Leveraging Graph Networks to Model Environments in Reinforcement Learning

Viswanath Chadalapaka, Volkan Ustun, Lixing Liu
University of Southern California, Institute for Creative Technologies
vchad@ict.usc.edu, ustun@ict.usc.edu, lxliu@ict.usc.edu

Abstract

This paper proposes leveraging graph neural networks (GNNs) to model an agent's environment to construct superior policy networks in reinforcement learning (RL). To this end, we explore the effects of different combinations of GNNs and graph network pooling functions on policy performance. We also run experiments at different levels of problem complexity, which affect how easily we expect an agent to learn an optimal policy and therefore show whether or not graph networks are effective at various problem complexity levels. The efficacy of our approach is shown via experimentation in a partially-observable, non-stationary environment that parallels the highly-practical scenario of a military training exercise with human trainees, where the learning goal is to become the best sparring partner possible for human trainees. Our results present that our models can generate better-performing sparring partners by employing GNNs, as demonstrated by these experiments in the proof-of-concept environment. We also explore our model's applicability in Multi-Agent RL scenarios. Our code is available online at `https://github.com/Derposoft/GNNsAsEnvs`.

Introduction

Capable, adaptive, and dynamic synthetic characters as sparring and training partners for use in complicated, realistic simulations taking place on geo-specific terrain are valuable to establish efficient, scalable, and mobile systems that train human trainees. For example, many military training scenarios (e.g., military skirmish and scouting exercises) stand to benefit from the development of such systems. To this end, models that train synthetic characters in abstractions of these complex environments are valuable study targets. Earlier work introduced one such practical simplification: a graph-based environment which was a discretized abstraction of a Unity simulation (Liu et al. 2021). Such a graph-based environment simplified the original problem into learning how to take actions and engage with other agents on these graph structures while still allowing the transfer of learned behavior policies back to the original Unity simulation environment (Ustun et al. 2020). Generating the optimal sequence of these actions is a combinatorial optimization (CO) problem: for example, a reconnaissance operation that requires one to take care not to be spotted by enemy patrols involves carefully executing a sequence of movements in a specific order (e.g., moving from cover after a patrol turns around and not before). Choosing the actions of this sequence is a problem that is generally intractable to solve outright, given that most real-world environments are generally stochastic, non-stationary, and sometimes partially observable in nature.

This paper explores the application of graph neural networks (GNNs) (Gilmer et al. 2017) to model similar graph-like environments to drive the learning of higher-performing policies in Reinforcement Learning (RL) experiments for these scenarios. One reason for choosing these networks is that GNNs have successfully learned to generate near-optimal solutions by exploiting similarities between problem instances for other CO routing problems, including the traveling salesman and vehicle routing problems (Cappart et al. 2021).

We build on these results by showing that when we utilize GNNs to capture features and relations of the scenario via leveraging these graph-based environments, the RL models can learn policy networks that outperform traditional, feed-forward (FF) neural networks for more general CO problems in military training exercises. For example, our results show that the inductive bias introduced by using GNNs within policy networks to model an agent's environment results in higher rewards and, therefore, more capable training partners. Our work additionally includes: (1) an exploration of the effects of various graph network pooling functions; (2) a comparison between GNNs vs. FFs at different levels of problem complexity; and (3) a comparison of GNNs vs. FFs in Multi-Agent RL (MARL) scenarios.

Related Work

Using GNNs as CO Heuristics. GNNs have been shown to provide high-scoring heuristics for solving well-known CO problems, such as the Travelling Salesperson Problem (TSP), in which the goal is to choose the shortest tour about a fully-connected graph. Recent work has shown that GNNs such as the Graph Attention Network (GAT) (Veličković et al. 2018) have the ability to produce high-quality heuristics for both TSP, and its more-general counterpart, the Vehicle Routing Problem (VRP) in a reinforcement learning (RL) setting (Kool, van Hoof, and Welling 2019). The simpler Graph Convolutional Network (GCN) (Bresson and Lau-

Copyright © 2023 by the authors. All rights reserved.

rent 2017) has also been shown to learn valuable heuristics for TSP approximation, both in supervised learning environments (Joshi et al. 2022) as well as within reinforcement learning scenarios (Devailly, Larocque, and Charlin 2021). Our work seeks to leverage and extend this useful property of GNNs by applying them to the broader class of CO problems that occur in the non-stationary and partially-observable environments of military simulations.

Using GNNs to Model Collaboration. Our paper focuses on using GNNs to model an environment. Conversely, in the past, work has been done to use GNNs to model the agents themselves: inter-agent interactions have been captured via graph connections both with and without the use of attention mechanisms (Li et al. 2020; Shang et al. 2021) to promote better collaboration. Intra-agent collaborative behavior (e.g., among the joints or structural parts of an agent) has also been successfully modeled to drive performance gains with graph networks (Wang et al. 2018), albeit with limited practical success (Kurin et al. 2020). Our approach is similar in that agent encodings are embedded into a graph, which can serve as a mechanism to capture cross-agent interactions. However, our work extends these previous methods in that the structure of the environment is also allowed to affect the way that collaborative behavior is learned.

Graph Neural Networks

Graph neural networks (GNNs) are a type of neural network architecture that can process graph data. At a high level, GNNs accomplish this by computing node-level representations formed via aggregation of neighboring node features (aka *message passing*), for a certain number of iterations (or *layers*). After computing node representations, a single output vector representation can then be computed by pooling together all (or some subset) of the node representations via a *pooling function*. In this paper, to test the efficacy of GNNs in modeling the environment, we test 3 different variations of GNNs, each with their own method of aggregation as described below. We also experiment with 3 different types of pooling functions which are discussed in the Experiments section. To help clarify our descriptions, we formally consider a graph $G = (V, E)$ with $V(|V| = n)$ and E is the set of nodes and edges, respectively, with v_i referring to the i-th node of G and e_{ij} referring to an edge between v_i and v_j. Furthermore, let x_i^l refer to the feature vector corresponding to v_i at a given layer l of the GNN, with x_i^0 (or just x_i) being the initial value of the node representation. Finally, let $N(v_i)$ be the set of node indices that are in the immediate neighborhood of v_i, i.e. $N(v_i) = \{j | e_{ij} \in E\}$.

1. **Graph Convolutional Networks (GCNs)** (Bresson and Laurent 2017) update the representation of node v_i at each layer l by using learned weights W^l to transform neighboring nodes before simply summing them and applying a $ReLU$ nonlinearity:

$$x_i^{l+1} = ReLU(\sum_{j \in N(v_i)} W^l x_j^l)$$

2. **Graph Transformers (GTs)** (Dwivedi and Bresson 2021) are one type of GNN that use neighborhood-level attention to update node representations. At each layer l, for each attention head k, the node representation is updated in a way that is very similar to that of a regular transformer. First, the regular self-attention computation is performed over the neighborhood, followed by adding residuals and taking the norm, passing the values through an FF network, and then adding residuals and norming once more. The following set of equations describes this process:

$$\hat{x_i}^{l+1} = O_h^l \|_{k=1}^H Attention_{j \in N(v_i)}(Q^{k,l}, K^{k,l}, V^{k,l})$$

$$\hat{x_i^{l+1}} = Norm(\hat{x_i}^{l+1} + x_i^l)$$

$$x_i^{l+1} = Norm(FF(\hat{x_i}^{\hat{l+1}}) + \hat{x_i^{l+1}})$$

where $\|_{k=1}^H$ means concatenating each of the k attention head outputs, and $Attention_{j \in N(v_i)}$ refers to self-attention computed over the node representations of those nodes neighboring v_i. $O_h^l \in \mathbb{R}^{d \times d}$ is a set of parameters meant to learn the relative importance of each attention head and $Q^{k,l}, K^{k,l}, V^{k,l} \in \mathbb{R}^{d \times d}$ refer to the regular query, key, and value matrices used in the attention computation.

3. **Graph Attention Networks (GATs)** are a second type of GNN that also use neighborhood-level attention to update node representations. However, attention is computed in a different way in a GAT than in the GT. In this paper, we use a slightly more expressive form of a GAT called GATv2 (Brody, Alon, and Yahav 2022), described below. Rather than using query, key, and value matrices, a much simpler method of using 2 weight matrices is used instead. First, two values are calculated as follows:

$$e(x_i, x_j) = a^T LeakyReLU(W \cdot [x_i \| x_j])$$

$$a_{ij} = Softmax_{j \in N(v_i)}(e(x_i, x_j))$$

where $\|$ again means concatenation, and W and a are learned parameters. The resulting value $e(x_i, x_j)$ is a simple way to obtain an importance score between x_i and x_j, and a_{ij} can be considered a form of attention between the two, normalized over each of v_i's neighbors. The node representations are then simply updated to be a weighted average of neighboring node representations, after applying a sigmoid activation function:

$$x_i^{l+1} = \sigma(\Sigma_{j \in N(v_i)} a_{ij} W x_j)$$

Approach

In this section, we first describe our RL framework and RL environment. Then, we detail the different policy models that we use in our experiments.

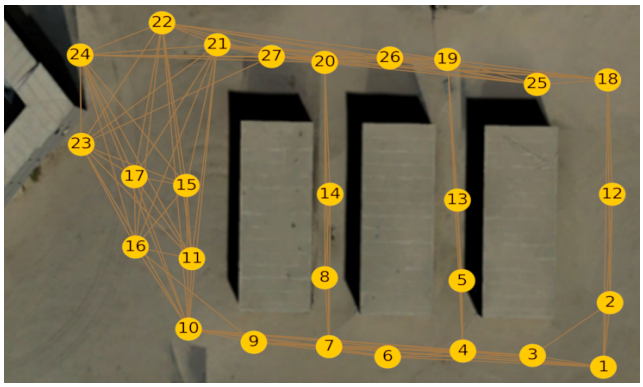

Figure 1: Visual representation of the graph-based terrain abstraction that we use for our experiments.

Reinforcement Learning Setup

We employ the RLLib library as the RL framework to run our experiments (Liang et al. 2018). For our RL algorithm, we use the PPO policy gradient method of updating our policy networks (Schulman et al. 2017). The introduction of a value network and the use of actor-critic algorithms (e.g. A2C/A3C) were also tested (Babaeizadeh et al. 2017), although their performance with graph network policies was much poorer than that of PPO.

Environment

Our RL environment is a 27-node graph-based abstraction of a real terrain map on which military training takes place (Fig. 1), adapted from a previous work employing pretrained GATs for collaboration problems (Liu et al. 2021), implemented via the OpenAI Gym library (Brockman et al. 2016). The objective of the simulation is for each of the two teams, red and blue, to deal damage to the other team while minimizing damage to their own team: the reward is increased for a team if they do damage to the other team while minimizing damage taken to their own, and vice versa. This setup provides us with a way to verify the efficacy of our models in a semi-realistic, multi-agent scenario.

The simulation ends either when the max step count of 20 is reached, or if one of the teams is completely wiped out. Our environment consequently gives rise to the following CO problem:

What is the optimal sequence of actions that can be made to maximize the reward for a specific agent?

Observations and Actions: Observations are a concatenation of the following information:

1. One-hot encoding for agent position on the graph
2. Indicators for enemies within sight on each node
3. 4-Directional indicators for each enemy agent
4. Indicators for allies on each node

Since the terrain graph of the environment has 27 nodes, this results in $27+27+4+27 = 85$ dimensions in the case of 1 red vs 1 blue (1v1) and $27+27+4*2+27 = 89$ dimensions in the case of 2v2.

Since each agent must choose 1 of 5 possible movement options among $\{$*North*, *South*, *East*, *West*, *None*$\}$ and 1 of 3 possible directions to turn and face among $\{$*Left*, *Right*, *Straight*$\}$, this means there are $5x3 = 15$ possible actions in total.

Agents: There are two teams in the environment: red and blue. In our experiments, the blue team (representing human trainees) consists of opponents which run through a scripted path, while the red team follows a policy learned through RL. Upon initialization, the blue agents are spawned equal distances away from one another on the scripted blue route. The red agents, on the other hand, are spawned at random locations on the map. Agents can spawn with a specified amount of health points (HP), and every time an agent is shot at, its HP is decremented by 1.

Rewards: Each red gains a +4 reward for shooting any blue agent and -3 for getting shot. To incentivize agents to actively seek rewards, a further -1 per turn is applied on each turn that the agent stays still. An additional -2 penalty is applied per turn if two red agents both position themselves on the same node in the multi-agent scenario to encourage more collaborative movement instead of a copycat movement. At the end of the episode, the following two rewards are also added for each red-blue agent pair: a health-based reward based on the amount of damage sustained d is added per agent of $32 * (0.5)^d$ if $d <= 4$ else 0; and a "fast finish" reward of 16 if the given blue agent was killed before the 10th step with a decay of 1 for each step above 10.

In total, the maximum possible reward – though it may not necessarily be attainable given the randomized starting locations of the learning agents – varies as follows depending on the type of scenario the environment is configured for:

- $32 + 8 + 16 = 56$ for 1v1 2HP
- $32 + 20 + 16 = 68$ for 1v1 5HP
- $32 + 80 + 6 = 118$ for 1v1 20HP
- and $32 * 4 + 20 * 2 + 16 * 4 = 232$ for 2v2 5HP

Policies

First, we describe the architecture of each of our policy models. Then, we detail how node embeddings are generated for our GNNs from the environment observations.

Models: To measure the relative efficacy of GNN-based policies to learn abstract CO problems, we compare the average reward earned by four different policy networks: a baseline FF model and also three different GNN-based models.

1. An *FF* model which consists of one input layer, two hidden layers, and an output layer.
2. A *GCN-FF* model, in which the output of a GCN is concatenated with the original observation and fed through an FF network.

3. A *GT-FF* model, in which a GT and FF are combined in a similar way to the GCN-FF.
4. A *GATv2-FF* model, in which a GATv2 and FF are combined in a similar way to the GCN-FF.

For fairness, all of the models were designed to have nearly 50k parameters in total: the baseline and GT-FF had 49.4k parameters, the GATv2-FF had 50.8k parameters, and the GCN-FF had 48.5k parameters. The reason for this is to show that the inclusion of GNNs in a network can result in a performance increase even after normalizing for parameters. The following hyperparameters were chosen for each model: for the FF model, hidden layers of size 177 were used, and for all of the graph network-based models, hidden layers of size 169 were used, as well as four graph network layers with a hidden dimension of 4. The GATv2- and GT-FF models also used four attention heads. All models were also trained on the same hardware and with the same seeds.

Graph Node Embeddings: First, a graph of the same number of nodes and same connections as the environment's movement graph is constructed. Observations are then manually embedded at each of the nodes as a 6-dimensional node embedding. The embedding of a node i, $node_i$, is built in the following way:

$$node_i = (A_{t,i}, N_{red,i}, N_{blue,i}, R_{t+1,i}, B_{t+1,i}, [DIR_i])$$

In this scheme, $A_{t,i}$ is an indicator function for the presence of our RL agent being at node i at the current time t. Only one node on the graph will have this value set to 1 at once since, even in a MARL setting, we make predictions for a single agent at a time. We refer to this node as the Agent Node.

$N_{red,i}$ and $N_{blue,i}$ are counts of the number of red and blue agents at node i, respectively (since in this environment, multiple red and blue agents can overlap each other). $R_{t+1,i}$ and $B_{t+1,i}$ are indicator functions representing the *possibility* of a red agent appearing at node i at time $t+1$ and the *possibility* of a blue agent appearing node i at a time $t+1$. An agent is counted as having the *possibility* of appearing at a node i at a time $t+\tau$ if it is currently stationed τ or fewer degrees away from that node. $[DIR_i]$ is an indicator variable for whether or not any blue agent is on an adjacent node facing toward node i.

The reason for building node embeddings in this way is to ensure that the graph representation of the environment contains all the same information as the raw observations. In practice, these embedding could be learned from the raw observations instead of being manually constructed, but we decided to opt for the latter to better showcase our proof-of-concept.

Experiments

Evaluation

To compare each of the models, we first train each one ten different times using different seeds each time. To keep things simple, our experiments use seeds 0-9 inclusive. Each of those ten trained models is continuously evaluated during the training process (after each batch of 200 episodes) at seven different starting locations on the map. Given that evaluating a model at different starting locations is the same as evaluating it on seven different CO problems – since the optimal strategy differs at each of the different starting points – this means that in total, $10 \times 7 = 70$ different evaluations are run for each of the trained models. After training is over, we consider the evaluation score of each of the ten trained models to be the highest average evaluation score among all seven starting locations obtained at any point during the training process.

Each score is then compared to the score of the corresponding baseline result for that scenario (the FF model) via a 2-sample t-test. The resulting p-value of the t-tests done for each model in each scenario are provided in our tables, as well as a 95% confidence interval about each result.

Ablation Study

To test how GNN-based policies perform relative to pure FF policies at different levels of problem complexity and given different settings, we run each experiment multiple times to test the effects of the following factors.

Problem Complexity: The length of the sequence of actions that need to be made logically corresponds to the complexity of the problem at hand. Therefore, measuring performance metrics at different policy lengths allows us to compare model performances at different levels of problem complexity. In our experiments, setting the maximum agent HP is used as a proxy to determine model efficacy at different policy lengths. This is because episodes starting with low HP counts (e.g., 2HP, 5HP) can be expected to end sooner on average since agents die in just a few hits (and thus the team-wipe stopping condition can be met), whereas episodes starting with high HP counts (e.g., 20HP) will always end in 20 steps as a result of the max step stopping condition. To test a variety of policy lengths, experiments were run at initial HP counts of 2HP, 5HP, and 20HP. These HPs resulted in average episode lengths of 12.5, 17.5, and 20 steps, respectively.

Multiple Agents: Testing our models in a multi-agent setting is important to understand the applicability of graph networks in real-world scenarios involving either competition or collaboration. For that reason, in addition to the 1v1 simulations, 2v2 experiments were run at the 5HP setting.

Pooling Function: Generally, the use of GNNs ends with a pooling method (e.g., sum/max/mean) overall final node embeddings. In this paper, the output of this function is directly referred to as the GNN output. Action priors are then determined by concatenating this output with the original observation and applying an FF network and a softmax. In our experiments, a *local*, a *global*, and a *hybrid* pooling were tested in the 1v1 5HP scenario to compare their relative performances. The local pooling returns the embedding of

Model	Pool. Fn.	Reward@40k	p-val
FF	-	47.38 ± 16.26	-
GATv2-FF	global	**51.47 ± 20.86**	6e-3
	hybrid	47.98 ± 21.95	0.36
	local	49.54 ± 15.22	0.05
GCN-FF	global	48.87 ± 15.40	0.14
	hybrid	43.04 ± 12.61	0.99
	local	**51.78 ± 14.62**	6e-4
GT-FF	global	**53.70 ± 17.46**	1e-5
	hybrid	48.44 ± 15.03	0.21
	local	48.55 ± 20.36	0.23

Table 1: **Relative performance of GNN pooling functions.** Bold results indicate the best-performing combinations. - entries indicate that the given column does not apply to the given row.

the Agent Node, the global pooling returns mean node embedding among all nodes, and the hybrid pooling returns a concatenation of local and global pooling functions.

Results

Table 1 summarizes our exploratory experiments with pooling functions in the 1v1 5HP scenario. As compared to the baseline FF model, the global pooling performs 8.3% better, and the local pooling performs 5.4% better, with both methods obtaining statistical significance for at least one of the models. On the other hand, the hybrid pooling method not only underperformed relative to the baseline but also failed to significantly outperform it on any single model. Given that hybrid pooling contains information about both the local and global methods, its low performance is surprising and suggests that, in general, optimizing graph network parameters for multiple types of pooling functions simultaneously may make the training process for graph networks much harder. Since the hybrid pooling performance clearly lags behind the other two methods, the remainder of our experiments were carried out with local and global pooling only, and these results are shown in Table 2 and Table 3, where each section corresponds to a different scenario configuration.

Our main results using local pooling in Table 2 show that GNN-based policies tend to outperform the standard FF model with significance in the 1v1 5HP and 1v1 20HP scenarios but not in the 1v1 2HP scenario. This suggests that less-complex problems (such as the 2HP scenario) do not benefit as strongly from the use of GNNs as more complex ones (such as the 5HP or 20HP scenarios). This trend is best visualized in Fig. 2, which examines the average evaluation performance of the GCN-FF from each of the 7 starting points. Although the global pooling results (Table 3) are somewhat affected by this trend as well, the greater performance in the 1v1 2HP scenario indicates that global pooling may still perform well in less-complex scenarios as well.

Our 2v2 5HP scenario results show that GNN-based policies tend to outperform standard FF models in MARL scenarios by 6% and 2% for the local and global pooling settings, respectively. Although the variance of the 2v2 scenario outcomes dampens the statistical significance of their corresponding results, the overall better performance suggests that the inductive bias introduced by using GNNs to model the environment directly may have the ability to stimulate better collaborative learning. These results are also in line with those from previous works, which find GNNs to have a positive impact on learning collaborative behavior in MARL settings (Li et al. 2020; Shang et al. 2021).

Taking the average of the percentage performance improvements across all scenarios, we find that the GATv2-FF performed 3.51% better than the FF model, the GCN-FF

Scenario	Model	Reward@40k	p-val
1v1 2HP	FF	47.77 ± 16.81	-
	GATv2-FF	48.18 ± 17.74	0.39
	GCN-FF	**49.04 ± 15.40**	0.18
	GT-FF	48.22 ± 17.62	0.38
1v1 5HP	FF	47.38 ± 16.26	-
	GATv2-FF	49.54 ± 15.22	0.05
	GCN-FF	**51.78 ± 14.62**	6e-4
	GT-FF	48.55 ± 20.36	0.23
1v1 20HP	FF	49.90 ± 15.54	-
	GATv2-FF	48.97 ± 17.74	0.73
	GCN-FF	**55.44 ± 16.18**	4e-5
	GT-FF	51.47 ± 17.19	0.13
2v2 5HP	FF	126.05 ± 80.18	-
	GATv2-FF	133.88 ± 67.62	0.11
	GCN-FF	**135.58 ± 82.52**	0.08
	GT-FF	131.75 ± 82.70	0.21

Table 2: **Local pooling model performance across different scenarios.**

Scenario	Model	Reward@40k	p-val
1v1 2HP	FF	47.77 ± 16.81	-
	GATv2-FF	49.62 ± 15.44	0.09
	GCN-FF	47.54 ± 15.59	0.56
	GT-FF	**51.30 ± 13.11**	3e-3
1v1 5HP	FF	47.38 ± 16.26	-
	GATv2-FF	51.47 ± 20.86	6e-3
	GCN-FF	48.87 ± 15.40	0.14
	GT-FF	**53.70 ± 17.46**	1e-5
1v1 20HP	FF	49.90 ± 15.54	-
	GATv2-FF	51.41 ± 13.47	0.11
	GCN-FF	50.21 ± 15.46	0.40
	GT-FF	**52.02 ± 18.68**	0.07
2v2 5HP	FF	126.05 ± 80.18	-
	GATv2-FF	129.58 ± 69.35	0.29
	GCN-FF	**133.62 ± 91.20**	0.15
	GT-FF	125.60 ± 72.37	0.52

Table 3: **Global pooling model performance across different scenarios.**

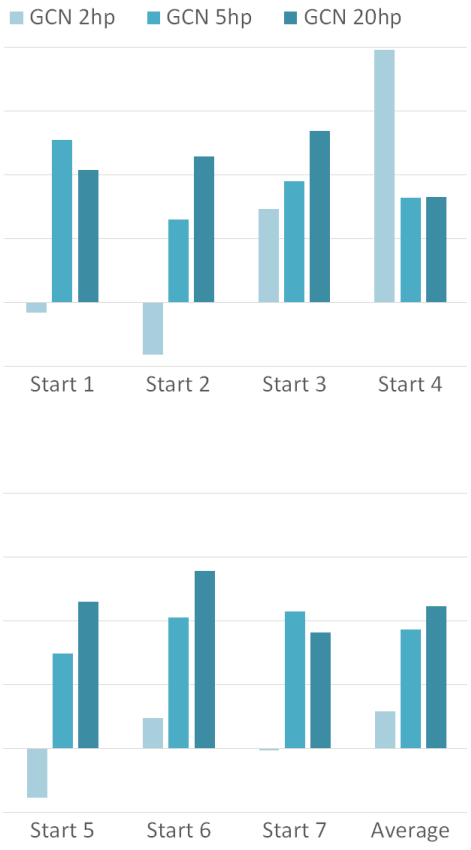

Figure 2: **Percent improvement of local pooling GCN performance relative to the baseline by starting locations.** The last graph shows the average performance increase across all 7 starting locations. Each horizontal tick represents a 5% difference from the baseline's performance at that starting location.

4.98% better, and the GT-FF 4.46% better. Therefore, the data shows that even simple GNN models (such as the GCN-FF) are well-suited for modeling an agent's environment in our scenarios and, conversely, that not all GNNs which employ attention mechanisms (such as the GT-FF) necessarily outperform baseline models to the same extent. Furthermore, our experience is that these attention-based models were much harder to tune and train, which may have contributed to this discrepancy. However, more complex scenarios may warrant attention mechanisms to achieve comparable improvements in performance.

Conclusion and Future Work

Our results show that leveraging GNNs in reinforcement learning experiments via modeling an agent's environment for proof-of-concept military training scenarios can yield significant performance gains when compared to employing only traditional feedforward neural networks when training a model to become a better sparring partner. Depending on the scenario, the best combination of GNN and pooling function is tied between the GCN with local pooling and the GT with global pooling. However, we generally find that GCNs have a practical edge due to their simplicity and speed. While dampened, these performance gains are also present in MARL settings, suggesting that learning collaborative behavior also benefits from using GNNs to model physical environments.

However, further investigation is needed to investigate why the addition (or exclusion) of certain node embeddings leads to better learning with GNNs using our methods. For example, although GNNs naturally propagate the information encoded in the two values $R_{t+1,i}$ and $B_{t+1,i}$ in our constructed node embeddings, their addition nevertheless resulted in a significant improvement to the pace of learning of the models, while the addition of more graph network layers did not. The ability to explain why this can occur would enhance the usability of our methods.

We plan to build on the results presented in this paper. Our projected future work toward the application of our results in adjacent problem spaces includes the following:

- **Latent Graph Generation:** Not all abstract CO problems have clear-cut, geographical graphs to represent the state space of their problem scenario. They do, however, all have at least one trivial state space graph representation, which corresponds to the links between problem states given a set of valid actions per state. Since this latter graph representation is often exponential in size, however, dynamic generation of learned, compact "latent" graph representations of a given CO problem is of great interest. If generated, latent graphs for other CO problems could be used in place of the geographical graphs used in this paper, which could extend our work's applicability to more problems.

- **Upstream Model Integration:** In practical settings, models are most valuable when they can easily make use of upstream results as features. Understanding how to interface upstream results to GNNs by devising a method to map them to individual node features would allow our method to better leverage information from the environment. For example, leveraging solutions to more abstract CO problems as features could help construct more effective node embeddings. If successful, such a pipeline could also support explaining why certain node embeddings work (or do not work.)

Acknowledgments

Research was sponsored by the Army Research Office and was accomplished under Cooperative Agreement Number W911NF-20-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation.

References

Babaeizadeh, M.; Frosio, I.; Tyree, S.; Clemons, J.; and Kautz, J. 2017. Reinforcement learning through asynchronous advantage actor-critic on a GPU. In *International Conference on Learning Representations*.

Bresson, X., and Laurent, T. 2017. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

Brody, S.; Alon, U.; and Yahav, E. 2022. How attentive are graph attention networks? In *International Conference on Learning Representations*.

Cappart, Q.; Chételat, D.; Khalil, E. B.; Lodi, A.; Morris, C.; and Veličković, P. 2021. Combinatorial optimization and reasoning with graph neural networks. In Zhou, Z.-H., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 4348–4355. International Joint Conferences on Artificial Intelligence Organization. Survey Track.

Devailly, F.-X.; Larocque, D.; and Charlin, L. 2021. Ig-rl: Inductive graph reinforcement learning for massive-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*.

Dwivedi, V. P., and Bresson, X. 2021. A generalization of transformer networks to graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *International conference on machine learning*, 1263–1272. PMLR.

Joshi, C. K.; Cappart, Q.; Rousseau, L.-M.; and Laurent, T. 2022. Learning the travelling salesperson problem requires rethinking generalization. *Constraints* 27(1–2):70–98.

Kool, W.; van Hoof, H.; and Welling, M. 2019. Attention, learn to solve routing problems! In *International Conference on Learning Representations*.

Kurin, V.; Igl, M.; Rocktäschel, T.; Boehmer, W.; and Whiteson, S. 2020. My body is a cage: the role of morphology in graph-based incompatible control. *arXiv preprint arXiv:2010.01856*.

Li, S.; Gupta, J. K.; Morales, P.; Allen, R.; and Kochenderfer, M. J. 2020. Deep implicit coordination graphs for multi-agent reinforcement learning. *arXiv preprint arXiv:2006.11438*.

Liang, E.; Liaw, R.; Nishihara, R.; Moritz, P.; Fox, R.; Goldberg, K.; Gonzalez, J. E.; Jordan, M. I.; and Stoica, I. 2018. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning (ICML)*.

Liu, L.; Gurney, N.; McCullough, K.; and Ustun, V. 2021. Graph neural network based behavior prediction to support multi-agent reinforcement learning in military training simulations. In *2021 Winter Simulation Conference (WSC)*, 1–12.

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

Shang, W.; Espeholt, L.; Raichuk, A.; and Salimans, T. 2021. Agent-centric representations for multi-agent reinforcement learning. *arXiv preprint arXiv:2104.09402*.

Ustun, V.; Kumar, R.; Reilly, A.; Sajjadi, S.; and Miller, A. 2020. Adaptive synthetic characters for military training. *I/ITSEC*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph attention networks. In *International Conference on Learning Representations*.

Wang, T.; Liao, R.; Ba, J.; and Fidler, S. 2018. Nervenet: Learning structured policy with graph neural networks. In *International conference on learning representations*.