When to Switch: Planning and Learning For Partially Observable Multi-Agent Pathfinding

Alexey Skrynnik, Anton Andreychuk, Konstantin Yakovlev, Aleksandr Panov

***Abstract*—Multi-agent pathfinding is a problem that involves finding a set of non-conflicting paths for a set of agents confined to a graph. In this work, we study a MAPF setting, where the environment is only partially observable for each agent, i.e. an agent observes the obstacles and other agents only within a limited field-of-view. Moreover, we assume that the agents do not communicate and do not share knowledge on their goals, intended actions, etc. The task is to construct a policy that maps the agent's observations to actions. Our contribution is multifold. First, we propose two novel policies for solving partially observable multi-agent pathfinding: one based on heuristic search and another one based on reinforcement learning. Next, we introduce a mixed policy that is based on switching between the two. We suggest three different switch scenarios: the heuristic, the deterministic, and the learnable one. A thorough empirical evaluation of all the proposed policies in a variety of setups shows that the mixing policy demonstrates the best performance, is able to generalize well to the unseen maps and problem instances, and, additionally, outperforms the state-of-the-art counterparts (PRIMAL2 and PICO). The source-code is available at https://github.com/AIRI-Institute/when-to-switch.**

***Index Terms*—MAPF, PO-MAPF, Reinforcement Learning, Planning**

I. INTRODUCTION

Multi-agent pathfinding (MAPF) is a challenging problem with topical applications in robotics, video games, logistics, etc. Typically, in MAPF, agents are confined to a graph, and at each timestep, an agent can either move to an adjacent vertex or stay put [1]. The task of each agent is to reach a predefined goal vertex. If the graph is undirected, the solution can be found in polynomial time [2] while finding the optimal solution w.r.t. a range of the objective functions is NP-hard [3]. Moreover, if the graph is directed, even the decision variant of MAPF is intractable [4].

Currently, multiple variants of MAPF formulations are considered. [5] considers agents of different sizes. In [6], MAPF with non-uniform cost actions is studied. [7] proposes a method that does not assume discrete time steps. An online variant of MAPF, where some agents appear after the other has already started executing the plan, is studied in [8]. MAPF with possibly delaying agents is explored in [9]. A lot of papers have studied the lifelong variant(s) of MAPF, where each finished agent is assigned a new goal immediately; see, e.g. [10]. MAPF combined with task allocation is considered in [11].

Corresponding author: skrynnikalexey@gmail.com

Anton Andreychuk is with AIRI, Moscow, Russia. Alexey Skrynnik, Aleksandr Panov and Konstantin Yakovlev are with Federal Research Center for Computer Science and Control of Russian Academy of Science and AIRI, Moscow, Russia. Konstantin Yakovlev is also with HSE University.

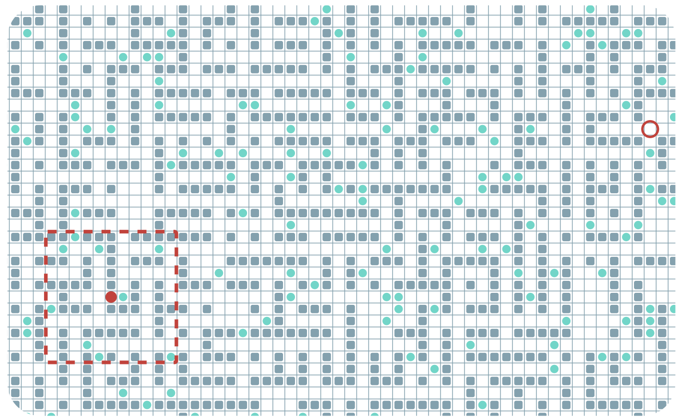

Fig. 1. A PO-MAPF instance: The red agent (like any other) observes only a local patch of the environment within its field-of-view (inside a dashed square). The goal location of this agent is marked with the empty red circle (in the upper-right portion of the map). The other agents are shown in teal.

Overall, MAPF is an extensively studied problem these days. Numerous algorithms exist that take into account the specifics of different MAPF formulations. Still, the vast majority of such formulations assume that the environment is fully observable and that there is a centralized controller, which possesses all the information and is actually in charge of solving MAPF. By contrast, in this work, we focus on a variant of MAPF when the environment is only partially observable for each agent: PO-MAPF. Fig 1 depicts an instance of PO-MAPF. PO-MAPF has no centralized controller, and each agent at each timestep has to decide which action to take based on the local observation or the history of local observations. The latter means that at any time, an agent observes the obstacles and other agents only within a limited field-of-view. Besides, in this paper, we assume that the agents do not share any information with each other. This makes the PO-MAPF problem particularly challenging.

PO-MAPF requires different approaches compared to the fully observable centralized MAPF. In the former case, we do not seek for a set of conflict-free plans, but rather for a policy that maps agents' observations onto actions in such a way that it maximizes the odds of reaching the goal while avoiding the collisions and minimizing the number of actions performed.

To this end, we introduce two novel and conceptually different policies for PO-MAPF. The first one is based on the search-based re-planning (REPLAN). At each timestep, an agent builds the shortest path to its goal using a history of the egocentric observations by a heuristic search algorithm. Other agents are considered as obstacles that need to be avoided.

To mitigate the possible deadlocks and oscillating behavior of the agents, we augment re-planning with additional decision-making procedures that pick a greedy or wait action under certain conditions.

The second policy is a learnable one. It utilizes a specifically designed reinforcement learning algorithm: Evolving Policy Optimization with Memory (EPOM). EPOM uses an actor-critic architecture with a recurrent neural network as a state approximator. One of the novel features of EPOM setting it apart from similar approaches, is the mechanism of augmenting the current observation with a patch of the previously observed and memorized map. Not only does this help stabilize learning, but it also contributes to higher performance of the policy. To determine the hyperparameters of the model during the learning process, a population-based training approach is implemented [12].

The following features distinguish our learnable policy from the similar ones proposed in the literature earlier [13], [14], [15]):

- when training we do not rely on external guidance, i.e. on expert demonstrations from conventional MAPF solvers or single-agent planners;
- no involved reward-shaping is used for training (our reward function, as well as the loss function, is simplistic)
- our policy is agnostic to the observation range due to the introduced memory mechanism, i.e. being learned with one observation range; it is capable of functioning (without sacrificing the performance) with another observation range.

As a next step, we suggest and investigate a combination of REPLAN and EPOM, introducing a switch mechanism that executes both policies in parallel and outputs the final action based on the several proposed strategies: the heuristic, the deterministic, and the learnable one. Empirically, we show that one variant of such switch leads to a constantly better performance in a large variety of setups and outperforms the state-of-the-art counterparts: PRIMAL2 [16] and PICO [17].

Summarizing the above, the contributions of this work can be stated as follows:

- We study the PO-MAPF setup when no communication and data-sharing between the agents is possible, and introduce two novel policies tailored to this setting: the search-based one and the learnable one. To the best of our knowledge, we are the first to introduce the (well-performing) policies for PO-MAPF, when the information on other agents' goals, actions, or plans is not available.
- Further on, we introduce three novel ways to combine the aforementioned policies into a single hybrid policy that utilizes both the search-based (re)planning and the learning-based decision-making.
- We conduct a thorough empirical evaluation of the suggested techniques to show their scalability and ability to generalize well to the unseen maps and problem instances. We compare three our hybrid policies to the state-of-the-art competitors, PRIMAL2 and PICO, and show that the latter are outperformed by the approaches introduced in the paper.

II. RELATED WORK

MAPF is increasingly gaining attention recently, as well as the topic of using the learnable components in multi-agent systems [18], [19], [20]. Here we, first, briefly overview the works focusing on solving a conventional MAPF formulation, i.e. the one that assumes the existence of the centralized controller and the full knowledge of the environment. Then, we proceed to the sub-areas that are more closely related to our work, i.e. decentralized MAPF, learnable techniques in solving MAPF, and Multi-agent Reinforcement Learning.

a) Centralized MAPF: Most works on MAPF assume a central controller that is in charge of constructing conflict-free plans before the agents actually start moving in the environment. One of the early search-based algorithms to solve this variant of MAPF optimally is introduced in [21]. It augments the search in the joint actions space and employs several techniques to reduce the branching factor. Thus, this planner can be deemed as *fully coupled*. To deal with a high branching factor and huge action space, [22] introduces a sampling-based approach based on the RRT algorithm, called MA-RTT*. MA-RRT* got some extensions, such as MA-RRT*FN[23], that helped improve the usage of memory spent on trees. However, such sampling-based approaches are only applicable in cases of sparse scenarios with few agents (up to 10). M* [24] postpones the search in the combined action space until the conflict between the agents is encountered. Similarly, CBS [25], another prominent optimal MAPF solver, relies on individual re-planning that is triggered by the detection of conflicts in the set of plans. Thus, the latter two algorithms can be viewed as the *semi-coupled* MAPF solvers. Nevertheless, they are sill limited as they scale poorly to large numbers of agents. The most scalable yet suboptimal (and even incomplete in general) techniques are the ones based on what is known as prioritized planning [26], [27], [28]. In this case, individual planning for each agent is carried out sequentially (in accordance with the imposed priority ordering) and the previously planned agents are treated as dynamic obstacles. Thus, prioritized planners can be attributed as *fully decoupled*, i.e. planning for an agent cannot lead to altering the path of the other agent, which has already been constructed. In this work, we study another setting for MAPF, i.e. when each agent acts individually (based on its local observations), with no centralized controller. Still, we empirically compare our approach with the prioritized planning algorithm CA* [29], as it is a widely used MAPF baseline.

b) Decentralized MAPF: Algorithms like MAPP [30] or DiMPP [31] solve MAPF in a decentralized fashion, meaning that each agent performs a search individually and then starts moving along the path. When conflicts are detected, they are resolved locally, and the agents proceed. Notably, these algorithms assume a fully observable environment, unlike the method presented in this paper. Sometimes prioritized planning algorithms (described above) are characterized as decentralized, based on the fact that each agent conducts its own search. However, the agents in prioritized planning globally share the information about their planned paths, as the agents with lower priorities have to avoid the paths of the higher-priority agents. Thus, we do not attribute these algorithms as the decentralized ones.

Decentralized algorithms like ORCA [32], BVC [33], and others are also related to MAPF. However, these assume that the agents are not confined to a graph, like they are in MAPF. Rather, they are free to arbitrarily move in the workspace. In practice, these algorithms are prone to deadlocks and struggle to solve the instances where the coordination between the agents is needed. Another decentralized approach called DMA-RRT is introduced in [34]. Individual plans for the agents are built via the RRT algorithm, and the agents are able to communicate with each other, modify their plans to eliminate collisions, and improve the overall performance.

The main difference between the (decentralized) algorithm presented in this work and the aforementioned ones is that the former does not assume the full knowledge of the environment beforehand (as in MAPP) and allow the agents to move only through the graph, representing the environment (unlike ORCA or BVC), without any ability to communicate (unlike DMA-RRT).

c) Learning-Based MAPF: Recently, learning-based approaches capable of solving decentralized (and often, partially observable) MAPF have started gaining attention. [13] introduced a learnable policy called PRIMAL. Later, it was modified and extended to a lifelong setting in [16]. Both these works utilize expert demonstrations and non-trivial manually-shaped rewards for learning. Moreover, they assume that not only the current locations of the other agents but also the information about the agents' goal locations are included in the observation. Similar assumptions are adopted in [14], suggesting another learnable approach to decentralized PO-MAPF, which is tailored to the agents with a non-trivial dynamic model (e.g., quadrotors). Learnable methods that assume the full knowledge of the environment (but not the global knowledge of the other agents' locations) are proposed in [15], [35]. Another recently presented approach, PICO [17], is also tailored to solve PO-MAPF problems, but allows agents, that see each other in observations, to communicate.

Our method is different from the mentioned works in that it assumes zero information-sharing between the agents, meaning that the paths/goals of the other agents are not known and presented in the observation (unlike the mentioned works). Moreover, we do not rely on expert demonstrations for training and use simplistic reward function rather than involving hand-shaped rewards. In the empirical evaluation, we compare our method with PRIMAL2 and PICO.

d) Multi-Agent Reinforcement Learning (MARL) and Hybrid Polices: Reinforcement learning (RL) researchers also explore domains where multiple agents need to collaborate to achieve cooperative behaviors. These domains often include video games, such as Starcraft [36], characterized by large observation spaces and partial observability. Many algorithms for learning cooperative behaviors assume partial decentralization of agent training and rely on information-sharing among agents.

For instance, QMIX [37] utilizes a mixing neural network that has access to the global state during training, while FACMAC [38] employs a decentralizable joint action-value function with per-agent factorization. Additionally, there is considerable interest in enabling agents to communicate with each other [39], [40], [41].

In contrast to existing approaches, our method exhibits scalability to a large number of agents and larger environments (with a large global state), while maintaining a more gradual decline in performance.

The effects of the on-policy method investigated in this paper under the conditions of using the experience gained using other polices are also covered in the literature. When using well-known methods, such as Deep Deterministic Policy Gradient (DDPG) [42] and Twin Delayed DDPG [43], a particular focus is on the features of policy gradient algorithms in the off-policy setting. Works, such as [44], [45], consider the stability of on-policy approaches together with off-policy methods or in the presence of irreversible events. In our work, we pay attention to the noise effect in the recurrent memory block, which serves as a state approximator, and show that in switches for PO-MAPF environments, it does not lead to irreversible degradation of overall performance.

III. Problem Statement

First, we revoke the conventional MAPF formulation and then introduce the PO-MAPF problem.

a) MAPF: Consider n agents confined to an undirected graph $G=(V,E)$ and a discretized timeline $T=\{0,1,2,...\}$. Initially, at $t=0$, the agents are located at their start vertices $Starts=\{start_1,...,start_n\}$, while their goal vertices are given, too: $Goals=\{goal_1,...,goal_n\}$. At each timestep, an agent can either wait in its current vertex or move to an adjacent one. The duration of the wait/move action is 1 timestep. The individual plan, pl_i, is a sequence of actions performed at consecutive timesteps that brings the agent i from $start_i$ to $goal_i$. Two individual plans are said to contain a vertex conflict if the agents following them occupy the same graph vertex at the same timestep. Similarly, an edge conflict occurs when the agents traverse the same edge in the opposite directions at the same timestep. The problem is to find a set of individual plans, one for each agent, such that any pair of them is conflict-free.

Notably, two different conventions on how agents behave at their target locations are known: stay-at-target and disappear-at-target. In this work, we assume that agents disappear upon reaching their goals, following [16] and [46].

b) Partially Observable MAPF: The principal difference between the classical MAPF and the PO-MAPF is that G is not given as the input explicitly, but instead, the observation function O is provided (the same for all agents). At each timestep, each agent obtains an observation $o_t=O(v,t)$, where v is the vertex occupied by the agent. For example, if G is a 4-connected grid, o_t can contain information about which neighboring cells are blocked/unblocked, which of them are occupied by the other agents, etc. The problem of achieving the goal vertex for each agent now boils down to sequential decision-making, i.e. at each timestep, an agent has to decide which action, either wait or move, to perform. The PO-MAPF problem is to construct a decision-making policy π—the same for all agents—that maps (the history of) observations onto actions. Indeed, π should maximize the chance of reaching the goal while minimizing the number of actions needed.

Depending on the PO-MAPF instance and on the policy π, the agents can continuously move around (or endlessly wait) without reaching their goals. To this end, the time limit (also known as the *episode length*) T_{max} is introduced and becomes a part of the PO-MAPF problem.

From the *engineering perspective*, the introduced formulation is inspired by the real multi-robotic systems. Partial observability is a direct consequence of the limited range of the conventional robotic sensors. In the case when the kinodynamic model of the robot is known and there is a robust controller, discrete actions correspond to a set of pre-computed motion primitives. Finally, in robotics, mapping algorithms often produce maps in the form of highly discretized occupancy grids that can be upscaled to coarser grids in which the robot fits to a cell (our setting).

c) Observation Model: The definition of PO-MAPF is agnostic to the observation function, which is assumed to be given as an input. In this work, we adopt the following assumptions to specify the observation model. First, the graph G is assumed to be a 4-connected grid composed of both blocked and unblocked cells. Second, the agent occupying the cell with the coordinates (i, j) is able to observe the status of the cells $(i \pm R, j \pm R)$, where R is the observation radius. Thus, the observation is a patch of a grid the size $(2 \cdot R+1) \times (2 \cdot R+1)$ centered at the currently occupied cell. Technically, this observation is represented as two matrices: the one that encodes the positions of the static obstacles and the other one that encodes the positions of the agents. We also include in the observation the current coordinates of the agent and its goal coordinates w.r.t. the relative coordinate frame, i.e. the one that is centered at the start location of the agent.

Crucially, any information regarding the other agents, except their current locations (e.g., their goals, paths (or path segments) to the goals, etc.), *is not included* in the observation.

d) Communication Model and Conflict Resolution: We assume that no communication is possible between the agents, i.e. they cannot share the information about their intended goals, future moves etc. We believe that PO-MAPF with no communication is the most restrictive and challenging variant of the problem to be solved. Under such assumptions, two (or more) agents can choose to move to the same cell at the same timestep, leading to a collision. To avoid this, several options can be considered: *i*) all agents stay where they are; *ii*) an arbitrarily chosen agent performs an action while the others stay put; or *iii*) the episode ends. We stick to the first option, which resembles the robotic applications: when two robots bump into each other, they stay where they are. As we use the discretized spatial representation, i.e. grid, "where they are" corresponds to the grid cells the agents occupy. [1]

[1]We have also tried to experiment with the second collision-resolution method: when one agent in a conflict is randomly chosen to be able to perform an action, while all others stay where they are. The performance of the policies suggested in the work is similar in this case.

IV. Search-Based Re-Planning for PO-MAPF

The idea of the search-based policy is to re-plan the individual path at every timestep upon obtaining a new observation. While re-planning, we do not distinguish between the static obstacles and the other agents, and consider the cells occupied by them as the blocked ones. The portions of the map that the agent has not seen so far are considered to be fully traversable for the planner. The portions that have been observed are remembered and used for planning. The latter can be done using any search-based algorithm, such as A* [47] or D*Lite [48]. D*Lite is typically thought of as the most prominent way for solving planning problems in environments with partial observability. Instead of re-planning the path from scratch after applying each action, it extensively reuses the previously built search tree to speed up the search. However, our preliminary tests have shown that sequential A* works faster than D*Lite. One reason for that might be that often traversable passages in the vicinity of the agent are blocked by other agents, so there is no actual path to the goal. In such cases, running A* from scratch detects unsolvability considerably faster compared with D*Lite, which, in effect, plans backwards from the goal.

The vanilla re-planning policy described above is prone to two problems: oscillating behavior and what is referred to as the "freezing robot" problem. Oscillating behavior occurs when the agent bumps into another one and seeks to detour it. However, the latter detours in the same manner, so at the next timestep, they face each other again and attempt to detour again—and this pattern loops over. "Freezing robot" occurs when an agent is not able to build a valid plan due to some other agents temporarily blocking narrow passages.

To mitigate these issues, we augment the policy with two additional techniques. The first technique is detecting loops in the agent's plans. We check if the first action of the currently constructed plan leads to a location that was visited within l steps prior. If it does, we substitute the planned action with the wait action with the p_{wait} probability. We experimented with setting l and p_{wait} to different values and ended up with $l = 2$ and $p_{wait} = 0.5$, as this values leads to a better performance. The second technique tells an agent to perform a greedy action that brings it closer to the goal in case the path cannot be found. The ablation study of the introduced enhancements is given in Section VII-D. Additionally, there could be cases for which a plan could not be found (e.g., when the path to the goal is blocked by other agents). Thus, we introduce the parameter N_{max}, which is used to limit the allowed number of iterations of the path-planning algorithm.

The high-level pseudocode of the search-based PO-MAPF policy, RePlan, is shown in Algorithm 1. It starts with updating the map (of the static obstacles) using the current observation (Line 1). After that, it executes the A* search algorithm that looks for an (optimal) path from the agent's current location to the target one with respect to the map and the positions of the other agents that are visible currently. If the plan is found, its first action is selected for the execution (Lines 3–4). Otherwise, a greedy action, i.e. the one that transfers the agent closer to the goal, is picked (Lines 5–6). After the action is picked, we check whether its execution will lead to a loop (Line 7). If the loop is detected, i.e. the chosen action transfers the agent to the position that was visited in the last k steps, with p_{wait} probability, the wait action is returned (Line 8).

Algorithm 1: High-level pseudocode for the search-based policy incorporating loop detection and greedy actions: REPLAN

Input: o — observation; map — map; pos — current position of the agent; $goal$ — position of the goal; $hist$ — sequence of the already performed actions. At the beginning of the episode, $plan = map = \emptyset$.

Output: a — action to perform at the current timestep; updated $hist$ and map.

```
1  map := MapUpdate(map, o);
2  plan := A*(map, pos, goal);
3  if plan ≠ ∅ then
4  |  a := GetFirstAction(plan);
5  else
6  |  a := GetGreedyAction(pos, map);
7  if DetectLoop(a, plan) then
8  |  With p_wait probability return wait;
9  hist := hist + a;
10 return a, map, hist
```

V. Policy Optimization for PO-MAPF

The interaction of an agent with the environment in PO-MAPF can be generally described as a partially observable Markov Decision Process (POMDP), which is a tuple (S, O, A, P, r, γ). Here S is the set of environment states, $o \in O$ is a partial observation of the state, A is the set of agent's actions, $r(s, a) : S \times A \to \mathbb{R}$ is a reward function, $P : S \times A \to S$ is a state transition function, and γ is the discount factor, which determines the relative importance of future rewards compared with immediate rewards. In a POMDP setting, the agent does not know the true state of the environment; however, it can observe it. In our setting, we assume the observations to be deterministic, i.e. there is one-to-one correspondence between the state and the observation the agent gets in it (as specified in Section III). A policy for POMDP is a function that maps a *belief state* onto the distribution over the actions, where the former summarizes the previous experience of the agent in the environment with no precise knowledge of the true state [49]. The goal is to find a policy that maximizes the expected discounted return: $\mathcal{G} = \mathbb{E}_\pi[\sum_{i=0}^{T_{max}} \gamma^i r(s_i, a_i)|s_0, a_0]$.

In this work, we rely on the actor-critic methods to learn the policy, as they are known to be powerful and versatile RL tools. Specifically, we utilize a seminal actor-critic algorithm, Proximal Policy Optimization (PPO) [50], which has shown effectiveness in many challenging domains [51], [52], [53].

Originally, PPO was designed for the agent operating in the fully observable environment; thus, it assumes knowing the true state of the environment at each timestep s_t.

To adapt PPO for the POMDP setting, we approximate the state s_t by a hidden state of a recurrent neural network (RNN) $h_t \approx s_t$ that depends on the previous hidden state and the current observation $h_t = f(h_{t-1}, o_t)$. Further, we will assume that the policy additionally depends not only on the current observation but also on the hidden state at the previous step: $\pi(a_t|o_t, h_{t-1})$ or briefly $\pi(o_t, h_{t-1})$.

A. Evolving Policy Optimization with Grid Memory

Our original variant of PPO, EPOM (Evolving Policy Optimization with the Grid Memory), learns the policy in a decentralized fashion, i.e. it does not require any information-sharing among the agents and utilizes the following distinctive features. First, as noted above, we employ RNN as the state approximator. Second, we explicitly memorize the static portion of the grid environment, i.e. the obstacles, and augment each observation with an enlarged patch of the memorized grid. Third, we rely on the specifically-designed population-based training (PBT) to encourage learning of the cooperative behaviors. The network architecture of the EPOM approach is presented in Fig. 3.

Note that although we leverage PPO in this work, the suggested enhancements, like the grid memory, can be used for any actor-critic RL method.

a) Grid Memory Module: The previously introduced search-based policy uses observations to construct and memorize the map of the static obstacles, which is indeed beneficial for solving PO-MAPF. However, incorporating such map memorization directly into the learnable policy is not straightforward, as the input size needs to be fixed while the environments used for training and evaluation may have different sizes.

To address this issue, we propose enhancing PPO with an additional Grid Memory module, inspired by REPLAN. This module explicitly stores and updates the map of the environment. At each step, the initial input of the obstacles matrix is extended with extra obstacles that are memorized during execution. This extended observation forms a patch (e.g., 15×15 in our experiments) which is used as input to the policy encoder. The scheme of the proposed approach is presented in Fig. 2. Changing the size of the obstacles matrix requires the corresponding adjustment of the agents matrix. In this case, additional cells are filled with zeros. The target or its projection is added to the extended field-of-view.

As demonstrated in SectionVII-E, grid memory significantly stabilizes the learning process and improves performance. Furthermore, it enables an agent trained with one observation radius to be deployed in a setting with a different observation radius without requiring retraining (refer to Section VII-E for experimental details).

b) Population-Based Training: Population-based training (PBT) is a technique for automated hyper-parameter tuning at the learning stage[54]. It has been successfully used for RL and resulted in more robust policies [12]. In this work, we employ PBT to adjust such parameters as the learning rate, batch size, and entropy coefficient. We use the success rate of the PO-MAPF instances as the PBT target objective, as

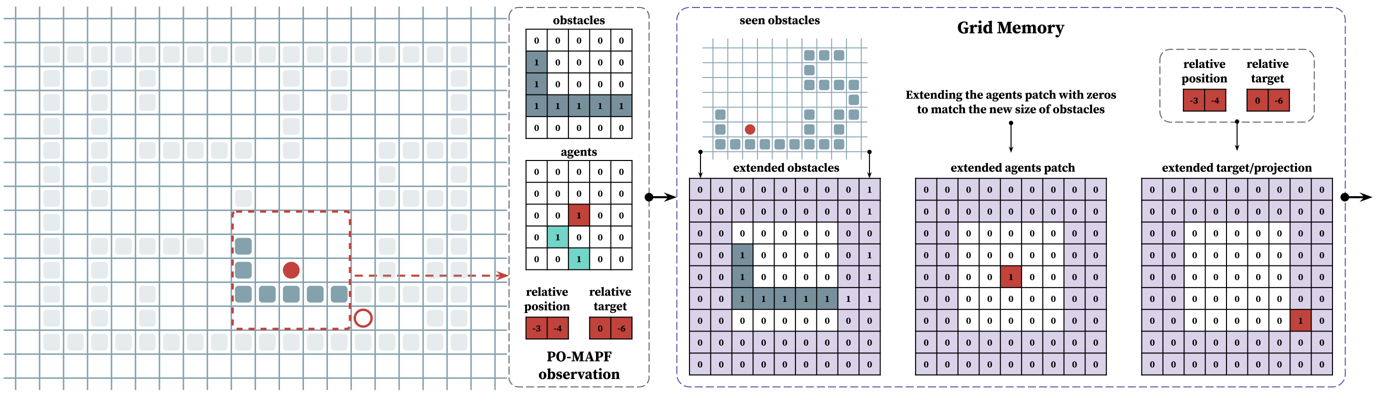

Fig. 2. The Grid Memory observation pre-processing of EPOM approach, which facilitates the storage and updating of an environment map. This module extends the initial input of the obstacles matrix with additional obstacles observed during execution, creating a patch-like extended observation.

opposed to the individual agents' rewards, to encourage the populations of the cooperative agents.

c) Reward: We do not use any complex reward-shaping and let the agent receive a non-zero reward only in one case: when it reaches the goal. At every step, it also receives a small negative reward of 0.0001. An additional negative reward of 0.0002 is added if the agent picks an action that leads to a collision. Indeed, this reward is based purely on an agent's observation, not the true state of the environment.

d) Learning: We use the PO-MAPF observation (as specified in Section III) for learning. Matrices that encode the obstacles' and the other agents' positions are passed through Grid Memory, which extends the observation as described above. Additionally, another matrix, which encodes the goal projection (similarly to PRIMAL [13]), is formed and passed to the encoder. This is done to enable goal conditioning inside the encoder. Furthermore, we concatenate the output of the encoder with the normalized coordinates of the agent's current position and the target position. The resultant embedding is passed to the actor-critic heads of EPOM. We use a ResNet-based encoder and a GRU for the actor-critic. The scheme of the neural network is presented in Fig. 3. More details on the training hyperparameters are provided in VIII.

B. Dataset

Aiming at obtaining a versatile policy capable of solving a large variety of PO-MAPF problems, we create a heterogenous dataset for learning (and further evaluating) EPOM. In total, it consists of 239 maps of size 64×64 that model the environments with different topologies. These environments include the re-scaled multi-player game maps, `wc3` (Warcraft III), `sc1` (Starcraft I), taken from the MovingAI benchmark [1]; maps of the real cities, `street`, taken from the same benchmark; synthetically generated (by us) maps with random number of blocked cells, `random`; and maze maps, `maze`, which are procedurally generated (by us) using the code publicly available from the PRIMAL2 authors [16]. Examples of the maps are provided in Fig. 4.

a) Multiplayer Maps: These are the maps used in video games Starcraft I (`sc1`) and Warcraft III (`wc3`). `sc1` collection contains 74 maps, while `wc3` contains 35. The distinctive

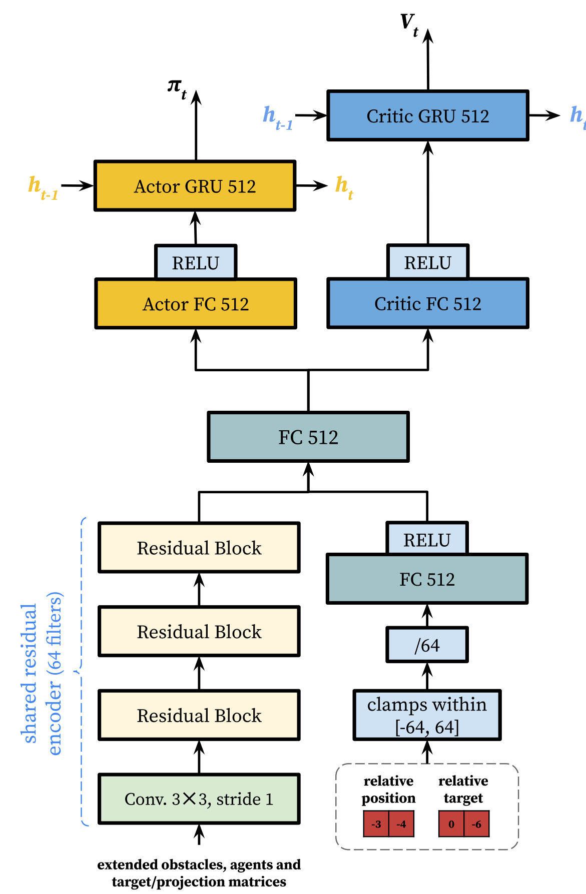

Fig. 3. The neural network architecture for the EPOM algorithm incorporates a ResNet-based encoder and GRU heads for the actor-critic. The network takes extended PO-MAPF observations from Grid Memory, which include obstacle and agent positions, as well as the target or its projection. The encoder generates an embedding, which is then concatenated with the normalized coordinates of the agent's current and target positions. To normalize the coordinates, the values are clamped within the range [-64, 64] and divided by 64. Finally, this embedding is fed to the actor-critic heads.

feature of these maps is their region-based structure. By the latter, we mean that, typically, on these maps, several areas of the free space are present that are connected by the (sometimes narrow) passages. This enforces the agents to resolve conflicts that are likely to occur along their paths. Another feature of these maps is the presence of (sometimes large) obstacles located in the middle of the map. This means that the paths of the agents are likely to contain detours and not just resemble straight-line segments to their targets.

b) Synthesized Maps: We generate two types of these maps: maze-like environments (50 maps) and random ones (50 maps).

Maze-like maps are generated using the tool created by the authors of the seminal learning-based PRIMAL2 MAPF solver. The main parameters that govern the generation are the corridor length (we vary this parameter from 2 to 10) and the obstacles' density (this parameter is varied from 25% to 75% with a 5% increment). To generate each of the 50 maps of our collection, we iteratively choose these two parameters randomly and invoke the generator. The resultant maps contain large number of corridors that are likely to trap the agents that enter these corridors from the opposite directions.

Maps with the randomly blocked cells are the ones that have no regular or predictable structure. We vary the obstacle density from 15% to 35%. Our preliminary tests showed that the 35% density is the most challenging. Lesser density results into more open areas where agents can easily surpass each other, while the higher density often results in creating several isolated regions on the map.

c) Street Maps: We use 30 maps generated from the real data taken from OpenStreetMap. Maps of this type in most of the cases contain large obstacles and wide open areas, though there might be some areas with small buildings and narrow passages between them.

As said before, in total our dataset is comprised of 239 maps. We split it to the training-test parts in proportion 80/20, i.e. 80% of the maps are used for training, while the other 20% of the maps are used for testing. In such a way we are able to evaluate how well our learnable policy is able to generalize to the unseen maps (as no map used for testing was seen while training).

When learning, we randomly sample the map from the training part of the dataset and populate it with 64 agents whose start and target locations are picked randomly. Noteworthy, for testing purposes we use different number of agents, up to 500. This, again help us to assess how well the policy is able to generalize to higher number of agents.

In total, EPOM has been trained for 1 billion steps on a single TITAN RTX GPU in ≈ 8 hours.

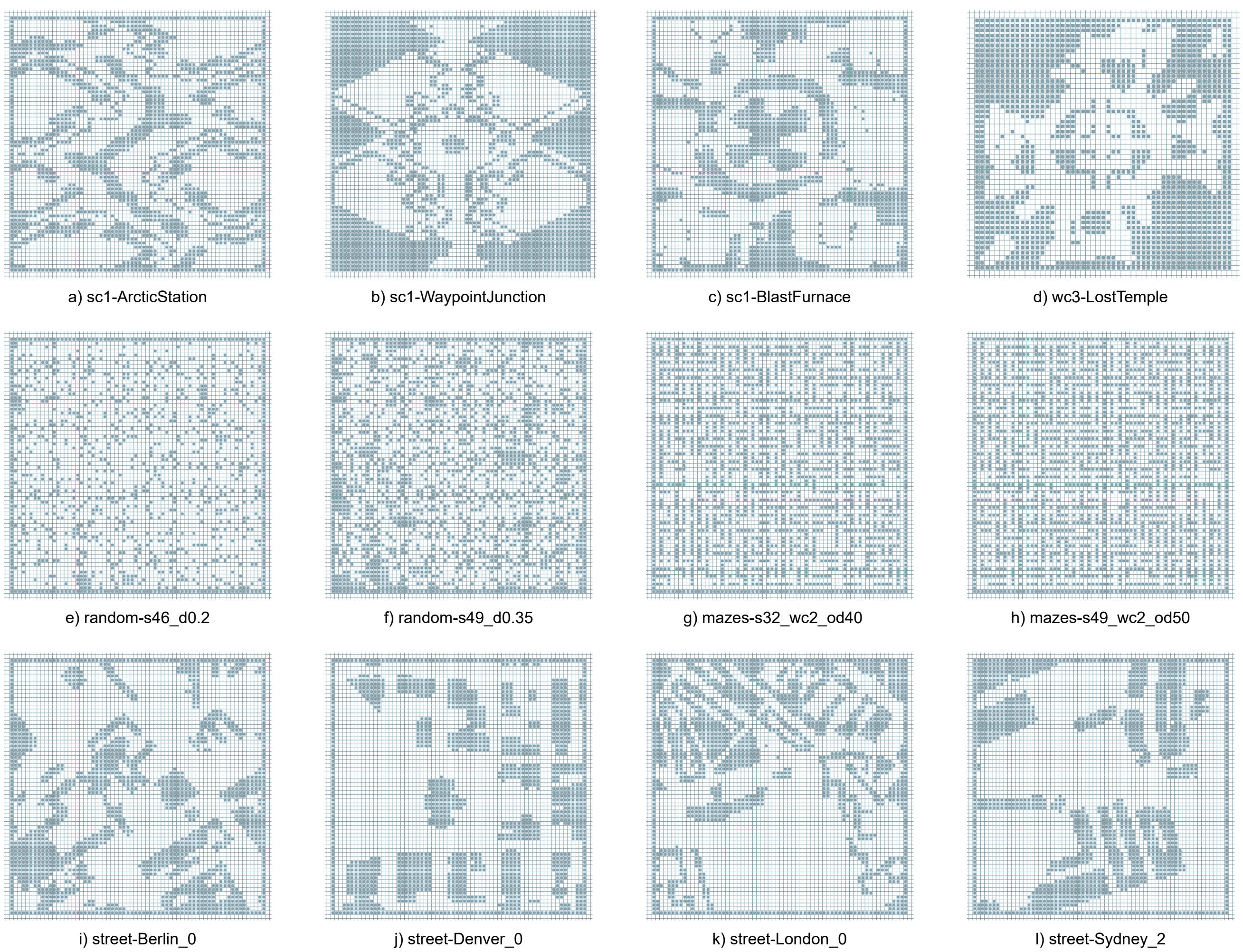

Fig. 4. Examples of the maps from a diverse dataset of 239 maps of size 64×64 representing various environments for training and evaluating PO-MAPF solvers The dataset includes re-scaled multiplayer game maps (`wc3` and `sc1`) from the MovingAI benchmark, real city maps (`street`) from the same benchmark, synthetically generated maps (`random`) with random blocked cells, and procedurally generated maze maps (`maze`).

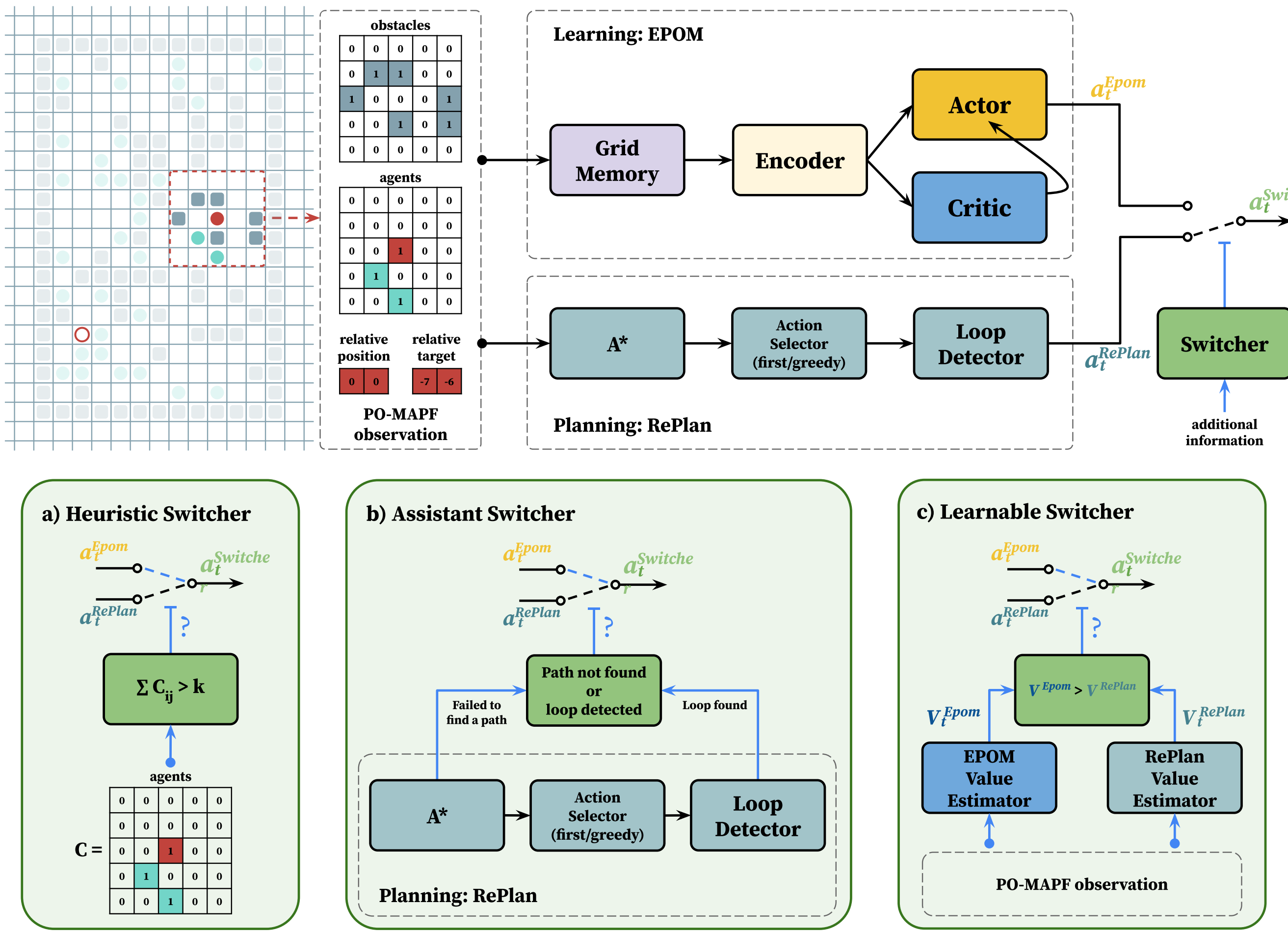

Fig. 5. The general pipeline of the switching approach is as follows. The observation space of the environment consists of two matrices that encode obstacles and agents, as well as the agent's relative position and target. This information is fed into the learning component (i.e. EPOM) and the planning component (i.e. RePlan). Then, the switcher decides which action to take based on the additional information. Subfigures (a), (b), and (c) show different implementations of the switcher. The Heuristic Switcher selects EPOM when the agent count threshold is reached in the observation. The Assistant Switcher transfers control to EPOM when it fails to find a path or detects a loop. Finally, the Learnable Switcher trains additional value estimators to evaluate each policy for the current observation and greedily selects the best one.

VI. Switching Between the Learnable and Planning-Based Policies

The introduced policies designed to solve PO-MAPF, RePlan and EPOM presumably have both advantages and drawbacks.

EPOM requires a prepared set of the environments on which the policy will be trained. An incorrectly compiled set can lead to a weak generalization. RePlan's performance, on the other hand, largely depends on the set of the hand-crafted heuristics.

To this end, we suggest several ways for integrating RePlan and EPOM that follow the general pipeline depicted in Fig. 5.

This pipeline includes a switcher that, having access to the outputs of both policies, as well as to the current observation, makes a final decision as to which action should be performed. Note that both policies in the switcher are executed in parallel, i.e. at each timestep, they both receive the observation, update the internal variables, and output an action. We consider the following switchers in our work.

a) Heuristic-Based Switcher: This switcher (HSwitcher) relies on the assumption that one may identify a set of key features that impact the effectiveness of each policy, and design a heuristic based on these features. Candidate features are the density of obstacles, the number of observed agents, the distance to the goal, etc. The algorithm of the Heuristic Switcher consists of identifying significant features from observation and applying a set of fixed rules based on preliminary experiments on the effectiveness of two policies. In our work, we leverage an empirical observation that sometimes in dense environments, RePlan performs worse than EPOM and vice versa. Thus, we suggest switching from RePlan to EPOM when the number of agents in the agent's field-of-view is greater than a given threshold k (in our experiments, we use $k = 6$).

b) Assistant Switcher: This switcher (ASwitcher) is based on the assumption that RePlan, in general, copes well with the problem at hand and should only be aided when it is unable to construct a plan or when it detects the loop. In these cases, we switch to the EPOM action. Note that, contrary to HSwitcher, this switching technique does not rely on ad-hoc

TABLE I
SUCCESS RATES OF THE EVALUATED PO-MAPF SOLVERS W.R.T. DIFFERENT MAP TYPES. IN ADDITION TO THE SUCCESS RATES, WE INCLUDE THE STANDARD DEVIATION COMPUTED FOR EACH MAP AND NUMBER OF AGENTS, WHICH IS THEN AVERAGED ACROSS ALL INSTANCES.

agent	`mazes`	`random`	`sc1`	`street`	`wc3`	average success rate
REPLAN	92.41% ±16.07	66.72% ±20.84	53.29% ±18.06	87.1% ±19.88	55.51% ±28.71	69.72% ±19.66
EPOM	80.38% ±31.24	52.81% ±22.66	17.26% ±15.25	52.94% ±40.03	35.31% ±17.28	45.95% ±23.98
Assistant Switcher	97.36% ±14.73	**77.84%** ±13.13	**67.1%** ±17.06	**95.1%** ±12.27	82.97% ±14.53	**81.71%** ±14.75
Heuristic Switcher	97.18% ±5.53	73.98% ±7.16	43.73% ±13.75	78.19% ±20.54	43.73% ±12.96	66.92% ±11.28
Learnable Switcher	**99.39%** ±3.11	75.44% ±5.87	55.16% ±14.83	94.17% ±15.61	**83.78%** ±8.71	77.88 % ±9.66

heuristics and is deterministic.

c) Learnable Switcher: This switcher (LSwitcher) is implemented using a learnable greedy switching policy π^{sw}. Recall that our agent has access to two policies: π^{RePlan} and π^{EPOM}; thus, it can conduct a classical policy evaluation on a certain set of environments. If we introduce two new approximators with two-parameter sets θ^{RePlan} and θ^{EPOM}, the agent can adjust these parameters to evaluate values V^{RePlan} and V^{EPOM}—the expected values of the states conditioned to the respective policy are used till the end of the episode. In this case, the greedy policy for switching at the state o_t to the next $\mathcal{N}$ steps will look as follows:

$$\pi^{sw}(o_t, h_{t-1}) = \begin{cases} \pi^{RePlan}, \text{if } V^{RePlan}(o_t) > V^{EPOM}(o_t), \\ \pi^{EPOM}(o_t, h_{t-1}), \text{otherwise.} \end{cases}$$

We train LSwitcher using the training part of our dataset in the same way as EPOM. The only difference is that while EPOM is trained on 64 agents, LSwitcher is trained on the varying number of agents (from 50 to 300) for the latter to make correct value predictions for different numbers of agents. We use non-recurrent architecture for LSwitcher with the same encoder as in EPOM extended with two MLP 512 layers. For each training epoch, we sample 10^6 pairs $\langle o_i, R_i \rangle$, where R_i is a return. To decorrelate samples, we use only 20% of data from each episode. The final values of the MSE loss are 0.016 and 0.013 for REPLAN and EPOM, respectively (0.035 and 0.036 for the validation phase).

Finally, we only allow switching in LSwitcher when $\mathcal{N}$ timesteps have been completed by the previously active policy. We set the value of $\mathcal{N}$ to 50 based on the results of the preliminary experiments. Setting it lower has resulted in worse performance, while setting it higher has not led to an improvement.

VII. EXPERIMENTAL EVALUATION

A. Evaluation of the Suggested PO-MAPF Solvers

We evaluate all the suggested PO-MAPF solvers on the test split of our dataset (20% of 239 maps that were not used while training). For each map, we randomly generate PO-MAPF instances that contain between 50 and 300 agents with an increment of 50 agents. One hundred different instances per each map for each number of agents is generated. The time limit (maximal episode length) is set to 512 steps.

The main evaluation metric is the success rate: the fraction of the test instances for which all the agents reach their goals within the time limit. We also track the independent success rate and the averaged episode length. The former is the ratio of agents that successfully reach their goals in a single test run, while the latter indicates how many steps each of the agents performs before reaching the goal (on average). Note that in case the agent has not reached its target location, its episode length is equal to the limit, i.e. to 512.

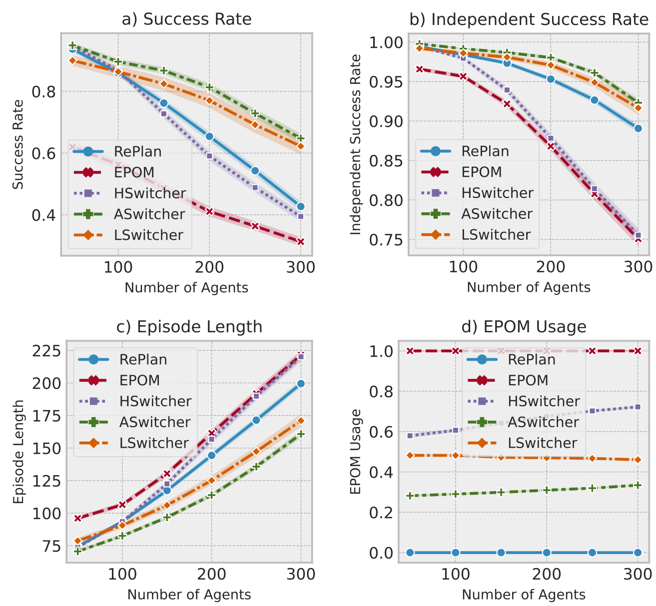

Fig. 6. (a) Success rate, (b) independent success rate, (c) episode length, and (d) EPOM usage per each number of agents averaged over all the evaluated instances. The shaded area reports confidence intervals 95%.

Success rates of all the PO-MAPF policies w.r.t. different map types are presented in Table I. Clearly, REPLAN shows better success rates compared with EPOM. Having visualized and analyzed various runs of these two policies, we make the following important observation. While the overall performance of EPOM may seem underwhelming, it actually performs better than REPLAN when it comes to the coordination of the agents in the confined areas. This explains why the hybrid polices, i.e. ASwitcher and LSwitcher, demonstrate a notable boost in performance. On the one hand, they leverage the capability of REPLAN to rapidly progress toward the target; on the other, they utilize EPOM for conflict resolution in congested areas.

Another view of the results is presented in Fig. 6. Here the metrics are shown w.r.t. the number of agents (averaged across all the test instances). In general, the observed trends support the claim that ASwitcher and LSwitcher outperform the other policies. In addition to the main metrics, Fig. 7 (d) displays the percentage of actions performed using the EPOM algorithm. LSwitcher utilizes EPOM approximately 50% of the time, while ASwitcher employs it less frequently. However, this percentage tends to increase as the number of agents increases. This occurs because in such cases, RePlan often fails to find a path or detects a loop.

B. Comparison with Other Solvers

Next, we compare our switching approaches with the other methods described in the literature.

The first competitor is a centralized MAPF-algorithm: Cooperative A* [29]. It relies on prioritized planning to eliminate the conflicts leveraging access to the full state of the environment. Thus, it is technically not a PO-MAPF solver. The second approach is the state-of-the-art RL-based algorithm that solves PO-MAPF problems: PRIMAL2 [13]. The core difference between PRIMAL2 and switchers is that the former assumes that each agent knows the goals of the other agents that are within its field-of-view, while our solvers rely on more restrictive assumptions (no information about the other agents, except their locations, is accessible). We use the code and the trained model provided by the authors of the approach[2].

The last competitor is PICO [17] – another recently presented RL-based approach capable to solve PO-MAPF problems. Unlike PRIMAL2 and our methods, PICO allows agents to communicate with each other. Moreover, originally PICO was tailored to solve PO-MAPF problems with agents that do not disappear when reaching the goals. Thus, for a fair comparison the code of PICO, taken from the original repository[3], was modified such that the agents disappear when reach their goal locations. The authors of the algorithm haven't provided the trained model, so we trained the model ourselves in the same way as it was described in the paper (on 20×20 grids with randomly placed obstacles and 8 agents only). It is also worth to note that the implementations of both PRIMAL2 and PICO approaches assume that agents perform their actions sequentially within one timestep. As a result, in cases when two or more agents try to occupy the same grid cell simultaneously, the agent with higher priority occupies it. We have modified the code of all other evaluated approaches to follow the same logic.

For the first experiment, we use the maps and the instances taken from the PICO repository. The maps are represented by 20×20 grids with randomly placed blocked cells with the density of up to 30%. The instances consist of randomly placed start and goal locations for $8, 16, 32, 64$ agents. The episode length is set to 256, while the size of the field-of-view is set to 11×11.

The results of this experiment are presented in Fig. 7. As can be seen, all the approaches are able to solve all the instances when the grid is completely empty. However, with the rising amount of blocked cells, the success rate of PICO and PRIMAL2 decreases, especially on the maps with 30% density of obstacles, where they are able to solve only half of the instances with only eight agents. By contrast, all the switchers solve more than 80% of instances with 64 agents on the maps with 30% blocked cells. As expected, the absolute winner is Cooperative A*, which is actually relying of the full observation to solve the problem.

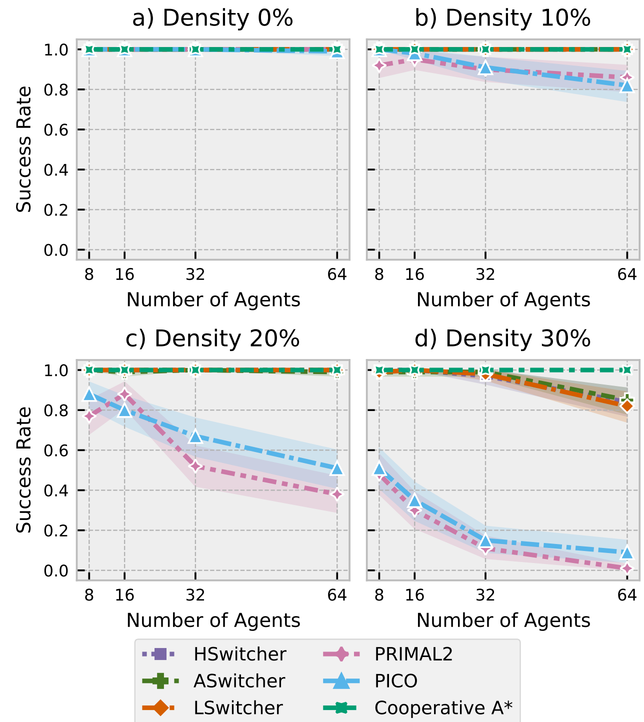

Fig. 7. Comparison of the suggested approaches with PICO and PRIMAL2 on 20×20 grids taken from the PICO repository. The main difference between these maps is the obstacle density considered: 0%, 10%, 20%, and 30%. Maps with 0% density are quite simple, and all algorithms perform well. For maps with higher density, Switchers show the best results. The shaded area represents 95% confidence intervals.

While PRIMAL2 shows relatively good results on the maps from PICO's dataset, it wasn't trained to solve these instances. Instead, it was trained and focused to solve instances on maze-like maps. Thus, we have made an additional experiment where PRIMAL2 and switchers are additionally compared on maze-like maps, generated by the tool taken from PRIMAL2 repository. For this purpose we have reused the test part of the `mazes` dataset. In contrast to previous experiments, the number of agents in the most challenging instances for this experiment is increased to 500. The episode length is set to 512, while the size of field-of-view is left the same – 11×11.

The results of this experiment are depicted in Fig. 8, where both cooperative and independent success rates are shown. The evident outsider in this experiment – HSwitcher, that has issues while solving instances with 300+ agents. At the same time all the rest approaches can successfully solve almost all the instances with 300 agents. However, when the number of agents exceeds 400, only Cooperative A* and ASwitcher are able to solve more than 95% of the instances. On the most challenging instances, with 500 agents, the cooperative success rates of PRIMAL2 and LSwitcher drop down to about 50% while ASwitcher still able to solve more than 80% of the instances.

Overall the conducted experiments have shown that the

[2]https://github.com/marmotlab/PRIMAL2
[3]https://github.com/mail-ecnu/PICO

suggested approaches, especially ASwitcher, can compete with existing state-of-the-art RL-based approaches and outperform them even in scenarios for which the latter were specifically trained.

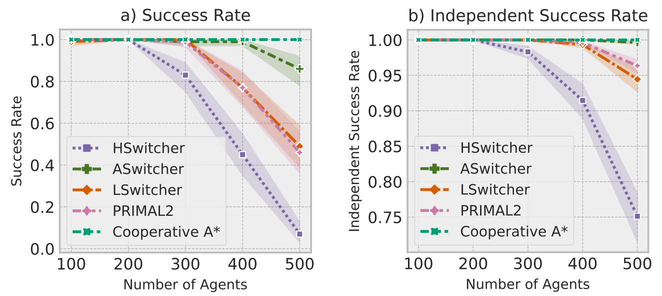

Fig. 8. Comparison of switchers with other approaches on the `mazes` maps. Cooperative A* has access to the full state of the environment, so it shows the best results, solving all the presented instances. The best algorithm among those working in the PO-MAPF setting is ASwitcher. The shaded area represents the 95% confidence intervals.

C. Scalability on Lifelong PO-MAPF

In these experiments, a more practical setting of the automated warehouse is modeled. In this setting, an agent does not disappear upon reaching its goal, but rather it is immediately assigned to another one. We use the warehouse map from the MAPF MovingAI Benchmark [1] for these experiments and limit the episode length to 1000. In contrast to previous experiments, the size of the map is much larger than 64×64. It is now 159×61, allowing us to test the scalability of the proposed approach with an increased number of agents in the environment. We have tested up to 600 agents. The considered metric is the throughput, i.e. the number of accomplished goals (deliveries) per one timestep.

Fig. 9. The plot (a) demonstrates that in Lifelong PO-MAPF experiments on the warehouse map, EPOM performs close to the best-performing switcher, ASwitcher, based on average throughput. In plot (b), it is observed that the steps per second remain constant even with an increasing number of agents. Additionally, ASwitcher algorithm's speed improves with more agents as EPOM is utilized more frequently.

The results are presented in Fig. 9 a). Notably, the performance of EPOM in such setups is impressive. It even outperforms ASwitcher for certain numbers of agents. On the other hand, RePlan fares poorly. This confirms our hypothesis that the former has a better collision-resolution ability, which is very important when agents are constantly moving in the environment.

Fig. 9 (b) shows the average number of steps per second for each agent in the environment. As can be seen, the speed of the EPOM algorithm remains constant regardless of the number of agents. The other algorithms also do not degrade with an increase in the number of agents, except for the ASwitcher algorithm, which becomes faster with more agents. We attribute this to the fact that with a large number of agents, RePlan quickly either terminates without finding a path or detects a loop and then transfers control to EPOM, which operates faster.

D. RePlan Enhancements Ablation

To evaluate how the suggested enhancements, i.e. loop detection and greedy actions, improve the vanilla policy, we conduct an empirical evaluation involving 64×64 grid with 30% of randomly blocked cells and 50–300 agents whose starts and goals are chosen randomly. The time limit is set to 512 timesteps. Fig. 10 depicts the independent success rate: the ratio of the agents that reached their goals within the time limit.

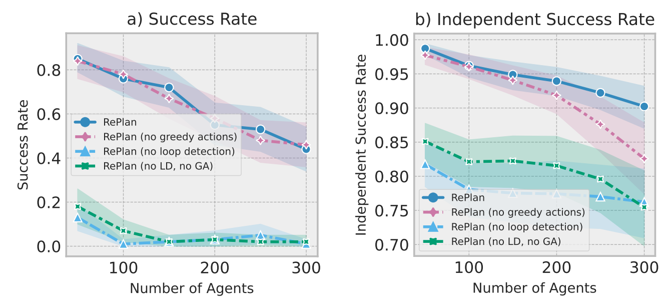

Fig. 10. Performance of the different variants of search-based PO-MAPF policy. Disabling loop detection (LD) in RePlan significantly worsens the results. RePlan with both greedy actions (GA) enabled and disabled exhibit similar success rates, but the variant with greedy actions demonstrates better results in terms of independent success rate. The shaded area represents the 95% confidence intervals.

As can be seen, the performance of the vanilla policy is poor: even the instances that only contain 50 agents cannot be solved efficiently. Adding greedy actions on its own does not improve the performance. The enhancement that dramatically improves the policy, though, is the loop detection; adding greedy actions on top of it further improves the performance.

E. Grid Memory Ablation

To evaluate how the suggested Grid Memory mechanism affects the learning process, we run a specifically designed experiment involving one agent (using the maps from our training set). We vary the observation radius of the agent in the range $1, 2, 3, 4, 5$ and train the agent either with or without Grid Memory (whose size was 15×15). The aggregated learning curves for 30M steps (averaged across all the observation radii) are shown in Fig. 11. As can be seen, Grid Memory indeed stabilizes the learning process (the dispersion is lower) and leads to a better result (the success rate is higher, and the episode length is lower).

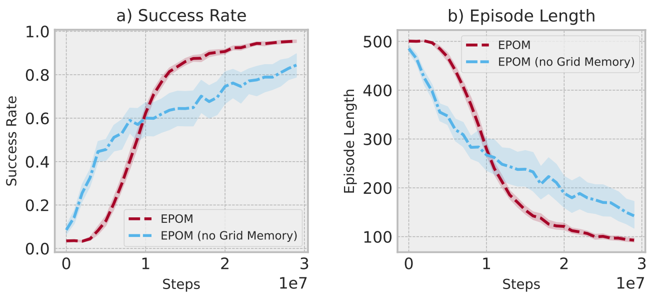

Fig. 11. The effect of the suggested Grid Memory mechanism for single-agent learning in PO-MAPF scenarios. The shaded area reports confidence intervals 95%. The use of Grid Memory allows the agent to achieve higher scores in terms of success rate (a) and shorter episode length (b).

F. EPOM Ablations

In this experiment, we tested how the use of an RNN and changing the observation radius R in the environment affects the quality of the solutions produced by the EPOM algorithm. We compared a regular EPOM ($R = 5$), EPOM with a smaller field of view ($R = 3$), and EPOM with an even smaller field of view ($R = 1$), as well as a regular EPOM that resets h_t at each step, thereby not providing the agent with all the previously observed information. The results are shown in Fig. 12. For this experiment, we used the life-long setting and the warehouse map. The results are averaged over six seeds.

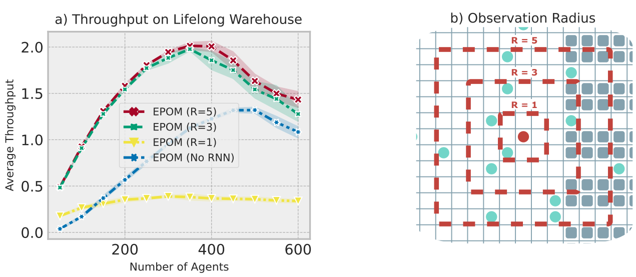

Fig. 12. (a) The performance of the EPOM algorithm when RNN is disabled, as well as when the observation radius changes in the environment. The shaded area represents the 95% confidence intervals. (b) An example of different observation radii on the Lifelong Warehouse map.

As can be observed from the figure, the algorithm that does not utilize an RNN consistently yields significantly worse results. This underscores the importance of incorporating the RNN component into the algorithm. It can also be seen that EPOM with a viewing radius of $R = 3$ shows close results to the regular EPOM. This demonstrates the ability of Grid Memory to work with different observations without retraining the neural network. However, significantly worse results are shown for $R = 1$ due to the fact that the agent cannot foresee other agents (outside its field of view) that may try to move to an adjacent cell, causing a conflict.

VIII. Hyperparameters

Table II reports the hyperparameters used in the experiments. Due to the significant training time required for the EPOM algorithm, we have not performed an exhaustive hyperparameter search. Instead, we have employed parameters that have exhibited good performance in reinforcement learning problems. We have mainly relied on the default settings of the Sample Factory library[4], along with configurations that have demonstrated success in single-agent pathfinding problems within stochastic environments[5]. To effectively train the algorithm for specific tasks, it is recommended to consider key parameters, namely batch size and learning rate. Selecting suitable values for these parameters has yielded noteworthy enhancements in comparison to alternative choices.

For LSwitcher, we have tuned the batch size parameter. The parameter $\mathcal{N}$ has been separately determined using grid search over values ranging from 1 to 100 with increments of 25.

For HSwitcher and RePlan, we have conducted a hyperparameter search using the maps employed in training EPOM, and the table reports the best parameters found through this search. The value of $N_{max} = 10000$ has been chosen empirically as the smallest value that did not worsen the results.

TABLE II
HYPERPARAMETERS FOR EPOM, LSWITCHER, HSWITCHER AND REPLAN APPROACHES

EPOM Hyperparameters	
grid memory radius	7
learning rate	1e-4
γ	0.99
adam ϵ	1e-6
adam β_1	0.9
adam β_2	0.999
rollout	32
recurrence rollaut	32
clip ratio	0.1
clip value	1.0
batch size	4096
num batches per iteration	1
num epochs	1
max grad norm	4.0
entropy loss coeff	0.01
value loss coeff	0.5
max policy lag	100
PBT period env steps	5e6
PBT start mutation	2e7
PBT replace fraction	0.3
PBT mutation rate	0.15
PBT replace gap	0.1

LSwitcher Hyperparameters	
learning rate	$1e-4$
γ	0.99
adam ϵ	$1e-6$
adam β_1	0.9
adam β_2	0.999
num epochs	7
batch size	512
shuffle	True
$\mathcal{N}$	50

HSwitcher Hyperparameters	
k	6

RePlan Hyperparameters	
l loop detection	2
p_{wait}	0.5
N_{max}	10000

IX. Limitations

Similarly to the vast majority of the MAPF-related papers, in this work, we intrinsically assume that the agents have perfect localization and mapping capabilities, as we mainly concentrate on the planning and decision-making aspects of the problem at hand. Moreover, we assume that the obstacles are static part of the environment. It would be interesting to study problem variants when the obstacles can rather appear/disappear (closing/opening doors) or move (someone has moved a chair) in a stochastic fashion. Notably, we have recently presented a preliminary study for a single-agent pathfinding in a presence of stochastic obstacles in [55].

We assume that the agents cannot communicate and share MAPF-related data, e.g. their goals, intended paths, further

[4]https://github.com/alex-petrenko/sample-factory
[5]https://github.com/Tviskaron/pathfinding-in-stochastic-envs

actions etc. The reason we have decided to adapt these limiting assumptions is that we wanted to obtain a solution to the most-restrictive problem setting on the presumption that this can serve as the lower bound, and adding more MAPF-related data to a decision-making policy is likely to only increase the performance. Indeed, we believe that information exchange could boost the performance of the proposed approach.

The last but not least, similarly to the other prominent learnable methods that are tailored to (PO)-MAPF, e.g. PRIMAL [13], PRIMAL2 [16], DHC [56], PICO [17], etc., we do not provide theoretical guarantees that the agents will reach their destinations. On the other hand, numerous experiments (in this paper and in the ones referenced above) confirm that practically-wise learnable methods are powerful and scalable tools to solve non-trivial MAPF problems.

X. Conclusion

In this work, we have investigated a challenging variant of the multi-agent pathfinding problem, i.e. the one with partial observability and no inter-agent communication. We have introduced two policies to solve such kind of problems: the planning-based one and the learning-based one. The latter is learned in a decentralized fashion without any external guidance and sophisticated reward-shaping. We have also proposed a hybrid policy that combines the search-based and the learning-based ones and introduced three different ways of such combination, which are all based on the parallel running of the policies.

The conducted experimental evaluation on a wide range of different setups provides a clear evidence of the following. First, the suggested idea of combining the policies is worthwhile, as two of the suggested switching policies notably outperform the solo ones. Second, this idea leads to outperforming the state-of-the-art competitors that also utilize decentralized learning.

Possible directions for future research include further enhancing the switching techniques, especially the learnable ones and considering even more challenging PO-MAPF settings (e.g. stay-at-target behavior).

List of Notation

G	Undirected graph used in MAPF formulation
$T_{\max}$	Time limit or episode length
S	State space, the set of all possible states in the environment
O	Observation function, returns the observation o_t given the current state
P	State transition function, which maps a state-action pair to the next state in the system
R	Observation radius, size of the observation grid: $(2 \cdot R+1) \times (2 \cdot R+1)$
A	Action space, set of all possible actions
$r(s,a)$	Reward function, returns a real-valued reward given the current state and action
γ	Discount factor, value between 0 and 1, determines the importance of future rewards
π	Decision-making policy, maps states to actions
$\mathcal{G}$	Expected discounted return, expected sum of discounted rewards over time
θ	Set of parameters of a neural network, defines the network's behavior
h_t	Hidden state of the neural network, calculated based on previous hidden state and current observation
π^{RePlan}	RePlan policy, decision-making policy based on a search-based re-planning approach
π^{EPOM}	EPOM policy, reinforcement learning policy based on the Evolving Policy Optimization with Memory algorithm
π^{sw}	Switching policy, decides between the RePlan policy and the EPOM policy
k	Threshold that determines the policy to be used in heuristic switcher based on the number of agents observed
V^{EPOM}	Expected value of states conditioned on the EPOM policy until the end of the episode
V^{RePlan}	Expected value of states conditioned on the RePlan policy
l	A hyperparameter used to detect loops in an agent's plans by checking if the first action of the current plan leads to a previously visited location within l steps.
N_{max}	The parameter is used to limit the allowed number of iterations of the pathplanning algorithm (the number of expansions). It is necessary for cases when the path to the goal is blocked by other agents and cannot be found
$\mathcal{N}$	Number of steps to transfer control between the π^{RePlan} and π^{EPOM} policies in the learnable switcher

References

[1] Roni Stern et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the 12th Annual Symposium on Combinatorial Search (SoCS 2019)*, pages 151–158, 2019.

[2] D. Kornhauser et al. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS) 1984*, pages 241–250, 1984.

[3] Jingjin Yu et al. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32(5):1163–1177, 2016.

[4] Bernhard Nebel. On the computational complexity of multi-agent pathfinding on directed graphs. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS 2020)*, pages 212–216, 2020.

[5] Jiaoyang Li et al. Multi-agent path finding for large agents. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, pages 7627–7634, 2019.

[6] Thayne Walker. Extended increasing cost tree search for non-unit cost domains. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 534–540, 2018.

[7] Anton Andreychuk et al. Multi-agent pathfinding with continuous time. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, pages 39–45, 2019.

[8] Jiri Svancara et al. Online multi-agent pathfinding. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*, pages 7732–7739, 2019.

[9] Dor Atzmon et al. Robust multi-agent path finding and executing. *Journal of Artificial Intelligence Research*, 67:549–579, 2020.

[10] Hang Ma et al. Lifelong multi-agent path finding for online pickup and delivery tasks. In *AAMAS 2017*, pages 837–845, 2017.

[11] Wolfgang Honig et al. Conflict-based search with optimal task assignment. In *AAMAS 2018*, pages 757–765, 2018.

[12] Aleksei Petrenko et al. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *ICML*, 2020.

[13] Guillaume Sartoretti et al. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019.

[14] Benjamin Riviere et al. 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, 2020.

[15] Zuxin Liu et al. Mapper: Multi-agent path planning with evolutionary reinforcement learning in mixed dynamic environments. In *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2020)*, pages 11748–11754, 2020.

[16] Mehul Damani et al. Primal _2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong. *IEEE Robotics and Automation Letters*, 6(2):2666–2673, 2021.

[17] Wenhao Li, Hongjun Chen, Bo Jin, Wenzhe Tan, Hongyuan Zha, and Xiangfeng Wang. Multi-agent path finding with prioritized communication learning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 10695–10701. IEEE, 2022.

[18] Yongliang Yang and Cheng-Zhong Xu. Adaptive fuzzy leader–follower synchronization of constrained heterogeneous multiagent systems. *IEEE Transactions on Fuzzy Systems*, 30(1):205–219, 2020.

[19] Yuanzheng Li, Shangyang He, Yang Li, Yang Shi, and Zhigang Zeng. Federated multiagent deep reinforcement learning approach via physics-informed reward for multimicrogrid energy management. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[20] Jianye Hao, Tianpei Yang, Hongyao Tang, Chenjia Bai, Jinyi Liu, Zhaopeng Meng, Peng Liu, and Zhen Wang. Exploration in deep reinforcement learning: From single-agent to multiagent domain. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[21] T. S. Standley. Finding optimal solutions to cooperative pathfinding problems. In *Proceedings of The 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*, pages 173–178, 2010.

[22] Michal Čáp, Peter Novák, JiYí Vokrínek, and Michal Pěchouček. Multi-agent rrt: sampling-based cooperative pathfinding. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1263–1264, 2013.

[23] Jinmingwu Jiang and Kaigui Wu. Cooperative pathfinding based on memory-efficient multi-agent rrt. *IEEE Access*, 8:168743–168750, 2020.

[24] Glenn Wagner and Howie Choset. M*: A complete multirobot path planning algorithm with performance bounds. In *Proceedings of The 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2011)*, pages 3260–3267, 2011.

[25] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Conflict-based search for optimal multiagent path finding. *Artificial Intelligence Journal*, 218:40–66, 2015.

[26] M. Erdmann and T. Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2:1419–1424, 1987.

[27] M. Čáp, P. Novák, A. Kleiner, and M. Selecký. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE Transactions on Automation Science and Engineering*, 12(3):835–849, 2015.

[28] Konstantin Yakovlev, Anton Andreychuk, and Vitaly Vorobyev. Prioritized multi-agent path finding for differential drive robots. In *Proceedings of the 2019 European Conference on Mobile Robots (ECMR 2019)*, pages 1–6. IEEE, 2019.

[29] David Silver. Cooperative pathfinding. In *Proceedings of the First AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 117–122, 2005.

[30] Ko-Hsin Cindy Wang and Adi Botea. Mapp: a scalable multi-agent path planning algorithm with tractability and completeness guarantees. *Journal of Artificial Intelligence Research*, 42:55–90, 2011.

[31] Satyendra Singh Chouhan and Rajdeep Niyogi. Dimpp: a complete distributed algorithm for multi-agent path planning. *Journal of Experimental & Theoretical Artificial Intelligence*, pages 1–20, 2017.

[32] Van Den Berg et al. Reciprocal n-body collision avoidance. *Robotics research*, pages 3–19, 2011.

[33] Dingjiang Zhou, Zijian Wang, Saptarshi Bandyopadhyay, and Mac Schwager. Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells. *IEEE Robotics and Automation Letters*, 2(2):1047–1054, 2017.

[34] Vishnu R Desaraju and Jonathan P How. Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees. In *2011 IEEE International Conference on Robotics and Automation*, pages 4956–4961. IEEE, 2011.

[35] Binyu Wang et al. Mobile robot path planning in dynamic environments through globally guided reinforcement learning. *IEEE Robotics and Automation Letters*, 5(4):6932–6939, 2020.

[36] Mikayel Samvelyan et al. The StarCraft multi-agent challenge. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, volume 4, pages 2186–2188, 2019.

[37] Tabish Rashid et al. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement Learning. *35th International Conference on Machine Learning, ICML 2018*, 10:6846–6859, 2018.

[38] Bei Peng, Tabish Rashid, Christian Schroeder de Witt, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Böhmer, and Shimon Whiteson. Facmac: Factored multi-agent centralised policy gradients. *Advances in Neural Information Processing Systems*, 34:12208–12221, 2021.

[39] Guangzheng Hu, Yuanheng Zhu, Dongbin Zhao, Mengchen Zhao, and Jianye Hao. Event-triggered communication network with limited-bandwidth constraint for multi-agent reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[40] Zhiqiang Pu, Huimu Wang, Zhen Liu, Jianqiang Yi, and Shiguang Wu. Attention enhanced reinforcement learning for multi agent cooperation. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[41] Shanqi Liu, Weiwei Liu, Wenzhou Chen, Guanzhong Tian, Jun Chen, Yao Tong, Junjie Cao, and Yong Liu. Learning multi-agent cooperation via considering actions of teammates. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[42] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[43] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[44] Pietro Falco, Abdallah Attawia, Matteo Saveriano, and Dongheui Lee. On policy learning robust to irreversible events: An application to robotic in-hand manipulation. *IEEE Robotics and Automation Letters*, 3(3):1482–1489, 2018.

[45] Shixiang (Shane) Gu, Timothy Lillicrap, Richard E Turner, Zoubin Ghahramani, Bernhard Schölkopf, and Sergey Levine. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[46] Florian Laurent, Manuel Schneider, Christian Scheller, Jeremy Watson, Jiaoyang Li, Zhe Chen, Yi Zheng, Shao-Hung Chan, Konstantin Makhnev, Oleg Svidchenko, et al. Flatland competition 2020: Mapf and marl for efficient train coordination on a grid world. In *NeurIPS 2020 Competition and Demonstration Track*, pages 275–301. PMLR, 2021.

[47] Peter Hart. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[48] Sven Koenig and Maxim Likhachev. D* lite. In *Proceedings of the 18th AAI Conference on Artificial Intelligence (AAAI 2002)*, pages 476–483, 2002.

[49] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, may 1998.

[50] John Schulman et al. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[51] Christopher Berner et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[52] Chao Yu et al. The surprising effectiveness of ppo in cooperative multi-agent games, 2021.

[53] Karl Cobbe. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[54] Max Jaderberg et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[55] Alexey Skrynnik, Anton Andreychuk, Konstantin Yakovlev, and Aleksandr Panov. Pathfinding in stochastic environments: learning vs planning. *PeerJ Computer Science*, 8:e1056, 2022.

[56] Ziyuan Ma, Yudong Luo, and Hang Ma. Distributed heuristic multi-agent path finding with communication. In *2021 IEEE International Conference on Robotics and Automation (ICRA 2021)*, pages 8699–8705. IEEE, 2021.

Alexey Skrynnik received his M.S. degree in computer science from Rybinsk State Aviation Technical University in Rybinsk, Russia, in 2017. Since 2018, he has been a junior researcher at the Federal Research Center ”Computer Science and Control” of the Russian Academy of Sciences. From 2021, he has been working as a researcher at the Artificial Intelligence Research Institute in the Neurosymbolic Integration research group. His current research focuses on reinforcement learning, learning and planning, and multi-agent systems.

Anton Andreychuk received his M.S. degree in Computer Science from Peoples' Friendship University of Russia in 2017. He has continued his education as a PhD student in Peoples' Friendship University of Russia till 2021. From 2021, he has been working as a researcher at the Artificial Intelligence Research Institute in the Neurosymbolic Integration research group. His current research focuses on heuristic search, single and multi-agent pathfinding.

Konstantin Yakovlev received his PhD in Computer Science in 2010 from the Institute for Systems Analysis of Russian Academy of Sciences, Moscow, Russia.

He is currently the leading researcher at the Federal Research Center for Computer Science and Control of Russian Academy of Sciences and also affiliated with AIRI, where he holds the same position, as well as with HSE University and Moscow Institute of Physics and Technology. His research interests include (but are not limited to) heuristic search, single- and multi-agent pathfinding, motion planning, multi-agent systems and robotics.

Aleksandr I. Panov earned an M.S. in Computer Science from the Moscow Institute of Physics and Technology, Moscow, Russia, 2011 and a Ph.D. in Theoretical Computer Science from the Institute for Systems Analysis, Moscow, Russia, in 2015.

Since 2010, he has been a research fellow with the Federal Research Center “Computer Science and Control” of the Russian Academy of Sciences. Since 2018, he has headed the Cognitive Dynamic System Laboratory at the Moscow Institute of Physics and Technology, Moscow, Russia. He authored three books and more than 100 research papers. In 2021, he joined the research group on Neurosymbolic Integration at the Artificial Intelligence Research Institute. His academic focus areas include behavior planning, reinforcement learning, embodied AI, and cognitive robotics.