IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON COLLABORATIVE INTELLIGENCE FOR INTERNET OF VEHICLES

Received September 2, 2021, accepted September 5, 2021, date of publication September 9, 2021,
date of current version September 17, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3111321

Hybrid Policy Learning for Multi-Agent Pathfinding

ALEXEY SKRYNNIK[1], ALEXANDRA YAKOVLEVA[2], VASILII DAVYDOV[2], KONSTANTIN YAKOVLEV[1,2], AND ALEKSANDR I. PANOV[1,2]

[1]Federal Research Center "Computer Science and Control," Russian Academy of Sciences, 119333 Moscow, Russia
[2]Moscow Institute of Physics and Technology, Moscow Region, 141700 Dolgoprudny, Russia

Corresponding author: Alexey Skrynnik (skrynnik@isa.ru)

This work was supported by the Ministry of Science and Higher Education of the Russian Federation under Project 075-15-2020-799.

ABSTRACT In this work we study the behavior of groups of autonomous vehicles, which are the part of the Internet of Vehicles systems. One of the challenging modes of operation of such systems is the case when the observability of each vehicle is limited and the global/local communication is unstable, e.g. in the crowded parking lots. In such scenarios the vehicles have to rely on the local observations and exhibit cooperative behavior to ensure safe and efficient trips. This type of problems can be abstracted to the so-called multi-agent pathfinding when a group of agents, confined to a graph, have to find collision-free paths to their goals (ideally, minimizing an objective function e.g. travel time). Widely used algorithms for solving this problem rely on the assumption that a central controller exists for which the full state of the environment (i.e. the agents current positions, their targets, configuration of the static obstacles etc.) is known and they cannot be straightforwardly be adapted to the partially-observable setups. To this end, we suggest a novel approach which is based on the decomposition of the problem into the two sub-tasks: reaching the goal and avoiding the collisions. To accomplish each of this task we utilize reinforcement learning methods such as Deep Monte Carlo Tree Search, Q-mixing networks, and policy gradients methods to design the policies that map the agents' observations to actions. Next, we introduce the policy-mixing mechanism to end up with a single hybrid policy that allows each agent to exhibit both types of behavior – the individual one (reaching the goal) and the cooperative one (avoiding the collisions with other agents). We conduct an extensive empirical evaluation that shows that the suggested hybrid-policy outperforms standalone stat-of-the-art reinforcement learning methods for this kind of problems by a notable margin.

INDEX TERMS Multiagent systems, path planning, machine learning, intelligent transportation systems, reinforcement learning, Monte-Carlo Tree Search.

I. INTRODUCTION

The number of commercial and personal vehicles steadily grows causing pressure on the existing transportation networks. One of the ways to mitigate this issue to a certain extent is to increase the autonomy of the vehicles. The expectation is that autonomous vehicles inter-connected to the Internet of vehicles will use the shared resources (transportation networks) in a smarter way [1], [2], reducing the risk of accidents, traffic congestion and increasing the safety and comfort for their passengers. The Internet of Vehicles (IoV) in a smart city is already beginning to be practically implemented through the introduction of V2X technologies [3]. In addition to questions about what data to collect and how to organize the communication for building an Internet of vehicles, the role of tasks related to how to use this data to improve the efficiency of each autonomous vehicle working in connection with others is increasing [4], [5].

According to the Society of Automotive Engineers (SAE), 6 levels of vehicle autonomy exist,[1] with 0 being conventional vehicle lacking any systems that might control any aspects of vehicle's motion and 6 correspondings to a fully-autonomous vehicle capable of operating without any guidance by a human driver. In this work, we are focusing on the vehicles of higher degrees of autonomy.

The associate editor coordinating the review of this manuscript and approving it for publication was Celimuge Wu.

[1]https://saaq.gouv.qc.ca/fileadmin/documents/publications/classification-society-automotive-sngineers-en.pdf

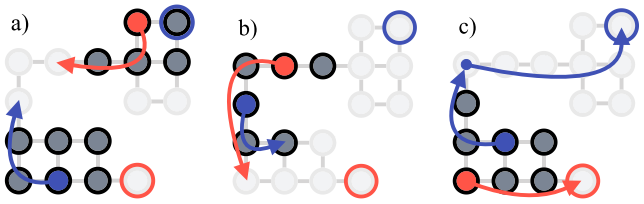


FIGURE 1. Example of the cooperative behavior in the multi-agent partially observable environment. a) The agents move to their goals relying on partial observations of the grid world (vertexes shaded with light gray are no visible to the agents at the current time step). b) Then agents see each other, and one of them lets the other pass by stepping aside. c) After resolving the conflict, the agents continue moving towards their goals.

One of the challenging scenarios for such vehicles is moving in a heterogeneous environment with numerous other moving entities such as pedestrians, autonomous vehicles, non-autonomous vehicles, etc. A characteristic example is a navigation on a parking lot. This setup is challenging for the following reasons. First, conventional means of organizing traffic such as traffic lights, road lanes, road signs, etc. are absent and vehicles have to safely navigate to their goals without them. Second, the connection to the IoV (and peer-to-peer connection as well) may suffer significantly, thus the centralized or semi-centralized (like in [6]) coordination of vehicles movements is limited. Consequently, in this setting, vehicles have to predict the behavior of other vehicles in the environment to avoid conflicts and deadlocks. An example of such scenario is presented in Figure 1.

The described problem can be abstracted to the so-called Multi-agent Pathfinding (MAPF) [7], a well-studied problem with a range of different algorithms tailored to solve it. These algorithms differ in assumptions they make on how mobile agents may move through the workspace, how the collisions are defined etc. Still, most of those algorithms assume full knowledge of the environment. I.e. they accept as input not only the map of the environment but also locations of all the agents and their goals. Under such assumption MAPF algorithms are capable to produce collision-free plans for all the agents. This approach is hard to implement in the considered scenario when numerous vehicles constantly enter and leave the parking lot, perfect localization and mapping are burdensome, and connection to the Internet of vehicles may be unstable. In the presence of such disturbances, it is reasonable for each vehicle (agent) to operate individually, relying on the local observations. The approaches to such partially-observable multi-agent navigation are, indeed, known, see the seminal work on ORCA [8] for example. However, they often suffer from the deadlocks as they do not take the cooperative aspect of the problem into account (i.e. some agents have to stop progressing towards the goals in certain cases in order to let the other agents go). To overcome this problem in this work we suggest utilizing machine learning, and more specifically – reinforcement learning (RL) [9]. We are inspired by the successful experience of colleagues who have already used machine learning methods for the IoV related problems [10], [11].

In this paper, we propose to use reinforcement learning [9], [12] to generate the behavior of each autonomous agent in a multi-agent partially-observable environment, which in our case is an abstraction for the Internet of vehicles. Model-free reinforcement learning methods have shown excellent results in behavior generation tasks for single agents [13]–[15] and cooperative environments [16], [17]. Modern deep reinforcement learning algorithms cope well with complex observation space (visual environments) [18] and stochastic environmental conditions [19]. Nevertheless, some cases in multi-agent environments, in which long-term prediction of the purposeful behavior of other agents is essential, are hardly tackled by most of the model-free approaches. To account for the purposeful actions of the other agents, the so-called model-based approaches play an important role. They have shown impressive results in such environments as chess and Go [20], [21] where planning ahead is much needed. On the other hand, model-based approaches are very demanding on the quality of the model of the environmental dynamics and often require access to an unlimited launch of the environment simulator.

In this paper, we suggest utilizing both model-free and model-based RL in the following fashion. We decompose partially-observable MAPF into two subtasks: reaching the goal and avoiding collisions with other agents and deadlocks. The first one is handled by the model-free RL module which outputs the suggestion on which action to take (i.e. actions' probability distribution) provided with the current observation of the agent. The second subtask is handled by either the model-based RL module or another model-free RL module. All modules are trained separately and then are integrated into a resultant hybrid policy (either model-free + model-based or model-free + model-free) via the mixing mechanism. Such an approach leads to very promising results as our experimental studies suggest.

The main contributions of our work can be summarized as follows:

- we present a variant of the MAPF problem which is an abstraction of the challenging IoV setting when the communication and observation are limited,
- we decompose the pathfinding problem into the two subproblems: avoiding static obstacles and resolving inter-agent conflicts in the local neighborhood,
- we suggest an original hybrid model-free + model-based/model-free RL method that handles both problems in the integrated way,
- we develop an original simulator – POGEMA (Partially Observable Grid Environment for Multi-Agent scenarios) that allows generating MAPF problems of different levels of complexity for further empirical evaluation,
- we show empirically that the suggested hybrid is an efficient tool to solving the considered tasks and it outperforms pure model-free/model-based RL approaches.

The rest of the paper is organized as follows. Section II provides an overview of the related works from the MAPF

domain. The problem statement is presented in section III followed by the description of the suggested method in section IV. We describe the developed simulator in section V. Empirical evaluation and results are presented in section VI. Section VII concludes.

II. RELATED WORK

Multi-Agent Pathfinding (MAPF) is a challenging problem that is being actively investigated in AI and robotics communities.

The first methods for solving the problem were planning algorithms searching for the minimum cost path in the joint action space, e.g. A^* [22]. Such algorithms are extremely ineffective in the case of a large number of agents due to the exponential growth of computational complexity.

Currently used classical approaches suffer less from the curse of dimension, but are still time-consuming. Planning algorithms can be divided into two types: centralized and decentralized. Centralized ones rely on a control center that accumulates information on all agent's positions. This group of algorithms includes conflict-based search algorithm [23] and its enhanced versions [24], [25] guaranteeing optimality and prioritized planning algorithms [26] that are capable to build near-optimal solutions under conditions. Decentralized planning considers the problem in the setting when access to information about other agents is limited. One of the most cited decentralized algorithms ORCA [27] modifies agent's velocities on the fly to prevent collisions. Combinations of ORCA with centralized planning algorithms are also known [28].

Planning via Monte Carlo Tree Search (MCTS) which we utilize is not new for MAPF. It was recently adapted to solve Numberlink puzzles [29]. Another MCTS-based approach [30] showed great scalability on the benchmark SysAdmin domain.

Many reinforcement learning approaches address MAPF by learning separated policies or Q-functions, which means that each of them considers only a single agent. Such methods require the use of additional trickery to get agents to interact in the desired way. In actor-critic approaches [31] the critic network is usually trained in centralized fashion [32], [33] to enhance cooperation. Several off-policy learning algorithms [34]–[36] suggest to decompose optimized joint Q-function into action-values of single agents or pairs of them. Heuristics could be also helpful in collaboration tasks, e.g. [37] introduces new loss functions to prevent agents from blocking each other. The advantage of our approach is the absence of built into training heuristic rules. There are also some approaches additionally relying on exact agents behavior modeling [32], [37].

Probably the most natural way of learning agents to take into account other agents is to design observation containing the necessary information about other agents as it was done in [37]. Our simulator POGEMA returns complex observation that combines the location of static obstacles, positions of agents, and their targets in the area of visibility.

III. PROBLEM STATEMENT

Consider n homogeneous agents, e.g. representing driverless vehicles, navigating the shared environment which is discretized to a 4-connected grid. The task of each agent is to reach the given goal position (grid cell) from the start one. To do this at each time step the agent can either stay at its current position or move to a neighboring one. The sequence of move/wait actions is called a plan and denoted $p_i = (a_1, a_2, \ldots, a_k)$, where i is the agent's index. The cost of the plan is defined by the time step by which the plan ends, $cost(p_i) = k$. Lower-cost plans correspond to faster ways of reaching the goal.

Two plans are said to be conflict-free if the agents following them never collide. We consider the two common types of collisions: vertex collision and edge collision. Vertex collision occurs when two agents occupy the same grid cell at the same time step (e.g. the first agent waits in the cell and the second agent moves to it). Edge collision occurs when two agents swap vertices using the same edge at the same time step. In general, a few more collision types can be defined, see [38], but for the sake of clarity, we limit ourselves to the vertex and edge collisions in this work. Still, the method we are presenting is agnostic to the collision definition and can support other collision types provided that the corresponding collision detection mechanism is implemented.

Unlike numerous works on multi-agent pathfinding, we assume that the global state of the environment, i.e. the global grid-map, the positions of all agents, their goals, etc., is not known. Instead, each agent possess only a local observation at each time step. This observation includes:

- the local map of size l, i.e. the patch of the global grid-map centered at agent's position, where l is the predefined parameter,
- the positions of the other agents if they fall within the local map (positions of the other agents are unknown),
- global information about the agent's goal in the form of a position (x_g, y_g), if it falls within the local map, or in the form of its direction (projection on the edge of the local map).

The problem now is to design a common individual policy $\pi(s)$ for an agent that maps observations to actions, s.t. when each agent executes this policy it reaches its goal and no collisions happen along the way. The cost of the resulting set of plans can be defined as the sum of costs of the individual plans, or the maximum over such costs. In this work we are not aiming at designing the policy that guarantees to produce the set of plans of the least possible cost, however producing lower-cost joint plans is, obviously, preferable.

In contrast to the standard MAPF problem statement, we suggest that it is enough for an agent to learn a policy due to interaction with the environment for a certain number of episodes. Furthermore, we enable the agent to simulate its actions with a copy of the environment, thus giving it access to an ideal model of transitions in the observation space, thus not violating the conditions of partial observability.

Formally, the interaction of an agent with the environment is described as Markov decision process (MDP) (S, A, P, r, γ) where S is the set of environment states, $a \in A$ is the set of agent's actions preformed in the environment, $r(s, a){:}S \times A \rightarrow \mathbb{R}$ is a reward function, and γ is the discount factor. The agent chooses its action based on the policy π which is a function $\pi(a|s){:}A \times S \rightarrow [0, 1]$ that is the main part of the actor. In value-based approaches, the policy is formed by estimation of the value of action-state pairs: $Q(a_t, s_t) = r(s_t, a_t) + \mathbb{E}(\sum_{i=1}^{\infty} \gamma^i r(s_{t+i}, a_{t+i}))$. The optimal policy π is then $\pi(s_t) = argmax_{a \in A} Q(s_t, a)$.

In the partially observable Markov decision process (POMDP), it is assumed that a state of the environment is not fully known to the agent. Instead it is able to receive only a partial observation $o \in O$ of the state. Thus POMDP is a tuple (S, O, A, P, r, γ). The policy now is a mapping from observations to actions: $\pi(a|o){:}A \times O \rightarrow [0, 1]$. Q-function in this setting is also dependent on observation rather than on state: $Q(a_t, o_t) = r(s_t, a_t) + \mathbb{E}(\sum_{i=1}^{\infty} \gamma^i r(s_{t+i}, a_{t+i}))$. The estimation of the value of the state $V(s)$ is then $V(s_t) = max_{a \in A} Q(s_t, a)$.

In model-based reinforcement learning, it is assumed that the agent explicitly has access to the transition function of the environment P or to some approximation of it $\hat{P}$. This function is usually used to generate additional experience for effective training of the critic and the actor. In our case, when solving the MAPF problem, only a certain approximation of the true model $\hat{P}$ is available to the agent since the actions of other agents cannot be predicted entirely and are replaced only by some approximating policy, for example, a random one.

IV. METHOD

To solve the stated problem we suggest to decompose the partially-observable MAPF into the two sub-problems: progressive movement of each agent towards the goal with the avoidance of static obstacles and avoiding conflicts with other agents. Accordingly, we develop two learnable policies for the agent: a policy for reaching the goal and a policy for resolving the conflicts.

We chose Proximal Policy Optimization (PPO) to learn a goal-reaching policy as one of the most versatile methods for solving a wide range of RL problems [14], [39], including procedurally-generated navigation tasks [40]. To learn the collision avoidance policy we have tried both model-free and model-based approaches, QMIX and MCTS respectively. QMIX is one of the widely used off-policy algorithms for multi-agent learning [35]. MCTS is known to show great results in highly complex antagonistic games [41] (Chess, Go, etc) and we chose to adapt it to our challenging problem, i.e. to multi-agent cooperative path-finding with limited observations. As our experimental results show this choice was worthwhile.

Both modules are learned in parallel. After learning, at the inference phase we apply a mixing mechanism to end up with the hybrid policy that combines both types of behavior (goal-reaching and collision-avoidance). The general scheme of the proposed approach is shown in Figure 2. Please note, that MCTS conflict-avoidance module is depicted, while it is interchangeable by the QMIX one. Overall, the suggested architecture of the hybrid policy is flexible and one can plug any other RL policy for goal reaching or collision avoidance, with only minor changes.

Next, we describe both learnable policies and the mixing mechanism in more detail.

A. GOAL REACHING MODULE

The Proximal Policy Optimization (PPO) [42] approach has proven effective in many RL problems, especially in large-scale learning setups [39]. PPO is a variant of advantage actor-critic, which relies on specialized clipping in the objective function to penalize the new policy for getting far from the old one. PPO uses Generalized Advantage Estimation [43], a well-known approach for variance reduction.

The pathfinding problem in a partially observable environment is quite complex, even for a single agent. The agent needs to master several tasks: avoiding obstacles (sometimes dynamic), exploring the environment, and moving towards the target. The reward signal in such an environment is sparse; the agent receives it only at the end of the episode. If one trains an agent in such scenarios, then even accidentally getting to the target will be almost impossible. To overcome this problem, we suggest using curriculum learning.

Curriculum learning in RL aims to order tasks to provide curricula for training an agent, improving its performance on the target task. We can design training curricula for POGEMA using parameters of the environment (e.g. max distance to the target, obstacle density). In Figure 3, we show an example of the progressively increasing complexity of the tasks. In this work, we use human expert knowledge to provide such a curriculum plan. We denote PPO trained using that plan as cPPO.

B. CONFLICT RESOLUTION MODULE

1) MODEL-BASED APPROACH

The central mechanism of the Conflict resolution module is the Monte-Carlo Tree Search mechanism. Monte-Carlo Tree Search (MCTS) is a powerful technique widely used to solve decision-making problems when an agent has to decide which action to take next observing the current state of the world. Most often MCTS-based methods are used in the context of the adversarial two-player games and they show great success in solving this type of problems [44], [45]. Even bigger popularity came to MCTS in 2016 when AlphaGo [20] showed super-human performance in Go (which is a very challenging game to computers). AlphaGo was a mixture of human-designed game heuristics, MCTS, and deep learning. Its enhanced successor, AlphaZero [41], did not require any human knowledge and mastered not only Go but a range of other board games like chess and shogi. Inspired by the success of these model-based approaches we adopted the idea of mixing together planning (search) and deep learning to the considered conflict resolution problem. Next, we describe

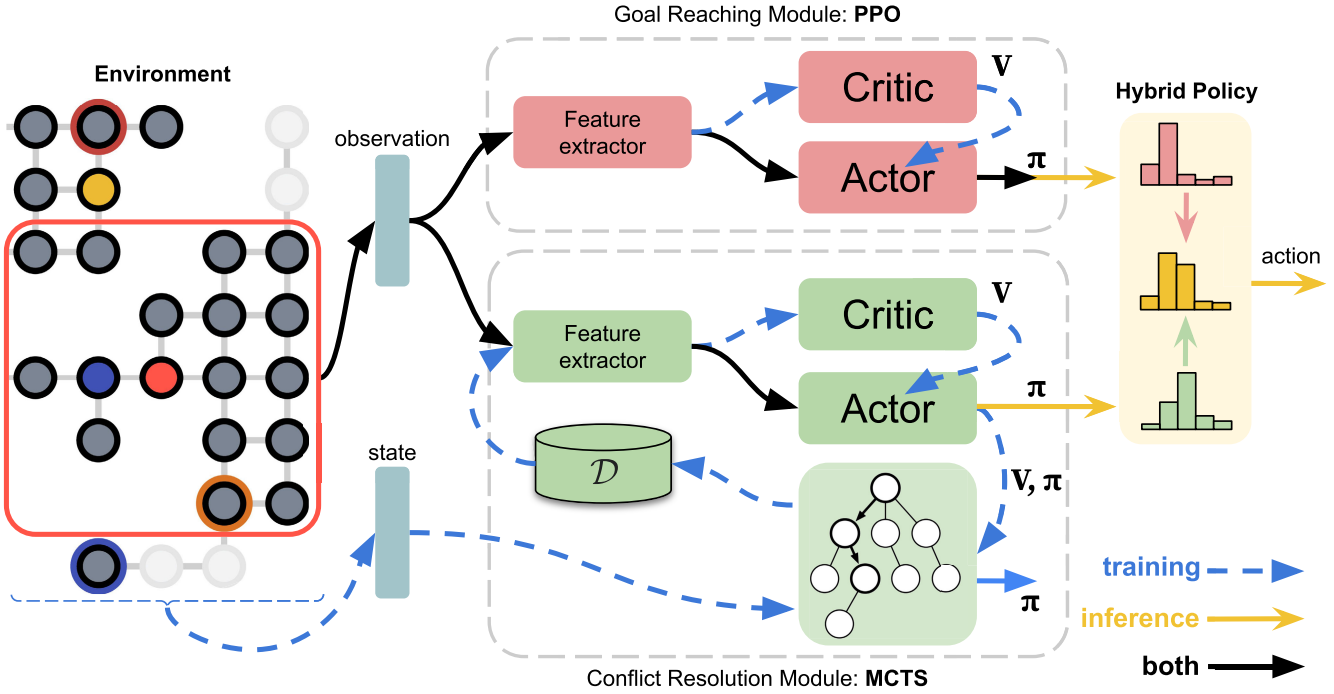


FIGURE 2. The hybrid policy approach. Three main modules are identified: Goal Reaching Module, Conflict Resolution Module, and the Hybrid Policy module. The latter receives the distributions over the actions and combines them so the agent demonstrates both types of behavior: reaching the goal and avoiding conflicts with other agents.

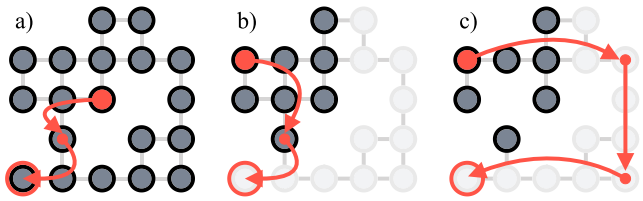


FIGURE 3. Example of progressively increasing difficulty of the scenario. a) The target is in the agent's field of vision, and the path to it is open. b) The target is out of the agent's field of vision, but the path to it is quite simple and is in the same direction. c) The target is out of observation of the agent and it's difficult to reach. The agent must properly explore the environment to find it.

the necessary background of MCTS and its combination with deep learning and finally propose our variant of model-based learning algorithm to address the partially-observable multi-agent pathfinding.

Basic MCTS: MCTS solves the problem of choosing an action (game move) provided with the current state of the world/game (vanilla MCTS assumes a fully observable environment, where states and observations are equivalents). It does so by virtue of simulating the most promising variants of how the game develops in the future. This process can be thought of as planning (i.e. MCTS reasons about the actions and their consequences). Technically this planning is realized as a tree-search process. The node in the tree corresponds to a game state, the outgoing edges correspond to the actions applicable in that state. At each iteration of the search the four following steps are made: 1) descending the tree to a leaf along the most promising path, 2) adding the successor, 3) simulating the game from the successor using a default policy (usually random) to get the reward (e.g. 1 if we win, 0 if we lose), 4) propagating the reward backward. The notion of promising states/actions is formalized via a range of numeric variables that are assigned to each node/edge. For each node, we store the average reward that was achieved from the corresponding game-state, V_s. For each edge we store the average reward achieved when the corresponding action was chosen from the source state of that edge, Q_{sa}. Initially, these variables are zeroes and they are updated at each back-propagation step. They are decreased in case the current simulation ends with a loss, and increased if the game ends with a win. Intuitively we want to penalize the states/actions that are likely to lead to losing the game and do the opposite for the states/actions that are likely to lead to a win. V_s is also used at step 1 of a search iteration when we are descending the current tree. Here we start at the root and for a node pick the children according to the UCT [46] formula:

$$UCT = V_s + c\sqrt{\ln N_{p(s)}/N_s} \tag{1}$$

Here N_s is the number of times the (child) node was visited before and $N_{p(s)}$ is the same for the parent node (these counters are stored within the nodes and are incremented at the back-propagation step). The rationale behind this formula is the following. On the one hand, we want to better explore the game variants that are achieved by the moves that promise good outcomes (to be sure that current promises are consistent). On the other hand, we want also to invest time

in evaluating the under-explored states/actions (as they may potentially lead to good results as well). Parameter c is the one that controls this *exploitation-exploration* trade-off (the higher it is the more we are likely to explore). In practice, c is chosen empirically.

After a dedicated number of search iterations (usually defined by the allotted time budget) MCTS stops. The principal result of MCTS is the partially-built game tree that contains collected statistics (node/edge visits counters, V_s, and Q_{sa} values). This output is utilized now to finally decide which action to choose from the current game state (the root of the node). Typically the action that leads to the most visited child (of the root) is chosen. In other words, we greedily pick an action that leads to the promising and sufficiently explored state of the game (the state from which we know how to win). A more general approach is to construct a stochastic policy out of the gained data, e.g. by distributing the probabilities of picking the actions proportionally to the numbers of visits of the root child nodes. Thus, the action that leads to the most visited node will have a higher chance of being selected, while other actions still have a chance to be applied.

MCTS With Deep Learning: Applying MCTS every time an agent has to take action in the environment (e.g. move in a game) might be very time-consuming, as to come up with the consistent policy one needs thousands of thousands of MCTS simulations. To mitigate this issue one may wish to construct a computationally efficient still powerful (i.e. being able to select good actions) approximator of MCTS via deep learning. In [41] such an approximator (in the form of a deep neural network) was proposed for a large class of adversarial two-player games and got the name *AlphaZero*.

AlphaZero is made to provide both the policy (distribution of actions probabilities) and the expected reward is given a state s. We will denote the approximated policy and reward as π and v respectively, while the ground-truth policy and reward (i.e. the ones that can be obtained via MCTS) as $\boldsymbol{\pi}^*$ and v^*. Technically, the approximator is a neural network, parameterized by a set of weights θ: $f_\theta(s) = (\pi, v)$. This network is trained in a supervised fashion on the samples $\{s, \boldsymbol{\pi}^*, v^*\}$ collected by the MCTS search with the following loss function $\mathcal{L}$:

$$\mathcal{L} = -\boldsymbol{\pi}^{*T} \log \boldsymbol{\pi} + (v^* - v)^2 \tag{2}$$

The examples $\{s, \boldsymbol{\pi}^*, v^*\}$ are collected during the episodes of self-play as follows. Consider the initial state of the game, s_0. An adapted MCTS (deep MCTS) is invoked on that state, i.e. the MCTS tree is built from s_0. Several principle features make this MCTS different from the vanilla one and we will describe them shortly. After building the tree we extract the policy, i.e. the probabilities of taking each action from s_0, and form a tuple $\{s_0, \boldsymbol{\pi}^*, \_\}$, here _ is the wildcard which will be filled later when the episode finishes (the game is over) and we get the actual reward. This tuple constitutes the first training example. Then we choose an action sampling from $\boldsymbol{\pi}^*$ and transfer to the new state s_1. We make s_1 the new root of the MCTS tree and run the search again (technically we re-use the data from the previous MCTS tree by keeping the sub-tree rooted in s_1 and pruning away the remaining irrelevant part). After this search finishes we get another training sample $\{s_1, \boldsymbol{\pi}^*, \_\}$. We continue collecting samples in that way until the game ends and we get the reward v^* (e.g. 1 if we win, 0 if we lose). This reward is used now for all the samples collected through the self-play and the fully defined samples are added to the training dataset which will further be used to train the MCTS approximator.

The last ingredient to be discussed is the adapted MCTS which is used to create a tree from which the policy sample is extracted. The major difference is that when a node is expanded and its child is added to the tree no (random) rollout is performed to get the reward. Instead, the value v predicted by f_θ is used as the reward and is back-propagated in the conventional MCTS fashion. In other words, we do not simulate the game until it ends but rather asks the neural network to predict the outcome immediately. Similarly, the predictions of f_θ are used when descending the tree. More formally, deep MCTS uses the following formula instead of (1):

$$U = Q_{sa} + C_s P_{sa} \cdot \frac{\sqrt{N_s}}{1 + N_{sa}}, \tag{3}$$

where Q_{sa} is average reward (for an action a taken in the state s), P_{sa} is an action probability given by the neural network, N_s is a number of visits of the node s and N_{sa} is a number of times move a from s was chosen.

Deep MCTS for Conflict Resolution: In this work we suggest utilizing the deep MCTS approach to the problem of resolving conflicts in partially-observable multi-agent path planning. In our setup we adopt the egoistic paradigm, i.e. we consider the agents as the dynamic obstacles between which the principal agent has to maneuver. Although not all multi-agent interactions can be reduced to this setting, the policy that we can learn from following this approach scales well and can solve numerous non-trivial problems, as will be shown in Section VI.

The input to our MCTS approximator f_θ, which is, indeed, a deep neural network, is the agent's observation o (as described in Section III). The output is the policy vector $\pi = (p_{up}, p_{down}, p_{right}, p_{left}, p_{wait})$ which assigns a probability to each of the 5 possible actions (go left/right/up/down or wait at place).

To train f_θ we follow the general pipeline of deep MCTS, i.e. we collect training samples and optimize the loss function 2. Within collecting these samples we use the tuples $((x_0, y_0), \ldots, (x_n, y_n), t)$ to identify nodes in the MCTS tree, where (x_i, y_i) are the coordinates of the ith agent and t is timestep. I.e. we grow the tree from the perspective of only one agent, while assuming that the others are greedily choosing the actions according to the currently available policy (i.e. the one that is provided by the neural network). Including t in the state description is essential to take the time dimension into account and to distinguish states with the same agents' positions but arising in different time steps (so we can avoid cycles and propagate the reward correctly).

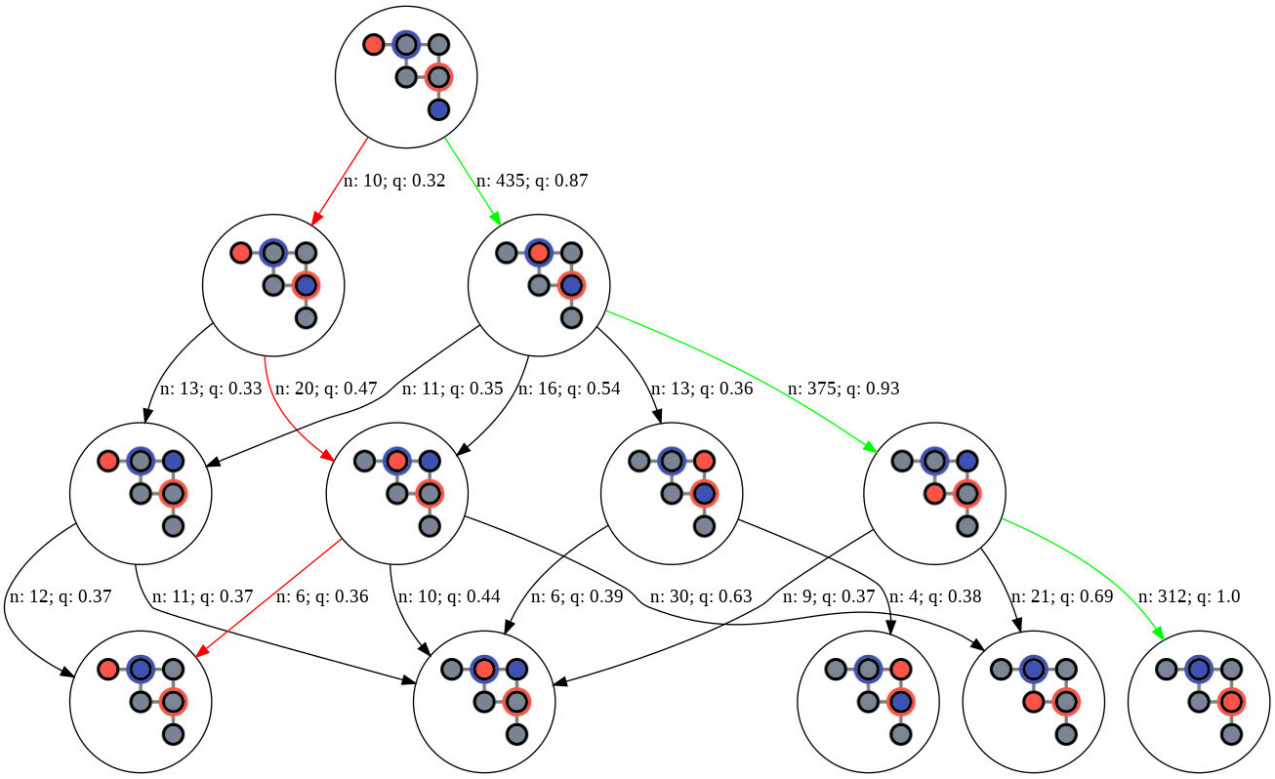


FIGURE 4. Example of subtree build by our Deep MCTS algorithm for conflict resolution for toy 3 × 3 environment with 2 agents. Positions of agents and their goals are shown in the corresponding tree nodes. Each edge is provided with collected statistics of edge visit counter *n* and *q* values. The green path corresponds to the most promising path in the tree. It ends up in the termination state for both agents. Another randomly chosen path is in red.

Importantly, the input to f_θ, which is used to compute P_{sa} and v while MCTS simulations, is always the agent's observation o. The training episode ends if the agent reaches the goal or takes the maximum number of steps in the environment. The reward that the agent gets (and which is used for training samples) takes into account both how many time steps the agent spent before the episode ends and how far it is from the goal (in case the latter was not reached). Intuitively, maximum reward corresponds to the fastest way of reaching the goal while not colliding with the obstacles (both static and dynamic). The exact formula of reward will be given further on.

Figure 4 illustrates subtree grown using the proposed method.

2) MODEL-FREE APPROACH

As another way for the conflict resolution module to work, we considered a model-free approach based on a coordinated change in the Q-functions of different agents. This algorithm is based on deep Q-learning that is a method to optimizes Q-function $Q(s, a|\theta)$. It uses deep Q-network as an approximator for this function, which is updated with the following formula:

$$\mathcal{L} = \sum_{i=1}^{b}[((r^i + \gamma max_{a^{i+1}} Q(s^{i+1}, a^{i+1}|\theta)) - Q(s^i, a^i|\theta))^2] \tag{4}$$

For multi-agent problems, DQN can be modified into the QMIX algorithm. QMIX uses the DQN approximator loss function to update the agents' weights. But instead of using individual Q functions for each agent, it uses joint Q-function Q_{tot} to extract additional information about other agents. It takes all the individual agents' Q-functions; mixes them, using its' mixing network, and computes single joint Q-function Q_{tot}. The weights for the mixing network are received from the hyper networks, that take the global state as input. The loss function uses Q_{tot} in the DQN loss then:

$$\mathcal{L} = \sum_{i=1}^{b}[(y^i_{tot} - Q_{tot}(\tau^i, u^i, s^i|\theta))^2] \tag{5}$$

$$y_{tot} = r + \gamma max_{t^{i+1}} Q_{tot}(\tau^{i+1}, u^{i+1}, s^{i+1}|\theta^-) \tag{6}$$

Here τ is a joint agents' action, u is a joint agents' observation. τ^{i+1}, u^{i+1}, s^{i+1} are joint actions to be performed, joint observations and state of the environment for the next step.

C. COMBINATION

The designed conflict resolution module is focused on learning the policy capable to avoid collisions with static obstacles and other agents. The designed goal-reaching module learns to choose actions that lead to the goal position. Indeed, to solve partially observable multi-agent pathfinding problems efficiently we need a combination of both of these

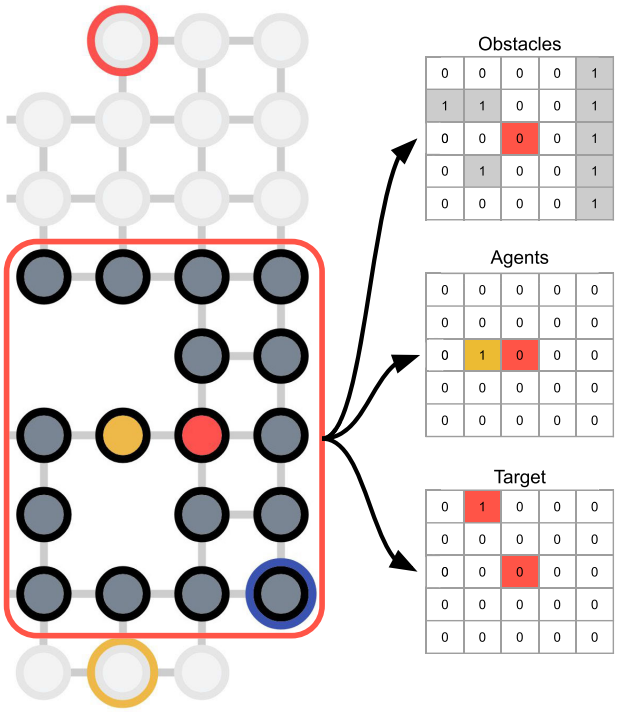


FIGURE 5. An example of the agent's observation for the agent is shown in red. The observation is comprised of three matrices: obstacles, other agents' positions, agent's goal (or its projection). If the observation spans beyond the boundaries of the grid, then the corresponding cells in the observation's obstacles matrix are considered as obstacles (see top right part of the figure), while the other matrices are filled with zeroes in these positions.

ingredients. To this end, we suggest a non-trainable mixer of the policies.

The mixer is initialized with several policies $\{\pi_i\}_{i=1}^{N}$. It takes the observation o as an input and outputs the vector of action probabilities $\pi(o)$. The output is calculated as a sum of predictions of policies, desired to be combined, according to the formula:

$$\pi(o) = \sum_{i=1}^{N} \pi_i(o) \tag{7}$$

Please note, that the mixing policy does not contain a switching rule that chooses which of the policies, goal-reaching or avoiding collision, to apply. Instead, we mix the action probabilities suggested by both policies and sample action from the resultant distribution. This simple yet effective way to combine reaching-the-goal-policy and avoid-the-conflicts policy allowed us to improve notably the performance compared to when only one policy (either goal-reaching or conflict resolving) is used (see Section VI for the details). Indeed, more involved switching mechanisms can be designed. We leave this for future work.

V. POGEMA SIMULATOR

We designed and implemented the environment simulator that takes all the specifics of the partially-observable multi-agent pathfinding into account. We call this simulator – POGEMA (Partially Observable Grid Environment for Multi-Agent Scenarios).

The following parameters define the environment in POGEMA:

- grid size $Size \geq 2$,
- obstacle density $Density \in [0, 1)$,
- number of agents $Agents \geq 1$,
- observation radius: agents get $1 \leq R \leq Size$ cells in each direction,
- the maximum number of steps in the environment before episode ends $Horizon \geq 1$,
- the distance to the goal for each agent $Dist$ (is an optional parameter, if it is not set, the distance to the goal for each agent is generated randomly).

The observation space O of each agent is a multidimensional matrix: $O : 3 \times (2 \times R + 1) \times (2 \times R + 1)$, that represents information about the environment around the agent within radius R. It includes the following 3 matrices.

Obstacle matrix: 1 encodes an obstacle, and 0 encodes its absence. If any cell of the agent's field of view is outside the environment, then it is encoded 1.

Agents' positions matrix: 1 encodes other agent in the cell, and 0 encodes his absence.

Self agent's target matrix: if the agent's goal is inside the observation field, then there is 1 in the cell, where it is located, and 0 in other cells. If the target does not fall into the field of view, then it is projected onto the nearest cell of the observation field. As a cell for projection, a cell is selected on the border of the visibility area, which either has the same coordinate along with one of the axes as the target cell or if there are no such cells, then the nearest corner cell of the visibility area is selected.

Figure 5 show an example of the observation for an agent (the one highlighted in red). Overall, the agent observes only a fraction of the environment, has access to the global map and states of the other agents (unless they fall within the agent's visibility range).

At any time step each agent has 5 actions available: stay in place, move vertically (up or down), or move horizontally (right or left). An agent can move to any free cell that is not occupied by an obstacle or other agent. If an agent moves to a cell with his own goal, then he is removed from the map and the episode is over for him.

A. REWARD

One of the natural (and general) ways to encode the reward is to assign each agent either 1 or 0 depending on whether it was able to reach its goal before the episode ended. We, however, choose to design a more informative continuous reward that gives a hint on how well the agent copes with its task. This reward is computed by the following formula.

$$v^* = \frac{c_{best\_path}}{c_{current\_path} + c_{path\_goal}} \tag{8}$$

Here c_{best_path} is the cost of the agent's individual optimal path, i.e. the number of time steps needed for the agent to

TABLE 1. Hyperparameters of the cPPO, QMIX, and MCTS algorithms.

Curriculum PPO	
feature extractor	MLP: $256 \times 512 \times 512$
actor network	MLP: 64×64
critic network	MLP: 64×64
activation function	Tanh
optimizer	Adam
learning rate	3e-4
rollout size	2048
batch size	64
update epochs	10
gamma	0.99
clip range	0.02
GAE λ	0.95
entropy coefficient	0.0
V func coefficient	0.5

QMIX	
agent network	MLP: $64 \times 64 \times 5$
mixing network	MLP: $256 \times 256 \times 1$
hyper network	MLP: $256 \times 256 \times 256$
activation function	ReLU
optimizer	RMSprop
learning rate	5e-4
optimizer epsilon	1e-5
optimizer alpha	0.99
batch size	128
buffer size	1000
gamma	0.99
target update interval	100
epsilon start	1
epsilon finish	0.05
epsilon anneal time	100000

MCTS	
feature extractor	4 blocks: [Conv2D $(64, 3, 1)$, BatchNorm2D (64), ReLU], Dropout
actor network	MLP: $128 \times 64 \times 5$
critic network	MLP: $128 \times 64 \times 1$
activation function	ReLU
optimizer	Adam
learning rate	1e-3
batch size	16
gamma	1
dropout probability	0.3
self-play episodes	100
self-play steps	5000

reach the goal along the shortest path that does not take other agents into account; $c_{current\_path}$ is the cost of the path actually undertaken by the agent in the episode; c_{path_goal} is the cost of the optimal path between the agent's current location (the one in which agent resides at the end of the episode) and its goal. The reward is given to the agent at the end of the episode. That is, if the agent managed to reach its goal via the optimal individual path then the reward is maximal. In any other case we penalize agent (via lower reward) for both *i*) not reaching the goal, *ii*) reaching the goal in a non-optimal way.

We compared the designed reward with other variants (the conventional 0-or-1 reward and the other one known from the literature on MAPF. This comparison was in favor of our reward. The details are presented in the next section.

VI. EXPERIMENTAL EVALUATION

We implemented, trained, and evaluated the performance of the suggested hybrid policies MCTS + cPPO, QMIX + cPPO and compared them to the standalone goal-reaching and conflict resolution policies (cPPO and MCTS, QMIX, QMIX + MCTS respectively) on a wide range of setups. We used PPO implementation from StableBaselines3 [47], QMIX implementation from PyMARL [48] and deep MCTS implementation of our own. Hyperparameters of the neural networks used within cPPO, QMIX, and Deep MCTS are presented in Table 1.

A. TRAINING SETUP

Each policy (cPPO, QMIX, MCTS) was trained on a range of multi-agent pathfinding problems using our POGEMA simulator. Obstacle density was set to 0.3 and observation radius to 5 for training. Training configurations are presented in Table 2. As one can note we used different sizes of the environment while training. The number of agents never exceeded 2. The maximum number of time steps before the training episode ended is shown in the 4th column of the table

TABLE 2. Training set configurations for cPPO, Deep MCTS, and QMIX algorithms. In contrast to cPPO and MCTS, QMIX was trained only on a single configuration setting because mixing network Q_{tot} is trained on a full environment state. Each stage of curriculum learning (PPO and MCTS) and training of QMIX was performed until the convergence curve reached a plateau, which corresponds to the steps column.ISR metric shows convergence results for each stage.

Curriculum PPO

#	size	agents	horizon	distance	steps	ISR (±std)
1.	6×6	1	50	3	1M	0.98 (±0.0)
2.	10×10	1	50	5	2M	0.89 (±0.01)
3.	15×15	1	50	8	3M	0.79 (±0.01)
4.	17×17	1	50	11	5M	0.76 (±0.02)
5.	32×32	1	100	20	5M	0.44 (±0.01)
6.	8×8	2	50	8	1M	0.88 (±0.01)

Deep MCTS

#	size	agents	horizon	distance	steps	ISR (±std)
1.	6×6	2	10	5	30K	0.54 (±0.03)
2.	10×10	2	10	5	30K	0.40 (±0.03)
3.	16×16	2	10	8	10K	0.22 (±0.0)

QMIX

#	size	agents	horizon	distance	steps	ISR (±std)
1.	15×15	2	100	8	2M	0.57 (±0.01)

(named *horizon*). The 5th column, *distance*, shows how far the goal was located from the start. I.e. for each problem instance, we generated the start location for an agent randomly and then picked a goal location randomly but conditioned that the cost of the individual optimal path from start to a goal does not exceed the given threshold. We also report the number of environment *steps* for each algorithm, on a particular configuration, in the last column of the table.

We utilized curriculum learning for PPO (thus – cPPO) starting from the smaller environments and closer targets and then gradually increasing both the environment size and the distance to the goal. A similar approach was adopted for MCTS training. QMIX, however, was trained on a single configuration as it trains Q_{tot} network on a full environment state.

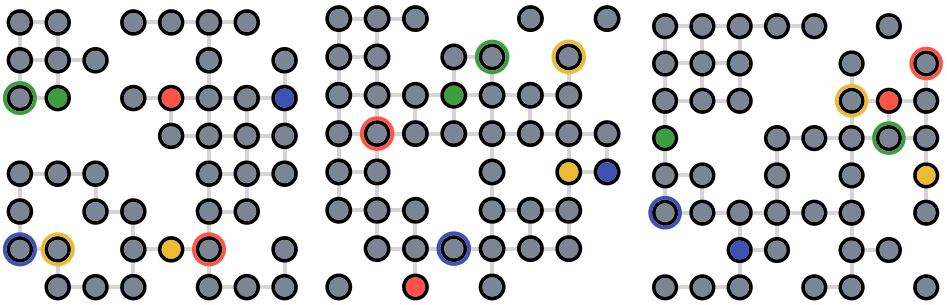


FIGURE 6. Examples of random configs from *id2-rnd*8 × 8-*4* configuration. The a average cooperative hardness for this set is 0.0, which means with the right interactions, agents will not interfere each other.

B. TEST SETUP AND EVALUATION METRICS

After training policies on the setups described above we tested and evaluated them (and their combinations) on a range of specifically created previously unseen instances. Indeed, the agent's visibility range and obstacle density were the same for test scenarios (5 and 0.3 respectively).

The first portion of the test scenarios was created randomly. These problems vary in environment's size (from 8 × 8 to 32 × 32) and number of agents (from 2 to 16). Start and goal locations for each agent in each problem instance were chosen randomly. Some examples of these setups are shown in Figure 6. In total 400 random problem instances with the specifications indicated in Table 3 were created (100 instances per each specification). The 4th column in the table shows the length of the evaluation episode (in time steps). The last column indicates the hardness of the problem. This indicator was computed as follows:

$$hardness = c_{MAPF\_solution} - \sum_{i=1}^{n} c_{path}^{(i)} \tag{9}$$

Here $c_{MAPF\_solution}$ is the cost of the conflict-free solution obtained by the off-the-shelf multi-agent path planner (which relied on the full knowledge of the environment); $c_{path}^{(i)}$ is the cost of the individual path for the ith agent (constructed without taking other agents into account). Intuitively this indicator shows how much cooperation is needed from the agents to successfully solve the problem instance. Indeed, the higher it is the harder the instance is (as it requires more involved cooperation from the agents). Indeed, different other indicators can be suggested to measure the complexity of the pathfinding problems, e.g. the total number of the graph vertices considered by the conventional systematic/heuristic search algorithm (the higher the number – the harder the problem is). In this work we were especially interested in the complexity arising from the need of the agents to cooperate, which explains the rationale behind defining the hardness indicator in the way described above.

Besides random problem instances, we evaluated the trained policies on the challenging problems where a high degree of cooperation is expected from the agents. To this end, we created a separate dataset called *cooperative* in the following way. For each specification of the random dataset, we generated 10,000 different problems and computed the *hardness* indicator for each of them. We then select the top-100 hard instances to form the cooperative dataset. Examples are shown in Figure 7. Indeed, these problems are much harder to solve than the random ones.

TABLE 3. Test configurations: random and cooperative. Random configurations are much easier to solve (as indicated in the *hardness* column) as they do not require involved cooperative behavior.

Random set

config	size	agents	horizon	hardness
id5-rnd8x8-2	8 × 8	2	32	0.0
id6-rnd8x8-4	8 × 8	4	32	1.5
id7-rnd16x16-8	16 × 16	8	64	2.0
id8-rnd32x32-16	32 × 32	16	128	6.5

Cooperative set

config	size	agents	horizon	hardness
id1-coop8x8-2	8 × 8	2	32	6.07
id2-coop8x8-4	8 × 8	4	32	15.19
id3-coop16x16-8	16 × 16	8	64	31.26
id4-coop32x32-16	32 × 32	16	128	48.05

We used the following metrics for evaluating the policies on the aforementioned datasets:

- *Individual success rate (ISR)*, which is the ratio of agents that have reached their goals before the imposed step limit. E.g. 0.8 means that 80% of agents managed to arrive to their target locations (while 20% do not) before the episode ends.
- *Cooperative success rate (CSR)*, which is the ratio of the successfully accomplished episodes, i.e. the episodes with ISR of 1. In other words, it is the percentage of problem instances that were successfully solved in a strong sense (all agents successfully reached their goals within the allotted time limit).

Indeed, CSR is a more demanding indicator and is expected to be lower compared to ISR (which is confirmed by our experimental results).

C. RESULTS

The results for each metric are divided into modules. First, we report the performance of the Conflict Resolution Module (MCTS, QMIX, MCTS + QMIX). Second, we show the results of the Goal Reaching Module, which is presented by cPPO. Third, we report the results of hybrid policies

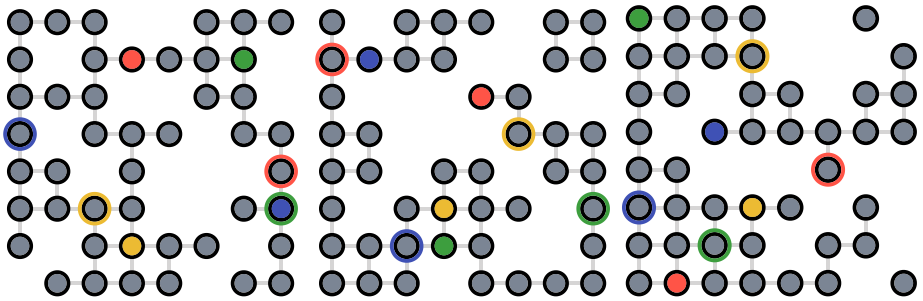


FIGURE 7. Examples of cooperative configs from *id2-coop*8 × 8-*4* configuration. The average cooperative hardness for this set is 6.07.

TABLE 4. Final results for all algorithms on eight configurations for ISR metric. Hybrid policies outperform other algorithms in all cooperative cases. All metrics are averaged over three runs.

Individual Success Rate (±std)						
	Conflict resolution			Goal reaching	Hybrid Policy	
config	QMIX	MCTS	QMIX+MCTS	cPPO	QMIX+cPPO	MCTS+cPPO
id1-coop08x08-02	0.397 (±0.015)	0.377 (±0.027)	0.517 (±0.006)	0.532 (±0.021)	**0.607 (±0.012)**	**0.607 (±0.01)**
id2-coop08x08-04	0.428 (±0.019)	0.307 (±0.004)	0.527 (±0.005)	0.497 (±0.022)	**0.578 (±0.008)**	0.559 (±0.014)
id3-coop16x16-08	0.330 (±0.003)	0.248 (±0.004)	0.358 (±0.004)	0.421 (±0.004)	**0.450 (±0.008)**	0.443 (±0.005)
id4-coop32x32-16	0.220 (±0.002)	0.173 (±0.002)	0.271 (±0.001)	0.364 (±0.004)	0.367 (±0.001)	**0.381 (±0.005)**
id5-rnd08x08-02	0.780 (±0.018)	0.715 (±0.011)	0.855 (±0.007)	**0.893 (±0.008)**	0.883 (±0.002)	0.880 (±0.004)
id6-rnd08x08-04	0.768 (±0.011)	0.655 (±0.011)	0.837 (±0.007)	0.852 (±0.007)	**0.885 (±0.005)**	0.852 (±0.001)
id7-rnd16x16-08	0.565 (±0.011)	0.486 (±0.006)	0.640 (±0.001)	0.718 (±0.004)	**0.735 (±0.007)**	0.730 (±0.003)
id8-rnd32x32-16	0.329 (±0.002)	0.246 (±0.005)	0.416 (±0.003)	0.565 (±0.004)	0.562 (±0.001)	**0.586 (±0.008)**

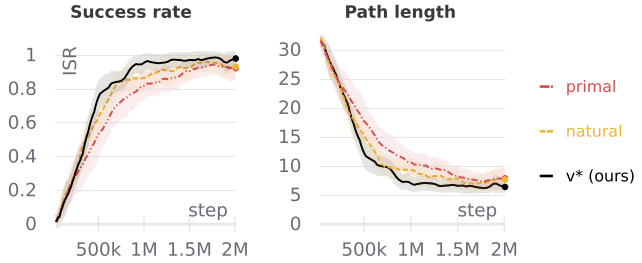


FIGURE 8. Comparison of reward functions with PPO algorithm trained on random 8 × 8 single-agent maps. *v** show better convergence and better path lengths than *PRIMAL* [37] and natural reward functions. Curves are averaged over 10 runs on unseen seeds from the same distribution. Shaded area reports standard deviation.

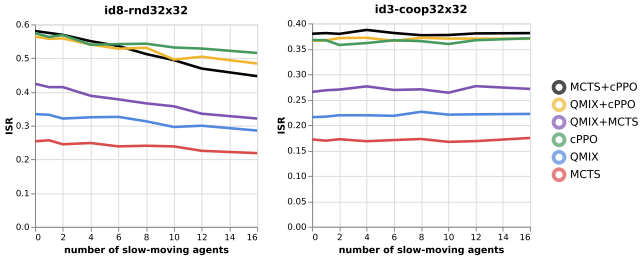


FIGURE 9. The performance of the algorithms in different vehicle velocities. The number of slow-moving agents indicates the number of agents which perform actions two times slower. Hybrid policy outperforms other approaches only for the low number of slow-moving agents on id8-rnd32 × 32 configuration. The case when most of the agents are slowed down is similar to the case when the *Horizon* is halved, which is consistent with the experimental results for different horizons. In the case of id8-coop32 × 32 configuration, increasing the number of slow-moving agents does not affect the results, which shows the dominance of the cooperative component of the environment.

(QMIX + cPPO, MCTS + cPPO), which combine the modules. During testing, we used stochastic versions of the algorithms, so for each configuration, we ran several experiments and averaged them.

We report results for ISR metric in Table 4. Hybrid policies, in all cooperative cases, show significantly better results than any other algorithms. The best of the single algorithms is cPPO, but in comparison, for example, to MCTS + cPPO it outperforms hybrid policy only on one of the eight configurations. Moreover, cPPO outperforms other approaches on *id5-rnd8x8-2* since this configuration doesn't require cooperative interaction at all.

The results for CSR metric are presented in Table 5. Both options of the Hybrid Policy significantly outperforms other approaches on the following configurations: *id1-coop08* × *08-02*, *d2-coop08* × *08-04*, *id6-rnd08* × *08-04*, *id7-rnd16* × *16-08*. None of the algorithms show results greater than 2% in configurations: *d3-coop16* × *16-0*, *id4-coop32* × *32-16*, *id8-rnd32* × *32-16*. As in the case of the ISR metric, cPPO performs significantly better, only in a scenario where cooperative interaction is not required: *id5-rnd8* × *8-2*.

Figure 10 shows the impact of the density, the horizon, the number of agents on the performance of the algorithms. We used *id8-rnd32* × *32* configuration for the all experiments of this figure. Hybrid policies and cPPO succeed in configuration with *density* ∈ {0.0, 0.1}, since these are fairly simple

TABLE 5. Final results for all algorithms on eight configurations for CSR metric. Hybrid policies outperform other algorithms in most cases. All metrics are averaged over three runs.

Cooperative Success Rate (±std)						
	Conflict resolution			Goal reaching	Hybrid Policy	
config	QMIX	MCTS	QMIX+MCTS	cPPO	QMIX+cPPO	MCTS+cPPO
id1-coop08x08-02	0.243 (±0.012)	0.217 (±0.009)	0.323 (±0.012)	0.370 (±0.024)	**0.430 (±0.008)**	0.427 (±0.026)
id2-coop08x08-04	0.020 (±0.008)	0.003 (±0.005)	0.083 (±0.005)	0.103 (±0.024)	0.113 (±0.017)	**0.127 (±0.009)**
id3-coop16x16-08	0.000 (±0.0)	0.000 (±0.0)	0.000 (±0.0)	**0.017** (±0.005)	0.003 (±0.005)	0.003 (±0.005)
id4-coop32x32-16	0.000 (±0.0)	0.000 (±0.0)	0.000 (±0.0)	0.000 (±0.0)	0.000 (±0.0)	0.000 (±0.0)
id5-rnd08x08-02	0.610 (±0.029)	0.537 (±0.009)	0.737 (±0.005)	**0.807 (±0.017)**	0.787 (±0.005)	0.783 (±0.012)
id6-rnd08x08-04	0.357 (±0.029)	0.203 (±0.021)	0.513 (±0.005)	0.590 (±0.014)	**0.647 (±0.017)**	0.593 (±0.009)
id7-rnd16x16-08	0.017 (±0.012)	0.000 (±0.0)	0.043 (±0.009)	0.113 (±0.017)	**0.150 (±0.016)**	0.147 (±0.005)
id8-rnd32x32-16	0.000 (±0.0)	0.000 (±0.0)	0.000 (±0.0)	**0.003 (±0.005)**	0.000 (±0.0)	**0.003 (±0.005)**

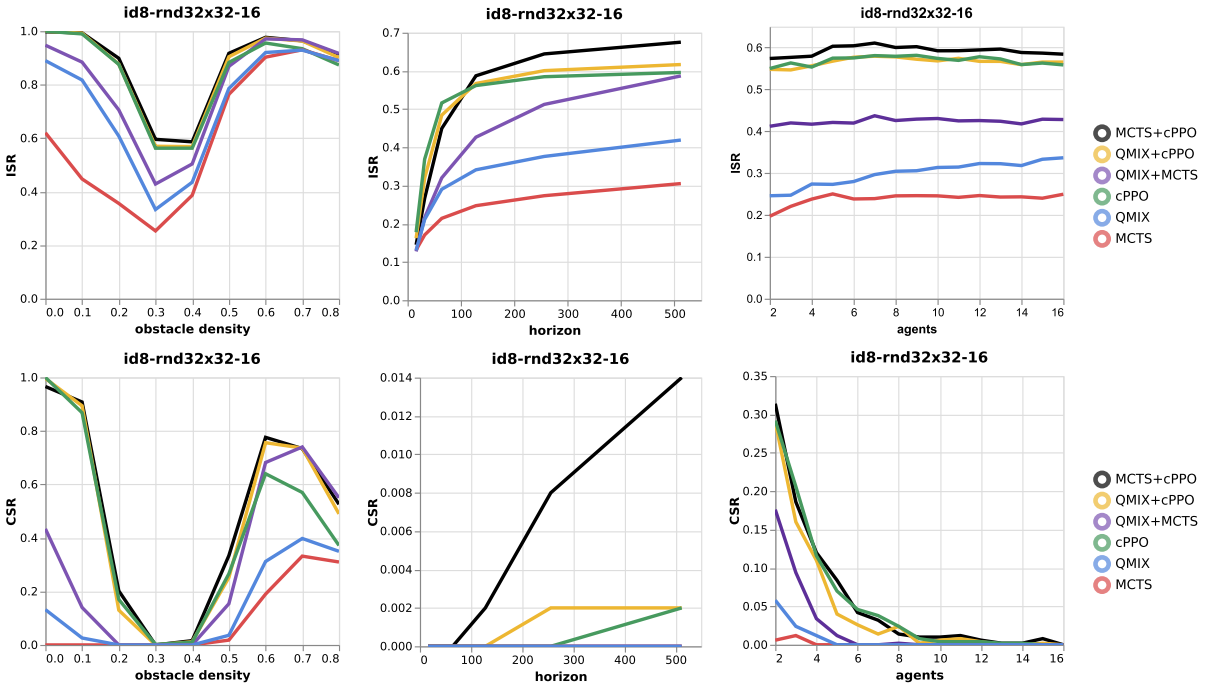


FIGURE 10. Evaluation results for different parameters of id8-rnd32 × 32 configuration. We study how each parameter (density, horizon, and the number of agents) affect the final results. In most cases, MCTS + cPPO shows better performance than other algorithms. The results of each curve are averaged over 5 runs. Obstacle density diagrams show that the most difficult densities are 0.3 and 0.4. The growth of the graph after the density of 0.4 is explained by the fact that with an increase in the number of obstacles, the environment is divided into several components of connectivity, the distance to the targets in which is less. Increasing of the Horizon, as expected, increases the performance of the algorithms. Changing the number of the agents for ISR metric has almost no effect on the results but significantly reduces the CSR metric results.

tasks for the agents. Also, hybrid policies show high performance on the configurations with *density* $\in \{0.6, 07, 0.8\}$. Then the number of obstacles increasing, the environment breaks up into several connected components, reducing the whole problem to solving several independent ones. The split into such small tasks, on the one hand, shortens the path length, and on the other, simplifies the task for the conflict resolution module. It can be seen that the cPPO results fall heavily for the CSR metric. This diagram also shows that the most difficult densities for agents are *density* $\in \{0.3, 0.4\}$.

Increasing the *horizon* parameter, as expected, allows agents to solve more environments. Increasing the number of *agents* has an insignificant effect on the ISR metric, but significantly reduces results for the CSR. Adding a new agent, on the one hand, reduces the chance of completing the whole task, but on the other hand, it increases the chance of getting an agent with a close distance to the target, which increases the result.

We present a comparison of reward function designs in Figure 8. Proposed v^* show faster convergence in comparison with natural (1 for success and 0 otherwise) and reward functions from *PRIMAL* [37]. The main advantages of the function are: It provides a positive reward signal, even if the agent didn't reach the target; It penalizes the agent (in addition to the discount factor) for choosing a non-optimal path.

Additionally, we report the results of experiments for agents with different speeds in Figure 9. We have modified the environment so that some of the agents perform actions

twice as slow. The slow-moving agent needs two steps to move to an adjacent cell. The ISR metric decreases slightly as the number of such agents increases on random configurations. In contrast, adding slow-moving agents in cooperative configurations does not affect the results. It emphasizes the reactive nature of RL algorithms, which allows RL agents not to worsen the performance even in environments with new dynamics.

VII. CONCLUSION AND DISCUSSION

In this work, we considered the challenging navigation problem arising in the context of autonomous vehicles lacking a stable connection to the Internet of Vehicles and having to rely on local observations to arrive to their goals. We suggested a novel method (a hybrid policy) that is based on the combination of two learnable modules: the one which is in charge of generating target-driven behavior and the one which generates cooperative behavior. These modules are based on state-of-art model-based and model-free reinforcement learning algorithms. They compliment each other and the hybrid policy exploiting them, indeed, outperforms individual policies, as results of our comparative empirical evaluation shows.

There are several avenues for future work. One of them is going beyond the 4-connected grid setup. Indeed, 4-connected grids are the most widely used graph models in the context of multi-agent pathfinding, however, it is known that allowing diagonal or any-angle moves results is much shorter and faster trips for the agents. Thus it will be beneficial to study partially-observable multi-agent pathfinding on general graphs. A further step along this line of research is taking the kinematic constraints of the agents (vehicles, robots, etc.) into account. Classical approaches that rely on an explicit model of agents' dynamics are computationally burdensome, while learnable ones may prove to be more efficient and to scale well to a large number of agents.

REFERENCES

[1] X. Chen, C. Wu, T. Chen, H. Zhang, Z. Liu, Y. Zhang, and M. Bennis, "Age of information aware radio resource management in vehicular networks: A proactive deep reinforcement learning perspective," *IEEE Trans. Wireless Commun.*, vol. 19, no. 4, pp. 2268–2281, Apr. 2020.

[2] X. Chen, C. Wu, Z. Liu, N. Zhang, and Y. Ji, "Computation offloading in beyond 5G networks: A distributed learning framework and applications," *IEEE Wireless Commun.*, vol. 28, no. 2, pp. 56–62, Apr. 2021.

[3] H. Zhou, W. Xu, J. Chen, and W. Wang, "Evolutionary V2X technologies toward the Internet of vehicles: Challenges and opportunities," *Proc. IEEE*, vol. 108, no. 2, pp. 308–323, Feb. 2020.

[4] D. R. Aleko and S. Djahel, "An IoT enabled traffic light controllers synchronization method for road traffic congestion mitigation," in *Proc. IEEE Int. Smart Cities Conf. (ISC)*, Oct. 2019, pp. 709–715.

[5] A. Lissac, S. Djahel, and J. Hodgkiss, "Infrastructure assisted automation of lane change manoeuvre for connected and autonomous vehicles," in *Proc. IEEE Int. Smart Cities Conf. (ISC)*, Oct. 2019, pp. 173–180.

[6] Y. Yang, H. Modares, K. G. Vamvoudakis, Y. Yin, and D. C. Wunsch, "Dynamic intermittent feedback design for H_∞ containment control on a directed graph," *IEEE Trans. Cybern.*, vol. 50, no. 8, pp. 3752–3765, Aug. 2019.

[7] R. Stern, NR Sturtevant, A Felner, S Koenig, H. Ma, T. T. Walker, J. Li, D. Atzmon, L. Cohen, T. S. Kumar, and R. Barták, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proc. 12th Annu. Symp. Combinat. Search (SoCS)*, 2019, pp. 151–158.

[8] J. van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in *Proc. IEEE Int. Conf. Robot. Automat.*, May 2008, pp. 1928–1935.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Bradford Books, 2018, p. 552.

[10] C. Wu, Z. Liu, F. Liu, T. Yoshinaga, Y. Ji, and J. Li, "Collaborative learning of communication routes in edge-enabled multi-access vehicular environment," *IEEE Trans. Cognit. Commun. Netw.*, vol. 6, no. 4, pp. 1155–1165, Dec. 2020.

[11] F. Rasheed, K.-L.-A. Yau, R. M. Noor, C. Wu, and Y.-C. Low, "Deep reinforcement learning for traffic signal control: A review," *IEEE Access*, vol. 8, pp. 208016–208044, 2020.

[12] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artif. Intell.*, vol. 101, nos. 1–2, pp. 99–134, May 1998.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*. [Online]. Available: https://arxiv.org/abs/1312.5602

[14] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *Int. J. Robot. Res.*, vol 39, pp. 1–27, Aug. 2018.

[15] A. Skrynnik, A. Staroverov, E. Aitygulov, K. Aksenov, V. Davydov, and A. I. Panov, "Forgetful experience replay in hierarchical reinforcement learning from expert demonstrations," *Knowl.-Based Syst.*, vol. 218, Apr. 2021, Art. no. 106844.

[16] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, "Emergent complexity via multi-agent competition," 2017, *arXiv:1710.03748*. [Online]. Available: https://arxiv.org/abs/1710.03748

[17] X. Xu, T. Huang, P. Wei, A. Narayan, and T.-Y. Leong, "Hierarchial reinforcement learning in StarCraft II with human expertise in subgoals selection," in *Proc. AAAI Conf.*, 2019. [Online]. Available: https://arxiv.org/abs/2008.03444

[18] D. S. Chaplot, D. P. Gandhi, A. Gupta, and R. R. Salakhutdinov, "Object goal navigation using goal-oriented semantic exploration," in *Advances in Neural Information Processing Systems*, vol. 33, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds. Red Hook, NY, USA: Curran Associates, 2020, pp. 4247–4258. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/2c75cf2681788adaca63aa95ae028b22-Paper.pdf

[19] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proc. 34th Int. Conf. Mach. Learn.*, PMLR, vol. 70, 2017, pp. 2778–2787.

[20] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–503, 2016.

[21] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, and T. Lillicrap, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[22] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.

[23] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multiagent path finding," *Artif. Intell. J.*, vol. 218, pp. 40–66, Feb. 2015.

[24] E. Boyarski, A. Felner, R. Stern, G. Sharon, O. Betzalel, D. Tolpin, and E. Shimony, "ICBS: Improved conflict-based search algorithm for multi-agent pathfinding," in *Proc. 24th Int. Joint Conf. Artif. Intell. (IJCAI)*, 2015, pp. 740–746.

[25] A. Felner, J. Li, E. Boyarski, H. Ma, L. Cohen, T. S. Kumar, and S. Koenig, "Adding heuristics to conflict-based search for multi-agent path finding," in *Proc. 28th Int. Conf. Automated Planning Scheduling (ICAPS)*, 2018, pp. 83–87.

[26] K. Yakovlev, A. Andreychuk, and V. Vorobyev, "Prioritized multi-agent path finding for differential drive robots," in *Proc. Eur. Conf. Mobile Robots (ECMR)*, Sep. 2019, pp. 1–6.

[27] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research* (Springer Tracts in Advanced Robotics), vol. 70, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds. Berlin, Germany: Springer, 2011, doi: 10.1007/978-3-642-19457-3_1.

[28] S. Dergachev, K. Yakovlev, and R. Prakapovich, "A combination of theta*, ORCA and push and rotate for multi-agent navigation," in *Interactive Collaborative Robotics* (Lecture Notes in Computer Science), vol. 12336, A. Ronzhin, G. Rigoll, and R. Meshcheryakov, Eds. Cham, Switzerland: Springer, 2020, doi: 10.1007/978-3-030-60337-3_6.

[29] M. S. Kiarostami, M. R. Daneshvaramoli, S. K. Monfared, D. Rahmati, and S. Gorgin, "Multi-agent non-overlapping pathfinding with Monte-Carlo tree search," in *Proc. IEEE Conf. Games (CoG)*, Aug. 2019, pp. 1–4.

[30] S. Choudhury, J. K. Gupta, P. Morales, and M. J. Kochenderfer, "Scalable anytime planning for multi-agent MDPs," in *Proc. 20th Int. Conf. Auton. Agents MultiAgent Syst. (AAMAS)*, May 2021, pp. 341–349.

[31] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1008–1014.

[32] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, vol. 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/68a9750337a418a86fe06c1991a1d64c-Paper.pdf

[33] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Autonomous Agents and Multiagent Systems* (Lecture Notes in Computer Science), vol. 10642, G. Sukthankar and J. Rodriguez-Aguilar, Eds. Cham, Switzerland: Springer, 2017, doi: 10.1007/978-3-319-71682-4_5.

[34] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning," in *Proc. 17th Int. Conf. Auton. Agents MultiAgent Syst. (AAMAS)*, Jul. 2018, pp. 2085–2087.

[35] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. N. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *Proc. ICML*, 2018, pp. 4292–4301.

[36] W. Böhmer, V. Kurin, and S. Whiteson, "Deep coordination graphs," in *Proc. 37th Int. Conf. Mach. Learn.*, PMLR, vol. 119, 2020, pp. 980–991.

[37] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. S. Kumar, S. Koenig, and H. Choset, "PRIMAL: Pathfinding via reinforcement and imitation multi-Agent learning," *IEEE Robot. Autom. Lett.*, vol. 4, no. 3, pp. 2378–2385, Jul. 2019.

[38] R. Stern, "Multi-agent path finding—An overview," in *Artificial Intelligence* (Lecture Notes in Computer Science), vol. 11866, G. Osipov, A. Panov, and K. Yakovlev, Eds. Cham, Switzerland: Springer, 2019, doi: 10.1007/978-3-030-33274-7_6.

[39] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, and R. Józefowicz, "Dota 2 with large scale deep reinforcement learning," 2019, *arXiv:1912.06680*. [Online]. Available: https://arxiv.org/abs/1912.06680

[40] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging procedural generation to benchmark reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 2048–2056.

[41] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017, *arXiv:1712.01815*. [Online]. Available: https://arxiv.org/abs/1712.01815

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: http://arxiv.org/abs/1707.06347

[43] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015, *arXiv:1506.02438*. [Online]. Available: https://arxiv.org/abs/1506.02438

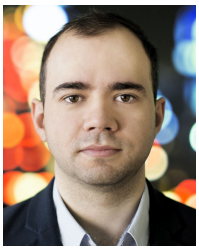
[44] C.-S. Lee, M.-H. Wang, G. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong, "The computational intelligence of MoGo revealed in Taiwan's computer go tournaments," *IEEE Trans. Comput. Intell. AI Games*, vol. 1, no. 1, pp. 73–89, Mar. 2009.

[45] M. Enzenberger, M. Muller, B. Arneson, and R. Segal, "Fuego—An open-source framework for board games and go engine based on Monte Carlo tree search," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 4, pp. 259–270, Dec. 2010.

[46] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Proc. 17th Eur. Conf. Mach. Learn.* Berlin, Germany: Springer-Verlag, 2006, pp. 282–293.

[47] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. (2019). *Stable Baselines3*. [Online]. Available: https://github.com/DLR-RM/stable-baselines3

[48] M. Samvelyan, T. Rashid, C. S. de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C.-M. Hung, P. H. S. Torr, J. Foerster, and S. Whiteson, "The StarCraft multi-agent challenge," 2019, *arXiv:1902.04043*. [Online]. Available: https://arxiv.org/abs/1902.04043



ALEXEY SKRYNNIK received the M.S. degree in computer science from Rybinsk State Aviation Technical University, Rybinsk, Russia, in 2017. He is currently pursuing the Ph.D. degree with the Federal Research Center "Computer Science and Control," Artificial Intelligence Research Institute, Russian Academy of Sciences, under the supervision of A. I. Panov.

Since 2018, he has been a Researcher with the Federal Research Center "Computer Science and Control," Artificial Intelligence Research Institute, Russian Academy of Sciences. His current research interests include reinforcement learning, learning and planning, and machine learning.

ALEXANDRA YAKOVLEVA received the B.S. degree in applied mathematics and physics from Moscow Institute of Physics and Technology (MIPT), Moscow, Russia, in 2020, where she is currently pursuing the M.S. degree.

Since 2020, she has been a Research Engineer in multi-agent reinforcement learning with the Cognitive Dynamic System Laboratory, MIPT.

VASILII DAVYDOV received the M.S. degree in fundamental informatics and information technologies from Moscow Aviation Institute, Moscow, Russia, in 2021.

Since 2020, he has been a Research Engineer with the Cognitive Dynamic System Laboratory, Moscow Institute of Physics and Technology.

KONSTANTIN YAKOVLEV received the Ph.D. degree in computer science from the Institute for Systems Analysis, Russian Academy of Sciences, Moscow, Russia, in 2010.

He is currently a Leading Researcher with the Federal Research Center "Computer Science and Control," Russian Academy of Sciences, and also affiliated with Moscow Institute of Physics and Technology and HSE University. His research interests include heuristic search, single and multi-agent pathfinding, motion planning, multi-agent systems, and robotics.

ALEKSANDR I. PANOV received the M.S. degree in computer science from Moscow Institute of Physics and Technology, Moscow, Russia, in 2011, and the Ph.D. degree in theoretical computer science from the Institute for Systems Analysis, Moscow, in 2015.

Since 2010, he has been a Research Fellow with the Federal Research Center "Computer Science and Control," Russian Academy of Sciences. Since 2018, he has been the Head of the Cognitive Dynamic System Laboratory, Moscow Institute of Physics and Technology. He is the author of three books and more than 90 articles. His research interests include behavior planning, reinforcement learning, semiotics, and robotics.