# Learn to Follow: Decentralized Lifelong Multi-Agent Pathfinding via Planning and Learning

**Alexey Skrynnik**[1,2], **Anton Andreychuk**[1] , **Maria Nesterova**[2,3], **Konstantin Yakovlev**[1,2], **Aleksandr Panov**[1,3]

[1]AIRI, Moscow, Russia
[2]Federal Research Center for Computer Science and Control of Russian Academy of Sciences, Moscow, Russia
[3]MIPT, Dolgoprudny, Russia
skrynnikalexey@gmail.com, andreychuk@airi.net, minesterova@yandex.ru, yakovlev@isa.ru, panov@airi.net

## Abstract

Multi-agent Pathfinding (MAPF) problem generally asks to find a set of conflict-free paths for a set of agents confined to a graph and is typically solved in a centralized fashion. Conversely, in this work, we investigate the decentralized MAPF setting, when the central controller that possesses all the information on the agents' locations and goals is absent and the agents have to sequentially decide the actions on their own without having access to the full state of the environment. We focus on the practically important lifelong variant of MAPF, which involves continuously assigning new goals to the agents upon arrival to the previous ones. To address this complex problem, we propose a method that integrates two complementary approaches: planning with heuristic search and reinforcement learning through policy optimization. Planning is utilized to construct and re-plan individual paths. We enhance our planning algorithm with a dedicated technique tailored to avoid congestion and increase the throughput of the system. We employ reinforcement learning to discover the collision avoidance policies that effectively guide the agents along the paths. The policy is implemented as a neural network and is effectively trained without any reward-shaping or external guidance. We evaluate our method on a wide range of setups comparing it to the state-of-the-art solvers. The results show that our method consistently outperforms the learnable competitors, showing higher throughput and better ability to generalize to the maps that were unseen at the training stage. Moreover our solver outperforms a rule-based one in terms of throughput and is an order of magnitude faster than a state-of-the-art search-based solver. The code is available at https://github.com/AIRI-Institute/learn-to-follow.

## Introduction

Multi-agent pathfinding (MAPF) (Stern et al. 2019) is a challenging problem that has been getting increasing attention recently. It is often studied in the AI community with the following assumptions. The agents are confined to a graph, and at each timestep, an agent can either move to an adjacent vertex or stay at the current one. A central controller possesses information about the graph and the agents' start and goal locations. This unit is in charge of constructing a set of conflict-free plans for all the agents. Thus, a typical setting for MAPF can be attributed as *centralized* and *fully observable*.



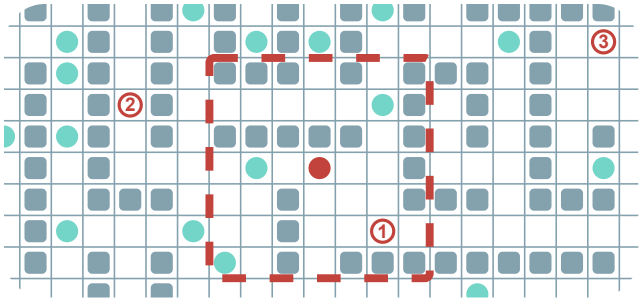


Figure 1: An example of a decentralized LMAPF instance. Agents are depicted as filled circles. The dashed line illustrates the red agent's ego-centric field-of-view, where the other observed agents are colored in teal. The red circles with numbers represent the goals that the agent needs to reach. The next goal is only revealed to the agent when the current one is achieved.

In many real-world domains, however, it is not possible, from the engineering perspective, to design such a central controller that has a stable connection to all the agents (robots) and obtains a full knowledge of the environment all the time. For example, consider a fleet of service robots delivering some items in a human-shared environment, e.g., the robots delivering medicine in the hospital. Each of these robots is likely to have access to the global map of the environment (e.g., the floor plan), possibly refined through the robot's sensors. However, the connection to the central controller may not be consistent. Thus, the latter may not have accurate data on the robots' locations and, consequently, cannot provide valid MAPF solutions. In such scenarios, *decentralized approaches* to the MAPF problems, when the robots themselves have to decide their future paths based on their local observations, as depicted in Fig. 1, are essential. In this work, we aim to develop such an efficient decentralized approach.

It is natural to frame the decentralized MAPF problem as a sequential decision-making problem where, at each timestep, each agent must choose and execute an action that will advance it toward its goal while ensuring that the other agents can also reach their goals. The result of solving this problem is a policy that, at each moment, specifies which

action to execute. To form such a policy, learnable methods are commonly used, such as reinforcement learning (RL), which is particularly beneficial in tasks with incomplete information (Mnih et al. 2015; Rashid et al. 2018; Hafner et al. 2021). However, even state-of-the-art RL methods generally struggle with solving long-horizon problems with the involved causal structure (Milani et al. 2020; Hafner et al. 2023), and they are often inferior to the search-based, planning methods when solving problems with hard combinatorial structure (Kansky et al. 2023).

Indeed, numerous learnable methods tailored to MAPF settings are already known, such as PRIMAL (Sartoretti et al. 2019), PRIMAL2 (Damani et al. 2021), DHC (Ma, Luo, and Ma 2021), PICO (Li et al. 2022), SCRIMP (Wang et al. 2023) to name a few. These methods either rely on the complex training procedures that typically involve manual reward-shaping, external demonstrations etc., or on communication (data sharing) between the agents. Moreover, these methods often do not generalize well, i.e. their performance degrades significantly when they solve problem instances on the maps that are not alike the ones used for training.

To this end, we suggest that the MAPF problem should not be solved directly by RL, but rather in combination and vivid interaction with a heuristic search algorithm. This idea is put into practice via the following pipeline. Each agent plans an individual path to its goal by a heuristic search algorithm without taking the other agents into account. Moreover, an additional technique is introduced for planning that is dedicated specifically to dispersing the agents over the workspace via penalizing the paths that are likely to cause deadlocks. Upon path construction, a learnable policy, developed through decentralized training, is then invoked to follow the planned path, making necessary detours to avoid collisions and allow other agents to progress towards their goals.

Empirically, we compare our method, which we name FOLLOWER, to a range of both learnable and non-learnable state-of-the-art competitors and show that it *i*) consistently outperforms the learnable competitors in terms of solution quality; *ii*) better generalizes to the unseen environments compared to the other learnable solvers; *iii*) outperforms a state-of-the-art rule-based centralized solver in terms of solution quality; *iv*) scales much better to the large numbers of agents in terms of computation time compared to the state-of-the-art search-based centralized solver.

## Related Works

**Lifelong MAPF** LMAPF is an extension of MAPF when the new goals are assigned to the agents when they reach their current ones. Similarly, in (online) multi-agent pickup and delivery (MAPD), agents are continuously assigned tasks comprising two locations that the agent has to visit in a strict order: pickup location and delivery location. Typically, the assignment problem is not considered in LMAPF/MAPD. However, some works also consider task assignment, such as (Liu et al. 2019; Chen et al. 2021).

Ma et al. (2017) propose several variants to tackle MAPD differing in the amount of data the agents share. Yet, even the decoupled (as attributed by the authors) algorithms based on Token Swapping rely on global information, i.e., the one provided by the central unit. An enhanced Token Swapping variant that considers kinematic constraints was introduced in (Ma et al. 2019b). In (Okumura et al. 2019) an efficient rule-based re-planning approach to solve MAPF that is naturally capable of solving LMAPF/MAPD problems is introduced – PIBT (Priority Inheritance with Backtracking). It does not rely on the several restrictive assumptions of Token Swapping and is empirically shown to outperform the latter. We compare with PIBT and demonstrate that our method provides solutions of the better quality.

Finally, one of the most recent and effective LMAPF solvers is the RHCR (Rolling-Horizon Collision Resolution) algorithm presented in (Li et al. 2021). It draws upon the idea of bounded planning, i.e., constructing not a complete plan but rather its initial part. RHCR is a centralized solver that relies on the full knowledge of the agents' locations, current paths, goals, etc. In this work, we empirically compare with RHCR and show that our method scales better to large number of agents when the computation time is capped.

**Decentralized MAPF** This setting entails that the paths/actions of the agents are not decided by a central unit but by the agents themselves. Numerous approaches, especially the ones tailored to the robotics applications, boil this problem down to reactive control (Lumelsky and Harinarayan 1997; Van den Berg, Lin, and Manocha 2008; Zhu, Brito, and Alonso-Mora 2022). These methods, however, are often prone to deadlocks. Several MAPF algorithms can also be implemented in a decentralized manner. For example, Wang and Botea (2011) introduce MAPP algorithm that relies on individual pathfinding for each agent and a set of rules to determine priorities and choose actions to avoid conflicts when they occur along the paths. In general, most rule-based MAPF solvers, like the previously mentioned PIBT (Okumura et al. 2019), or another seminal MAPF solver Push And Rotate (de Wilde, ter Mors, and Witteveen 2013), can be implemented in such a way that each agent locally decides its actions. However, in this case, the implicit assumption is that the agents can communicate to share relevant information (or that they have access to the global MAPF-related data). By contrast, our work assumes that the agents cannot reliably communicate with each other or a central unit, which significantly increases the complexity of the problem.

**Learnable MAPF** This direction has recently received an increased attention. In (Sartoretti et al. 2019), a seminal PRIMAL method was introduced. It utilizes reinforcement learning and imitation learning to solve MAPF in a decentralized fashion. Later in (Damani et al. 2021), it was enhanced and tailored explicitly to LMAPF. The new version was named PRIMAL2. Since numerous learning-based MAPF solvers have emerged, it has become common to compare against PRIMAL/PRIMAL2 (we also compare with it in our work). For example, Riviere et al. (2020) propose another learning-based approach tailored explicitly to agents with a non-trivial dynamic model, such as quadrotors. Ma, Luo, and Ma (2021) describe DHC – a method that efficiently utilizes the agents' communications to solve

decentralized MAPF. Another communication-based learnable approach, PICO, is presented in (Li et al. 2022) and yet another in the most recent paper by (Wang et al. 2023). Overall, currently, there is a wide range of learnable decentralized MAPF solvers. In this work, we compare our method with the state-of-the-art learnable competitors and show that the former produces better quality solutions and better generalizes to the unseen maps.

**MARL and HRL** Multi-Agent Reinforcement Learning (MARL) (Wong et al. 2023) is a separate direction in RL that specifically considers the multi-agent setting. Mainly, MARL approaches consider game environments (like Starcraft (Samvelyan et al. 2019)) in which pathfinding is not of primary importance. However, several MARL methods, such as QMIX (Rashid et al. 2018) and MAPPO (Yu et al. 2022), have been adapted specifically for the MAPF task (Skrynnik et al. 2021). However, they rely on information sharing between the agents.

Learnable low-level policies and heuristic sub-goal allocation procedures are commonplace in many hierarchical RL (HRL) approaches tailored to single-agent problems. However, such techniques are rarely explored in MARL (Wang et al. 2022). Existing studies primarily demonstrate their results within simplistic environments (Tang et al. 2018), leaving ample room for further research. Among these, PoEM (Liu et al. 2016), a method closely related to ours, utilizes preexisting demonstrations to identify sub-goals, implying that its application is limited without such demonstrations. In contrast to our approach, all the methods we are aware of present their findings using scenarios with a few agents.

## Background

**Multi-agent Pathfinding** In (Classical) Multi-agent pathfinding (Stern et al. 2019), the timeline is discretized to timesteps and the workspace, where $M$ agents operate, is discretized to a graph $G = (V, E)$, whose vertices correspond to the locations and the edges to the transitions between these locations. $M$ start and goal vertices are given, and each agent $i$ has to reach its goal $g_i \in V$ from the start $s_i \in V$. At each timestep, an agent can either stay in its current vertex or move to an adjacent one. An individual plan for an agent $p_i$[1] is a sequence of actions that transfers it between two designated vertices. The plan's cost is equal to the number of actions comprising it.

The MAPF problem asks to find a set of $M$ plans s.t. each agent reaches the goal without colliding with the others. Formally, two collisions are typically distinguished: a vertex collision, where the agents occupy the same vertex at the same timestep, and an edge collision, where the agents use the same edge at the same timestep.

*Lifelong MAPF* (LMAPF) is a variant of MAPF where immediately after an agent reaches its goal, it is assigned to another one (via an external assignment procedure) and has to continue its operation.

[1]In MAPF literature, a plan is typically denoted with $\pi$. However, in RL, this is reserved to denote the policy. As we use both MAPF and RL approaches in this work, we denote a plan as $p$.

**The Considered Decentralized LMAPF Problem** Let a set of agents operate in the shared environment, represented as a graph $G = (V, E)$. The timeline is discretized into the timesteps $T = 0, 1, ..., T_{max}$, where $T_{max}$ is the episode length. Each agent is located initially at the start vertex and is assigned to the current goal vertex. If it reaches the latter before the episode ends, it is immediately assigned another goal vertex. We assume that the *goal assignment* unit is external to the system, and the agents' behavior does not affect the goal assignments. Each agent is allowed to perform the following actions: wait at the current vertex and move to an adjacent vertex. The duration of each action is uniform, i.e., one timestep. We assume that the outcomes of the actions are deterministic and no inaccuracies occur when executing the actions.

Each agent has a complete knowledge of the graph $G$. However, it observes the other agents only *locally*. When observing them, no communication occurs. Thus, an agent does not know the current goals or intended paths of the other agents. It only observes their locations. The observation function can be defined differently depending on the type of graph. In our experiments, we use 4-connected grids and assume that an agent observes the other agents in the area of the size $m \times m$, centered at the agent's current position.

Our task is to construct an individual policy $\pi$ for each agent, i.e., the function that takes as input a graph (global information) and (a history of) observations (local information) and outputs a distribution over actions. Equipped with such policy, an agent at each time step samples an action from the distribution suggested by $\pi$ and executes it in the environment. This continues until timestep $T_{max}$ is reached when the episode ends. Upon that, we compute the *throughput* as the ratio of the number of goals achieved by all agents to episode length. We use it to compare different policies: we assert that $\pi_1$ outperforms $\pi_2$ if the throughput of the former is higher.

**Partially Observable Markov Decision Process** We consider a partially observable multi-agent Markov decision process defined as $M = \langle S, A, U, P, R, O, \mathcal{O}, \gamma \rangle$. At each timestep, each agent $u \in U$, with $U = \{1, \ldots, n\}$, chooses an action $a^{(u)} \in A$. These actions form a joint action $\mathbf{j} \in \mathbf{J} = A^n$, influencing the environment's state transition as per the function $P(s'|s, \mathbf{j}) : S \times \mathbf{J} \times S \to [0, 1]$.

After that, each agent receives an individual observation $o^{(u)} \in O$ based on the global observation function $\mathcal{O}(s, a) : S \times A \to O$, and an individual scalar reward $R(s, u, \mathbf{j}) : S \times U \times \mathbf{J} \to \mathbb{R}$, which depends on the current state, joint action and may be different for different agents. Discount factor $0 \leq \gamma \leq 1$ determines the importance of future rewards.

To make decisions, each agent maintains an action-observation history $\tau^{(u)} \in T = (O \times A)^*$. The latter is used to condition a stochastic policy $\pi^{(u)}(a^{(u)}|\tau^{(u)}) : T \times A \to [0, 1]$. The aim is to obtain (to learn) a policy $\pi^{(u)}$ for each individual agent that maximizes the expected cumulative reward over time.

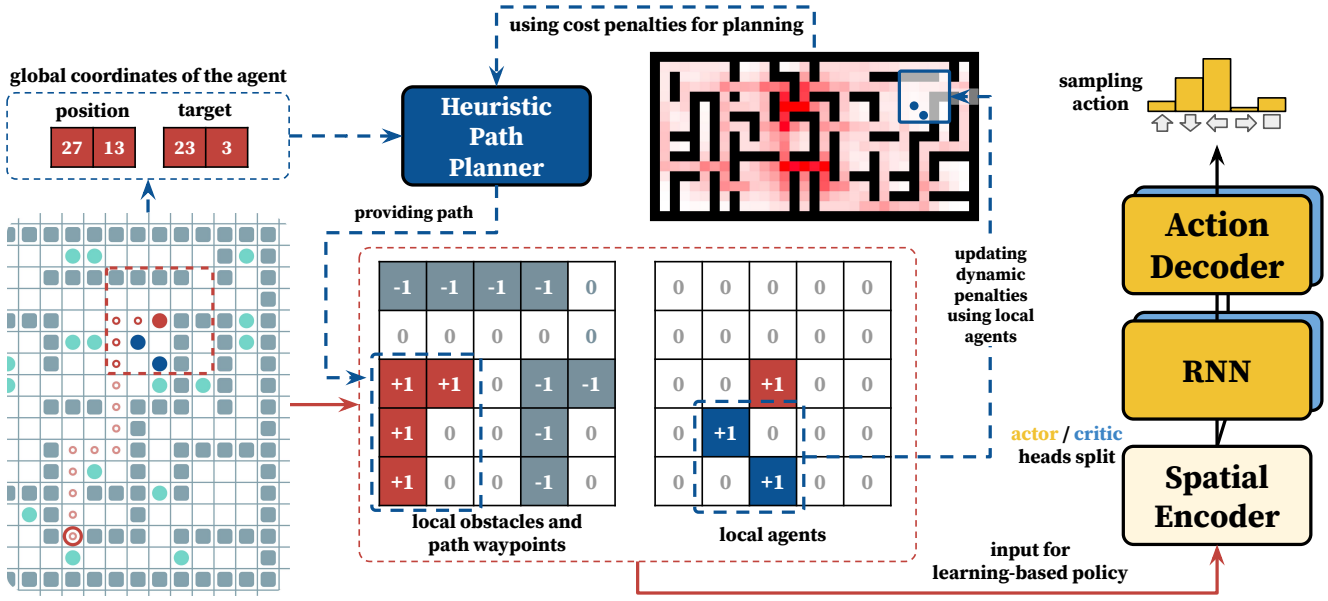


Figure 2: The general pipeline of the FOLLOWER approach. The action selection policy for each agent is decentralized and consists of two modules: Heuristic Path Planner, which addresses the long-term path planning problem, and Learnable Follower, which addresses the short-term conflict resolution task.

## Learn to Follow

The suggested approach, which we dub FOLLOWER, is comprised of the two complementary modules combined into a coherent pipeline shown in Fig. 2. First, a *Heuristic Path Planner* is used to construct an individual path to the goal. Then, a *Learnable Follower* is invoked to follow this path.

### Heuristic Path Planner

The aim of this module is to build a path from the current location of the agent to the goal. The static obstacles are taken into account, while the other agents are not; therefore, the constructed path may go through them. The rationale behind this is that the collision avoidance will be handled later on by the path following policy.

A crucial design choice is which individual path to build. On the one hand, paths with the minimal length are desirable. On the other hand, when the number of agents is high and each agent is following its shortest path, a congestion often arises in the bottleneck parts of the map, such as corridors or doors. This degrades the overall performance dramatically. To this end, we suggest searching not for the shortest paths but rather for the evenly dispersed paths. Intuitively, we wish to distribute the agents across the map to decrease congestion and increase the throughput. This technique is implemented as follows.

Instead of assuming that the transition costs used by a search algorithm (we use A* in our experiments) are uniform, we compute the individual varying transition costs associated with the cells. The individual cost of a transition to a cell is the sum of two components, the static and the dynamic one:

$$cost(c,t) = cost_{st}(c) + cost_{dyn}(c,t). \quad (1)$$

The static cost component depends solely on the topology of the map and does not change through the episode. The dynamic cost component, conversely, is based on the history of the observations of the agent and is dynamically updated.

To estimate the static cost of each cell, we, first, compute the average cost of the paths starting in this cell and ending in all other free cells (we use BFS algorithm for that):

$$avg\_cost(c) = \sum_{c' \in V_{free}(c)} \frac{path\_cost(c,c')}{|V_{free}(c)|}, \quad (2)$$

where $V_{free}(c)$ denotes the vertices reachable from $c$.

Intuitively, the lower values of $avg\_cost(c)$ indicate that a higher number of (the shortest) paths pass through $c$, and, thus, the latter is a potential congestion attractor. Consecutively, the transition to $c$ should be penalized. This is implemented as follows:

$$cost_{st}(c) = \frac{\max_{c' \in V}(avg\_cost(c'))}{avg\_cost(c)}, \quad (3)$$

In other words, the static transition cost to a cell is 1 only if it is the "most rarely used" cell of the grid, while the transition costs to the other (more frequently used) cells are higher.

The dynamic cost, $cost_{dyn}(c,t)$, is based on the personal experience of an agent and changes during the episode. It is computed as follows.

$$cost_{dyn}(c,t) = \sum_{t' \in [0,t]} AgentAtCell(c,t'), \quad (4)$$

where $AgentAtCell(c, t')$ is a function that returns 1 *iff* some agent was observed (by the current agent) at cell $c$ at timestep $t'$ and returns 0 otherwise.

Intuitively, the dynamic cost penalizes transitions to the cells that are frequently used by the other agents. Indeed, each agent maintains its own dynamic costs. Moreover, to avoid the negative impact of over-accumulating the dynamic penalties, whenever an agent reaches its goal it resets the dynamic costs of all grid cells.

Empirically, both the precomputed transition costs and the individual dynamic costs contribute toward greater efficiency of our solver as will be shown later.

### Learnable Follower

This module implements a learnable policy tailored to follow the provided path while avoiding the collisions with the other agents. The policy function is approximated by a (deep) neural network and, as the agents are assumed to be homogeneous, a single network is utilized during training (a technique referred to as *policy sharing*).

The input to the neural network represents the local observation of an agent and is comprised of a $2 \times m \times m$ tensor, where $m$ is the observation range. The channels of the tensor encode the locations of the static obstacles combined with the current path and the other agents; see Fig. 2.

The input goes through the *Spatial Encoder* first, and then the network is split into the actor and critic heads, with the *RNN blocks* designed to memorize the observation history. The output of the actor is the *Action Decoder*, which produces an action distribution. The *Critic Head* generates a value estimate, which is needed for training purposes only.

The pipeline employs a policy optimization algorithm, rewarding the agent with $+r$ for reaching the first waypoint (i.e. the next grid cell on the constructed path). If the agent deviates from or approaches the waypoint, the heuristic path planner is reactivated. This is advantageous in situations where taking a detour to avoid congestion with other agents is beneficial in achieving the overall goal. The focus on reaching the first waypoint provides a dense reward signal. While the agent is rewarded for reaching the nearest waypoint, its decision-making extends beyond the immediate vicinity of that waypoint. It's important to note that the FOLLOWER aims to maximize rewards by navigating through multiple waypoints en route to the global goal. It takes into account potential long-term cumulative rewards, such as allowing another agent to pass and then following the path, instead of obstructing each other.

The task of the learning process is to optimize the shared policy $\pi_\theta^u$ (i.e. the same policy for each agent) to maximize the expected cumulative reward. During the training process, rollouts (sequences of observations, rewards, and actions) are gathered from multiple environments with varying numbers of agents. The shared policy $\pi_\theta$ (actor network) is continually updated using the PPO clipped loss (Schulman et al. 2017).

In practice, the observation history $\tau^u$ is effectively modeled using a recurrent neural network (RNN) integrated into the actor and critic heads. The actor network is parameterized by $\theta$, while the critic network is parameterized by $\phi$. In our approach, we specifically utilize the GRU architecture (Chung et al. 2014).

During the decentralized inference, each agent uses a copy of the trained weights, and the other parameters remain unchanged. The proposed FOLLOWER scheme, despite its simplicity, allows the agent to separate the two components of the overall policy transparently and does not require the involvement of any expert data for training. Finally, the reward function used is simple and does not require involved manual shaping.

## Experimental Evaluation

To evaluate the efficiency of the proposed method, we conduct a set of experiments, comparing it with the state-of-the-art LMAPF algorithms on different maps. The training and evaluation of the presented approaches is held in fast and scalable POGEMA[2] environment.

The path planner of FOLLOWER is based on A*. The learnable policy is implemented as the neural network of the following architecture. The *Spatial Encoder* is a ResNet (He et al. 2016) with an additional Multi-Layer Perceptron (MLP) in the output layer. The *Action Decoder* and the *Critic Head* are recurrent neural networks, based on the GRU. The total number of parameters is 5M. Moreover, we developed an additional fast variant of FOLLOWER, FOLLOWERLITE, which has only 3,678 parameters, excludes the RNN component (see the Arxiv version of the paper for more details[3]) and is implemented fully in C++.

For training the episode length was set to 512. The agent's field-of-view was $11 \times 11$, the number of agents varied in range: 128, 256. The reward $r$ was a small positive number, i.e. $r = 0.01$. More details about tuning the hyperparameters are reported in the Arxiv version of the paper. Upon fixing the parameters, the final policy of FOLLOWER is trained for 1 billion steps using a single NVIDIA A100 in approximately 18 hours. FOLLOWERLITE is trained for 20 million steps with a single NVIDIA TITAN RTX GPU in approximately 30 minutes.

### Comparison With the Learnable Methods

In the first series of experiments, we compare our method with the state-of-the-art learnable MAPF solvers – SCRIMP (Wang et al. 2023), PRIMAL2 (Damani et al. 2021) and PICO (Li et al. 2022). PRIMAL2 is a seminal approach specifically tailored for solving LMAPF problems. SCRIMP and PICO are the decentralized MAPF solvers that were (straightforwardly) adopted by us to handle LMAPF setting. In the experiments we utilize the environmental conflict-handling mechanism from PRIMAL2 – when two or more agents decide to move to the same cell, only one of them succeeds while the rest stay put. Noteworthy, SCRIMP has a dedicated negotiation procedure for conflict resolution, which we did not modify.

As learnable methods assume training on a certain type of maps, we use the maps suggested by the authors of the respective baselines for a fair comparison. Specifically, we

[2]https://github.com/AIRI-Institute/pogema

[3]https://arxiv.org/abs/2310.01207

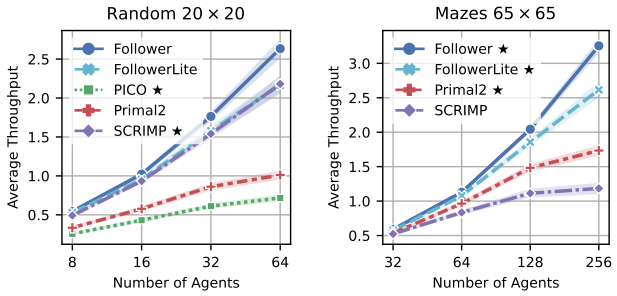


Figure 3: Average throughput on random and maze-like maps. The shaded area indicates 95% confidence intervals. The symbol ⋆ marks the approaches that were trained on the corresponding type of maps.

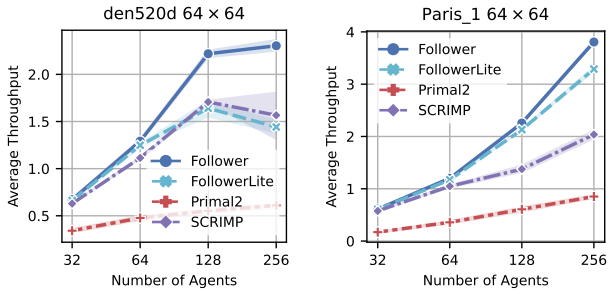


Figure 4: The results on MovingAI `den520d` and `Paris_1` maps. The results are averaged over 10 runs for each data point. The shaded area indicates 95% confidence intervals.

made a comparison on two types of maps – the maze-like maps of size $65 \times 65$ on which PRIMAL2 was originally trained, and $20 \times 20$ grids with randomly placed obstacles, that were used for training PICO and SCRIMP. We use the readily available weights for PRIMAL2 and SCRIMP neural networks from the authors' repositories. PICO was trained by us using the open-source code of its authors. Our solvers, FOLLOWER and FOLLOWERLITE, were trained on the maze-like maps only.

For evaluation, each solver is faced with 10 different maze-like and 40 random maps that were not used during training. Each map is populated with an increasing number of agents, ranging from 32 to 256 agents for maze-like maps and from 8 to 64 for random ones. Start and goal locations for the agents are generated randomly in a reproducible way (so each solver gets the same starts and goals). The length of the episode is set to 512.

The results of the first series of experiments are depicted in Fig. 3. The OX-axis shows the number of agents, and the OY-axis demonstrates the average throughput. Overall, on both types of maps FOLLOWER demonstrates the best results, notably outperforming all the competitors. The main competitor on the maze-like maps, PRIMAL2, shows almost twice less throughput on the instances with 256 agents. The main competitor on random maps, SCRIMP, shows results equal to the lightweight version of FOLLOWER, i.e. FOLLOWERLITE. However, the results of SCRIMP on the maze-like maps are much worse, that indicates its low ability to generalize. PICO demonstrates the worst results on random maps out of all the evaluated approaches, though it was trained on this type of maps. Therefore, we exclude PICO from the rest of the experiments.

**Out-of-distribution evaluation.** An important quality of any learnable algorithm is its *generalization*, i.e. the ability to solve problem instances that are not similar to the ones that have been used for training. We have already seen that FOLLOWER generalizes well and can outperform SCRIMP on random maps though FOLLOWER was not trained on this type of maps. Now we run an additional evaluation where we compare FOLLOWER, FOLLOWERLITE, PRIMAL2 and SCRIMP on two (unseen during learning) maps from the well-known in the MAPF community MovingAI benchmark (Stern et al. 2019): `den520d` and `Paris_1`. The former map is taken from a video game, while the latter one corresponds to the part of a real city. Their topologies are quite different from the one of the maps used for training. Both of the maps were downscaled to the size of $64 \times 64$.

The results of these experiments are presented in Fig. 4. Again FOLLOWER significantly outperforms all the competitors. PRIMAL2 demonstrates very low throughput on both maps, that indicates its poor ability for generalization. Compared to PRIMAL2, SCRIMP shows itself much better in terms of generalization, but in the best case it is only able to demonstrate the results comparable to the lightweight version of our approach, i.e. FOLLOWERLITE. Additional results of the out-of-distribution experiments are presented in the Arxiv version of the paper.

## Comparison With Non-Learnable Approaches

We use two non-learnable approaches for comparison – RHCR[4] (Li et al. 2021) and PIBT[5] (Okumura et al. 2022). These are two different centralized approaches: RHCR is the state-of-the-art search-based method aiming at the high-quality LMAPF solutions at the expense of the limited scalability, while PIBT is the state-of-the-art rule-based approach that is extremely fast, but provides solutions of a moderate quality.

RHCR solver requires setting a time limit for planning. We set it either to 1 or 10 seconds (both variants are reported with the according names). We chose PBS (Ma et al. 2019a) as the planning method inside RHCR since it showed the best results in the original paper. We have also tuned the planning horizon $(2, 5, 10, 20)$, the re-planning rate $(1, 5)$ and found that the best throughput is achieved by RHCR when the first parameter is set to 20 and the second one to 5 (see the Arxiv version of the paper for more details). We use these values in our experiments. The other parameters of RHCR are left default.

The comparison is performed on the `warehouse` map from the original RHCR paper (Li et al. 2021). The maximum number of agents is limited to 192, due to the restric-

[4]https://github.com/Jiaoyang-Li/RHCR

[5]https://github.com/Kei18/pibt2

tions for starting locations introduced in (Li et al. 2021). We generated 10 random instances for each number of agents for evaluation.

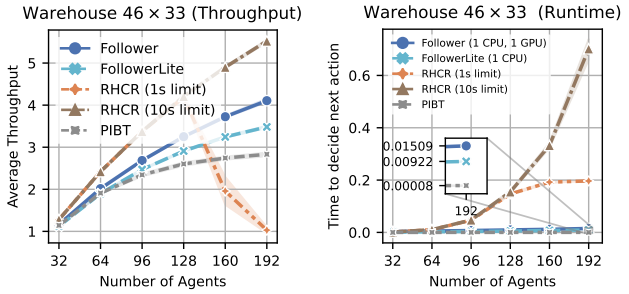


Figure 5: Average throughput and runtime (seconds per joint action) on `warehouse` map. The shaded area indicates 95% confidence intervals.

The results are presented in Fig. 5. As can be seen on the left part of the Fig. 5, both versions of RHCR significantly outperform the other solvers when the number of agents is up to 128. However, when this number increases to 160 and 192, the performance of RHCR with a 1s time cap degrades dramatically. It is then outperformed by both FOLLOWER and FOLLOWERLITE. This highlights the principal limitation of the centralized approach: it does not scale well to a large number of agents, especially when a strict time limit for finding a MAPF solution is imposed. PIBT does not have such a problem with scalability but its throughput is the lowest among all the evaluated methods.

To better understand how the runtime of the evaluated methods is affected by the increasing number of agents, see the right pane of the Fig. 5. In this plot each data point indicates how much time on average is spent to decide the next action for all agents. Indeed, FOLLOWER needs much less time to choose an action and, consequtively, scales better to the increasing number of agents compared to RHCR. To make it fair we ran all the solvers on a single CPU. We measured the time (in seconds) required for a solver to decide the next action *for all agents*. As expected, PIBT is the fastest approach, as its rule-based procedures are computationally cheap compared to the ones used by RHCR and FOLLOWER. Recall, however that the throughput of PIBT is inferior. Moreover, in practice our method can be parallelized, i.e. run individually on each agent, while the others can not. Running FOLLOWER individually on each agent will result in decreasing the runtime.

## Ablations

In this experiment, we investigate the impact of different components on the performance of FOLLOWER. To this end we turn them off and run the resultant solver on the `warehouse` map from the RHCR paper. Specifically, FOLLOWER (no RL) omits the learnable policy. At each timestep, it plans a path, taking into account other observable agents as static obstacles, and selects its first action for the execution. If no path is found a random action is picked. FOLLOWER (no dynamic cost) and FOLLOWER (no static cost) use both planning and learning components, but they do not utilize one of the introduced techniques that penalize transitions to the frequently used cells.

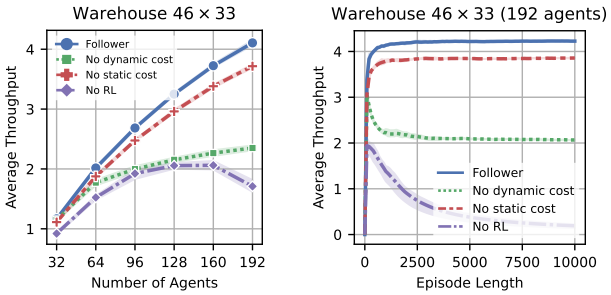


Figure 6: Impact of FOLLOWER components on it's performance. The shaded area indicates 95% confidence intervals.

The results are shown in Fig. 6 (left pane). First, note that the performance of FOLLOWER (no RL) is inferior, which justifies the importance of the learnable policy and its contribution to the efficiency of FOLLOWER. The same can be said about the cost-penalizing techniques. Overall, it is clear that *all* components of FOLLOWER are crucial; omitting any of them results in a notable degradation of performance.

In addition, we run FOLLOWER with different episode lengths (up to $10,000$), as the initial distribution of the agents can be very different from the distribution that happens after some time. The results are shown in Fig. 6 (right pane). Notably, the absense of RL policy and dynamic cost accumulation lead to a very low throughput in the limit. We explain this by the congestion that occurs and grows like a rolling snowball and prevent the agents from reaching their goals. Indeed, FOLLOWER copes well with this as its throughput monotonically increases with the episode length and then plateaus.

## Summary

The proposed approach surpasses learnable decentralized competitors, especially when the number of agents is large or when dealing with maps different from the training ones. Both the learnable component and the cost-penalizing techniques are essential to FOLLOWER's performance. Furthermore, FOLLOWER scales much better than a state-of-the-art search-based solver and provides solutions of better quality compared to modern rule-based solver.

## Conclusion

In this study, we addressed the challenging problem of decentralized lifelong multi-agent pathfinding. We proposed a solution that leverages heuristic search for long-term planning and reinforcement learning for short-term conflict resolution. Our method consistently outperforms decentralized learnable competitors. Moreover, it provides a better trade-off between the scalability and the solution quality compared to the modern search-based and rule-based planners. Directions for future research may include: enriching the action space of the agents, handling uncertain observations and external stochastic events.

## Acknowledgments

This work was partially supported by the Analytical Center for the Government of the Russian Federation in accordance with the subsidy agreement (agreement identifier 000000D730321P5Q0002; grant No. 70-2021-00138).

## References

Chen, Z.; Alonso-Mora, J.; Bai, X.; Harabor, D. D.; and Stuckey, P. J. 2021. Integrated task assignment and path planning for capacitated multi-agent pickup and delivery. *IEEE Robotics and Automation Letters*, 6(3): 5816–5823.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Damani, M.; Luo, Z.; Wenzel, E.; and Sartoretti, G. 2021. PRIMAL _2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong. *IEEE Robotics and Automation Letters*, 6(2): 2666–2673.

de Wilde, B.; ter Mors, A. W.; and Witteveen, C. 2013. Push and rotate: cooperative multi-agent path planning. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, 87–94.

Hafner, D.; Lillicrap, T.; Norouzi, M.; and Ba, J. 2021. Mastering atari with discrete world models. In *ICLR*.

Hafner, D.; Pasukonis, J.; Ba, J.; and Lillicrap, T. 2023. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Kansky, K.; Vaidyanath, S.; Swingle, S.; Lou, X.; Lazaro-Gredilla, M.; and George, D. 2023. PushWorld: A benchmark for manipulation planning with tools and movable obstacles. arXiv:2301.10289.

Li, J.; Tinka, A.; Kiesel, S.; Durham, J. W.; Kumar, T. S.; and Koenig, S. 2021. Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021)*, 11272–11281.

Li, W.; Chen, H.; Jin, B.; Tan, W.; Zha, H.; and Wang, X. 2022. Multi-Agent Path Finding with Prioritized Communication Learning. *2022 International Conference on Robotics and Automation (ICRA)*, 10695–10701.

Liu, M.; Amato, C.; Anesta, E. P.; Griffith, J. D.; and How, J. P. 2016. Learning for Decentralized Control of Multiagent Systems in Large, Partially-Observable Stochastic Environments. In *AAAI Conference on Artificial Intelligence*.

Liu, M.; Ma, H.; Li, J.; and Koenig, S. 2019. Task and path planning for multi-agent pickup and delivery. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019)*, 1152–1160.

Lumelsky, V. J.; and Harinarayan, K. 1997. Decentralized motion planning for multiple mobile robots: The cocktail party model. *Autonomous Robots*, 4(1): 121–135.

Ma, H.; Harabor, D.; Stuckey, P. J.; Li, J.; and Koenig, S. 2019a. Searching with consistent prioritization for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7643–7650.

Ma, H.; Hönig, W.; Kumar, T. K. S.; Ayanian, N.; and Koenig, S. 2019b. Lifelong Path Planning with Kinematic Constraints for Multi-Agent Pickup and Delivery. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, 7651–7658.

Ma, H.; Li, J.; Kumar, T.; and Koenig, S. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2017)*, 837–845.

Ma, Z.; Luo, Y.; and Ma, H. 2021. Distributed heuristic multi-agent path finding with communication. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 8699–8705. IEEE.

Milani, S.; Topin, N.; Houghton, B.; Guss, W. H.; Mohanty, S. P.; Vinyals, O.; and Kuno, N. S. 2020. The MineRL Competition on Sample-Efficient Reinforcement Learning Using Human Priors: A Retrospective. In *NeurIPS Competition track*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. a.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2019. Priority inheritance with backtracking for iterative multi-agent path finding. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 535–542.

Okumura, K.; Machida, M.; Défago, X.; and Tamura, Y. 2022. Priority inheritance with backtracking for iterative multi-agent path finding. *Artificial Intelligence*, 310: 103752.

Rashid, T.; Samvelyan, M.; De Witt, C. S.; Farquhar, G.; Foerster, J.; and Whiteson, S. 2018. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement Learning. In *35th International Conference on Machine Learning, ICML 2018*, volume 10, 6846–6859. ISBN 9781510867963.

Riviere, B.; Hönig, W.; Yue, Y.; and Chung, S.-J. 2020. Glas: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning. *IEEE Robotics and Automation Letters*, 5(3): 4249–4256.

Samvelyan, M.; Rashid, T.; De Witt, C. S.; Farquhar, G.; Nardelli, N.; Rudner, T. G.; Hung, C. M.; Torr, P. H.; Foerster, J.; and Whiteson, S. 2019. The StarCraft multi-agent challenge. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, volume 4, 2186–2188. ISBN 9781510892002.

Sartoretti, G.; Kerr, J.; Shi, Y.; Wagner, G.; Kumar, T. S.; Koenig, S.; and Choset, H. 2019. Primal: Pathfinding via

reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3): 2378–2385.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Skrynnik, A.; Yakovleva, A.; Davydov, V.; Yakovlev, K.; and Panov, A. I. 2021. Hybrid Policy Learning for Multi-Agent Pathfinding. *IEEE Access*, 9: 126034–126047.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. S.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the 12th Annual Symposium on Combinatorial Search (SoCS 2019)*, 151–158.

Tang, H.; Hao, J.; Lv, T.; Chen, Y.; Zhang, Z.; Jia, H.; Ren, C.; Zheng, Y.; Fan, C.; and Wang, L. 2018. Hierarchical Deep Multiagent Reinforcement Learning. *ArXiv*, abs/1809.09332.

Van den Berg, J.; Lin, M.; and Manocha, D. 2008. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proceedings of The 2008 IEEE International Conference on Robotics and Automation (ICRA 2008)*, 1928–1935. IEEE.

Wang, K.-H. C.; and Botea, A. 2011. MAPP: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research*, 42: 55–90.

Wang, X.; Wang, S.; Liang, X.; Zhao, D.; Huang, J.; Xu, X.; Dai, B.; and Miao, Q. 2022. Deep Reinforcement Learning: A Survey. *IEEE transactions on neural networks and learning systems*, PP.

Wang, Y.; Xiang, B.; Huang, S.; and Sartoretti, G. 2023. SCRIMP: Scalable Communication for Reinforcement-and Imitation-Learning-Based Multi-Agent Pathfinding. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, 2598–2600.

Wong, A.; Bäck, T.; Kononova, A. V.; and Plaat, A. 2023. Deep multiagent reinforcement learning: Challenges and directions. *Artificial Intelligence Review*, 56(6): 5023–5056.

Yu, C.; Velu, A.; Vinitsky, E.; Gao, J.; Wang, Y.; Bayen, A.; and Wu, Y. 2022. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35: 24611–24624.

Zhu, H.; Brito, B.; and Alonso-Mora, J. 2022. Decentralized probabilistic multi-robot collision avoidance using buffered uncertainty-aware Voronoi cells. *Autonomous Robots*, 46(2): 401–420.