2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Graph Neural Network-based Multi-agent Reinforcement Learning for Resilient Distributed Coordination of Multi-Robot Systems

Anthony Goeckner[1], Yueyuan Sui[1], Nicolas Martinet[1,2], Xinliang Li[1], and Qi Zhu[1]

Abstract— **Existing multi-agent coordination techniques are often fragile and vulnerable to anomalies such as agent attrition and communication disturbances, which are quite common in the real-world deployment of systems like field robotics. To better prepare these systems for the real world, we present a graph neural network (GNN)-based multi-agent reinforcement learning (MARL) method for resilient distributed coordination of a multi-robot system. Our method, Multi-Agent Graph Embedding-based Coordination (MAGEC), is trained using multi-agent proximal policy optimization (PPO) and enables distributed coordination around global objectives under agent attrition, partial observability, and limited or disturbed communications. We use a multi-robot patrolling scenario to demonstrate our MAGEC method in a ROS 2-based simulator and then compare its performance with prior coordination approaches. Results demonstrate that MAGEC outperforms existing methods in several experiments involving agent attrition and communication disturbance, and provides competitive results in scenarios without such anomalies.**

I. Introduction

Multi-agent systems surround us every day: vehicles on the road, airplanes in the sky, and soon, robots in multitudes. For truly useful autonomy in these systems, individual agents must coordinate their actions with those around them. However, current multi-agent coordination techniques are often fragile, susceptible to both drastic changes such as agent attrition or to the slightest disturbances such as poor communications, especially given the inherent partial observability of our physical world. In this paper, we present new work towards a novel graph-based reinforcement learning approach to multi-agent coordination that is robust to such disturbances.

A multi-robot system that depends on inter-agent coordination cannot be fielded in environments which are particularly disturbance-prone or adversarial without robustness to those disturbances. This problem was highlighted extensively in the DARPA OFFSET program, which fielded nearly two hundred autonomous ground and air vehicles in urban combat scenarios [1]. Experimentation during the OFFSET program revealed significant coordination deficiencies in the presence of frequent agent attrition and a difficult communication environment. Most existing works do not address these disturbances effectively or at all.

Many real-world environments (and associated problems) can be naturally represented by graphs. For example, a road network (route planning), dense forest (swarm motion planning), or warehouse (task assignment) are graph environments at heart. These environments are then well suited for modeling and analysis using Graph Neural Networks (GNNs), which aggregate information from throughout the graph using a "message-passing" process to learn an "embedding", or representation, of the graph. This allows GNNs to capture the attributes of individual entities in the graph and the complex relationships between them.

The combination of GNNs and multi-agent reinforcement learning (MARL) is therefore a natural fit for solving many multi-robot challenges. Until now though, GNNs have not been heavily used in MARL and multi-robot coordination tasks.

We aim to change this using MAGEC, the Multi-Agent Graph Embedding-based Coordination algorithm. It is a MARL framework based on multi-agent proximal policy optimization (MAPPO) and inductive GNNs. Our approach is capable of multi-agent coordination in the face of agent attrition, poor or nonexistent communication, and partial observability of the environment. It is applicable to the general class of problems in which agents must traverse a graph-based environment. In this paper, we apply MAGEC to the multi-robot patrolling scenario, which we evaluate using our custom multi-robot simulation environment. Our evaluation includes comparison with benchmark algorithms for the patrolling scenario and provides analysis of our various design decisions. **Our contributions are as follows:**

- MAGEC, a robust GNN-based MARL method for multi-agent coordination in the face of disturbances, delayed rewards, and sparse actions.
- An inductive graph embedding method that accounts for both node and edge features.
- A novel "neighbor scoring" GNN which enables navigation in a graph-based environment.
- Training and simulation tools for MARL-based multi-robot patrolling.

II. Related Works

A. Graph Neural Networks

Many multi-robot coordination problems, including ours, can be easily represented by a graph. However, traditional neural networks accept only fixed-size inputs such as bitmap observations. In the case of a graph, the input size may vary based on the graph structure. To handle this, the so-called "message-passing" paradigm was developed first in [2] and then in many later works such as [3] to learn graph embeddings.

[1]Department of Electrical and Computer Engineering, Northwestern University, Evanston, Illinois, USA. Corresponding author: Anthony Goeckner (anthony.goeckner@northwestern.edu).
[2]CNRS, Inria, Rennes, France.

However, many of these prior works represent graphs in a transductive manner and do not generalize well to graphs different from those on which the embedding was originally trained. To overcome this challenge, Hamilton et al. devise an *inductive* embedding method called "GraphSAGE" [4]. This method was further improved upon by Xu et al. in [5] with the addition of skip connections between GraphSAGE convolutional layers.

Zhou et al. proposed a GNN framework [6] that integrates node and edge features to boost long-distance information transmission and enhance node embeddings. Our independently developed research also adopts combined node and edge embedding, though our method is inductive and may be applied to graphs and nodes unseen during training.

B. Multi-Agent RL using Graph Neural Networks

Compared to other methods, the use of GNNs in MARL has been minimal. In [7][8], a single-layer GNN is used to interpret LIDAR observations. In [9], [10], a GNN is used during the training stage of a MARL algorithm to perform state-action value factorization. Hu et al. use a GNN for learned multi-robot coordination, though they only consider the agents and not the environment as part of the graph [11].

C. Algorithms for Patrolling

A series of studies have progressively enhanced understanding of and methodology for the multi-robot patrolling problem. Portugal and Rocha proposed two Bayesian-based strategies, namely the *Greedy Bayesian Strategy* (GBS) and the *State Exchange Bayesian Strategy* (SEBS) [12]. GBS emphasized immediate gains by making locally optimal choices and SEBS extended this by integrating inter-robot communication attrition robustness. Portugal and Rocha later introduced the *Concurrent Bayesian Learning Strategy* (CBLS) [13], which utilized reward-based learning. Farinelli et al. developed two solutions for patrolling in uncertain environments: DTA-Greedy and the more intricate DTAP, an auction-based task allocation mechanism [14]. Wiandt et al. presented a self-organizing algorithm that could autonomously partition a graph for agent patrolling [15]. Kobayashi et al. proposed the Local Reactive (LR) algorithm [16], which focuses on patrolling while maintaining base station situation awareness. More recently, ElGibreen and Youcef-Toumi introduced an online DTA method for uncertain environments, emphasizing heterogeneous agent capabilities [17]. The *Adaptive Heuristic-based Patrolling Algorithm* (AHPA) was introduced in [18] and was shown to be robust to agent attrition while using minimal communication resources.

D. Algorithms for MARL Patrolling

More recently, MARL has been used for multi-robot patrolling. Guo et al. use graph attention networks with MARL [19]. However, their method does not appear to generalize to different environments without retraining. Tong et al. present a MARL approach [20] for patrolling, though it only uses simple grid environments with single-step actions and immediate reward.

III. Problem Definition

We start by introducing the motivating scenario, multi-robot patrolling, and then provide the mathematical formulation for our multi-agent system (MAS).

In the following, we use subscript notation to indicate a value pertaining to an agent, and superscript notation to indicate an agent's belief about some value. For example, s_i^j is agent j's belief about the state of agent i, and s_i is the true state of agent i. When necessary, we denote a value as a function of time t, such as $s_i(t)$.

A. The Patrolling Problem

Multi-Robot Patrolling is a common benchmark problem for evaluation of multi-agent coordination algorithms such as task allocation [12]. We use the patrolling formulation of [18] and adapt it to this problem. In the patrolling problem, agents of the MAS must repeatedly visit a set V of *observation points* or nodes, attempting to minimize one of several metrics such as $\bar{\zeta} = \frac{1}{m}\sum_{v\in V}\zeta(v)$, the average idleness time or $\zeta_{\max} = \max_{v\in V}\zeta(v)$, the worst node idleness time.

Unlike in some other patrolling works, such as [12], we assume that the last visitation time of nodes can be observed from the environment (see Section IV-E.1).

B. Multi-Agent System Definition

Using the notation of [21], we formulate the multi-agent system as a decentralized partially-observable Markov decision process (Dec-POMDP) described by the tuple $\langle \mathcal{A}, S, O, A, T, R, \gamma \rangle$. The set $\mathcal{A}$ is a finite set of agents. As shorthand, we let $n = |\mathcal{A}|$.

The environment is modeled as a graph $G = \{V, E\}$ in Euclidean space $\mathbb{R}^2$, where V is the set of vertices and E is the set of edges. As shorthand, we let $m = |V|$ indicate the number of vertices. Graph G is known a-priori to all agents.

The set S is a finite set of possible system states and

$$s = \{s_i\}_{i\in\mathcal{A}} \cup \{s_v\}_{v\in V} \quad \forall s \in S$$

where s_i is the state of agent i and s_v the state of node v. Due to partial observability and communication disturbances, the true state s of the system may not be known with certainty. Rather, each agent only knows with certainty a subset of the system state, s^i.

The set O is the joint observation space, where $O = \{O_i\}_{i\in\mathcal{A}}$. The observation space O_i of each agent is described as the state of all vertices and agents within some radius r of the agent i.

Information is shared between agents by observation and explicit communication. Agent i may observe the state s_j of other agents and vertices within radius r:

$$o_i = \{s_j | j \in V \cup \mathcal{A} : dist(i, j) \leq r\}$$

Agents may only travel between nodes which are joined by an edge in E, and travel is considered an action in A_i, the set of available actions for agent i. The action space is defined as $A_i = \{0, \ldots, \Delta(G) - 1\}$, where $\Delta(G)$ is the maximum degree of graph G. We also define $A = \{A_i\}_{i\in\mathcal{A}}$ as the joint action space over all agents.

A transition function $T : S \times A \times S \rightarrow [0, 1]$ describes the probability of transition from one system state to another given some joint action. Finally, the function $R : S \rightarrow \mathbb{R}^n$ describes the reward to each agent for a particular system state, and γ is a discount factor on the reward. We assume that T and R are defined as part of the environment and are unknown a priori.

The optimization objective of the agent is to find a policy $\pi_i(s^i)$ which provides the agent's best action given the estimated system state s^i. We assume that spaces S, O, A are known in advance by all agents and that the policy π_i may be shared such that it is the same on all agents: $\pi^i = \pi^j \ \forall i, j \in \mathcal{A}$. Then, we may train a policy using centralized training and decentralized execution (CTDE).

IV. Design Methodology

To overcome the serious and realistic limitations imposed by the problem formulation, including partial observability, disturbed communications, and agent attrition, we present our Multi-Agent Graph Embedding-based Coordination (MAGEC) approach. MAGEC uses an actor-critic architecture, with a custom k-convolution GNN serving as the actor and a basic multi-layer perceptron as the critic. Given graph-based inputs, the actor must select an edge of the graph for the agent to next traverse.

Training of MAGEC is performed in an entirely inarcane manner by means of the multi-agent proximal policy optimization (MAPPO) algorithm [22], with some modifications.

A. Overall Architecture

We must enable distributed coordination while still optimizing towards a global objective. Therefore, we develop a reinforcement learning architecture based on multi-agent proximal policy optimization (MAPPO) [22]. We leverage the centralized training and decentralized execution (CTDE) paradigm to enable distributed coordination among the agents. After training is complete, the actor network is used as a policy mapping states to actions, while the critic is discarded. Training uses a shared and omniscient critic $\hat{V}$ for all agents, enabling optimization of agent policies based on the global objective (see Section III-A). All agents use a shared policy π, allowing for varying numbers of agents and for adaptation to agent attrition.

The actor and critic networks have different architectures, reflecting their distinct purposes. The critic network is extraordinarily simple, consisting only of a multi-layer perceptron (MLP). It takes as input the global information necessary to judge overall performance, using feature-engineered information such as the normalized idleness time of all nodes and an adjacency matrix. Its output is merely the value of the current state. This critic provides good estimations of value, but its architecture is far too simple to enable effective coordination for agents using the limited (partially-observed) information available at execution time.

Therefore, we develop an actor neural network which is far more capable than the critic. The actor is based on the message-passing GNN paradigm, which we describe along with our GNN architecture in Section IV-C. First, a graph-based observation of the agent's surroundings (Section IV-E.1) is passed into the GNN. Neighbor scoring is then performed on the GNN output using an MLP (Section IV-D), and the output of neighbor scoring is then passed through another MLP, the "selector". The "selector" MLP output is interpreted as a categorical distribution over the discrete action space. Each action represents a neighbor of the agent's current node to which the agent should travel (Section IV-B).

The overall architecture is presented in Fig. 1, and proceeding sections describe components in greater detail.

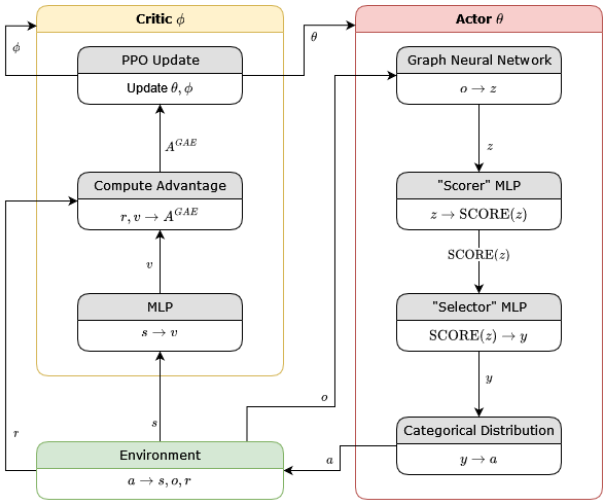

Fig. 1. The overall MAGEC training architecture is seen above. Note that the critic is only used during training (CTDE). Please see Fig. 3 for details of the GNN block.

B. Discrete Wayfinding in a Graph

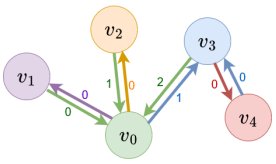

Fig. 2. An example of neighbor indexing. Note that v_3 is neighbor 2 of v_0, but v_0 is neighbor 1 of v_3. Indexing is enforced by the environment observation mechanism.

Let $\mathcal{N}(v)$ be the neighbourhood of node v. By definition, this set of neighbors has no particular order. However, to enable generalizable discrete wayfinding, we must enforce an ordering upon these neighbors. As discussed in Section IV-E.1, we ensure that neighbors of each node are always presented in the same (undefined) order in every observation. Neighbors are assigned identifiers in the range $0, \ldots, |\mathcal{N}(v) - 1|$ as shown in Fig. 2. In order for each node's neighbors to use this same range of identifiers, the undirected graph G must first be converted into a bidirected graph. This process is described in more detail in Section IV-E.1.

The agent's policy then outputs an edge index for the neighbor to traverse which corresponds with the desired next node to visit. We use action masking to ensure that the action selected is never greater than the degree of the current node. If an agent is already traversing an edge, we use masking to ensure that the only option is to continue until complete.

C. Graph Neural Network Design

Our GNN is based on the GraphSAGE algorithm [4][5]. However, GraphSAGE does not account for edge attributes, which are critical to wayfinding in a graph. The edge attributes in our graph contain both the weight (length) of the edge and an edge identifier as described in Section IV-B.

To enable consideration of these edge attributes in the convolutional layers, we modify the GraphSAGE embedding generation algorithm as shown in Algorithm 1. As each "message" is passed between nodes, we concatenate the transmitted node features x_v with the features $x_{u,v}$ of the edge that is being traversed to create an augmented feature vector, $\mathring{x}_{u,v}$. Message passing otherwise proceeds as normal in GraphSAGE. This is an efficient and simple method which enables our GNN to consider both node and edge features. Our message-passing framework is shown in Fig. 3.

Algorithm 1 GraphSAGE Embedding Generation (Forward Propagation) with Edge Attributes

Input: Graph $G(V, E)$; input features $\{x_v, \forall v \in V\}$; edge features $\{x_{u,v}, \forall u, v \in V\}$; depth K; weight matrices $W_k, \forall k \in \{1, \ldots, K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, \ldots, K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^V$
Output: Vector representations z_v for all $v \in V$
$h_v^0 \leftarrow x_v, \forall v \in V$
for $k = 1$ **to** K **do**
  for all $v \in V$ **do**
    $\mathring{x}_{u,v} \leftarrow \text{CONCAT}(h_v^{k-1}, x_{u,v})\ \forall u \in \mathcal{N}(v)$
    $h_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathring{x}_{u,v}\}_{\forall u \in \mathcal{N}(v)})$
    $h_v^k \leftarrow \sigma\left(\mathbf{W}_k \cdot \text{CONCAT}(h_v^{k-1}, h_{\mathcal{N}(v)}^k)\right)$
  end for
  $h_v^k \leftarrow \frac{h_v^k}{\|h_v^k\|_2}, \forall v \in V$
end for
$z_v \leftarrow h_v^K, \forall v \in V$

With each message-passing convolutional layer described in Algorithm 1, the graph embedding h_v^k gains information about nodes an additional hop from v, for all nodes $v \in V$. Therefore, to effectively embed a graph environment in which many hops must be traversed, k must be greater than one. This is in contrast with other GNN-based MARL works, such as [7][8], where a value of $k = 1$ ensures that only immediate neighbors are represented in the embedding. To balance computational cost and effectiveness of the embedding, we select a value of $k = 10$ which enables strong performance in large graphs. As in previous works, we find that skip connections must be included between message-passing layers for efficient training, so we implement "jumping knowledge" skip connections as described in [5].

In especially large graphs, this fixed value of $k = 10$ ensures that the environment is only partial observable, though we find that our method is easily able to overcome such limitation. We also enforce additional partial observability mechanisms during evaluation, described in Section IV-E.1.

D. Neighbor Scoring

Existing works such as [23] and [7] combine graph neural networks with MARL, but they do not attempt to solve the discrete wayfinding problem described in Section IV-B. To address those challenges involved in selection of an edge to traverse, we devise a strategy which we term, "neighbor scoring".

In the forward propagation step of our algorithm, we first create graph embeddings from the perspective of each node using the GNN described in Section IV-C. We specify a node of interest, v, which represents the agent for which we desire to select an action (see Section IV-E.1 for more information about agent representations). The graph embedding from the perspective of each neighbor $u \in \mathcal{N}(v)$ is pulled from the output of the GNN and passed through the neighbor-scorer MLP, as seen in Fig. 1. This neighbor-scorer MLP provides a single score for each neighboring node. All scores are then passed to a "selection" MLP, the output of which forms a categorical distribution over the available actions. We then sample from that categorical distribution to select an action. Borrowing from the notation of Algorithm 1, the neighbor scoring mechanism may be seen as follows:

$$y_v \sim \text{SELECTION}(\langle \text{SCORE}(z_u) \rangle_{u \in \mathcal{N}(v)})$$

The neighbor scoring mechanism can be seen as a "choke-point" which creates compact latent representations (in our case, a single "score" value) of each neighbor's graph embedding. This enables the selection MLP to effectively distinguish between different neighbors' utilities.

We ensure that neighbor scores passed to the selection MLP are always in order of their neighbor index. This enables the selection MLP to effectively learn which embeddings correspond with which neighbor indices, and thus, with which action.

To enable generalization to different numbers of nodes and agents, we train the network with a fixed maximum number of neighbors per node. For nodes with fewer neighbors, we use padding to ensure that the input to the neighbor-scorer MLP remains the same size.

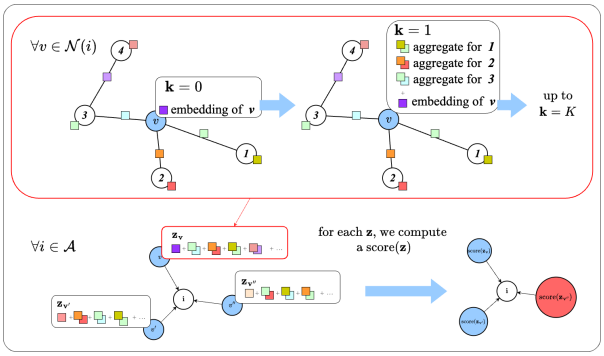

Fig. 3. Illustration of GNN computing node embeddings through iterative neighborhood aggregation. A scoring function is applied to the embeddings for decision-making. Note that node and edge features are concatenated during message-passing.

E. Training Environment

1) Observations: Agents may observe the state of other agents and nodes within a limited observation radius. Any other agents within the observation radius are added to graph G as nodes, with edges indicating the agent's current position relative to other nodes, thus forming an augmented graph $\mathring{G}$. Each node in $\mathring{G}$ is populated with features including an encoded type (agent or observation point), normalized node idleness time ζ, and node degree. Edges in $\mathring{G}$ are populated with features including normalized distances and an identifier as described in Section IV-B. The graph $\mathring{G}$ is then converted to a digraph to enable edge indexing (see Section IV-B) and fed into the GNN-based policy π to generate an action.

2) Actions: As described in Section III, the action space for an agent $i \in \mathcal{A}$ is defined as $A_i = \{0, \ldots, \Delta(G) - 1\}$, where $\Delta(G)$ is the maximum degree of graph G. Agents will move to the neighbor with the index of the selected action.

3) Reward: We must optimize for a global objective: the minimization of average idleness time $\bar{\zeta}$. To achieve this, we provide a global "terminal" reward $r_{\text{terminal},i}(t) = \frac{t}{\bar{\zeta}} \;\; \forall i \in \mathcal{A}$ to all agents at the end of every episode. We also use a local reward which is awarded to individual agents upon reaching a node v. This reward is based on the ratio between the idleness of the visited node $\zeta(v_i)$ and the average idleness of all nodes such that $r_{\text{local},i}(t) = \frac{\zeta(v_i)}{\bar{\zeta}+\epsilon} \;\; \forall i \in \mathcal{A}$, where ϵ is a small value which avoids division by 0. The overall reward function may be written as

$$r_i = \begin{cases} \alpha\, r_{\text{local, i}} + \beta\, r_{\text{terminal, i}} & \text{if } t = \mathcal{T} - 1 \\ \alpha\, r_{\text{local, i}} & \text{otherwise} \end{cases}$$

where $\mathcal{T}$ is the preconfigured episode length. This reward r_i is awarded to every agent $i \in \mathcal{A}$ at every step $t = 0, \ldots, \mathcal{T} - 1$. In testing, we use $\alpha = 1.0$ and $\beta = 0.5$.

F. Training Algorithm

Training of the agents occurs in a centralized fashion following a modified version of the MAPPO algorithm [22] seen in Algorithm 2. The rewards provided by our patrolling environment (see Section IV-E.3) are sparse; the agent is only rewarded upon visiting nodes and upon termination of the episode. Further, the patrolling problem described in Section III involves *sparse actions* which require multiple time steps to complete. Therefore, after sampling an action $a_i(t)$ from the policy, agent i's next action $a_i(t + \Delta t)$ need not be sampled from the policy until Δt steps have passed from t. We use this revelation to reduce the number of samples stored in the replay buffer, greatly increasing the speed of training. Unfortunately, due to technical limitations of our implementation, we are unable to skip steps for each agent asynchronously. Rather, we only skip steps synchronously; when one agent needs to sample the policy for a new action, all other agents must also take that step. However, the total number of steps taken in an episode is still greatly reduced thanks to the extremely sparse action nature of our problem.

To account for rewards which are given during skipped steps, we further enhance the algorithm by adding the sum of those rewards to the replay buffer.

$$R_i(s(t), t) = \sum_{k=t}^{t+\Delta t} R_i(s(k), k)$$

The cumulative reward $R_i(s(t), t)$ is then associated with $a_i(t)$ and added to the replay buffer. The rewards for skipped steps $t \ldots t + \Delta t$ are not added to the buffer individually.

Similar modifications to the MAPPO algorithm were first made by Yoshitake and Abbeel in [24], and we use their adapted version of Generalized Advantage Estimation (GAE) here to great success. Yoshitake and Abbeel modify GAE equations to account for skipped steps as follows:

$$\delta_i(t) = R_i(s(t), t) + \gamma^{\Delta t} \hat{V}(s(t+1)) - \hat{V}(s(t))$$

$$\text{GAE}_i(t) = \delta_i(t) + (\gamma\lambda)^{\Delta t}\delta_i(t+1) + \cdots + (\gamma\lambda)^{\Delta\mathcal{T}-1}\delta_i(\mathcal{T}-1)$$

As seen above, the discounting factors γ are merely raised to the number of steps between samples Δt. Without skipping, Δt is clearly 1 and the modified GAE and TD equations are then equivalent to the originals.

Algorithm 2 Training Algorithm

```
Initialize Policy π to parameters θ
Initialize Critic V̂ to parameters φ
for all episodes e ∈ 0 ... e_max do
    Initialize Rollout buffer B = ⟨s(0), o(0)⟩
    for all t ∈ 0 ... T do
        Δt ← number of steps until next action required
        a(t) ← π(o(t))
        Take Δt steps in environment using action a(t)
        s(t + Δt), o(t + Δt) ← values from environment
        r ← Σ_{τ=t}^{Δt} R(s(τ))
        v ← V̂(s(t))
        B += ⟨s(t), o(t), a(t), r, v, Δt⟩
    end for
    Compute returns Â using modified GAE over B
    for all mini-batches b ∈ B do
        Update φ using mini-batch data b.
        Update θ using mini-batch data b and critic V̂.
    end for
end for
```

We find experimentally that use of agent attrition is unnecessary during training and that agents will adapt to attrition during execution nonetheless. This is likely because the problem is formulated as a DEC-POMDP such that the policy $\pi(s)$ has no time dependency and only needs the current state of the system.

G. Execution

Use of the trained policies in our patrolling scenario is straightforward and is consistent with typical CTDE practice. Since the critic was only used during training as a surrogate of the value function $V_\pi(s)$, it is not required in the execution phase. The policy trained provides a mapping from state $s(t)$ to appropriate action(s): $\pi(s(t)) \rightarrow a(t)$. Therefore,

the policy may be straightforwardly applied to agents during execution. Since the policy is shared amongst all agents and operates only on the agent's belief about the global state $s^i(t)$, it may be executed by any number of homogeneous agents regardless of the number of agents used in training.

H. Implementation

We implement the training and basic execution environment using PettingZoo [25], a multi-agent reinforcement learning environment library based on OpenAI's popular Gym library. The PettingZoo environment is implemented as a parallel environment (rather than agent-environment cycle) in which all agents perform actions simultaneously before any observations are taken. We call our environment "patrolling_zoo", and release it as an open-source project[1].

Our training code is heavily based on that used by Yu et al. in their original MAPPO paper [22], though we have performed extensive work to integrate it with our "patrolling_zoo" environment and modify as described in Section IV-F, along with the implementation of new critic and GNN-based actor networks.

V. Evaluation Methodology

In this section, we describe our evaluation methodology and test setup. Importantly, while the training and initial evaluation of our approach was performed in a simple graph environment using the PettingZoo library, we also choose to evaluate our algorithm using a multi-agent robotics simulator which better highlights our approach's robustness to disturbances and applicability to real scenarios such as the multi-robot patrolling problem.

A. Experimental Setup

The primary tool used in evaluation is the Grex Machina multi-agent framework[2], developed by the IDEAS Lab at Northwestern University for multi-agent research and based on the Robot Operating System 2 (ROS 2) [26]. Grex allows agents to be operated either in one of multiple available simulators (Gazebo, Flatland, etc.) or on physical robots, all without changing a single line of code. The simulator that we select, Flatland, injects artificial Gaussian noise into sensor readings, creating a more realistic test of our algorithm than the PettingZoo training environment.

To test our algorithm on the patrolling scenario, we integrate existing state-of-the-art and benchmark patrolling algorithms and environments into the Grex framework. Many such algorithms and environments are from a patrolling simulator previously built by Portugal et al. [12], which we have adapted and integrated into Grex[3]. Portugal's original patrolling simulator was used in a large number of patrolling algorithms that are still considered to be state-of-the-art, including [12][13][14][15][17][18].

Selection of appropriate algorithms for comparison with this work is critical, but is highly difficult due to varying assumptions and objectives between the previous approaches. We compare with the AHPA algorithm [18], SEBS [12], and CBLS [13]. All three benchmarks have strong performance in the face of agent attrition. However, none of them make use of environmental observations. To account for this, we first perform an experiment using unlimited observations and undisturbed communications, theorizing that this will show the best performance of all algorithms. We also attempt to find a middle ground with an observation radius of 40 m for subsequent tests and a variety of communication success rates. All algorithms are tested with and without attrition.

The primary metric that we use for performance comparison is the average idleness of all nodes $\hat{\zeta}$ over time, as discussed in Section III-A. The use of this metric was established by [12] and it has appeared in almost all related works since that point. Therefore, we find it to be a suitable metric for judging the overall efficacy of our algorithm.

B. Generalization

To ensure that the learned policy π is not overfit and can generalize to different environments, we train and test with entirely different graphs and agent counts. For the results shown in this paper, training is performed on the "Milwaukee" graph with four agents. Evaluation is performed with six agents on the "Cumberland" graph (see Fig. 4), the same environment commonly used for testing in previous works [12].

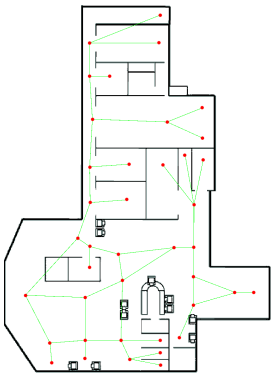

Fig. 4. Agents were trained on the "Milwaukee" graph and patrol on an entirely different one, "Cumberland" (above), demonstrating the generalizability of our approach. Above, red dots indicate nodes and green lines indicate edges in the patrol graph. Black lines indicate obstacles which agents must navigate around.

C. Disturbances

We model agent attrition in the simplest way possible: at two fixed points in time throughout the experiments, we choose an agent and remove it from the simulation. This simplicity provides good performance comparisons between algorithms by allowing the moment of attrition to be seen as an inflection point, where algorithms either suffer in performance or continue unfazed.

As with attrition, we keep the communication disturbance as simple as possible, using a Bernoulli loss model with fixed reception probability. We ensure that all algorithms publish agent telemetry (positions, etc.) at a rate no more than one Hertz. Other messages, such as attrition notifications or goal-reached notifications, are sent on-demand. We apply the communication disturbance to all of these messages.

VI. Results

We find that our method is highly effective and suitable for use in multi-agent systems that must be fielded in environments where the risk of agent attrition or of communication disturbance is great. In this section, we present and analyze the findings of our work.

[1] https://github.com/NU-IDEAS-Lab/patrolling_zoo
[2] https://github.com/NU-IDEAS-Lab/grex
[3] https://github.com/NU-IDEAS-Lab/patrolling_sim

A. Training Performance

The policy was trained for 350,000 environment steps, broken into environment episodes of 200 steps each, with five copies of the environment in parallel, for a total of 350 PPO episodes. In Fig. 5, both average reward and average idleness can be seen to improve and converge over time. Our improvements to MAPPO and the use of graph observations with a GNN greatly improved training time and feasibility over our previous (non-GNN) attempts. Experimentally, we found that using hyperparameters $\alpha = 1.0$, $\beta = 0.5$, and $\gamma = 0.99$ resulted in the best training performance, effectively balancing global and local rewards.

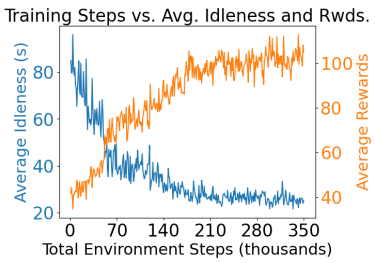

Fig. 5. Graphs showing the average episode reward and evaluated average idleness over training period. Training completes in a mere 350,000 environment steps.

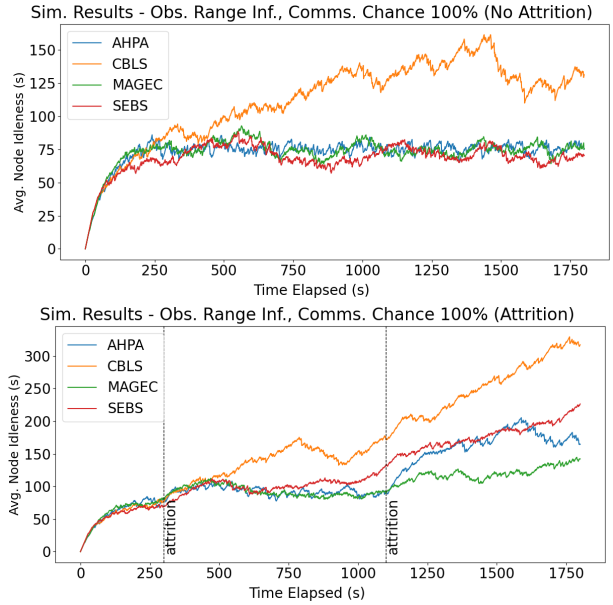

Fig. 6. Test results using observation radius inf m and communication success rate of 100%. Above is the test without attrition, and below is the test with two attrition events. Our MAGEC algorithm is shown in green.

B. Simulation Performance

In evaluation of our algorithm's performance in simulation, we focus on how the "average idleness" metric described in Section III-A changes throughout the course of a 30-minute evaluation run. Due to artificial sensor noise in the simulator, we perform three evaluation runs of every experiment and then average the results across runs.

Though direct comparison between algorithms that make different assumptions is difficult, we believe that our method performs well. MAGEC achieves stable standard deviations of idleness throughout all tests, indicating that nodes in the graph are visited with similar frequencies. To provide the fairest comparison possible, we first test with an infinite observation range and 100% communication success rate, on the theory that all algorithms will be able to achieve their maximum performance without being hampered by differing assumptions. MAGEC performs far better than the existing algorithms in this attrition scenario (Fig. 6), especially after the second attrition event where MAGEC's steady performance degrades far less than the benchmarks.

It also outperforms the benchmark algorithms in terms of average idleness, our primary metric, in attrition scenarios such as the one shown in Fig. 7 which uses a 10% communication success rate and limited observation range.

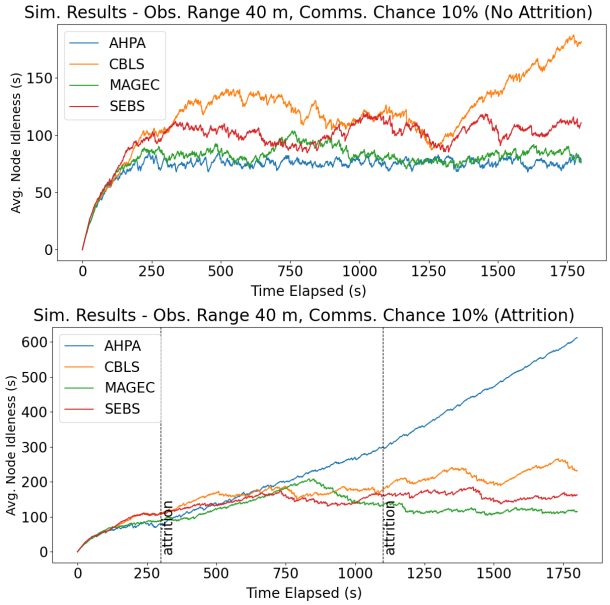

Fig. 7. Test results using observation radius 40 m and communication success rate of 10%. At top, the test without attrition, and at bottom, the test with two attrition events. Our algorithm, MAGEC, is shown in green.

Other algorithms do not appear to handle attrition well when faced with heavy communication losses. For example, AHPA handles attrition very poorly when its attrition notification message is lost, resulting in extremely high average idleness and standard deviation of idleness after attrition. CBLS also struggles when faced with message losses and attrition. However, SEBS fairs well and comes close to MAGEC in some scenarios.

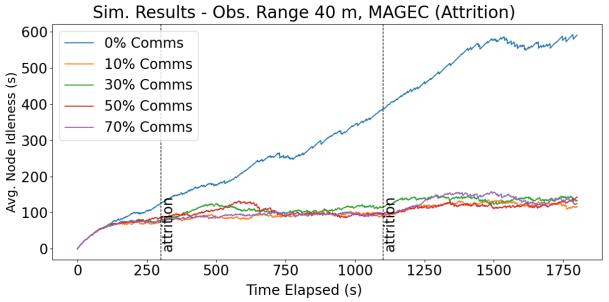

Fig. 8. Test results using observation radius 40 m and various communication success rates for MAGEC. Zero-communication performance is poor, but MAGEC's performance remains strong even with only 10% comms.

We also test MAGEC's performance with various obser-

vation ranges and communication success rates. As seen in Fig. 8, performance drops off significantly with zero communications, but otherwise is minimally impacted.

In non-attrition scenarios, MAGEC is competitive, even though it is sometimes outperformed by AHPA. We attribute this to AHPA's extremely deterministic patrol order, which results in a lower standard deviation of idleness and thus lower average idleness when attrition is not a factor. For AHPA, communication losses are also only important when attrition occurs, because AHPA does not use any messaging other than an attrition notification method. This makes AHPA a strong contender in the non-attrition, disturbed communications scenarios, but as expected, MAGEC performs best in the experiments involving agent attrition.

VII. Conclusion

The performance of MAGEC, the Multi-Agent Graph Embedding-based Coordination algorithm, provides evidence that GNN-based MARL can be highly effective in an entire class of graph-environment problems such as multi-robot patrolling, vehicle routing, and swarm navigation. MAGEC is shown to effectively coordinate robots even with agent attrition, partial observability, and significant communication loss. Further, MAGEC operates in an environment with delayed rewards and sparse actions. These are realistic disturbances and limitations that multi-robot systems must overcome before they can be widely fielded. MAGEC provides a solid foundation for future work in this critical direction.

However, MAGEC is not magic, and more problems remain to be solved. Future work may attempt to integrate better methods for prediction of unobservable state (such as that performed by SEBS) which will enable far better performance in limited-communication scenarios.

Regardless, our method is pioneering in its use of k-layer GNNs paired with MARL for multi-robot coordination. It represents a general solution for agents which must coordinate movement in any environment representable as a graph, and we hope that MAGEC will form the basis of many robust multi-robot systems to come.

Acknowledgment

Special thanks to Dr. Yixuan Wang, Mr. Simon Zhan, and Dr. Stephen Xia for their advice on a million issues. We also graciously acknowledge support from National Science Foundation grants 1724341, 2038853, and 2324936.

References

[1] E. Taranta, A. Seiwert, A. Goeckner, K. Nguyen, and E. Cherry, "From Warfighting Needs to Robot Actuation: A Complete Rapid Integration Swarming Solution," *FR*, vol. 3, pp. 460–515, Jan. 2023.

[2] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural Message Passing for Quantum Chemistry," in *Proceedings of the 34th International Conference on Machine Learning*, pp. 1263–1272, PMLR, July 2017.

[3] T. Yao, Y. Wang, K. Zhang, and S. Liang, "Improving the Expressiveness of K-hop Message-Passing GNNs by Injecting Contextualized Substructure Information," in *KDD*, KDD '23, (New York, NY, USA), pp. 3070–3081, Association for Computing Machinery, Aug. 2023.

[4] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," Sept. 2018.

[5] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation Learning on Graphs with Jumping Knowledge Networks," in *Proceedings of the 35th International Conference on Machine Learning*, pp. 5453–5462, PMLR, July 2018.

[6] Y. Zhou, H. Huo, Z. Hou, L. Bu, J. Mao, Y. Wang, X. Lv, and F. Bu, "Co-embedding of edges and nodes with deep graph convolutional neural networks," *Sci Rep*, vol. 13, p. 16966, Oct. 2023.

[7] S. Zhang, K. Garg, and C. Fan, "Neural Graph Control Barrier Functions Guided Distributed Collision-avoidance Multi-agent Control," in *7th Annual Conference on Robot Learning*, Aug. 2023.

[8] S. Zhang, O. So, K. Garg, and C. Fan, "GCBF+: A Neural Graph Control Barrier Function Framework for Distributed Safe Multi-Agent Control," Jan. 2024.

[9] N. Naderializadeh, F. H. Hung, S. Soleyman, and D. Khosla, "Graph Convolutional Value Decomposition in Multi-Agent Reinforcement Learning," Feb. 2021.

[10] S. Ding, W. Du, L. Ding, J. Zhang, L. Guo, and B. An, "Multi-agent Reinforcement Learning With Graphical Mutual Information Maximization," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–10, 2023.

[11] Y. Hu, J. Fu, and G. Wen, "Graph Soft Actor–Critic Reinforcement Learning for Large-Scale Distributed Multirobot Coordination," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2023.

[12] D. Portugal and R. P. Rocha, "Distributed multi-robot patrol: A scalable and fault-tolerant framework," *Robotics and Autonomous Systems*, vol. 61, pp. 1572–1587, Dec. 2013.

[13] D. Portugal and R. P. Rocha, "Cooperative multi-robot patrol with Bayesian learning," *Auton Robot*, vol. 40, pp. 929–953, June 2016.

[14] A. Farinelli, L. Iocchi, and D. Nardi, "Distributed on-line dynamic task assignment for multi-robot patrolling," *Auton Robot*, vol. 41, pp. 1321–1345, Aug. 2017.

[15] B. Wiandt, V. Simon, and A. Kőkuti, "Self-organized graph partitioning approach for multi-agent patrolling in generic graphs," in *IEEE EUROCON 2017 -17th International Conference on Smart Technologies*, pp. 605–610, July 2017.

[16] K. Kobayashi, S. Ueno, and T. Higuchi, "Multi-Robot Patrol Algorithm with Distributed Coordination and Consciousness of the Base Station's Situation Awareness," Oct. 2023.

[17] H. ElGibreen and K. Youcef-Toumi, "Dynamic task allocation in an uncertain environment with heterogeneous multi-agents," *Auton Robot*, vol. 43, pp. 1639–1664, Oct. 2019.

[18] A. Goeckner, X. Li, E. Wei, and Q. Zhu, "Attrition-Aware Adaptation for Multi-Agent Patrolling," *IEEE Robotics and Automation Letters*, vol. 9, pp. 7230–7237, Aug. 2024.

[19] L. Guo, H. Pan, X. Duan, and J. He, "Balancing Efficiency and Unpredictability in Multi-robot Patrolling: A MARL-Based Approach," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3504–3509, May 2023.

[20] C. Tong, A. Harwood, M. A. Rodriguez, and R. O. Sinnott, "An Energy-aware and Fault-tolerant Deep Reinforcement Learning based approach for Multi-agent Patrolling Problems," June 2023.

[21] C. Boutilier, "Planning, learning and coordination in multiagent decision processes," in *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*, TARK '96, (San Francisco, CA, USA), pp. 195–210, Morgan Kaufmann Publishers Inc., Mar. 1996.

[22] 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," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24611–24624, Dec. 2022.

[23] S. Nayak, K. Choi, W. Ding, S. Dolan, K. Gopalakrishnan, and H. Balakrishnan, "Scalable Multi-Agent Reinforcement Learning through Intelligent Information Aggregation," May 2023.

[24] H. Yoshitake and P. Abbeel, "The Impact of Overall Optimization on Warehouse Automation," Aug. 2023.

[25] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente, N. Williams, Y. Lokesh, and P. Ravi, "PettingZoo: Gym for Multi-Agent Reinforcement Learning," in *Advances in Neural Information Processing Systems*, vol. 34, pp. 15032–15043, Curran Associates, Inc., 2021.

[26] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, May 2022.