Scaling Team Coordination on Graphs with Reinforcement Learning

Manshi Limbu, Zechen Hu, Xuan Wang, Daigo Shishika[†], and Xuesu Xiao[†]

***Abstract*— This paper studies Reinforcement Learning (RL) techniques to enable team coordination behaviors in graph environments with support actions among teammates to reduce the costs of traversing certain risky edges in a centralized manner. While classical approaches can solve this non-standard multi-agent path planning problem by converting the original Environment Graph (EG) into a Joint State Graph (JSG) to implicitly incorporate the support actions, those methods do not scale well to large graphs and teams. To address this curse of dimensionality, we propose to use RL to enable agents to *learn* such graph traversal and teammate supporting behaviors in a data-driven manner. Specifically, through a new formulation of the team coordination on graphs with risky edges problem into Markov Decision Processes (MDPs) with a novel state and action space, we investigate how RL can solve it in two paradigms: First, we use RL for a team of agents to learn how to coordinate and reach the goal with minimal cost on a single EG. We show that RL efficiently solves problems with up to 20/4 or 25/3 nodes/agents, using a fraction of the time needed for JSG to solve such complex problems; Second, we learn a general RL policy for any N-node EGs to produce efficient supporting behaviors. We present extensive experiments and compare our RL approaches against their classical counterparts.**

I. Introduction

Multi-robot systems have been studied with the premise of increased efficacy using many low-capability robots as opposed to a small number of high-capability robots. In such a setting, the coordination between low-capability teammates is essential to achieve the whole team's high efficacy. This paper is interested in a scenario where a team of robots cooperatively traverse a challenging environment by "supporting" each other. Support can take the form of, for example, providing a different vantage point for better situational awareness, or physically interacting with the environment to reduce risk (e.g., holding a ladder). We abstract these notions to the actions that can be taken on a graph environment to study how multi-robot teams can efficiently traverse an environment when such cooperation is possible.

Team coordination on graphs with state-dependent edge cost [1] is a recently proposed problem, in which a team of agents move on an Environment Graph (EG) and provide support actions for teammates to reduce the cost to traverse certain risky edges with the goal of achieving minimal traversal cost for the whole team to reach the goal(s). Prior methods convert the EG with risky edges (whose traversal cost depends on whether a teammate is supporting the traversal from a certain support node) into a Joint State Graph (JSG) which implicitly incorporates the support actions and then applies graph-search algorithms to find the minimal cost path on the JSG. However, JSG does not scale well to larger graphs and team sizes, while its extended version Critical Joint State Graph (CJSG) can only efficiently handle two agents on graphs where the ratio between risky edge and normal edge is low.

[†]Equally advising authors, George Mason University. `{klimbu2, zhu3, xwang64, dshishik, xiao}@gmu.edu`. Work supported by ARL Tactical Behaviors for Autonomous Maneuver (TBAM) CRA. This work has taken place in the RobotiXX Laboratory at GMU. RobotiXX research is supported by ARO (W911NF2220242, W911NF2320004, W911NF2420027), AFCENT, Google DeepMind, Clearpath Robotics, and Raytheon Technologies.

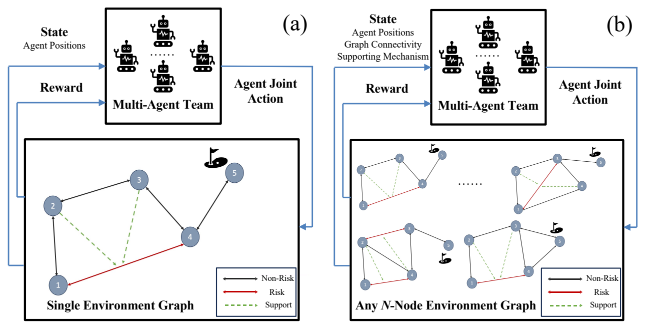

Fig. 1: Team coordination with reinforcement learning on a single graph (a) and on multiple graphs (b) with risky edges and supporting behaviors to reduce risk.

Reinforcement Learning (RL) has the potential to allow agents to learn from trial-and-error experiences by exploring on the graph. It has the potential to generate coordination behaviors on the EG without constructing JSG or CJSG and searching on those large-scale, densely connected graphs. In this paper, we investigate how the original centralized problem of team coordination on graphs with risky edges can be converted into Markov Decision Processes (MDPs) with graph-dependent state and action spaces and then be solved by state-of-the-art RL techniques. To be specific, we are interested in using RL to solve the team coordination problem in two paradigms. In the first paradigm (Fig. 1 (a)), RL is used to solve the team coordination on one single EG with more than two agents. Our experiment results show that the total time including training and inference of the RL policy can be faster than the time used by the JSG approach, including JSG construction and search time, in complex problems with many agents and nodes. RL can also extend to more than two agents, which CJSG cannot solve; In the second paradigm (Fig. 1 (b)), we use RL to solve any N-node graph with a novel formulation of the agent state, graph connectivity, and supporting mechanism encoded in the state space. Our experiment results show that a general RL policy can be learned for any graph and support structure up to ten nodes to produce efficient supporting behaviors. To be best of our knowledge, this work is the first to utilize RL to solve a multi-robot coordination problem on *graphs*.

II. RELATED WORK

We review related work on multi-agent systems and multi-agent reinforcement learning techniques.

A. Multi-Agent Systems

Multi-agent systems have received tremendous attention from different fields, ranging from robotics to sensor network [2], to execute a variety of tasks, e.g., environment sampling [3], search and rescue [4]–[6], and surveillance [7]. While enjoying the benefits of accomplishing a complex task with a team of agents, challenges arise correspondingly. For example, task allocation [8], consensus and formation control [9], [10], collision-avoidance [11], [12], and communication and synchronization [13] are problems that do not exist in single-agent scenarios.

Researchers have studied varying levels of "teaming". Earlier works have focused more on the scalability but with less concern on the inter-agent dependency leading to a simple "divide and conquer" approach [7], [14]. Some more recent works have considered how heterogeneity improves the team performance, including robot teams with heterogeneous capabilities and/or heterogeneous policies (i.e., specialized roles) that arise in teaming behavior [15]. In this paper, we are interested in considering a scenario where the team members may dynamically take different roles to cooperatively perform a task, i.e., through coordination.

B. Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) [16] is an extension of traditional RL that addresses settings with multiple agents that operate either cooperatively, competitively, or coexist in mixed settings. MARL algorithms can generally be categorized based on their learning paradigms—centralized, decentralized, and hybrid approaches [16]. In centralized learning, the training process has access to the full states and actions of all agents, enabling a comprehensive learning framework. In contrast, decentralized learning limits each agent to its local observations, making it more suitable for scenarios where global state information is either not available or not practical to use. Based on the approach to solve the learning problem, MARL is also categorized into value-based methods like QMIX [17] and VDN [18], and policy-based methods such as MAPPO [19], MADDPG [20], and MAA2C [21]. Many of these techniques are extended and adapted from single-agent RL algorithms like Q-Learning [22] and PPO [23], and some integrate both value and policy-based strategies like Actor-Critic methods.

In terms of using MARL to facilitate agent coordination, approaches like DICG [24] and DCG [25] use graphs as a tool to represent the interaction topology between agents, but not as the environment iteself. In these settings, each node in the graph represents an agent and edges represent communication or influence pathways between them. Agents learn to coordinate their actions either through explicit message-passing along the graph structures (in the case of DCG) or through implicit coordination that leverages the graph topology (as in DICG).

In cases where the environment can be efficiently represented as a graph [1], these conventional MARL methods lack specialized mechanisms to exploit the graph topology or the node-edge relationships. This research gap is what this work aims to fill. In a centralized manner, we convert the original problem of team coordination on graphs into the form of an MDP to present not only agent states, but also the states of the environment around the agents. We show RL has the potential to solve these problems more efficiently when more agents and larger graphs are of interest, comparing to state-of-the-art classical methods.

III. PROBLEM FORMULATION

We first present our original problem formulation of team coordination on graphs with state-dependent edge costs [1], which, in this work, is converted into a novel MDP formulation with a new state and action space, along with a state transition function. We also design a reward function based on the traversal cost to enable reinforcement learning to efficiently solve this problem in two different settings: learning to solve for a single graph and learning to solve for multiple graphs with the same number of nodes.

A. Team Coordination on Graphs

A team of N agents travel on a strongly connected Environment Graph (EG) denoted by $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ is a set of nodes, and $\mathbb{E}$ is a set of edges, $\mathbb{E} \subset \mathbb{V} \times \mathbb{V}$, from a start node set $\mathbb{V}_0 \subset \mathbb{V}$ to reach a goal node set $\mathbb{V}_g \subset \mathbb{V}$ while minimizing the cost of traversal. The nominal cost for traversing the edge $e_{i,j} \in \mathbb{E}$ is given as a constant, $c_{i,j}$ for $i, j \in \mathbb{V}$, when traveling on that edge without support. On the EG, each edge $e_{i,j}$ is associated with a set of support nodes, $\mathcal{Z}_{i,j} \subseteq \mathbb{V}$. If this set is non-empty, an agent at $v \in \mathcal{Z}_{i,j}$ can provide support for the agent traversing $e_{i,j}$ and reduce the nominal traversal cost $c_{i,j}$ to $\tilde{c}_{i,j}$. The action set for any agent at node i is given as $A_i = \{\{a_{i,j}\}_{j \in \mathcal{N}_i}, a_s\}$, where $\mathcal{N}_i$ is the neighborhood of i, and $a_{i,j}$ is the action to move to node j given that $j \in \mathcal{N}_i$. The action a_s is the support while inducing an extra support cost $\tilde{c}$.

For agent $n \in \mathbb{N}$, denote its action sequence and node visiting sequence as $\mathcal{A}^n = \{a_i^n\}_{i=0}^{T-1}$ and $\mathcal{V}^n = \{v_i^n\}_{i=0}^{T}$ respectively, where T is the total time step to reach the goal, $v_0^n \in \mathbb{V}_0$ and $v_T^n \in \mathbb{V}_g$ are the start and goal node, and $a_i^n \in A_{v_i^n}$. In general, the cost of each action taken by an agent $n \in \mathbb{N}$ at each step i is a function $C^n(\cdot)$ of the positions and actions of all agents at i: $c_i^n = C^n(\{v_i^j, a_i^j\}_{j \in \mathbb{N}})$. The goal of the team is to find the team action sequences $\{\mathcal{A}^j\}_{j \in \mathbb{N}}$ in order to minimize the accumulated cost of the entire team along the entire traversal to the goal:

$$\min_{\{\mathcal{A}^j\}_{j \in \mathbb{N}}} \sum_{n \in \mathbb{N}} \sum_{i=0}^{T-1} c_i^n. \tag{1}$$

B. MDP Formulation

We convert our original problem into a MDP formulation with a novel state and action space, along with a state transition and reward function.

1) State Space: A Markovian state needs to incorporate all necessary information along with the action to determine the next state and current reward. Part of our new state S includes all individual agent positions at each time step t, i.e., $\{v_t^n\}_{n\in\mathbb{N}}$, which is represented as a one-hot vector for each agent and therefore the joint agent state has dimensionality $|\mathbb{V}| \cdot |\mathbb{N}|$, i.e., $P_{|\mathbb{V}|\cdot|\mathbb{N}|\times 1} \in \mathbb{P}_{|\mathbb{V}|\cdot|\mathbb{N}|\times 1}$. Our state also needs to consider the EG to be Markovian. Therefore, we include the graph connectivity and supporting mechanism as part of S. We use the adjacency matrix of the EG to encode graph connectivity, i.e., a $|\mathbb{V}| \times |\mathbb{V}|$ matrix $\text{ADJ}_{|\mathbb{V}|\times|\mathbb{V}|} \in \mathbb{ADJ}_{|\mathbb{V}|\times|\mathbb{V}|}$, with each entry $\text{adj}_{i,j}$ denoting the nominal traversal cost between node i and j or set as ∞ if the edge $e_{i,j}$ does not exist. We also include as part of S the supporting mechanism as a supporting tensor $\text{SUP}_{|\mathbb{V}|\times|\mathbb{V}|\times|\mathbb{V}|} \in \mathbb{SUP}_{|\mathbb{V}|\times|\mathbb{V}|\times|\mathbb{V}|}$. Each entry $\text{sup}_{i,j,k}$ denotes the reduced traversal cost between node j and k, if there is an agent taking support action from node i. $\text{sup}_{i,j,k}$ remains the nominal cost of $e_{j,k}$ if support is not possible from i for $e_{j,k}$. Therefore, our new state space is defined as

$$\mathbb{S} := \mathbb{P}_{|\mathbb{V}|\cdot|\mathbb{N}|\times 1} \times \mathbb{ADJ}_{|\mathbb{V}|\times|\mathbb{V}|} \times \mathbb{SUP}_{|\mathbb{V}|\times|\mathbb{V}|\times|\mathbb{V}|}. \quad (2)$$

2) Action Space: Each agent is able to take the action of moving to any neighboring nodes (including staying at the current node and inducing zero cost) or supporting. To assure the action space for each agent has the same dimensionality across all states in our state space $\mathbb{S}$, we define the action for agent n to be a one-hot vector $a^n_{(|\mathbb{V}|+1)\times 1} \in A_{(|\mathbb{V}|+1)\times 1}$, denoting which node the agent moves to (or stays at the current node). The last dimension of $a^n_{(|\mathbb{V}|+1)\times 1}$ denotes the agent is taking the supporting action. In a centralized manner, the joint action space of the entire team can be defined as

$$\mathbb{A} := A^1 \times A^2, ..., \times A^N. \quad (3)$$

Considering that it is impossible to move from some node to others if there is no edge connecting them, we employ invalid action masking for those cases (details in Sec. IV).

3) Reward Function: Based on the defined state space (Eqn. (2)) and action space (Eqn. (3)), we define our reward function based on the negative cost induced by the entire team:

$$R_t := \mathbb{R}(S_t, A_t) = -\sum_{n\in\mathbb{N}} c_t^n, \quad (4)$$

along with a high reward when all agents reach the goal(s). In order to encourage the team to quickly reach the goal, in addition to this original reward, we also provide reward shaping (details in Sec. IV).

4) State Transition Function: Since our problem is formulated on graphs, the centralized state transition function $S_{t+1} \sim \mathcal{T}(\cdot|S_t, A_t)$ follows the graph adjacency matrix $\text{ADJ}_{|\mathbb{V}|\times|\mathbb{V}|}$, which is also part of the state space. While in our current implementation we simplify the state transition function as a deterministic function, we leave the formulation general enough to account for future nondeterministic cases, e.g., taking the action to move from node i to j has a non-zero probability of staying at i or moving to another node.

5) Full MDP: The full MDP is therefore formulated as a tuple, $(\mathbb{S}, \mathbb{A}, \mathcal{T}, \gamma, \mathbb{R})$, with γ as a discount factor which we set to 0.95 in our implementation. The overall object of RL is to learn a policy $\pi : \mathbb{S} \rightarrow \mathbb{A}$ that can be used to select team actions in a centralized manner to maximize the expected cumulative reward over time, i.e.,

$$J = \mathbb{E}_{(S_t,A_t)\sim\pi}[\sum_{t=0}^{\infty} \gamma^t R_t]. \quad (5)$$

C. Reinforcement Learning for Single and Multiple EG*(s)*

In this work, we use different RL algorithms, i.e., Q-Learning [26] and PPO [23], to optimize the expected cumulative reward (Eqn. (5)) in two different settings: learning for a single EG and learning for multiple N-node EGs.

1) Single EG*:* We start with a simplified version of the state space (Eqn. (2)) and remove the adjacency matrix and supporting tensor, i.e., $\mathbb{S} := \mathbb{P}_{|\mathbb{V}|\cdot|\mathbb{N}|\times 1}$. In this setting, we only aim at learning a policy π that works on one EG with a pre-defined supporting mechanism, so that the policy learns what is the optimal team joint action for every team joint state in the simplified state space. Using RL, our goal is to achieve faster solution time, including both RL training and inference, compared to the original JSG method [1], i.e., JSG construction and shortest-path search time. We also aim at extending to teams with more than two agents, which is the maximal number of agents CJSG can efficiently solve. Furthermore, the CJSG solution time will significantly increase when the ratio between the number of risky edges and the total number of edges is large, because support graph then becomes dense and therefore CJSG essentially becomes JSG. While it is counter-intuitive to expect RL to outperform search-based JSG and CJSG, we hypothesize that RL has the potential to address large-scale problems with many nodes and agents and high risky edge ratio.

2) Multiple EG*s:* Second, we also use RL to solve the full MDP. In this setting, a policy is learned to solve any EG with any supporting mechanism given a pre-defined node number N. The full state space (Eqn. (2)) can be divided into different disconnected subspaces (imagine no matter what actions the team take, they cannot be teleportated from one EG to another). While the training time for such a policy can be much longer, the learned policy can be reused when a new N-node EG with a new supporting mechanism is encountered. For this setting, we compare the RL inference time to solve the problem against their classical counterparts.

IV. IMPLEMENTATIONS

Based on our MDP problem formation of the original team coordination on graphs problem, we present implementation details on the RL algorithms, reward shaping, and invalid action masking.

A. RL Implementation

We employ two distinct RL techniques, Q-Learning [26] and Proximal Policy Optimization (PPO) [23], to tackle the multi-agent team coordination problem. To accelerate the learning process, we conduct reward shaping and integrate invalid action masking for both algorithms.

1) Q-Learning: We first use Q-Learning, a value based method, to solve our problem formulated as an MDP. In our centralized problem formulation, a global Q-table is used to maintain the Q-values for all joint states and actions for all agents. Actions are selected by alternating between exploration and exploitation using an ϵ-greedy policy. Both state transitions and rewards are managed globally. We update our Q-function $Q(s, a)$ based on the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R^*(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right], \tag{6}$$

where $R^*(s, a)$ represents a shaped reward detailed in Sec. IV-B, α is the learning rate, and γ is the discount factor.

Q-Learning encounters scalability issues when facing larger problems involving more agents and nodes, as the extensive Q-table does not fit within memory constraints. As the problem size increases, Q-Learning's performance begins to degrade and eventually fails to solve very large problems.

2) PPO: To address such scalability challenges, we also implement an on-policy method, PPO. Like its Q-Learning counterpart, our PPO model is also centralized, but it employs a multi-discrete, one-hot encoded representation for the state space defined in Eqn. (2). Actions for multiple agents are taken from a multi-discrete action space, facilitating simultaneous actions of all agents as defined in Eqn. (3). An important component of our PPO implementation is the modified clipped loss function, formalized as follows:

$$\begin{aligned} L_{\text{clip}} = \mathbb{E}[\min \left(\frac{\pi(a|s)}{\pi_{\text{old}}(a|s)} A^*(s,a), \right. \\ \left. \text{clip}\left(\frac{\pi(a|s)}{\pi_{\text{old}}(a|s)}, 1-\epsilon, 1+\epsilon \right) A^*(s,a) \right)], \end{aligned} \tag{7}$$

where $\frac{\pi(a|s)}{\pi_{\text{old}}(a|s)}$ is the likelihood ratio that compares how likely the current policy $\pi(a|s)$ scores an action a given a state s to how likely the old policy $\pi_{\text{old}}(a|s)$ scores the same action, clip$(\cdot, 1-\epsilon, 1+\epsilon)$ is a clipping function that limits the value within the range $[1-\epsilon, 1+\epsilon]$, and the advantage function $A^*(s, a)$ incorporates the shaped reward $R^*(s, a)$ (Sec. IV-B) to efficiently guide policy optimization.

B. Reward Shaping

The original problem formulation for team coordination on graphs is to minimize the entire team's traversal cost on the way to the goal(s) as shown in Eqn. (1), which needs to be converted into a reward function and a return as shown in Eqn. (4) and (5) for RL respectively. We also shape our reward function to encourage the agents to explore different coordination options to potentially reduce cost and improve reward. Our reward function includes the following terms:

a) Goal Reward:

$$r_g = \begin{cases} +10, & \text{if all agents arrive at goal(s),} \\ -0.01, & \text{otherwise.} \end{cases}$$

This reward term assigns +10 when all agents reaching their goal node(s) on the graph. The episode will then be terminated. To encourage moving to the goal fast, every other time step will be penalized by a -0.01 reward.

b) Movement Reward:

$$r_m = -\sum_{n \in \mathbb{N}} c_t^n = -\sum_{n \in \mathbb{N}} C^n(\{v_t^j, a_t^j\}_{j \in \mathbb{N}}).$$

This reward is computed by summing up and negating the individual costs incurred by all the agents taking one action at one time step t, including incurring regular, risky, or reduced cost when traversing the edge, providing support, and do nothing. In our experiments, normal edge traversal cost is around 1, while risky edges cost around 2, which can be reduced to around 0.5 when being supported.

c) Coordination Reward:

$$r_c = \alpha \times \text{CC} - \beta \times \text{RC},$$

where CC and RC denotes the total count of coordination (providing support to reduce risk) and the total count of an agent traversing a risky edge without support with $\alpha = 2$ and $\beta = 5$. This reward is additional to the original cost function in Eqn. (1), since coordination is not absolutely necessary and traversing risky edges without support does not have to be avoided, if the total traversal cost can be kept low. But we find that in practice it helps to encourage the agents to explore different coordination strategies to eventually reduce traversal cost or improve accumulated reward overall.

d) Final Reward: The final reward is therefore a weighted sum of all aforementioned reward terms:

$$r = w_1 r_g + w_2 r_m + w_3 r_c,$$

where w_1, w_2, and w_3 are weights for the reward terms and set to 1, 1, and 0.2, based on empirical results.

C. Invalid Action Masking

In our MDP formulation, we represent the individual agent's action as a one-hot vector, indicating the node the agent moves to. However, such an action definition inherently includes numerous invalid actions for nodes that are not directly connected to the agent's current position. Given that valid actions are defined as moving to neighboring nodes or supporting, we implement invalid action masking [27] to restrict permissible actions based on the graph structure at different nodes. This approach excludes invalid actions from consideration during the decision-making process.

In the case of Q-Learning, invalid actions are masked and only valid actions are considered when exploiting Q-values and when exploring the action space randomly. We design a mask function $m = M(s, a) \in \{0, 1\}$, returning 1 if action a is valid and 0 if invalid in a given state s. Note that in a centralized manner, any invalid action from any agent in the team will cause the total action to be invalid. During exploration, an action a is selected according to an ϵ-greedy policy based on the Q-values and the mask function $M(s, a)$:

TABLE I: Solution time for 2, 3, and 4 agents in JSG, Q-Learning and PPO respectively.

Graph	Nodes	2 Agents			3 Agents			4 Agents		
		JSG	Q-Learning	PPO	JSG	Q-Learning	PPO	JSG	Q-Learning	PPO
Sparse	5	**0.001**	1.228	58.39	**0.037**	2.863	83.66	**1.093**	9.978	88.74
	10	**0.014**	3.654	81.39	**1.494**	10.35	226.2	157.7	**102.7**	355.8
	15	**0.057**	5.922	201.02	**14.88**	27.88	326.4	3652	–	**962.2**
	20	**0.172**	13.86	560.3	80.16	**45.31**	701.5	–	–	**1045**
	25	**0.394**	–	730.5	**281.0**	–	1432	–	–	–
Moderate	5	**0.002**	0.293	56.97	**0.052**	3.469	74.03	**1.689**	14.34	88.21
	10	**0.022**	2.362	66.17	**3.007**	20.36	146.3	600.5	751.0	**352.2**
	15	**0.088**	2.389	189.6	25.49	**22.79**	317.7	9492	–	**949.4**
	20	**0.277**	3.587	531.0	160.04	**58.26**	683.3	–	–	**1032**
	25	**0.641**	5.720	677.5	571.1	**181.8**	1372	–	–	–
Dense	5	**0.002**	0.874	57.32	**0.072**	1.855	72.44	**0.072**	6.921	89.71
	10	**0.035**	1.963	64.35	**7.927**	15.11	142.3	4312	696.9	**344.8**
	15	**0.109**	6.671	186.4	**39.49**	129.1	317.7	46455	–	**944.1**
	20	**0.433**	2.616	646.2	481.4	**65.22**	677.5	–	–	**1018**
	25	**0.915**	5.192	700.9	1660	7821	**1349**	–	–	–

$$a = \begin{cases} \text{argmax}_{a'}(Q(s,a') \times M(s,a')), & \text{with prob. } 1-\epsilon, \\ \text{random action } a', \text{ s.t. } M(s,a')=1, & \text{with prob. } \epsilon. \end{cases} \tag{8}$$

Here, the first case picks the valid action that maximizes the masked Q-value with probability $1-\epsilon$, while the second case picks a random valid action (according to $M(s,a')$) with probability ϵ.

In PPO, the policy's action selection mechanism is refined by modifying action log probabilities to account for the validity of actions in the current state. This is achieved by applying a mask to action probabilities:

$$\pi(a \mid s) = \frac{\exp(\log \pi(a \mid s))}{\sum_{a', \text{ s.t. } M(s,a')=1} \exp(\log \pi(a' \mid s))}, \tag{9}$$

Here, action probabilities, $\pi(a \mid s)$, are recalculated to consider only valid actions (where $M(s,a')=1$) as determined by the mask function $M(s,a)$ effectively eliminating the chance of selecting invalid actions.

We observe that reward shaping (Sec. IV-B) and invalid action masking enhance sample efficiency, reduce training time and accelerate convergence for both Q-learning and PPO.

V. Results

With our new problem formulation and implementation, we conduct extensive experiments to study how RL can efficiently enable team coordination on graphs with risky edges. We present experiment results of using RL to solve both single and multiple EG(s).

A. *RL for Single* EG

The first set of experiments is to use RL to solve one single EG. Despite that the classical JSG approach can be efficient in solving simple coordination problems on small graphs with a small number of agents, we hypothesize that RL has the potential to solve complex problems faster. Note that since CJSG can only address problems with up to two agents, and our goal is to extend to more than two, we do not include CJSG in our comparison.

Specifically, we experiment with 5, 10, 15, 20, and 25 nodes with 2, 3, and 4 agents in three types of graph connectivity, i.e., sparse, moderate, and dense, using JSG, Q-Learning, and PPO. We randomly create 15 EGs as our test set. All experiment results are presented in Tab. I. For both RL approaches, training is terminated when the cumulative reward no longer changes more than 0.2 for 500 steps.

The results in Tab. I shows that JSG outperforms both Q-Learning and PPO in all two-agent cases, indicating that the cost to construct a JSG for two agents and search on such a graph is minimal even with up to 25 nodes. However, RL's superiority starts to show when the number of agents and nodes and graph connectivity start to increase. For three agents, Q-Learning starts to outperform JSG in 20-node sparse graphs, while it completely overtakes JSG for moderate graphs with more than 15 nodes. Q-Learning and PPO outperform JSG on 20-node and 25-node dense graphs respectively. In most cases with four agents, PPO (and Q-Learning) is the fastest, except for small 5-node graphs. Notice that "–" denotes that the algorithm fails to find a solution, i.e., JSG runs out of memory during graph construction, Q-Learning's Q-table becomes intractably large, or PPO does not converge. All three methods fail to produce a solution for the most difficult case at the lower right of Tab. I, i.e., four agents on 25-node dense graphs, while PPO is the only one that can solve for four agents on 20-node dense graphs.

For solving one EG, while RL cannot guarantee optimality, we observe that in most cases RL can achieve optimal solutions, and near-optimal ones in others. We use the four agents in 10- and 15-node graphs as an example and present the optimality vs. time plots in Fig. 2. We also implement a naive approach, in which all agents do not seek coordination, but just move towards the goal with the minimal cost path.

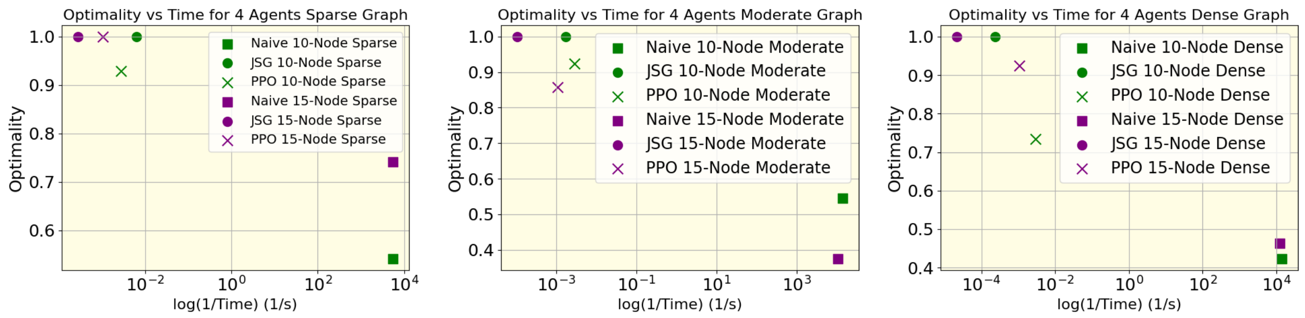

Fig. 2: Single EG: Optimality vs. Time plots for four agents on sparse, moderate, and dense graphs.

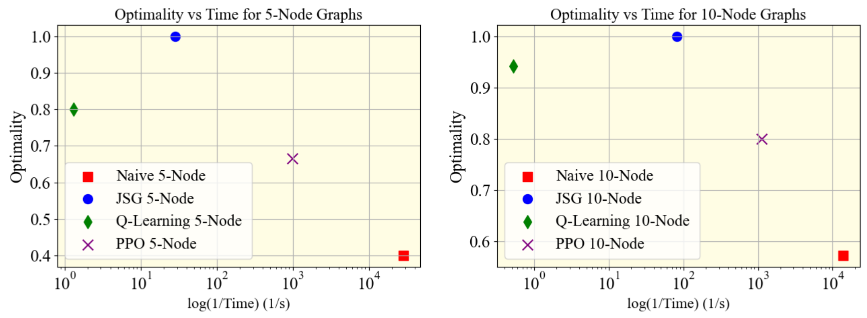

Fig. 3: Multiple EG: Optimality vs. Time plots for two agents on any 5- or 10-node graphs.

The naive approach is very efficient in terms of time, since it only needs to call a shortest-path algorithm once and the solutions for all agents remain the same, but without coordination the naive approach will incur a large cost. On the other hand, JSG is provably optimal [1], so we use the ratio between a solution's cost and the optimal JSG cost as an indication of solution optimality. In Fig. 2, the horizontal axis is the reciprocal of the solution time in log scale, while the vertical axis is the optimality value between 0 and 1. Naive approach is expected to appear in the lower right corner by achieving very short solution time but very low path quality, whereas JSG should be in the upper left corner with optimality value 1 and a very long solution time. For four agents on sparse 10-node graphs, PPO does not produce any advantage over JSG, by achieving lower optimality with longer time. But for sparse 15-node graphs, PPO outperforms JSG by achieving optimality with better time efficiency. For moderate 10- and 15-node graphs, PPO achieves a middle ground in terms of both optimality and time between JSG and the naive approach. We observe the same optimality-time trend in dense graphs of 10 and 15 nodes. In all scenarios, the worst optimality ratio PPO can achieve is more than 70% of JSG's absolute optimality value, but mostly with better time efficiency than JSG.

B. *RL for Multiple* EG

The second set of experiments is to use RL to solve any N-node graphs. This is a very difficult task considering the variations in graph connectivities and supporting mechanisms with a large number of nodes. Therefore, we only limit our experiments within up to 10 nodes and 2 agents and leave extending to more complex problems to future work, potentially with decentralized approaches. Notice that when RL converges to a good policy, despite the long time it may require, this policy can then be used as an available tool to solve any team coordination problem on any N-node graph in the future. In the second set of experiments, training usually takes hours, which is considered as a one-time cost. Once trained, we compare RL's inference time and optimality of solving any N-node graph with JSG and the naive approach, as shown in Fig. 3. Q-Learning does not scale well to this challenging problem by underperforming JSG in terms of both optimality and time , while PPO can find the middle ground in terms of optimality and time between JSG and the naive approach. The PPO results (magenta crosses) in both graphs indicate that for any 5-node and 10-node graph, on average PPO can solve it with 70% and 80% optimality and half of the time compared to JSG respectively.

VI. CONCLUSIONS

We study RL techniques to enable team coordination behaviors in graph environments with support actions among teammates to reduce edge traversal costs in a centralized manner. By converting the original team coordination on graphs with risky edges problem into a novel MDP formulation, we are able to apply RL to solve it. Our proposed state space is able to capture not only robot positions, but also graph connectivities and supporting mechanisms. Our experiment results indicate that while classical approaches can solve simple problems with smaller number of nodes and agents very efficiently with optimality guarantee, RL has the potential to outperform classical approaches in larger graphs with more agents. However, there is still room to improve with respect to graph scale, team size, optimality, and efficiency with better state and action space and shaping reward design. Another promising direction is to move towards the decentralized regime to keep improving on scalability with provable and bounded reduction on optimality.

References

[1] M. Limbu, Z. Hu, S. Oughourli, X. Wang, X. Xiao, and D. Shishika, "Team coordination on graphs with state-dependent edge cost," in *20230 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2023.

[2] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *Ieee Access*, vol. 6, pp. 28573–28593, 2018.

[3] J. Bellingham, "Autonomous ocean sampling network," *Moss Landing, CA: Monterey Bay Aquarium Research Institute*, 2006.

[4] X. Xiao, J. Dufek, and R. R. Murphy, "Autonomous visual assistance for robot operations using a tethered uav," in *Field and Service Robotics: Results of the 12th International Conference*, pp. 15–29, Springer, 2021.

[5] X. Xiao, J. Dufek, T. Woodbury, and R. Murphy, "Uav assisted usv visual navigation for marine mass casualty incident response," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6105–6110, IEEE, 2017.

[6] X. Xiao, J. Dufek, and R. Murphy, "Visual servoing for teleoperation using a tethered uav," in *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pp. 147–152, IEEE, 2017.

[7] B. Liu, X. Xiao, and P. Stone, "Team orienteering coverage planning with uncertain reward," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9728–9733, IEEE, 2021.

[8] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," *Cooperative robots and sensor networks 2015*, pp. 31–51, 2015.

[9] W. Ren and R. W. Beard, "Consensus seeking in multiagent systems under dynamically changing interaction topologies," *IEEE Transactions on Automatic Control*, vol. 50, no. 5, pp. 655–661, 2005.

[10] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Transactions on Automatic Control*, vol. 51, no. 3, pp. 401–420, 2006.

[11] X. Wang, Y. Zhou, and W. Jin, "D3g: Learning multi-robot coordination from demonstrations," *arXiv preprint arXiv:2207.08892*, 2022.

[12] J. Hart, R. Mirsky, X. Xiao, S. Tejeda, B. Mahajan, J. Goo, K. Baldauf, S. Owen, and P. Stone, "Using human-inspired signals to disambiguate navigational intentions," in *Social Robotics: 12th International Conference, ICSR 2020, Golden, CO, USA, November 14–18, 2020, Proceedings*, pp. 320–331, Springer, 2020.

[13] M. Khonji, R. Alyassi, W. Merkt, A. Karapetyan, X. Huang, S. Hong, J. Dias, and B. Williams, "Multi-agent chance-constrained stochastic shortest path with application to risk-aware intelligent intersection," *arXiv preprint arXiv:2210.01766*, 2022.

[14] K.-K. Oh, M.-C. Park, and H.-S. Ahn, "A survey of multi-agent formation control," *Automatica*, vol. 53, pp. 424–440, 2015.

[15] Y. Zheng, Y. Zhu, and L. Wang, "Consensus of heterogeneous multi-agent systems," *IET Control Theory & Applications*, vol. 5, no. 16, pp. 1881–1888, 2011.

[16] S. V. Albrecht, F. Christianos, and L. Schäfer, *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2023.

[17] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," 2018.

[18] 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," 2017.

[19] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative, multi-agent games," 2022.

[20] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," 2020.

[21] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1086–1095, 2019.

[22] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, May 1992.

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

[24] S. Li, J. K. Gupta, P. Morales, R. Allen, and M. J. Kochenderfer, "Deep implicit coordination graphs for multi-agent reinforcement learning," 2021.

[25] W. Böhmer, V. Kurin, and S. Whiteson, "Deep coordination graphs," 2020.

[26] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[27] S. Huang and S. Ontañ ón, "A closer look at invalid action masking in policy gradient algorithms," *The International FLAIRS Conference Proceedings*, vol. 35, may 2022.