

Judgelight: Trajectory-Level Post-Optimization for Multi-Agent Path Finding via Closed-Subwalk Collapsing

Yimin Tang¹, Sven Koenig^{1,2}, and Erdem Bıyık¹

¹ Thomas Lord Department of Computer Science, University of Southern California
{yimintan,biyik}@usc.edu

² Department of Computer Science, University of California Irvine
svenk@uci.edu

Abstract. Multi-Agent Path Finding (MAPF) is an NP-hard problem with applications in warehouse automation and multi-robot coordination. Learning-based MAPF solvers offer fast and scalable planning but often produce feasible trajectories that contain unnecessary or oscillatory movements. We propose **Judgelight**, a post-optimization layer that improves trajectory quality after a MAPF solver generates a feasible schedule. Judgelight collapses closed subwalks in agents’ trajectories to remove redundant movements while preserving all feasibility constraints. We formalize this process as **MAPF-Collapse**, prove that it is NP-hard, and present an exact optimization approach by formulating it as integer linear programming (ILP) problem. Experimental results show Judgelight consistently reduces solution cost by around 20%, particularly for learning-based solvers, producing trajectories that are better suited for real-world deployment.

Keywords: Multi-Agent Path Finding · NP-hardness · Deep Learning

1 Introduction

Multi-Agent Path Finding (MAPF) is an NP-hard problem [41,54] that plans collision-free trajectories for multiple agents moving from given start locations to distinct goal locations on a known graph while optimizing a cost criterion. MAPF is a fundamental abstraction for coordination in domains such as warehouse automation, transportation, and aerial swarms. Lots of search-based methods are designed to solve this problem, including Conflict-Based Search (CBS) [32], M^* [48], LaCAM [27], and MAPF-LNS2 [15]. These methods provide strong correctness guarantees and high-quality solutions, but their runtime increase substantially as the number of agents and the level of congestion grow, motivating complementary approaches that deliver fast decisions at scale.

Driven by the success of deep neural networks in perception, control, and decision making [14,34,1], learning-based MAPF solvers have attracted increasing attention [3]. Some learning-based methods, such as RAILGUN [46], adopt a *centralized* formulation that uses global information as neural network input

features and outputs the actions for all agents simultaneously. Others follow a *decentralized* formulation: each agent maps a local observation (often a limited field-of-view, FOV) to a single-step action, and all agents act either synchronously or in a fixed order. Representative methods include PRIMAL [31], MAPPER [20], MAGAT [19], SCRIMP [53], MAPF-GPT [4], and SILLM [13]. These approaches typically rely on imitation learning (IL) and/or reinforcement learning (RL).

Learned policies face two persistent challenges that limit their direct use as drop-in MAPF solvers. First, MAPF is safety-critical: even a single invalid action might cause a collision or a deadlock. In practice, learned policies are commonly wrapped by lightweight validity checks, and invalid moves are often patched by forcing one or more agents to wait. Recent work such as CS-PIBT [47] helps learning-based methods avoid producing collision-inducing next actions by using neural network output probabilities to guide the search order in the search-based algorithm PIBT [28], thereby effectively increasing success rates under congestion. While these corrections are useful as they may prevent collisions, they only focus on feasibility at the *next-action* level. When the trajectories are already feasible, their post-processing do not improve the solution quality.

Second, even when a learned solver outputs feasible trajectories, the resulting motion may exhibit *unnecessary movements* (e.g., back-and-forth movements) [47] that are undesirable in real deployments. In warehouse settings, such redundant motion increases energy consumption and mechanical wear, and may elevate operational risk by introducing avoidable interactions with other robots and environmental uncertainties. Importantly, this inefficiency might persist even when no nearby agents constrain the motion [47], indicating that trajectory quality issues are not solely attributable to collision resolution.

In this work, we propose **Judgelight**, a post-optimization layer that improves trajectory quality *after* a MAPF solver (learning-based or search-based) produces a feasible schedule. Judgelight targets redundant motion by allowing the solver to *collapse closed subwalks* in individual agents’ trajectories while maintaining all MAPF feasibility constraints. We formalize this post-processing task as a new optimization problem, **MAPF-Collapse**, whose objective is to minimize the total number of move actions (treating waits as low-cost) subject to feasibility. This formulation captures the deployment-motivated preference that an agent should stay in place unless movement is necessary. Our main contributions are:

1. We introduce **MAPF-Collapse**, a trajectory-level post-optimization problem that takes a feasible MAPF schedule as input and reduces redundant movements through collapse operations.
2. We analyze the computational complexity of MAPF-Collapse and prove that it is **NP-hard**.
3. We develop an exact optimization approach, **Judgelight**, for MAPF-Collapse and demonstrate that it consistently improves solution quality, particularly for learning-based MAPF solvers, producing trajectories that are more suitable for real-world warehouse-style operations.

2 Background: MAPF Problem Definition

The MAPF problem is defined as follows: Let $I = \{1, 2, \dots, N\}$ denote a set of N agents. $G = (V, E)$ represents an undirected graph, where each vertex $v \in V$ represents a possible location of an agent in the workspace, and each edge $e \in E$ is a unit-cost edge between two vertices that moves an agent from one vertex to the other. Self-loop edges are also allowed, which represent “wait-in-place” actions. Each agent $i \in I$ has a start location $s_i \in V$ and a goal location $g_i \in V$. It also holds that $s_i \neq s_j$ and $g_i \neq g_j$ when $i \neq j \forall i, j \in I$. The task is to plan a collision-free path for each agent i from s_i to g_i .

Each action of agents, either waiting in place or moving to an adjacent vertex, takes one time unit. Let $v_t^i \in V$ be the location of agent i at timestep t . Let $\pi_i = [v_0^i, v_1^i, \dots, v_{T^i}^i]$ denote a path of agent i from its start location v_0^i to its target $v_{T^i}^i$. We assume that agents rest at their targets after completing their paths, i.e., $v_t^i = v_{T^i}^i, \forall t > T^i$. We refer to the path with the minimum cost as the shortest path.

We consider two types of agent-agent collisions. The first type is *vertex collision*, where two agents i and j occupy the same vertex at the same timestep. The second type is *edge collision*, where two agents move in opposite directions along the same edge simultaneously. We use (i, j, t) to denote a vertex collision between agents i and j at timestep t or an edge collision between agents i and j at timestep t to $t + 1$. MAPF algorithms usually use *SoC (flowtime)* $\sum_{i=1}^N T^i$ as the cost function.

Overall, we need to find a set of paths $\{\pi_i \mid i \in I\}$ for all agents such that, for each agent i :

1. Agent i starts from its start location (i.e., $v_0^i = s_i$) and stops at its target location g_j (i.e., $v_{T^i}^i = g_j, \forall t \geq T^i$).
2. Every pair of adjacent vertices on path π_i is connected by an edge, i.e., $(v_t^i, v_{t+1}^i) \in E, \forall t \in \{0, 1, \dots, T^i\}$.
3. $\{\pi_i \mid i \in I\}$ is collision-free.

3 Related Work

3.1 Multi-Agent Path Finding (MAPF)

MAPF has been proved an NP-hard problem when optimality is required [54]. It has inspired a wide range of solutions for its related challenges. Decoupled strategies, as outlined in [33,52,21], approach the problem by independently planning paths for each agent before integrating these paths. In contrast, coupled approaches [39,40] devise a unified plan for all agents simultaneously. There also exist dynamically coupled methods [32,49] that consider agents planning independently at first and then together only when needed for resolving agent-agent collisions. Among these, Conflict-Based Search (CBS) algorithm [32] stands out as a centralized and optimal method for MAPF, with several bounded-suboptimal variants such as ECBS [5] and EECBS [16]. Some suboptimal MAPF algorithms,

such as Prioritized Planning (PP) [11,33], PBS [22], LaCAM [27] and their variant methods [9,15,26] exhibit better scalability and efficiency. However, these search-based algorithms always face the problem of search space dimensionality explosion as the problem size increases, making it difficult to produce a feasible solution within a limited time. Learning-based methods, which we discuss in the subsequent section, attempt to overcome the dimensionality issue by learning from large amounts of data or experience.

3.2 Learning-based MAPF

Given the success of deep learning, many learning-based MAPF methods have been proposed. Compared to search-based algorithms, these methods can typically produce plans in a short time and automatically learn heuristic functions. Most approaches use IL, RL, or a combination of both. Learning-based solvers can make end-to-end decisions using all available information and can be trained on various data types (e.g., MAPF and LMAPF). In contrast, search-based methods often require multi-stage decompositions with hand-crafted heuristics.

One early learning-based method for MAPF is PRIMAL [31], which is trained using both RL and IL. It is a decentralized algorithm that relies on a local field of view (FOV) around each agent to generate actions. MAPF-GPT [4] is a GPT-based model for MAPF, trained via IL on a large dataset. Other approaches incorporate communication mechanisms in a decentralized manner, such as GNN [18] and MAGAT [19], which employ graph neural networks for inter-agent communication, and SCRIMP [53], which uses a global communication mechanism based on transformers. SILLM [13] applies IL and achieves higher planning speed and throughput than search-based methods in certain LMAPF scenarios. RAILGUN [46] adopts a centralized IL formulation and outputs actions for all agents simultaneously. DeepFleet [2] is trained on real warehouse data and improves the efficiency of Amazon warehouse operations by 10%. Recent work such as CS-PIBT [47] improves learned local policies by combining them with PIBT [28] at the action-distribution level, effectively increasing success rates under congestion. However, these corrections primarily operate at the *next-action* level and focus on collision avoidance; they cannot detect whether an agent is actually making progress toward its goal or merely exhibiting oscillatory movement.

3.3 Lifelong MAPF

A generalization of the MAPF problem, Lifelong MAPF (LMAPF) continuously assigns new target locations to agents once they have reached their current targets. In LMAPF, agents do not need to arrive at their targets simultaneously. There are three main approaches to solving LMAPF: solving the problem as a whole [25], using MAPF methods but replanning all paths at each specified timestep [17,28], and replanning only when agents reach their current targets and are assigned new ones [7,12]. Recently, PIBT [28] has played a key role in LMAPF. Its core idea is that higher-priority agents can push lower-priority

agents to alternative locations, enabling progress toward their next targets and ensuring completeness. Similar to standard MAPF, LMAPF solutions may also exhibit unnecessary or oscillatory movements, and our method **Judgelight** can be naturally extended to this setting.

4 MAPF-Collapse

In this section, we first formulate the MAPF post-optimization problem, which we call **MAPF-Collapse**. We then prove that MAPF-Collapse is NP-hard.

We define the Multi-Agent Path Finding collapse (MAPF-Collapse) problem as follows. Let $I = \{1, 2, \dots, N\}$ denote a set of N agents. Let $G = (V, E)$ be an undirected graph, where each vertex $v \in V$ represents a possible location of an agent, and each edge $e \in E$ is a unit-cost edge between two vertices. As in standard MAPF, we allow self-loop edges, which represent *wait-in-place* actions.

Each agent $i \in I$ is associated with a discrete-time trajectory over a fixed time horizon $T \in \mathbb{N}$. The input is given as a matrix $M \in V^{N \times (T+1)}$, where $M[i, t] \in V$ denotes the location of agent i at timestep t , for $t = 0, 1, \dots, T$. We write $v_t^i = M[i, t]$ for notational consistency with MAPF. We assume the given matrix M satisfies all constraints in the MAPF problem definition.

Allowed Modification (Collapse): For any agent $i \in I$ and any indices $0 \leq a < b \leq T$ such that $v_a^i = v_b^i$, we may replace the subsequence:

$$[v_a^i, v_{a+1}^i, \dots, v_b^i]$$

by the constant sequence $[x, x, \dots, x]$, where $x = v_a^i = v_b^i$. This operation is called a *collapse of a closed subwalk*. Multiple collapses may be applied to different agents independently.

Let M' denote the modified schedule after applying some collapses, and let $v_t^i := M'[i, t]$. The modified schedule M' must satisfy the same constraints that mentioned in MAPF problem definition: Each agent finally stops at target locations, each pair of adjacent vertices must be connected and the path solution is collision-free.

Cost Function: Each action (waiting or moving) takes one time unit. Traversing an edge incurs unit cost, while waiting has lower cost. In this paper, we assume it to be zero. The total cost of a modified schedule M' is defined as

$$\text{cost}(M') = \sum_{i=1}^N |\{t \in \{0, \dots, T-1\} \mid v_t^i \neq v_{t+1}^i\}|.$$

The objective of the MAPF-Collapse problem is to find a set of collapses applied to the input schedule M such that the resulting schedule M' is feasible and minimizes $\text{cost}(M')$. Although MAPF-Collapse is defined based on the standard MAPF formulation, it can be readily extended to other MAPF variants, such

as Task and Path Finding (TAPF) [23,45,44] and LMAPF [17,28]. For example, given an LMAPF solution, one can remove all collapse actions associated with completed goals at their corresponding timesteps and then apply the same MAPF-Collapse formulation to optimize the remaining trajectories.

4.1 NP-hardness of MAPF-Collapse

We prove that MAPF-COLLAPSE is NP-hard via its decision version. Decision MAPF-Collapse (G, M, β) : Given a graph $G = (V, E)$, a horizon $T \in \mathbb{N}$, a feasible input schedule $M \in V^{N \times (T+1)}$, and a bound $\beta \in \mathbb{N}$, decide whether there exists a sequence of collapse operations applied to M that produces a feasible schedule M' such that $\text{cost}(M') \leq \beta$.

Lemma 1. *DECISION MAPF-COLLAPSE is in NP.*

Proof. A certificate can be the resulting schedule M' (of size $O(NT)$). We verify feasibility by checking (i) connectivity for every agent and timestep, (ii) absence of vertex collisions, and (iii) absence of edge swaps, all in $O(NT)$ time. We compute $\text{cost}(M')$ in $O(NT)$ time and compare it with β . Thus verification runs in polynomial time.

Theorem 1. *DECISION MAPF-COLLAPSE is NP-complete. MAPF-COLLAPSE (the optimization version) is NP-hard.*

Proof. We reduce from INDEPENDENT SET: given an undirected graph $H = (U, F)$ and an integer K , decide whether there exists an independent set $S \subseteq U$ with $|S| \geq K$.

Construction

Let $m := |F|$ and fix an indexing $F = \{e_r \mid r = 1, \dots, m\}$. We construct an instance (G, M, β) of DECISION MAPF-COLLAPSE with $N := |U| + |F|$ agents.

Vertices. For each $u \in U$ in INDEPENDENT SET, create two vertices x_u and p_u in G . For each edge $e \in F$, create three private vertices a_e, b_e, c_e in G . All created vertices are distinct.

Edges. Add $\{x_u, p_u\}$ to G for all $u \in U$. For each edge $e = (u, v) \in F$, add the following undirected edges to G (waiting is allowed by staying at the same vertex between consecutive timesteps, as in the problem definition):

$$\{a_e, x_u\}, \{x_u, b_e\}, \{b_e, c_e\}, \{c_e, a_e\}, \{a_e, x_v\}, \{x_v, b_e\}.$$

Agents and schedule M . We create one *vertex-agent* A_u for each $u \in U$ and one *edge-agent* B_{e_r} for each $e_r \in F$, so the total number of agents is $N := |U| + |F|$. Set the horizon (we subtract one because timesteps start from zero): $T = 7m - 1$.

– For each vertex-agent A_u :

$$M[A_u, 0] = x_u, \quad M[A_u, t] = p_u \quad (1 \leq t \leq T - 1), \quad M[A_u, T] = x_u.$$

Collapses and cost savings. For each $u \in U$, the subwalk of A_u on $[0, T]$ is closed (from x_u back to x_u). Collapsing it makes A_u stay at x_u for all times and saves exactly 2 moves (the moves $x_u \rightarrow p_u$ and $p_u \rightarrow x_u$). For each edge-agent B_{e_r} (block start $\Theta = \Theta_r$), there are two possible collapses:

$$[\Theta, \Theta + 4] \text{ (endpoints } a_{e_r}\text{)} \quad [\Theta + 2, \Theta + 6] \text{ (endpoints } b_{e_r}\text{)}.$$

Each saves exactly 4 moves (the four edge-traversals inside the collapsed interval), and they cannot both be applied since they overlap on $[\Theta + 2, \Theta + 4]$. Moreover, any other collapse of B_{e_r} yields zero saving: outside the block B_{e_r} only waits at a fixed vertex, and inside the block the only repeated vertices are a_{e_r} (at times $\Theta, \Theta + 4$) and b_{e_r} (at times $\Theta + 2, \Theta + 6$). We also note that for $e_r = (u, v)$, even if we do not collapse A_u or A_v , we can still apply one of the above collapses to B_{e_r} .

Bound. Let $C_0 := \text{cost}(M)$ and set

$$\beta := C_0 - 4m - 2K.$$

Since each edge-agent B_{e_r} can be collapsed once regardless of which vertex-agents are collapsed, we can always obtain a total saving of $4m$ from the edge-agents.

Correctness

(\Rightarrow) Let $S \subseteq U$ be an independent set with $|S| \geq K$. For each $u \in S$, collapse A_u on $[0, T]$ so that A_u stays at x_u for all times. For each edge $e_r = (u_r, v_r)$, modify B_{e_r} as follows: collapse $[\Theta_r, \Theta_r + 4]$ if $u_r \in S$; collapse $[\Theta_r + 2, \Theta_r + 6]$ if $v_r \in S$; otherwise apply either one (arbitrarily). Since S is independent, at most one endpoint of e_r lies in S , hence B_{e_r} is never required to apply both $[\Theta_r, \Theta_r + 4]$ and $[\Theta_r + 2, \Theta_r + 6]$. After these collapses, B_{e_r} avoids the uniquely occupied endpoint (if any) in its block, so no collisions arise. Therefore the resulting schedule M' is feasible. The total saving is at least $4m + 2|S| \geq 4m + 2K$, so $\text{cost}(M') \leq C_0 - (4m + 2K) = \beta$.

(\Leftarrow) Suppose there exists a feasible M' with $\text{cost}(M') \leq \beta$. Let

$$S := \{u \in U \mid A_u \text{ is collapsed on } [0, T]\}.$$

Each edge-agent B_{e_r} can save at most 4 since $[\Theta_r, \Theta_r + 4]$ and $[\Theta_r + 2, \Theta_r + 6]$ are mutually exclusive and no other collapse yields positive saving. Thus, the total saving from all edge-agents is at most $4m$. Because $\text{cost}(M') \leq \beta = C_0 - 4m - 2K$, the vertex-agents together must contribute saving at least $2K$, implying $|S| \geq K$.

It remains to show that S is independent. Suppose for contradiction that some edge $e_r = (u_r, v_r) \in F$ has $u_r, v_r \in S$. Then A_{u_r} occupies x_{u_r} at all times, so B_{e_r} must eliminate its visit to x_{u_r} at time $\Theta_r + 1$; the only positive-saving collapse that changes time $\Theta_r + 1$ is $[\Theta_r, \Theta_r + 4]$, hence $[\Theta_r, \Theta_r + 4]$ must be applied. Similarly, since A_{v_r} occupies x_{v_r} at all times, B_{e_r} must eliminate its visit to x_{v_r} at time $\Theta_r + 5$, which forces applying $[\Theta_r + 2, \Theta_r + 6]$. This contradicts the fact that $[\Theta_r, \Theta_r + 4]$ and $[\Theta_r + 2, \Theta_r + 6]$ cannot both be applied. Hence S is an independent set of size at least K .

NP-hardness Conclusion. Therefore, the graph H has an independent set of size at least K if and only if the constructed instance admits a feasible solution of cost at most β . The construction has size polynomial in $|U| + |F|$ and can be computed in polynomial time, so this is a polynomial-time reduction. Together with Lemma 1, it follows that DECISION MAPF-COLLAPSE is NP-complete. NP-completeness of the decision version implies NP-hardness of the optimization version. \square

4.2 Judgelight

We refer to our whole proposed post-optimization process as **Judgelight**. Here we show how to formulate MAPF-Collapse as an **Integer Linear Programming** (ILP) problem and demonstrate how Judgelight can help MAPF solvers improve the quality of their final solutions.

Throughout, we assume the input schedule $M \in V^{N \times (T+1)}$ is collision-free, which is already ensured by the problem definition. A *collapse action* is a tuple $c = (i, a, b, x)$ where $i \in [N]$, $0 \leq a < b \leq T$, $x = M[i, a] = M[i, b]$, and applying c rewrites all locations of agent i on timesteps $k \in [a, b]$ to x .

Candidate actions and weights. Let the set of all candidate actions be

$$\mathcal{C} := \{ c = (i, a, b, x) \mid 0 \leq a < b \leq T, x = M[i, a] = M[i, b] \}.$$

For each $c = (i, a, b, x) \in \mathcal{C}$, define its weight

$$w_c = \sum_{k=a}^{b-1} \mathbb{I}[M[i, k] \neq M[i, k+1]],$$

i.e., the number of edge traversals on $[a, b]$ in the original schedule. Collapsing $[a, b]$ turns all those traversals into waits, hence saves exactly w_c cost.

Decision variables and objective function. For each candidate $c \in \mathcal{C}$, introduce a binary variable $y_c \in \{0, 1\}$ indicating whether c is applied. We maximize the total saving:

$$\mathbf{maximize} \quad \sum_{c \in \mathcal{C}} w_c y_c$$

Under the non-overlap constraints below, the savings contributed by selected actions on the same agent do not double-count, and we have $\text{cost}(M') = \text{cost}(M) - \sum_{c \in \mathcal{C}} w_c y_c$. Therefore this objective is equivalent to minimizing $\text{cost}(M')$.

Constraints from relations between actions. Selecting actions is constrained by relations induced from M .

(1) *Mutual exclusion within an agent.* Within one agent, there may exist multiple candidate collapse actions. If two actions $c = (i, a, b, x)$ and $c' = (i, a', b', x')$ have intersecting time ranges, i.e.,

$$[a, b] \cap [a', b'] \neq \emptyset,$$

Algorithm 1 Build ILP relations from M

Require: Feasible schedule $M \in V^{N \times (T+1)}$

Ensure: Candidate actions \mathcal{C} with weights $\{w_c\}$; within-agent exclusions \mathcal{E}_{in} ; cross-agent exclusions $\mathcal{E}_{\text{cross}}$; dependencies \mathcal{D} ; invalid actions \mathcal{I}

- 1: Build $\text{Occ}[k][x] := \{j \in [N] \mid M[j, k] = x\}$ for all $k \in [0..T], x \in V$
- 2: For each agent i , precompute prefix move counts Pref_i where $\text{Pref}_i[0] = 0$, and $\text{Pref}_i[t+1] = \text{Pref}_i[t] + \mathbb{I}[M[i, t] \neq M[i, t+1]]$
- 3: $\mathcal{C} \leftarrow \emptyset$
- 4: **for** $i = 1$ to N **do**
- 5: Build timestep lists $\text{Times}_i[x] := \{k \mid M[i, k] = x\}$ for visited vertices x
- 6: **for all** x with $\text{Times}_i[x] \neq \emptyset$ **do**
- 7: **for all** (a, b) such that $a, b \in \text{Times}_i[x]$ and $a < b$ **do**
- 8: Add $c = (i, a, b, x)$ to \mathcal{C} with $w_c = \text{Pref}_i[b] - \text{Pref}_i[a]$
- 9: $\mathcal{E}_{\text{in}} \leftarrow$ all pairs of $c, c' \in \mathcal{C}$ on the same agent with $[a, b] \cap [a', b'] \neq \emptyset$
- 10: $\mathcal{E}_{\text{cross}} \leftarrow$ all pairs of $c, c' \in \mathcal{C}$ on different agents with $x = x'$ and $[a, b] \cap [a', b'] \neq \emptyset$
- 11: Build a Segment Tree Idx_j for each agent j over its actions in \mathcal{C} , which allows us to efficiently retrieve all actions related to a given position.
- 12: $\mathcal{D} \leftarrow \emptyset, \mathcal{I} \leftarrow \emptyset$
- 13: **for all** $c = (i, a, b, x) \in \mathcal{C}$ **do**
- 14: **for** $k = a$ to b **do**
- 15: **for all** $j \in \text{Occ}[k][x]$ with $j \neq i$ **do**
- 16: $\mathcal{S} \leftarrow \{c' = (j, a', b', x') \in \text{Idx}_j.\text{Query}(k) \mid x' \neq x\}$
- 17: **if** $\mathcal{S} = \emptyset$ **then**
- 18: $\mathcal{I} \leftarrow \mathcal{I} \cup \{c\}$
- 19: **break** k and j loops
- 20: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(c, \mathcal{S})\}$ $\triangleright y_c \leq \sum_{c' \in \mathcal{S}} y_{c'}$
- 21: **return** $(\mathcal{C}, \{w_c\}, \mathcal{E}_{\text{in}}, \mathcal{E}_{\text{cross}}, \mathcal{D}, \mathcal{I})$

then we select at most one of them, since otherwise the endpoints of one action would be rewritten by the other action, making the action invalid. This yields the constraint

$$y_c + y_{c'} \leq 1.$$

(2) *Mutual exclusion across agents (action-action collisions)*. Two actions on different agents may force them to occupy the same vertex at overlapping timesteps. For $c = (i, a, b, x)$ and $c' = (j, a', b', x')$ with $i \neq j$, if $x = x'$ and $[a, b] \cap [a', b'] \neq \emptyset$, then selecting both actions would create an unavoidable vertex collision. Hence we add the constraint

$$y_c + y_{c'} \leq 1$$

(3) *Dependency across agents (collision avoidance)*. A collapse may force an agent to occupy a vertex x over a time interval; to keep the schedule feasible, other agents that would occupy the same vertex at those timesteps must be “moved away” by selecting some collapse action on those agents. For example, if we have two agents $\{1, 2\}$ and the paths are (A, B, A) and (C, A, C) , then there are two collapse actions $c = (1, 0, 2, A)$ and $c' = (2, 0, 2, C)$, and selecting c requires that agent 2 also selects a suitable action so that collisions are avoided.

Formally, for each selected action $c = (i, a, b, x)$, for every other agent $j \neq i$, for all $k \in [a, b]$ such that $M[j, k] = x$, define the set of *suitable* actions on agent j :

$$\mathcal{C}(j, k, x) := \{c' = (j, a', b', x') \in \mathcal{C} \mid k \in [a', b'] \text{ and } x' \neq x\}.$$

Then collision avoidance is enforced by the implication

$$y_c \leq \sum_{c' \in \mathcal{C}(j, k, x)} y_{c'}$$

$$\forall c = (i, a, b, x) \in \mathcal{C}, \forall j \neq i, \forall k \in [a, b] \text{ with } M[j, k] = x$$

Therefore, the remaining task is to systematically extract (i) mutual-exclusion relations, (ii) dependency relations, and (iii) invalid actions from the input schedule M . We summarize the constraint construction procedure in Algorithm 1. We also note that Judgelight can be applied to any MAPF algorithm, not only learning-based methods, although its benefits are most prominent over learning-based solvers due to the inherent uncertainty of neural network policies. Moreover, in theory, Judgelight can be extended to other MAPF variants, such as those with rotation actions as well as TAPF and LMAPF, which are more representative of real-world warehouse scenarios.

In practice, when forming the ILP, we apply two essential optimizations:

1. For any path segment of the form “A...AA” or “AA...A”, in principle every pair of identical locations could be used to form a collapse action. However, all such actions are equivalent in terms of both cost savings and interactions with other agents. Therefore, we keep only the collapses defined by the first and last occurrences of “A” when constructing ILP variables. For example, for a path “AAABAAA”, there are originally $\binom{6}{2}$ possible collapse actions, but after this optimization, only $\binom{4}{2}$ actions remain. As a result, the number of ILP variables and constraints is significantly reduced without affecting the optimality of MAPF-COLLAPSE.
2. For oscillatory segments such as “ABABABA”, we first apply a length-3 filter that rewrites every subsequence of the form “ABA” into “AAA” whenever this transformation can be applied without introducing collisions. This optimization is necessary because, in practice, frequent back-and-forth motion can cause the number of ILP variables to become extremely large. For example, an alternating sequence “ABABAB...” of length n can generate $O(n^2)$ variables and $O(n^4)$ constraints, and n can be on the order of 100 depending on the map size. This preprocessing step may affect the optimality of the solution, but it is necessary in practice.

5 Experiments

For experiments, we primarily evaluate our Judgelight post-optimization on learning-based methods. We compare solution quality before and after applying Judgelight, using solution cost as the main metric. We adopt the POGEMA benchmark [35] to evaluate learning-based methods with Judgelight.

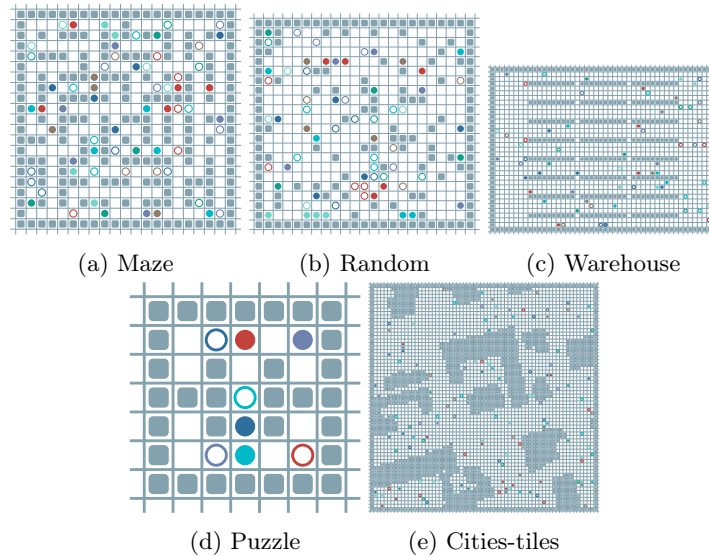


Fig. 3: Examples of POGEMA-tested maps. Maze and Random map sizes are 32×32 . Warehouse map is 33×46 . Puzzle map sizes are below 10×10 . Cities-tiles are 64×64 areas selected from larger Cities maps.

All experiments³ were conducted on a system running Ubuntu 22.04.1 LTS, equipped with an Intel Core i9-12900K CPU, 128 GB RAM, and an NVIDIA RTX 3080 GPU. We use the POGEMA benchmark with five different map types, as shown in Figure 3, resulting in a total of 3,296 test cases. For the ILP solver, we use Gurobi 13.0 and impose a time limit of 5 seconds on solving, but do not limit the time spent on ILP construction.

In POGEMA, the learning-based methods include VDN [42], QPLEX [51], SCRIMP [53], IQL [43], QMIX [30], DCC [24], MAMBA [10], Switcher [38], Follower [36], and MATS-LP [37]. However, based on the results reported in the RAILGUN paper [46], we evaluate SCRIMP, DCC, RAILGUN, MAMBA and Follower in this work, as they achieve good success rates on POGEMA. We also evaluate LaCAM as a representative suboptimal search-based method.

Here we introduce several evaluation metrics. **ISR** (Individual Success Ratio) is defined as the ratio of agents that eventually stop at their goal locations. **Saved SoC** denotes the reduction in SoC between the original solution and the solution obtained after applying Judgelight. **SoC Saving Ratio** is defined as Saved SoC divided by original solution SoC. **Agent density** is computed as the average, over all timesteps, of the number of agents within an agent’s field of view (FOV) divided by the number of non-obstacle grid cells within that FOV. All FOV sizes use the same setting in POGEMA. The default FOV size is 11×11 ; for DCC, it is 9×9 as DCC was trained on this FOV size. For the centralized methods RAILGUN and LaCAM, we use the default FOV to compute agent density.

³ Our code will be released as open source with the camera-ready version. We have provided demo videos in the supplementary material.

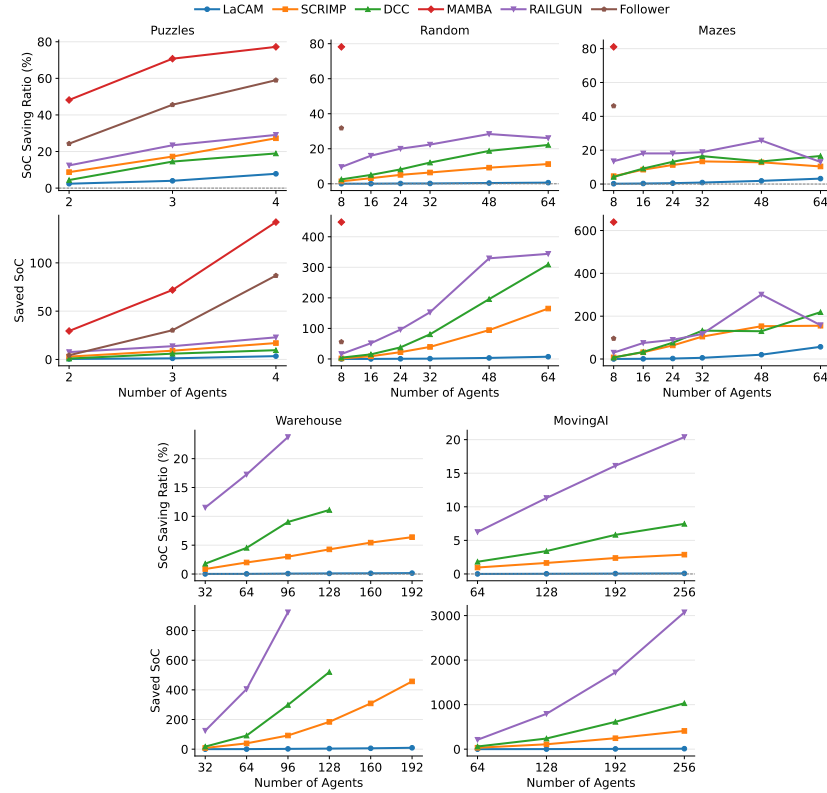


Fig. 4: This figure illustrates how Judgelight performance is affected by testcase difficulty and algorithm performance. We include only test cases for which the algorithm achieves $ISR = 1$, as Judgelight’s cost savings are not meaningful when agents exhibit largely random motion (even though the apparent savings could be high in such cases). Missing data indicate that the algorithm has no suitable test cases under the corresponding experimental settings.

It is important to note that all tested methods are designed for the standard MAPF problem formulation. Since standard MAPF does not assign a lower cost to wait actions compared to move actions, and learning-based policies are trained with standard MAPF data, the cost metric used in MAPF-Collapse is not perfectly aligned with the original objectives of some methods. As a result, the reported costs should be interpreted as illustrating the potential benefits that MAPF algorithms can gain from the Judgelight post-optimization under different scenarios, rather than as a direct comparison of their original optimization objectives.

In Figures 4 and 5, we report the total SoC savings achieved by Judgelight post-optimization for the MAPF problem. We observe a clear trend: as the number of agents increases, both the Saved SoC and the SoC Saving Ratio increase. For a fixed map type and map size, increasing the number of agents leads to

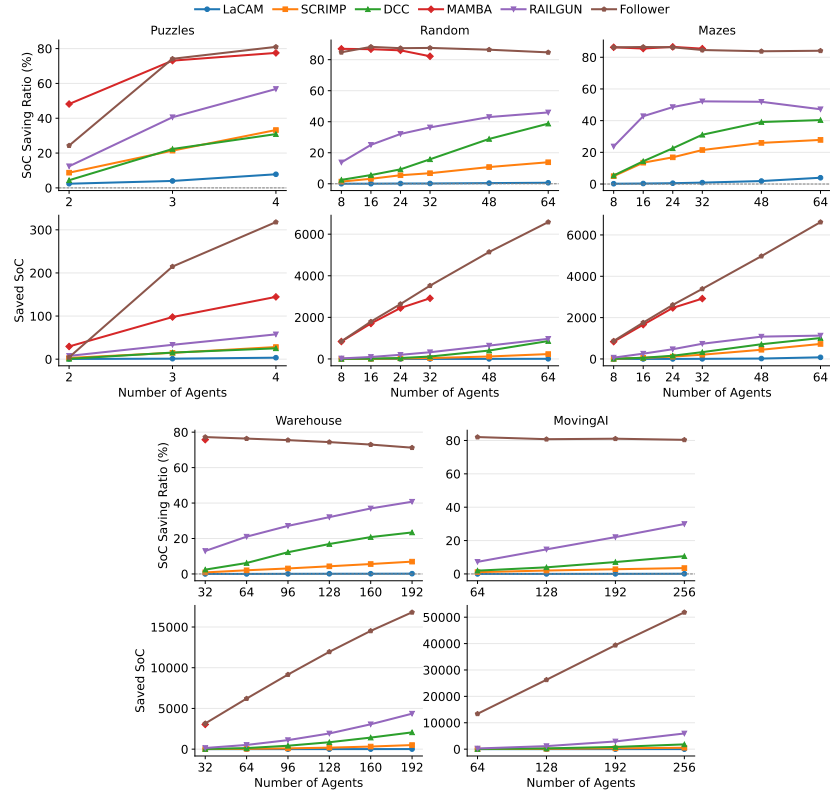


Fig. 5: In this figure, we include only test cases for which the algorithm achieves $\text{ISR} > 0.5$. This choice was made to provide more informative test cases. An $\text{ISR} > 0.5$ indicates that the learning-based policy is still partially effective on the test case. While such performance may not be sufficient for the MAPF task, it may still be useful for LMAPF.

higher agent density and a higher probability of path conflicts, making the test cases more difficult. As a result, suboptimal algorithms tend to explore multiple collision-avoidance behaviors. Since most learning-based methods predict only the next action, such exploratory behavior often manifests as oscillatory motion. After a collision-free solution is found, Judgelight can substantially reduce these redundant movements, achieving an average SoC reduction of approximately 20%–40%. As shown in Figure 5, we also observe that MAMBA and Follower exhibit consistently high SoC saving ratios across different map types. This is because these methods often fail to fully solve the MAPF problem; for agents that do not reach their goal locations, the policies tend to produce oscillatory behavior, causing unnecessary movements to constitute a large fraction of the overall solution cost. This observation indicates that Judgelight can also be beneficial in partially solved scenarios by encouraging agents that cannot reach their

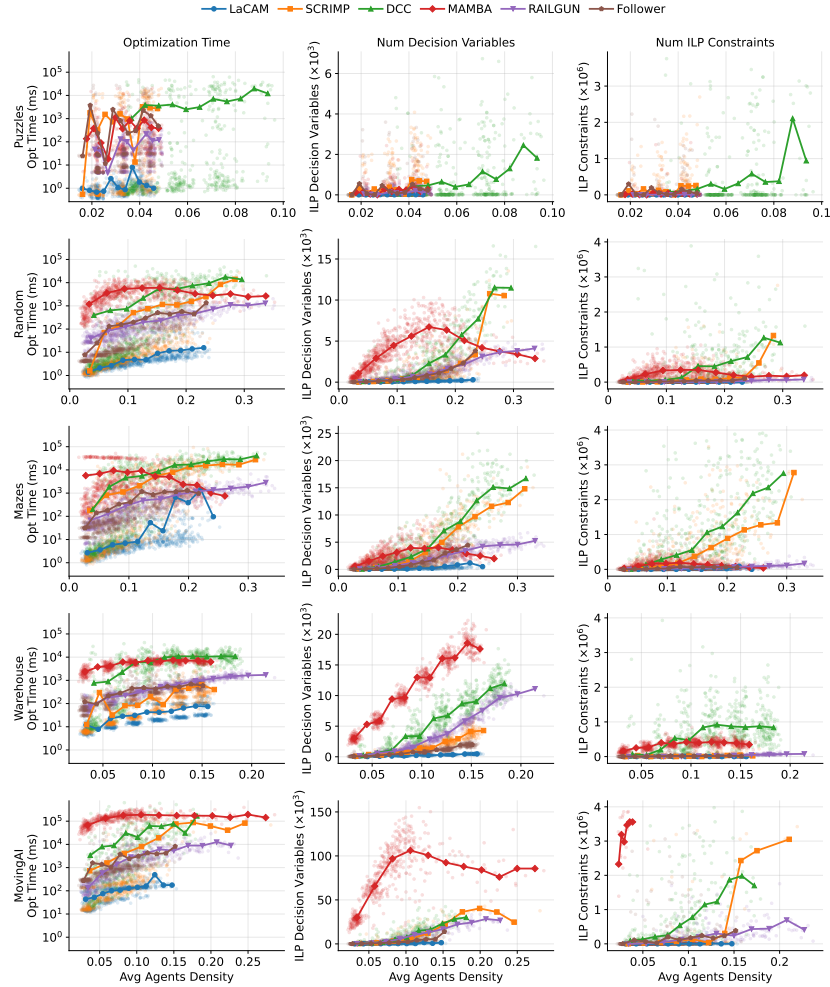


Fig. 6: This figure shows the relations between agent density and ILP generated actions/constraints, and also agent density and Judgelight running time. This figure include all testcases even the method’s ISR= 0. In ILP constraints, we didn’t show data points which larger than $4e6$ in the figure for making all method can be distinguish.

goals under the current policy to remain stationary, rather than executing unsafe and unnecessary actions.

Figure 6 focuses on the runtime performance of Judgelight. In real-world warehouse operations or warehouse competition settings [8], real-time planning is typically required, making Judgelight’s runtime performance critical for practical deployment. As shown in Figure 6, Judgelight’s runtime is mainly influenced by two factors: agent density and algorithm performance. Higher agent density corresponds to more difficult test cases, and in such scenarios learning-based

| | ISR ≥ 0 (All Runs) | ISR = 1 |
|----------|-------------------------|----------------------|
| LaCAM | 3286 / 3296 (99.70%) | 3266 / 3270 (99.88%) |
| SCRIMP | 2783 / 3296 (84.44%) | 2657 / 2699 (98.44%) |
| DCC | 1724 / 3296 (52.31%) | 1600 / 1799 (88.94%) |
| MAMBA | 880 / 3296 (26.70%) | 226 / 230 (98.26%) |
| RAILGUN | 2699 / 3296 (81.89%) | 1036 / 1041 (99.52%) |
| Follower | 2910 / 3296 (88.29%) | 110 / 110 (100.00%) |

Table 1: Judgelight runtime statistics measured by the proportion of test cases whose optimization time is within 1 second.

methods are more likely to generate unnecessary movements. This leads to a larger number of ILP variables and constraints, increasing both ILP construction time and solver time. We also observe that the number of ILP constraints has a greater impact on runtime than the number of variables. For example, in the warehouse scenario, although RAILGUN and DCC generate a similar number of ILP variables, DCC consistently exhibits longer runtimes because it produces more ILP constraints. Algorithm performance also strongly affects the size of the ILP, as shown in Table 1. LaCAM, which has strong MAPF-solving capability, generates relatively few ILP actions and constraints, resulting in runtimes that are consistently below 100ms. For learning-based methods such as SCRIMP, DCC, and RAILGUN, which achieve relatively good performance, Judgelight’s runtime is also below one second in most test cases.

6 Conclusion

Summary. Learning-based MAPF solvers often produce feasible solutions that exhibit unnecessary and oscillatory movements, especially under congestion or partial failures, due to their next-action-level decision making. We formalize the removal of such redundant motion as a trajectory-level post-processing problem, which we call **MAPF-Collapse**, prove that it is NP-hard, and develop an exact solution via an integer linear programming formulation. We refer to the resulting post-optimization framework as **Judgelight**. Experiments on the POGEMA benchmark show that Judgelight consistently reduces solution cost by around 20%–40% while remaining practical in runtime, with over 90% of solved MAPF test cases finishing within one second. These results demonstrate that Judgelight is an effective, solver-agnostic post-processing step for improving MAPF trajectory quality in real-world settings.

Future Work. Judgelight could be extended to other multi-agent planning settings, and enhanced to eliminate not only redundant movements but also unnecessary waiting actions. Another interesting idea is to use Judgelight to produce higher-quality data for imitation learning or reinforcement learning approaches to MAPF: it may increase the demonstration quality, generate noiseless preference data that can be used in addition to demonstrations [6] or with Direct Preference Optimization [29], or guide the policy optimization for better exploration [50].

References

1. Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al.: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
2. Agaskar, A., Siva, S., Pickering, W., O’Brien, K., Kekeh, C., Li, A., Sarker, B.G., Chua, A., Nemade, M., Thattai, C., Di, J., Iyengar, I., Dharoor, R., Kirouani, D., Erskine, J., Hegazy, T., Niekum, S., Khan, U.A., Pecora, F., Durham, J.W.: Deepfleet: Multi-agent foundation models for mobile robots (2025), <https://arxiv.org/abs/2508.08574>
3. Alkazzi, J.M., Okumura, K.: A comprehensive review on leveraging machine learning for multi-agent path finding. *IEEE Access* (2024)
4. Andreychuk, A., Yakovlev, K., Panov, A., Skrynnik, A.: Mapf-gpt: Imitation learning for multi-agent pathfinding at scale. arXiv preprint arXiv:2409.00134 (2024)
5. Barer, M., Sharon, G., Stern, R., Felner, A.: Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In: Proceedings of the International Symposium on Combinatorial Search (SoCS) (2014)
6. Biyik, E., Losey, D.P., Palan, M., Landolfi, N.C., Shevchuk, G., Sadigh, D.: Learning reward functions from diverse sources of human feedback: Optimally integrating demonstrations and preferences. *The International Journal of Robotics Research* **41**(1), 45–67 (2022)
7. Čáp, M., Vokřínek, J., Kleiner, A.: Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) (2015)
8. Chan, S.H., Chen, Z., Guo, T., Zhang, H., Zhang, Y., Harabor, D., Koenig, S., Wu, C., Yu, J.: The league of robot runners: Competition goals, designs, and implementation. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) (2024), <https://www.leagueofrobotrunners.org/>, competition series to foster MAPF research and benchmarking
9. Chan, S.H., Stern, R., Felner, A., Koenig, S.: Greedy priority-based search for suboptimal multi-agent path finding. In: Proceedings of the International Symposium on Combinatorial Search (SoCS) (2023)
10. Egorov, V., Shpilman, A.: Scalable multi-agent model-based reinforcement learning. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2022)
11. Erdmann, M., Lozano-Perez, T.: On multiple moving objects. *Algorithmica* (1987)
12. Grenouilleau, F., Van Hoesel, W.J., Hooker, J.N.: A multi-label a* algorithm for multi-agent pathfinding. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) (2019)
13. Jiang, H., Wang, Y., Veerapaneni, R., Duhan, T., Sartoretti, G., Li, J.: Deploying ten thousand robots: Scalable imitation learning for lifelong multi-agent path finding. In: 2025 IEEE International Conference on Robotics and Automation (ICRA). pp. 1–7. IEEE (2025)
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems (NeurIPS) (2012)
15. Li, J., Chen, Z., Harabor, D., Stuckey, P.J., Koenig, S.: Mapf-lns2: fast repairing for multi-agent path finding via large neighborhood search. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (2022)

16. Li, J., Ruml, W., Koenig, S.: Eecbs: A bounded-suboptimal search for multi-agent path finding. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (2021)
17. Li, J., Tinka, A., Kiesel, S., Durham, J.W., Kumar, T.S., Koenig, S.: Lifelong multi-agent path finding in large-scale warehouses. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (2021)
18. Li, Q., Gama, F., Ribeiro, A., Prorok, A.: Graph neural networks for decentralized multi-robot path planning. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2020)
19. Li, Q., Lin, W., Liu, Z., Prorok, A.: Message-aware graph attention networks for large-scale multi-robot path planning. *IEEE Robotics and Automation Letters* (2021)
20. Liu, Z., Chen, B., Zhou, H., Koushik, G., Hebert, M., Zhao, D.: Mapper: Multi-agent path planning with evolutionary reinforcement learning in mixed dynamic environments. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2020)
21. Luna, R.J., Bekris, K.E.: Push and swap: Fast cooperative path-finding with completeness guarantees. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) (2011)
22. Ma, H., Harabor, D., Stuckey, P.J., Li, J., Koenig, S.: Searching with consistent prioritization for multi-agent path finding. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (2019)
23. Ma, H., Koenig, S.: Optimal target assignment and path finding for teams of agents. *arXiv preprint arXiv:1612.05693* (2016)
24. Ma, Z., Luo, Y., Pan, J.: Learning selective communication for multi-agent path finding. *IEEE Robotics and Automation Letters* (2021)
25. Nguyen, V., Obermeier, P., Son, T., Schaub, T., Yeoh, W.: Generalized target assignment and path finding using answer set programming. In: Proceedings of the International Symposium on Combinatorial Search (SoCS) (2019)
26. Okumura, K.: Engineering lacam*: Towards real-time, large-scale, and near-optimal multi-agent pathfinding. *arXiv preprint* (2023)
27. Okumura, K.: Lacam: Search-based algorithm for quick multi-agent pathfinding. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (2023)
28. Okumura, K., Machida, M., D efago, X., Tamura, Y.: Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence* (2022)
29. Rafailov, R., Sharma, A., Mitchell, E., Manning, C.D., Ermon, S., Finn, C.: Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems* **36**, 53728–53741 (2023)
30. Rashid, T., Samvelyan, M., De Witt, C.S., Farquhar, G., Foerster, J., Whiteson, S.: Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research* (2020)
31. Sartoretti, G., Kerr, J., Shi, Y., Wagner, G., Kumar, T.S., Koenig, S., Choset, H.: Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters* (2019)
32. Sharon, G., Stern, R., Felner, A., Sturtevant, N.R.: Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* (2015)
33. Silver, D.: Cooperative pathfinding. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE) (2005)
34. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. *Nature* (2016)

35. Skrynnik, A., Andreychuk, A., Borzilov, A., Chernyavskiy, A., Yakovlev, K., Panov, A.: POGEMA: A benchmark platform for cooperative multi-agent pathfinding. In: International Conference on Learning Representations (ICLR) (2025)
36. Skrynnik, A., Andreychuk, A., Nesterova, M., Yakovlev, K., Panov, A.: Learn to follow: Decentralized lifelong multi-agent pathfinding via planning and learning. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (2024)
37. Skrynnik, A., Andreychuk, A., Yakovlev, K., Panov, A.: Decentralized monte carlo tree search for partially observable multi-agent pathfinding. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (2024)
38. Skrynnik, A., Andreychuk, A., Yakovlev, K., Panov, A.I.: When to switch: planning and learning for partially observable multi-agent pathfinding. *IEEE Transactions on Neural Networks and Learning Systems* (2023)
39. Standley, T.: Finding optimal solutions to cooperative pathfinding problems. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (2010)
40. Standley, T., Korf, R.: Complete algorithms for cooperative pathfinding problems. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJ-CAI) (2011)
41. Stern, R., Sturtevant, N., Felner, A., Koenig, S., Ma, H., Walker, T., Li, J., Atzmon, D., Cohen, L., Kumar, T., et al.: Multi-agent pathfinding: Definitions, variants, and benchmarks. In: Proceedings of the International Symposium on Combinatorial Search (SoCS) (2019)
42. Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W.M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J.Z., Tuyls, K., et al.: Value-decomposition networks for cooperative multi-agent learning based on team reward. In: Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2018)
43. Tan, M.: Multi-agent reinforcement learning: Independent vs. cooperative agents. In: International Conference on Machine Learning (ICML) (1993)
44. Tang, Y., Koenig, S., Li, J.: Ita-ecbs: A bounded-suboptimal algorithm for combined target-assignment and path-finding problem. In: Proceedings of the International Symposium on Combinatorial Search. vol. 17, pp. 134–142 (2024)
45. Tang, Y., Ren, Z., Li, J., Sycara, K.: Solving multi-agent target assignment and path finding with a single constraint tree. In: 2023 International Symposium on Multi-Robot and Multi-Agent Systems (MRS). pp. 8–14. IEEE (2023)
46. Tang, Y., Xiong, X., Xi, J., Li, J., Bıyık, E., Koenig, S.: Railgun: A unified convolutional policy for multi-agent path finding across different environments and tasks. In: International Conference on Intelligent Robots and Systems (IROS). IEEE (2025)
47. Veerapaneni, R., Wang, Q., Ren, K., Jakobsson, A., Li, J., Likhachev, M.: Improving learnt local mapf policies with heuristic search (2024), <https://arxiv.org/abs/2403.20300>
48. Wagner, G., Choset, H.: M*: A complete multirobot path planning algorithm with performance bounds. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2011)
49. Wagner, G., Choset, H.: Subdimensional expansion for multirobot path planning. *Artificial intelligence* (2015)
50. Wang, G., Liu, J., Li, X., Wu, F., Zhang, X., Chen, T., Chen, X.: Preference-guided reinforcement learning for efficient exploration. arXiv preprint arXiv:2407.06503 (2024)
51. Wang, J., Ren, Z., Liu, T., Yu, Y., Zhang, C.: Qplex: Duplex dueling multi-agent q-learning. In: International Conference on Learning Representations (ICLR) (2020)

52. Wang, K.H.C., Botea, A.: Fast and memory-efficient multi-agent pathfinding. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) (2008)
53. Wang, Y., Xiang, B., Huang, S., Sartoretti, G.: Scrimp: Scalable communication for reinforcement-and imitation-learning-based multi-agent pathfinding. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2023)
54. Yu, J., LaValle, S.: Structure and intractability of optimal multi-robot path planning on graphs. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (2013)