achterberg, "SCIP: solving constraint integer programs," *Math. Program. Comput.*, vol. 1, pp. 1–41, 2009.

[23] D. Silver, "Cooperative pathfinding," in *Proc. AAAI Conf. Artif. Intell. Interactive Digit. Entertainment*, 2005, pp. 117–122.

[24] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman & Co., 1979.

[25] H. Ma, J. Li, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," in *Proc. Int. Joint Conf. Auton. Agents Multiagent Syst.*, 2017, pp. 837–845.

[26] S. Chan, Z. Chen, T. Guo, H. Zhang, Y. Zhang, D. Harabor, S. Koenig, C. Wu, and J. Yu, "The league of robot runners competition: Goals, designs, and implementation," in *ICAPS 2024 Syst. Demo. track*, 2024, pp. –.

[27] F. Ho, R. Higa, T. Kato, and S. Nakadai, "A hierarchical approach for joint task allocation and path planning," *IEEE Robot. Autom. Lett.*, vol. 9, no. 11, pp. 10137–10144, 2024.