MARNN: A New Method for Multi-agent Reinforcement Learning

Aleksander Kalacun[1], **Iztok Fister, Jr.**[1]

[1] *Faculty of Electrical Engineering and Computer Science, University of Maribor*
E-mail: aleksander.kalacun@student.um.si

Abstract

This paper presents a new method for tackling dynamic partially observed multi-agent Markov decision processes with stochastic elements. We use this formulation to define and simulate real-world problems and present them on graphs. The current multi-agent reinforcement learning algorithms struggle in changing environments which lack consistency. The core idea of the algorithm is based on the deep autoregressive model GraphRNN, achieving long-term goal memorization and greater resilience to stochastic changes in environments. The policy making is based on graph convolution network encoders. Based on observations and information communicated from other agents via graph attention network encoders, agents decide on the best next state. To demonstrate its capabilities, new graph based environments were implemented. Using them, we evaluated the performance of multi-agent reinforcement learning algorithms in real-world type problems as an effective benchmark.

1 Introduction

Reinforcement learning (RL) has regained a lot of popularity compared to other deep learning paradigms in recent years [1]. Supervised learning may potentially be reaching the plateau and troubles reliability of solutions with human biases embedded in training datasets [2]. In contrast, the RL domain allows intelligent agents to come up with solutions in an environment by themselves. This results in avoiding any previous biases and producing creative, possibly yet unseen solutions. Although MARL progress has been driven by autonomous driving, recent LLM research renewed interest, as deep reasoning models increasingly rely on RL loops [3].
Multi-agent path finding problems have long been solved with optimal algorithms [4]. Still, we find it to be an illustrative benchmark for demonstrating the performance of algorithms, and compare it with the current state-of-the-art. Being abstracted as a graph, such environment setting is also flexible to be reshaped into a real-world problem. Examples include networking, hardware optimization, robotics, and drone swarms (from food delivery to space exploration). More distant fields include synthetic biology, finance, and sport simulations [5].

We explore the yet less known but promising topic of dynamic partially observed stochastic environments. While most algorithms struggle in changing settings where the Markov property does not hold [5], our robust method is resilient to such changes. Our main contributions in this paper are:

- New method (algorithm) of tackling dynamic partially observed stochastic Markov decision problems. The method was compared, benchmarked, and analyzed with the current space of state-of-the-art algorithms.
- Creation of new, custom-made dynamic decentralized partially observed environments, flexible for any further use to benchmark numerous MARL algorithms.

This paper is structured as follows: In Section 2, we present related work relevant to the research areas central to our study. Section 3 introduces key theoretical aspects of the topic. In Section 4, we describe our new environment and agent architecture in detail. Section 5 presents the experiments and results. Finally, the paper is concluded in Section 6.

2 Related Work

2.1 Multi-agent Reinforcement Learning and Communication for Path Finding

Approximation-based algorithms (e.g., RL) address the problematic (unbearable) time complexity of optimal multi-agent coordination algorithms (e.g. Dynamic Programming). Modeling real-world problems in simulated environments often requires taking into account stochastic (random) events and incomplete observations. Thus, partially observed Markov decision environments [6] are studied. Dynamic changes, random communication noise, and other random events are needed, in order to train real-world applicable models.
The problem of partial observability is overcome with inter-agent communication. Some works [7] have shown the use of a graph attention network (GAT) [8] approach to create a communication network with supervised learning. The agents are able to assign different attention values to different agents, depending on the agents' positions, observations, etc. The parameters and message encoding and decoding are learnt in the environment.

2.2 Machine Learning on Graphs

We focus on generative graph algorithms. Given a learning process, these algorithms create realistic graphs following training. Pioneering generative graph models is the employment of recurrent neural networks (RNNs) in graph setting, first introduced by the concept of GraphRNN [9]. There, RNNs were used for generating unique graphs by considering the graph as a sequence of edges and nodes. However, node classification tasks are approached more commonly with Graph neural networks (GNN) [10], due to their improved expressiveness. This approach considers a graph structure to be presented as layers of neighborhoods around a concerning node. In the following work [11], the authors showed a combination of these ideas, where the GNN layers increased expressiveness in sequence models (solving dynamic graph problems). Work [12] introduced concept of world models in MARL settings.

3 Theoretical Background

3.1 Markov Decision Processes (MDPs)

Markov Decision Process (MDP) is a mathematical framework for single-agent sequential decision-making [5, 2.2]. It can be extended further for multiple agents and stochastic environments. Formally, an MDP is defined as a tuple (S, A, T, R, γ), where:

- S is a finite set of states and A is a finite set of actions,
- $T : S \times A \times S \to [0, 1]$ is the transition probability function, denoting $T(s'|s, a)$, the probability of transitioning to state s' from s by taking action a
- $R : S \times A \times S \to \mathbb{R}$ is the reward function with $\gamma \in [0, 1]$ being the discount factor.

The goal is to determine the optimal policy $\pi^* : S \to A$ that maximizes the expected cumulative discounted reward:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t), s_{t+1}) \mid s_0 = s\right], \quad (1)$$

The Markov property: $P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...) = P(s_{t+1}|s_t, a_t)$. However, in dynamic environments with changing states this property is violated.

3.2 Partially Observable Markov Decision Processes (POMDPs)

POMDPs extend MDPs to situations with incomplete or noisy state information. Formally, a POMDP extends MDP and is defined by the tuple $(S, A, T, R, \Omega, O, \gamma)$ [6, 2.2], where: Ω is a finite set of observations and $O : S \times A \times \Omega \to [0, 1]$ is the observation probability function $O(o|s', a)$. Agents maintain a belief state $b \in \Delta(S)$ (a probability distribution over states), updated via Bayes' rule. The objective is to find a policy π that maps belief states to actions to maximize:

$$V^{\pi}(b) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(b_t), s_{t+1}) \mid b_0 = b\right]. \quad (2)$$

3.3 Graph Convolution Networks for MARL

Given a graph $G = (V, E)$ where nodes represent states and edges represent possible transitions, we employ Graph Convolutional Networks (GCNs) [13] for state representation learning:

$$h_v^{(k+1)} = \sigma\left(W^{(k)} \cdot \text{AGG}\left(\{h_u^{(k)} : u \in \mathcal{N}(v)\}\right)\right) \quad (3)$$

where $h_v^{(k)}$ is the hidden representation of node v at layer k, $\mathcal{N}(v)$ represents the neighbors of v, and AGG is some aggregation function. Matrix W is matrix of learnable weights and σ is some activation function.

4 Proposed method

4.1 Environment Description

We consider a complex partially observed environment in form of a graph. We found the inspiration in the Rware environment [5], generalized into a graph form. In such problems, we are facing the sparse reward problem, a common obstacle in the MARL paradigm [5]. In our custom-built environment (see Fig.1) agents have to carry packages from package states to the goal state, receiving a reward of +100 if able to do so. After a package is delivered, a new random state is assigned as a package state making our environment dynamic. Agents also face a -50 penalty for colliding at the same node and timestep. In each time step the agents observe the type of state of the node and all neighbor nodes (possible types being goal state, package state, and normal state). To force agents to make legal moves we give a reward of -10 for every illegal move. The environment is randomly regenerated with given edge probability for every new iteration.

4.1.1 Enhancing the Base environment

Given the premise of building an environment that imitates real-world problems, we added the following features: (i) we force agents to pick neighbour states to avoid reward mixing. (ii) The temporal dynamics of edge removal and creation and package changing (iii) agents memory decay (iv) asymmetric knowledge (simulating the addition of new agents over time), (v) observation noise (randomly corrupting agents' observations).
These features combined harm the simple observability of the environment, effectively lowering the performance of classical algorithms while simulating the real world more realistically.

4.2 Agent Architecture

The agents are built out of Long Short-Term Memory (LSTM) [14] cells in the actor critic architecture. The reasoning behind using recurrent neural networks is their capability to memorize their tendencies over multiple time steps. The hidden state is being transferred over time, countering an ever-changing dynamic environment and stochastic events where the Markov property does not hold. LSTMs were used instead of vanilla RNN for their better consistency and gates allowing forgetting of lost graph connections. Both, actor and critic are built out of LSTM neural networks. The difference being that critic

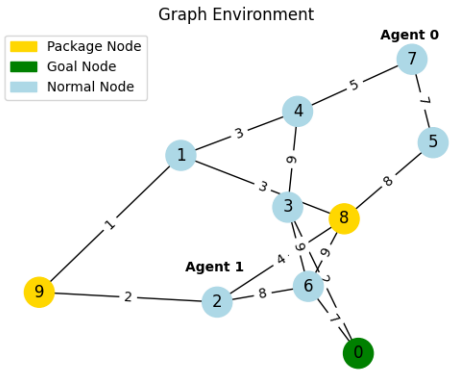


Figure 1: Graph based MARL environment.

outputs a scalar of predicted reward given the action sequence while actor outputs best possible next step following the policy. Formally, LSTM networks implicitly maintain belief representations (Section 3) through hidden states h_t^i for each agent i.

For the purposes of communication, we use the GAT [8] structure. The agents first encode their observations, and then pass them through a graph communication network. Attention is learnt to be given to the agents that carry the most important information.

Improving the expressiveness of each LSTM cell, we introduce Graph Convolution Network (GCN) encoders. Note, we define the observation space of only one layer ahead. Contrary to GraphRNN, GCN performs classification at every step among the neighbor states (by forming graph with central state and one layer of neighbor states). Thus, considering dynamic graph setting with stochastic elements, optimal policies must condition on history.

MARNN approximates $\pi^*(a_t|s_{0:t}, a_{0:t-1})$ with $\pi(a_t|h_t)$ where h_t is a learnt compression of history via LSTMs. Additionally, world model based on received information is gradually built. This helps agents plan ahead with limited observation. Note that world model is not shared among agents nor it is input with observations, thus we consider it less relevant. See algorithm 1.

Unlike reactive policies $\pi(a|o_t)$, MARNN's memory augmented policy $\pi(a|h_t)$ can: (i) remember previous package locations after relocation, (ii) adapt to edge deletions by recalling alternative paths, and (iii) maintain coordination despite observation noise. In summary, architecture addresses challenges in the following ways:

LSTM: Compresses unbounded history $o_{0:t}$ into fixed-size $h_t \in \mathbb{R}^d$, **GCN:** Handles variable graph topology via permutation-invariant aggregation, **GAT:** Enables selective communication without fixed topology assumptions.

5 Results

Hypotheses: We expect MARNN to: (H1) maintain performance in dynamic settings better than memoryless baselines, (H2) show higher sample efficiency due to memory, (H3) exhibit longer convergence time due to architectural complexity. To demonstrate the performance of MARNN we compare it to other state-of-the-art algorithms in our custom made environment using the EPyMARL framework [15].

Algorithm 1 MARNN

Input: Graph G_t, observations $\{o_t^i\}$, hidden states $\{h_{t-1}^i\}$
for each agent i **do**
  $h_t^i \leftarrow \text{LSTM}(o_t^i, h_{t-1}^i)$ ▷ Memory update
  $z_t^i \leftarrow \text{GCN}(G_t, \{h_t^j : j \in \mathcal{N}(i)\})$ ▷ Graph encoding
  $m_t^i \leftarrow \text{GAT}(\{h_t^j : j \neq i\})$ ▷ Communication
  $a_t^i \sim \pi(z_t^i, m_t^i)$ ▷ Action selection
end for
return actions $\{a_t^i\}$

5.0.1 *Experimental Setup*

We evaluated three graph environments: *small* (2 agents, 10 nodes), *medium* (3 agents, 15 nodes), *large* (6 agents, 30 nodes), and the *enhanced* (3 agents, 15 nodes, all dynamic features). Baselines: IQL [16], QMIX [17], VDN [18], MARNN (hidden dim: 64, LSTM layers: 2, learning rate: 5e-4, optimizer: RMSprop). Metrics: mean episode return and sample efficiency (measures performance with respect to environment interactions (visualized as area under learning curve), implying how quickly the algorithm learns). All experiments use 3 seeds, 140k environment steps. Table 1 shows the mean episode reward, and Figures 2 and 3 plot them versus environmental steps plots, demonstrating the algorithms' performance over time.

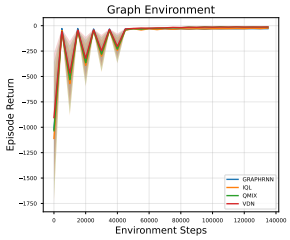
Table 1: Benchmark results for the base graph env.

Algorithm	Medium	Small	Large
IQL [16]	-20.6 ± 1.1	-6.0 ± 1.0	-56 ± 4.1
QMIX [17]	-13.6 ± 3.1	-6.5 ± 2.0	-45 ± 6.9
VDN [18]	-12.4 ± 0.9	-4.5 ± 2.0	-59 ± 0.7
MARNN	-25.0 ± 3.2	-10.5 ± 3.7	-79 ± 13

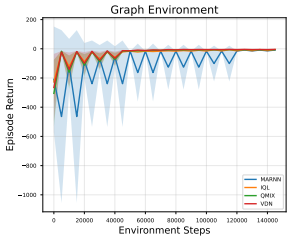
Table 2: Benchmark results for enhanced env. and sample efficiency (SE).

Algorith	Enh. Mean	Enh. SE	Med. SE
IQL	-1.8	0.88246%	0.90784%
QMIX	-2.1	0.87823%	0.87320%
VDN	-3.3	0.70847%	0.89575%
MARNN	-4.0	0.77477%	0.91802%

Considering the results for enhanced environment (Table 2, Fig. 3a), trends still seems to hold.



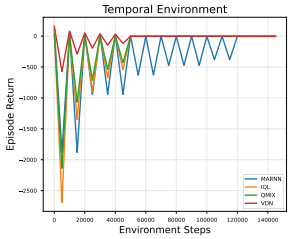
(a) Medium environment



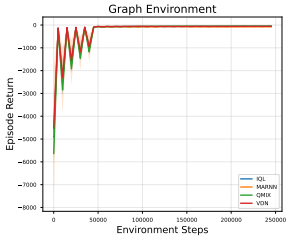
(b) Small environment

Figure 2: Experimental results on large and small env.

The results look promising when taking the sample efficiency (SE) into account (Table 2).



(a) enhanced environment



(b) Large environment

Figure 3: Experimental results on large and small env.

6 Conclusion and further work

Analysis: The experiments showed that our method is very close to other state-of-the-art methods while still not surpassing any performance wise. Note that, from the plots, we also read high variations among all the runs of the algorithms (visible as shade). Analyzing the plots, MARNN generally takes longer to settle on the final solution (also explainable as more diverse exploration). Consider this as a consequence of a complex algorithm structure with many components. Thus, notably, promising results come from the sample efficiency metric, where MARNN performed better than VDN, and was close to the rest of the algorithms. Considering the medium graph environment setting, the MARNN mean sample efficiency outperforms all the state-of-the-art algorithms. This overall validates H2 and H3: the architecture learns useful representations quickly but struggles with credit assignment across LSTM, GCN, and GAT components. Although our proposed method did not outperform the current state-of-the-art, the gap was surprisingly slim (even considering that the environment characteristics were tailored so as to favor MARNN's performance). We believe that advanced MARL approaches could further improve our method, potentially outperforming standard methods in niche environments. When building our environment, we sought balance between a theoretically simple environment and an accurate representation of real-world scenarios. We encourage further studies on more complex graph environments. Possible extensions to our environment include limited energy, package expiration, partial deliveries, and weighted rewards. We believe that if a deeper observation horizon is considered, the improvements of the GCN classification would be even greater. On the other hand, we observe all the results to be negative. This implies a too complex environment, testing which algorithms make the least mistakes. One of the aspects of reinforcement learning training we neglected is hyperparameter tuning. We acknowledged this variable early in benchmarking, but later decided that this brings another dimension of inconsistency between different algorithms runs. However, using an auto-tuning framework [19] might eliminate the uncertainties of such hyperparameter tuning process and enable the algorithms to reach their peak performance.

References

[1] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," 2nd ed., MIT Press, 2018.

[2] V. Udandarao et al., "No "Zero-Shot" Without Exponential Data: Pretraining Concept Frequency Determines Multimodal Model Performance"

[3] D. Guo et al. "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning,"

[4] G. Sharon et al. "Conflict-Based Search for Optimal Multi-Agent Path Finding," Artificial Intelligence, vol. 219, pp. 40–66, 2015.

[5] S. V. Albrecht, F. Christianos, and L. Schäfer, *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024.

[6] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs*. Springer, 2016.

[7] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning," in *IROS*, 2020

[8] P. Veličković et al., "Graph attention networks," in *ICLR*, 2018.

[9] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "GraphRNN: Generating realistic graphs with deep auto-regressive models," in *ICML*, 2018

[10] F. Scarselli et al. "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, 2009.

[11] A. Pareja et al., "EvolveGCN: Evolving graph convolutional networks for dynamic graphs," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 04, Apr. 2020.

[12] C. Guestrin et al., "Coordinated Reinforcement Learning", *ICML*, 2002

[13] T. Kipf, M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks", *ICLR*, 2017

[14] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Comput., vol. 9, no. 8, 1997.

[15] G. Papoudakis et al., "Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks," *NeurIPS*, 2024.

[16] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit Q-learning," in *ICML*, 2021,

[17] T. Rashid et al., "QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *ICML*, 2018

[18] P. Sunehag et al., "Value-decomposition networks for cooperative multi-agent learning," in *AAMAS*, 2018

[19] L. Pečnik and I. Fister, "NiaAML: AutoML framework based on stochastic population-based nature-inspired algorithms," *J. Open Source Softw.*, vol. 6, no. 61, Art. no. 2949, 2021