APPROVAL SHEET

Title of Thesis: Dynamic Edge Weighting in Relational Graph Convolutional Networks: Enhancing Sample Efficiency via Graph Attention in Reinforcement Learning

Name of Candidate: Prakhar Dixit
Masters of Science, 2024

Thesis and Abstract Approved: _____

Dr Tim Oates
Professor
Department of Computer Science and
Electrical Engineering

Date Approved: _____

Curriculum Vitae

Name: Prakhar Dixit.

Permanent Address: MY-FULL-ADDRESS.

Degree and date to be conferred: Masters of Science and December 2024.

Date of Birth: June 25th 1997.

Place of Birth: India.

Secondary Education: Army Public School,Lucknow, India.

Collegiate institutions attended:

University of Maryland Baltimore County, Masters of Science, 2024.
SRM Institute of Science and Technology, Bachelors of Technology in Computer Science,2019

Major: Computer Science.

Research publications:

Navardi, Mozhgan, Prakhar Dixit, Tejaswini Manjunath, Nicholas R. Waytowich, Tinoosh Mohsenin, and Tim Oates. "Toward real-world implementation of deep reinforcement learning for vision-based autonomous drone navigation with mission." UMBC Student Collection (2022).

Manjunath, Tejaswini, Mozhgan Navardi, Prakhar Dixit, Bharat Prakash, and Tinoosh Mohsenin. "ReProHRL: Towards multi-goal navigation in the real world using hierarchical agents." arXiv preprint arXiv:2308.08737 (2023).

ABSTRACT

Title of Thesis: Dynamic Edge Weighting in Relational Graph Convolutional Networks: Enhancing Sample Efficiency via Graph Attention in Reinforcement Learning Learning

Prakhar Dixit, Masters of Science, 2024

Thesis directed by: Dr. Tim Oates, Professor
Department of Computer Science and
Electrical Engineering

Reinforcement Learning (RL) has achieved remarkable success across a variety of domains, yet sample complexity remains a challenge, especially when actions are taken in the real world. To improve the effectiveness of RL, researchers have turned to deep neural networks that learn useful representations of the data that can improve and speed up the learning process. In particular, Graph Neural Networks (GNNs) have emerged as a powerful tool for handling data with inherent graph structures, such as social networks, communication systems, and biological networks. Despite these advances, traditional approaches using Relational Graph Convolutional Networks (R-GCNs) often rely on statically weighted graphs, which do not adequately capture the dynamic nature of many RL environments. To overcome this limitation, in this study, we propose an integration of Graph Attention within an R-GCN framework for model-free RL algorithms called REAGLE (Reinforcement learning with Edge Attention Networks) . This technique applies dynamic edge weighting, treating connections (relations) unequally based on their current relevance. By dynamically prioritizing relations, our model focuses computational resources on the most crucial information at each step, thereby improving sample efficiency. We validate our approach across different Minigrid environments, including the Boxworld domain, a grid-world navigation challenge designed to evaluate relational reasoning capabilities. Our results demonstrate that REAGLE surpasses conventional graph-based reinforcement learning algorithms by

achieving faster learning, better decision quality, and significantly lower sample complexity.

$$MER = \frac{1}{N} \sum_{i=1}^{N} R_i$$

Dynamic Edge Weighting in Relational Graph Convolutional Networks: Enhancing Sample Efficiency via Graph Attention in Reinforcement Learning

by

Prakhar Dixit

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
M.S. in Computer Science
2024

To my dearest mother, Thank you for your unwavering love and support throughout my academic journey and for always believing in me, even when no one else did. This thesis is a testament to your sacrifices and endless encouragement. I am grateful for everything you have done for me.Love you always.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to all those who have contributed to the completion of this thesis. I am honored to acknowledge their invaluable support and guidance throughout this journey.

First and foremost, I am sincerely grateful to my research advisor and chair, Dr. Tim Oates. His dedication, motivation, perseverance, and profound knowledge have been instrumental in the successful completion of this thesis. I extend my heartfelt thanks to him for his exceptional guidance, unwavering support, and insightful foresight. Without his mentorship, this thesis would not have been possible.

Additionally, I would like to extend my sincere gratitude to Dr. Tinoosh Mohsenin for being a part of my early days in research, supporting me wholeheartedly, and providing valuable opportunities while having faith in me. Her invaluable suggestions and guidance have greatly contributed to the progress of this thesis.

I would also like to express my appreciation to all the present and past members of the Coral Lab. Their direct and indirect contributions have played a significant role in shaping and refining the concepts presented in this thesis. Their unwavering support and encouragement have been instrumental in my development as a researcher.

I am sincerely grateful to my committee member, Dr. Manas Gaur, for agreeing to serve on my thesis committee and for his support and guidance throughout my master's program, as well as for his invaluable contributions to my thesis.

Lastly, I would like to express my deep appreciation to my family and friends for their constant support and encouragement. Their unwavering belief in my abilities and their presence throughout this journey have been a source of strength and motivation. I am truly grateful for their love, understanding, and encouragement.

To all those mentioned and the countless others who have contributed in various ways, I owe my heartfelt thanks. It is because of your support, guidance, and encouragement that I have been able to achieve this significant milestone in my academic journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Reinforcement learning (RL) [Sutton & Barto1998] is a powerful method in artificial intelligence [Hunt2014] for teaching agents to make decisions in uncertain situations. It stands as the third branch of machine learning alongside supervised and unsupervised learning. The core idea of RL is that the learning process to solve a decision-making problem involves a sequence of trial and error, where the agent, the main actor in RL, learns to distinguish valuable decisions from penalizing ones. This learning is guided by a reward signal [Dayan & Balleine2002], which provides feedback on the desirability of the actions taken. This interaction closely mirrors how human beings and animals learn and adapt their behaviors in the real world.

In an RL scenario, the agent interacts with its environment and takes actions based on its observations. These actions lead to certain outcomes, and the environment provides feedback in the form of rewards or penalties. The primary objective for the agent is to maximize the cumulative reward over time, essentially learning the best strategies to achieve this goal. By continuously interacting with the environment and receiving rewards, the agent gradually improves its decision-making policy [Cohen & Ranganath2007].

This technique has demonstrated remarkable capabilities across diverse domains such as robotics [Kober, Bagnell, & Peters2013], game AI [Torrado *et al.*2018], and natural lan-

guage processing [Sharma & Kaushik2017]. A persistent challenge in the field, however, is achieving low sample complexity—enabling efficient learning with fewer interactions from the environment. This challenge is pronounced in real-world applications where data acquisition can be costly or impractical [Kakade2003].

To improve the efficiency and effectiveness of RL, researchers have turned to representation learning [Bengio, Courville, & Vincent2013], which involves learning useful representations of the data that can simplify the learning process. Representation learning techniques, such as those used in deep learning, have proven to be extremely effective in capturing complex patterns in data. In particular, Graph Neural Networks (GNNs) have emerged as a powerful tool for handling data with inherent graph structures, such as social networks, communication systems, and biological networks.

In many real-world applications, data often exhibit these inherent graph structures, characterized by entities and their interactions represented as nodes and edges in a graph. Traditional graph-based RL methods often utilize static graphs, which are suitable for environments with stable, unchanging dynamics but are less effective in dynamic scenarios where the significance of nodes and edges can shift unexpectedly.

To address the limitations of existing methods, this thesis introduces a novel approach that combines Graph Attention Networks (GAT) with Relational Graph Convolutional Networks (R-GCN) within an off-policy Actor-Critic Reinforcement Learning framework [Liu, Song, & Zhang2023]. A key innovation of this method is the dynamic adjustment of graph edge weights based on their relevance to the current decision-making context. This is achieved by computing attention coefficients that adaptively prioritize critical environmental factors, thereby enhancing decision-making under uncertainty.

This approach significantly boosts computational efficiency by dynamically re-prioritizing connections between nodes at each decision-making step, ensuring the model's focus is maintained on pertinent information. This reduces the number of environment

interactions (samples) needed to learn an effective policy.

By introducing a novel graph-based model that addresses the challenge of high sample complexity, this work contributes significantly to the development of more robust and efficient RL systems capable of operating in dynamic real-world environments, thus pushing the boundaries of what artificial intelligence can achieve.

1.1 Contributions

The main contributions of this thesis are summarized as follows:

- **Improved Convergence Time**: Our framework reduces the number of environment interactions (samples) the agent needs to learn an effective policy, resulting in a faster convergence time for tasks compared to other methods.
- **Dynamic Edge Weighting in Graph-Based Reinforcement Learning**: We introduce a mechanism to dynamically update edge weights of nodes based on the current state of the agent and its environment using attention. By adjusting edge weights, we can emphasize or de-emphasize specific relationships, allowing the model to focus on the most relevant information for the current decision.
- **Evaluation and Validation**: We validate our approach through simulations with increasing levels of difficulty in complex environments. Our method demonstrates superior performance over conventional RL algorithms, achieving faster learning, better decision quality, and significantly lower sample complexity.

1.2 Overview

This thesis is divided into three parts. In the first part, we discuss the background and related work, defining all relevant terms and explaining each one. We also highlight some

past related work. The second part consists of the proposed approach and the architecture. The final part describes the environments used to test the approach, followed by the results and discussion.

Chapter 2

BACKGROUND AND RELATED WORK

In this chapter we provide a comprehensive background on the key concepts and related work that form the foundation of our research. We define and elaborate on several important terms and concepts, including relational graphs, Relational Graph Convolutional Networks (R-GCN), Graph Attention Networks (GAT), and reinforcement learning. Additionally, we review related work to contextualize our approach and highlight significant contributions in the field.

2.1 Background

2.1.1 Relational Graphs

A relational graph is defined as a structured, directed graph with multiple labels, represented symbolically as $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ and is used to represent relational data, where:

- $\mathcal{V}$ encompasses the collection of nodes, each symbolizing an entity.
- $\mathcal{R}$ comprises a set of labels that classify the types of relationships existing between the entities.
- $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{R} \times \mathcal{V}$ represents the collection of directed edges, each annotated with a type of relational label.

Within this framework, each relationship is encapsulated by a tuple (a, r, b), signifying:

- $a \in \mathcal{V}$ as the source entity,
- $r \in \mathcal{R}$ as the type of connection,
- $b \in \mathcal{V}$ as the recipient entity.

This tuple denotes a directional link from entity a to entity b under the classification r. The design of relational graphs enables the detailed modeling of complex entity interactions, providing an invaluable tool for applications ranging from network analysis to data integration.

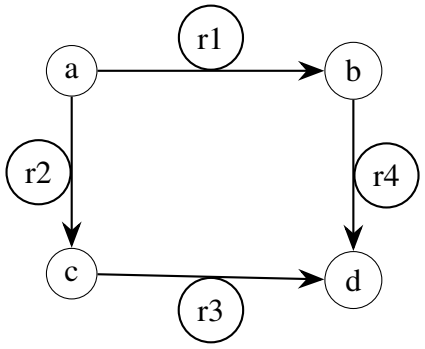

FIG. 2.1. An example of a relational graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$. Nodes $\mathcal{V}$ represent entities (a, b, c, d), edges $\mathcal{E}$ represent directed relationships, and labels $\mathcal{R}$ denote the types of relationships (r1, r2, r3, r4).

2.1.2 Role of Relational graphs

Relational graphs offer a significant advantage in reinforcement learning, particularly in environments where understanding the relationships between entities is critical for effective decision-making. By representing entities as nodes and their connections as edges with

specific types of relations, relational graphs capture the intricate structure and interactions within the environment. This approach contrasts with vector-based states, which flatten all information into a single vector, potentially losing valuable contextual information.

For instance, consider a social network in a multi-agent environment. Here, nodes represent individual agents, such as users in a social platform, while edges represent relationships like ”friend”, ”follower”, or ”blocked”. The relational graph explicitly captures the varying degrees and types of connections, enabling a model to better understand social interactions, such as the difference between close friendships and mere acquaintances. In contrast, using vector-based states by representing the environment using vectors would require encoding each agent and their relationships as a flat vector, which would lose the explicit relational structure and may obscure the importance of different types of relationships.

Another example could be in the context of hierarchical knowledge representation, such as in knowledge graphs. Nodes represent entities like people, places, or concepts, while edges represent relationships such as ”is_a”, ”part_of”, or ”owned_by”. This allows for an explicit representation of hierarchical and complex relationships, which is crucial for reasoning about the connections between different entities. A vector-based approach, on the other hand, would encode each entity and its relationships into a high-dimensional vector, potentially hiding the structure and making it more challenging to understand the hierarchy or connections between entities.

In this thesis, relational graphs form the backbone of the input structure upon which further learning and decision-making processes are built:

- **Input Graph Construction:** The environmental data is transformed into a relational graph using predefined relational determination rules.

  For example [”is_front”, ”is_back”, ”is_left”, ”is_right”] determines object positions

relative to the agent's direction.

- **Feature Integration and Attention Mechanism:** Node features are integrated into the graph, and processed through graph convolution techniques combined with an edge-wise attention mechanism to derive meaningful representations.

- **Decision Making:** The constructed graph serves as input to our reinforcement learning agent. By understanding the complex relationships between entities within the environment, the agent can make more informed decisions, leading to improved performance.

By leveraging the attention mechanism within relational graphs, our approach effectively captures the intricate relationships within complex environments, leading to better learning efficiency and improved decision-making accuracy for our reinforcement learning agent.

2.1.3 Relational Graph Convolutional Networks

Learning with graphs is a powerful method for capturing the complex relationships and structures inherent in many types of data [Zhang *et al.*2019]. The unique capability of graphs enables capturing the structural relations among data, and thus allows for harvesting more insights compared to analyzing data in isolation. However, it is often very challenging to solve the learning problems on graphs, because (1) many types of data are not originally structured as graphs, such as images and text data, and (2) for graph-structured data, the underlying connectivity patterns are often complex and diverse [Cho, Alahari, & Ponce2013]. Representation learning has achieved great success in recent years, providing a potential solution to these challenges by learning the representation of graphs in a low-dimensional Euclidean space, preserving the graph properties. Although tremendous efforts have been made to address the graph representation learning problem, many approaches still suffer

from their shallow learning mechanisms. Recently, deep learning models on graphs, such as Graph Neural Networks (GNNs), have emerged in machine learning and other related areas, demonstrating superior performance in various problems leading to the development of Graph Convolutional Networks (GCNs) [Kipf & Welling2016]. GCNs extend the concept of convolution from grid data (like images) to graph-structured data, enabling the application of deep learning techniques to graph-based problems.

Symbol	**Definition**
$h_i^{(l)}$	Hidden state of node i at layer l
$\mathcal{N}_i$	Set of neighboring nodes of node i
c_i	Normalization constant for node i
$W^{(l)}$	Weight matrix for layer l
σ	Non-linear activation function
$W_0^{(l)}$	Weight matrix for the self-loop
$\mathcal{R}$	Set of all relation types
$\mathcal{N}_i^r$	Set of neighboring nodes of node i under relation r
$c_{i,r}$	Normalization constant for relation r
$W_r^{(l)}$	Weight matrix for relation type r

Table 2.1. Notation table for Relational Graph Convolution Networks

In a Graph Convolutional Network (GCN), the hidden representation for each node i at the $(l+1)^{\text{th}}$ layer is computed by aggregating information from its neighbors. Mathematically, this is expressed as:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i} \frac{1}{c_i} W^{(l)} h_j^{(l)}\right) \tag{2.1}$$

where:

- $h_i^{(l)}$ represents the hidden state of node i at layer l.
- $\mathcal{N}_i$ denotes the set of neighboring nodes of node i.
- c_i is a normalization constant, often chosen as the degree of node i (i.e., the number of neighbors).
- $W^{(l)}$ is the weight matrix for layer l.
- σ is a non-linear activation function, such as ReLU.

GCNs work well with homogeneous graphs where all edges represent the same type of relationship. However, many real-world graphs are heterogeneous, containing multiple types of relationships. To address this, Relational Graph Convolutional Networks (R-GCNs) were introduced [Schlichtkrull *et al.*2017].

The key difference between GCN and R-GCN is the handling of edge types. In GCNs, the weight $W^{(l)}$ in equation (1) is shared by all edges. In contrast, R-GCNs use different weights for different edge types, allowing the model to capture the semantics of various relations. Specifically, the hidden representation of node i at the $(l+1)^{\text{th}}$ layer in R-GCN is computed as:

$$h_i^{(l+1)} = \sigma\left(W_0^{(l)} h_i^{(l)} + \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)}\right) \tag{2.2}$$

where:

- $h_i^{(l)}$ represents the hidden state of node i at layer l.
- $W_0^{(l)}$ is the weight matrix for the self-loop (relation of the node with itself).
- $\mathcal{R}$ denotes the set of all relation types.

- $\mathcal{N}_i^r$ represents the set of neighboring nodes of node i under relation r.
- $c_{i,r}$ is a normalization constant for relation r.
- $W_r^{(l)}$ is the weight matrix for relation type r.

In entity classification tasks, the normalization constant $c_{i,r}$ is typically set to $|\mathcal{N}_i^r|$, the number of neighbors of node i under relation r [Schlichtkrull *et al.*2017].

Our work presents a new perspective on R-GCNs: By leveraging Graph Attention Networks (GAT) within an R-GCN framework, we enable the model to dynamically prioritize connections based on their current relevance. This dynamic prioritization allows the model to focus computational resources on the most important information, enhancing learning efficiency and effectiveness in reinforcement learning tasks. Our approach provides a robust framework for handling the evolving importance of relationships in dynamic environments.

2.1.4 Graph Attention Networks

Graph Attention Networks (GATs) [Veličković *et al.*2017] are a variant of Graph Neural Networks (GNNs) that leverage attention mechanisms for feature learning on graphs. GATs offer a more detailed and adaptive approach to aggregating neighborhood information compared to traditional GNNs.

In standard GNNs, such as Graph Convolutional Networks (GCNs), the feature update of a node is typically the average of the features of its neighbors. This approach treats all neighbors equally and does not differentiate between the contributions of different neighbors.

GATs, on the other hand, assign an attention coefficient to each neighbor, indicating the importance of that neighbor's features for the feature update of the node. These

coefficients are computed using a shared self-attention mechanism, which calculates an attention score for each pair of nodes. The scores are then normalized across each node's neighborhood using a softmax function.

The attention-based approach allows GATs to assign different weights to different neighbors, providing a more adaptive and context-sensitive model. This means that GATs can focus more on the most relevant neighbors when updating a node's features, potentially capturing more meaningful and discriminative patterns in the graph. Additionally, GATs offer a level of interpretability, as the attention coefficients can be seen as indicating the importance of each neighbor.

2.1.5 Reinforcement Learning

Reinforcement learning (RL) [Kaelbling, Littman, & Moore1996] is a machine learning approach that enables an agent to learn decision-making through interactions with an environment. In RL (as seen the figure 2.2), the agent takes actions based on its current state and receives rewards as feedback. The objective is to maximize the cumulative reward over time by learning a policy that maps states to actions.

Agent The agent is the learner or decision maker in the RL framework. It interacts with the environment by taking actions and receiving feedback in the form of rewards. The agent's goal is to learn an optimal policy π that maximizes the expected cumulative reward over time.

Environment The environment represents the external system with which the agent interacts. It provides the agent with states S_t and rewards R_t based on the actions A_t taken by the agent. The environment's dynamics are often modeled as a Markov Decision Process (MDP).

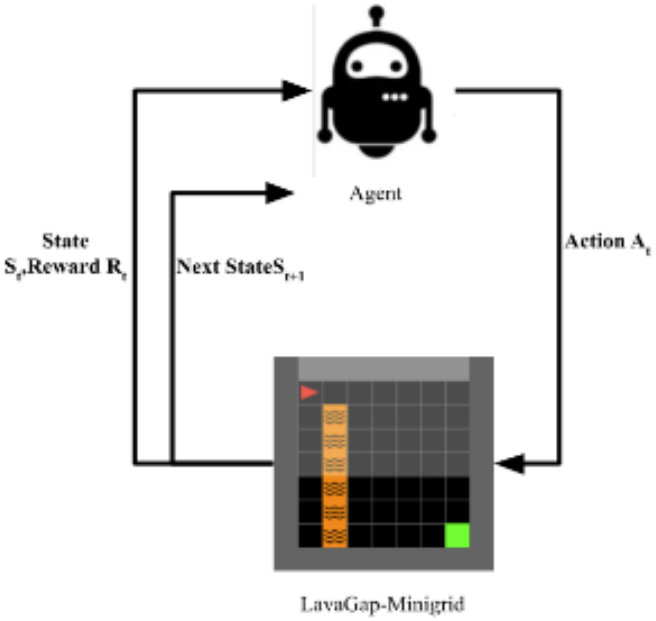

FIG. 2.2. Illustration of Reinforcement Learning process showing how the agent interacts with the environment to maximize the reward

Reward The reward R_t is a scalar feedback signal received by the agent after taking an action A_t in a state S_t. The reward function $r : S \times A \rightarrow R$ quantifies the immediate benefit of the action taken. The agent's objective is to maximize the total reward it receives over time, often defined as the return G_t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k} \tag{2.3}$$

where, G_t is the return, representing the total discounted reward from time step t onward. The term $\gamma \in [0, 1]$ is the discount factor that determines the importance of future rewards, and R_{t+k} is the reward received k time steps after time t.

Model The model in RL refers to the agent's representation of the environment's dynamics. A model can be used to predict the next state and reward given the current state and action. In model-based RL, the agent uses this model to plan and make decisions. The transition function P and the reward function r define the model:

$$P(s'|s,a) = P(S_{t+1} = s'|S_t = s, A_t = a) \tag{2.4}$$

$$r(s,a) = E[R_{t+1}|S_t = s, A_t = a] \tag{2.5}$$

In these equations, $P(s'|s,a)$ is the probability that the environment will transition to state s' given that the agent is currently in state s and takes action a. The term $P(S_{t+1} = s'|S_t = s, A_t = a)$ represents the same transition probability. The function $r(s,a)$ is the expected reward after taking action a in state s, and $E[R_{t+1}|S_t = s, A_t = a]$ denotes the expectation of the reward received at the next time step given the current state and action.

Policy The policy π is a mapping from states to probabilities of selecting each possible action. It defines the agent's behavior at any given time. A policy can be deterministic, where $\pi(s) = a$, or stochastic, where $\pi(a|s) = P(A_t = a|S_t = s)$.

The goal in RL is to find an optimal policy π^* that maximizes the expected return G_t from each state s. The value function $V^\pi(s)$ under policy π is defined as:

$$V^\pi(s) = E_\pi\left[G_t|S_t = s\right] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t = s\right] \tag{2.6}$$

Here, $V^\pi(s)$ is the value of state s under policy π, representing the expected return when starting from state s and following policy π. The term E_π denotes the expectation over all possible trajectories under policy π, G_t is the return from time step t onward, and $S_t = s$ indicates that the agent is in state s at time t.

Markov Decision Process A Markov Decision Process (MDP) [Puterman1990] is a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of the decision maker. An MDP is defined by a tuple (S, A, P, r, γ).

The equation for the value function $V^\pi(s)$ under policy π is given by [Peng1992]:

$$V^\pi(s) = \sum_{a \in A} \pi(a|s) \left[r(s,a) + \gamma \sum s' \in SP(s'|s,a) V^\pi(s') \right] \tag{2.7}$$

In this equation, $\pi(a|s)$ is the probability of taking action a in state s under policy π. The term $r(s,a)$ is the reward received after taking action a in state s, γ is the discount factor, $P(s'|s,a)$ is the probability of transitioning to state s' from state s after taking action a, and $V^\pi(s)$ is the value function, representing the expected return when starting from state s and following policy π.

For the optimal policy π^*, the value function $V^*(s)$ satisfies the Bellman optimality equation:

$$V^*(s) = \max_{a \in A} \left[r(s,a) + \gamma \sum s' \in SP(s'|s,a) V^*(s') \right] \tag{2.8}$$

Here, $V^*(s)$ is the optimal value function, representing the maximum expected return achievable from state s, and $\max_{a \in A}$ denotes the maximum over all possible actions a.

Similarly, the action-value function $Q^\pi(s,a)$ under policy π is defined as:

$$Q^\pi(s,a) = E_\pi \left[G_t | S_t = s, A_t = a \right] = r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^\pi(s') \tag{2.9}$$

In this equation, $Q^\pi(s,a)$ is the action-value function, representing the expected return when starting from state s, taking action a, and then following policy π. The term G_t is the

return from time step t onward.

For the optimal policy π^*, the action-value function $Q^*(s, a)$ satisfies:

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \max_{a' \in A} Q^*(s', a') \tag{2.10}$$

Here, $Q^*(s, a)$ is the optimal action-value function, representing the maximum expected return when starting from state s and taking action a, and $\max_{a' \in A}$ denotes the maximum over all possible actions a' in the subsequent state s'.

By solving these equations, the agent can determine the optimal policy and maximize its cumulative reward in the environment.

Symbol	**Definition**
S_t	State at time t
A_t	Action at time t
R_t	Reward at time t
G_t	Return, or cumulative reward, starting from time t
γ	Discount factor
$P(s'\|s, a)$	State transition probability function
$r(s, a)$	Reward function
$\pi(a\|s)$	Policy, mapping states to action probabilities
$V^\pi(s)$	Value function under policy π
$V^*(s)$	Optimal value function
$Q^\pi(s, a)$	Action-value function under policy π
$Q^*(s, a)$	Optimal action-value function

Table 2.2. Notation table for Reinforcement Learning

2.2 Related Work

This section provides a review of the existing literature on Graph Neural Networks (GNNs) [Zhou *et al.*2020] in the domain of reinforcement learning (RL), particularly focusing on the integration of attention mechanisms and the representation of relational data.

2.2.1 Graph Neural Networks in RL

One of the pioneering works in this area is NerveNet [Wang *et al.*2018], which utilizes GNNs to represent an RL policy. NerveNet adopts a similar message-passing scheme as Graph Convolutional Networks (GCNs) but focuses solely on node feature vectors without explicitly modeling relational data. It has been benchmarked on MuJoCo environments like Snake and Centipede, where it controls multi-joint robotic avatars. Each node in the graph corresponds to a movable part of the agent, with state information and actions directly attached to these nodes. This approach has shown superior in-distribution performance and generalization capabilities compared to traditional Multi-Layer Perceptrons (MLPs).

2.2.2 Attention Mechanisms in GNN-based RL

In terms of utilizing attention mechanisms within GNNs for RL, a notable work is the DRL-GAT-SA [Peng *et al.*2022]. This study introduces a graph attention reinforcement learning controller (GARL) based on a safety field model, combining graph attention networks with deep reinforcement learning to enhance the agent's situational awareness and safety. However, this model does not explicitly address relational representation learning nor does it focus on dynamic edge weighting, which are critical for adapting to complex, evolving environments.

Chapter 3

APPROACH AND ARCHITECTURE

3.1 Approach

In this chapter we describe how Relational Graph Convolutional Networks (R-GCNs) leverage attention mechanisms to achieve selective focus. This allows the model to extract the most relevant information from the graph structure at each step.

Inspired by the Grid-to-Graph Methodology [Jiang *et al.*2021] we use a set of relational determination rules to express relational inductive biases in relational graphs.

3.2 Relational Graph Construction

Traditionally, R-GCNs update each node's feature vector by aggregating transformed features of its neighbors, influenced by the specific types of relationships or edges that connect them. A key component in this aggregation process is the inclusion of self-loops, which allow nodes to incorporate their own features into the update, typically treated distinctly from other relational interactions.

Following the methodology outlined in the Grid-to-Graph paper [Jiang *et al.*2021], we treat self-loop edges as regular relational edges as all self-loop edges $W_0 x_b$ share the same

relation label, making the update rule for node a as follows:

$$y_a = \sum_{r \in R \cup \{0\}} \sum_{b \in N_a^r} \frac{1}{c_{a,r}} W_r x_b.$$

Here, each component of the equation represents:

- y_a: The updated feature vector of node a.
- R: The set of all types of relationships within the graph, with $\{0\}$ representing a self-loop.
- N_a^r: The set of neighbor nodes of a connected through relation type r.
- $c_{a,r}$: Normalization constant for node a under relation r, mitigating the influence of node degree on learning.
- W_r : Weight matrix corresponding to relationship type *r*.

This reformulation treats self-interactions (connections a node has with itself) the same way it treats connections with other nodes. This simplifies the model by eliminating the need for special rules for self-connections. By treating all connections equally, the model can learn more effectively and generalize better across different situations.

3.2.1 Relational Determination Rules

To construct the relational graph, we implement relational determination rules inspired by the methodologies outlined in the Grid-to-Graph paper [Jiang *et al.*2021]. Each rule is defined as follows:

$$r(a, b) \leftarrow \text{condition},$$

where $r(a, b)$ defines a relation from entity a to b with label r, and the condition describes a logical statement that must be true for the relation to be established. These rules are applied

based on the positions and other attributes of the objects, helping the model to understand and encode the spatial and relational structure of the environment effectively.

3.2.2 Encoding Relational Inductive Biases

The relational rules defined in the system, such as `is_front`, `is_back`, `is_left`, and `is_right` and more, specify conditions under which two objects a and b in a grid environment are related. These rules leverage the spatial positions and other attributes of objects to encode the structure of the environment into a graph model.

For example, the rule:

$$\text{is\_front}(a, b, \text{direction\_vec}) \leftarrow (b.pos - a.pos) \cdot \text{direction\_vec} > 0.1$$

states that object b is considered to be in front of object a if the dot product of the position vector difference $(b.pos - a.pos)$ with a given direction vector direction_vec is greater than 0.1. This establishes a 'front' relationship from a to b, provided the condition is satisfied.

3.2.3 Example of Relational Inductive Bias

Consider two objects, a and b, positioned in a simple grid where a is directly behind b from the perspective of an agent facing north. Applying the `is_front` rule determines that b is in front of a. This relational inductive bias assists the model in understanding and predicting actions based on the spatial arrangement of objects, enhancing its generalization capabilities.

Now we describe how the attention mechanism is applied on the RGCN nodes and edges.

3.3 Dynamic Edge Weighting via Attention Mechanism

The dynamic edge weighting in our model is realized through an attention mechanism integrated into the Relational Graph Convolutional Networks (R-GCNs), which is helpful for increasing sample efficiency in reinforcement learning scenarios. This mechanism allows the model to adaptively prioritize edges in the graph, which are critical for representing the dynamic relationships in the environment.

3.3.1 Attention Mechanism in R-GCN

Traditionally, in R-GCNs, the contribution of each neighbour is weighted equally or according to static, predefined parameters (as seen in Figures 3.1 - 3.4). However, such static weighting does not account for the varying relevance of information over different states or stages of learning. To address this, we introduce an attention mechanism that assigns a dynamic weighting to edges based on the current state of the environment and the historical interactions observed by the agent.

The attention coefficients are computed using a pairwise interaction function that considers the features of both the source and target nodes of each edge, as well as the edge's existing attributes. This function typically involves a learnable transformation followed by a softmax operation to ensure that the weights are normalized over all outgoing edges from a node:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}h_i \| \mathbf{W}h_j]))}{\sum_{k \in \mathcal{N}(i)} \exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}h_i \| \mathbf{W}h_k]))},$$

where $\mathbf{a}^T$ is a learnable parameter vector, each node's feature vector h_i and h_j is transformed using a shared weight matrix $\mathbf{W}$. The transformed features $[\mathbf{W}h_i \| \mathbf{W}h_j]$ of nodes i and j are concatenated, merging their information into a single vector. A learnable parameter vector $\mathbf{a}^T$ projects the concatenated feature vector to a scalar value $\mathbf{a}^T[\mathbf{W}h_i \| \mathbf{W}h_j]$, representing the raw attention score. **LeakyReLU** function is applied to the scalar output,

allowing small gradients for negative inputs to prevent gradients from vanishing, facilitating continuous learning. The output from the LeakyReLU is exponentiated to make all attention scores positive and to emphasize significant scores. The softmax function normalizes these scores across all neighbours k in node i's neighbourhood $\mathcal{N}(i)$, ensuring they sum to one and proportionally distributing focus based on feature interaction relevance. The final output of the equation is the normalized attention coefficient α_{ij} for the edge from node i to node j.

3.3.2 Applying Dynamic Weighting

Once the attention coefficients are determined, they modulate the aggregation of neighbouring features during the convolution operation in RGCNs (as seen in Figure 3.5 - 3.8). This modulation allows the network to focus more on the information that is currently most relevant, enhancing the responsiveness of the learning model to changes in the environment.Thus the new updated rule after applying dynamic weighting is.

$$y_a = \sum_{r \in R \cup \{0\}} \sum_{b \in N_a^r} \alpha_{ab} \frac{1}{c_{a,r}} W_r x_b.$$

where:

- α_{ab} denotes the attention coefficient for the edge from node a to node b, which adjusts the contribution of node b's features based on their relevance.
- $c_{a,r}$ is the normalization constant for the edge type r at node a, which helps in stabilizing the learning process.
- W_r represents the weight matrix for edge type r, applied to the features of node b (x_b).

- R is the set of all relation types in the graph, with 0 representing the self-loop relation for incorporating self-features.

- N_a^r is the set of neighbors of node a connected via relation type r.

3.4 Architecture

The state of the environment is initially represented as a feature map where each tile in a grid world is encoded as a binary-valued feature vector $\mathbf{x}$ (as seen in Fig. 3.9) These feature vectors, $\mathbf{X}$, are assigned to nodes. Following the Grid-to-Graph's set of relational determination rules [Jiang *et al.*2021], edges between these nodes are established, creating a relational graph $\mathbf{G}$ that embeds specific relational inductive biases. The node features are first projected into a higher-dimensional space using an embedding layer. These transformed embeddings are then propagated through two layers of Relational Graph Convolutional Networks (R-GCN), each configured with distinct weights for different types of relations. To enhance the stability of training, batch normalization is applied following each RGCN layer. Batch normalization normalizes each feature within a batch of samples, ensuring consistent feature scaling and accelerating convergence during training.

To adaptively modulate the importance of edges during training, we incorporate an attention mechanism. This mechanism computes attention coefficients based on matrices involving learnable parameters (as seen in Figure 3.10). The process involves transforming node features through a function equipped with learnable weights, from which the attention coefficients are derived. These coefficients dynamically scale the contributions of neighboring nodes' feature representations, enabling the model to prioritize edges based on their relevance to the current context and focus computation on the most critical information.

After processing through the RGCN and attention layers, the features are further refined through dense layers. The outputs from these dense layers are utilized to compute

both the policy logits and the baseline value. The policy logits direct the agent's actions, whereas the baseline value facilitates value estimation within the Actor-Critic algorithm.

We validate our proposed framework across various simulation environments, benchmarking its performance against traditional reinforcement learning algorithms. Our findings reveal that the attention-augmented R-GCN framework not only accelerates learning but also enhances decision quality while markedly reducing the sample complexity.

3.5 Use of Batch Normalization

Batch normalization [Ioffe & Szegedy2015] is an integral component of the REAGLE framework, particularly within the architecture of relational graph convolutional networks (R-GCNs) that are enhanced with dynamic edge weighting. This technique plays a critical role in stabilizing and accelerating the training process, which is essential for managing the high-dimensional data and complex model architectures often encountered in deep learning applications within reinforcement learning.

One of the primary benefits of batch normalization is its ability to reduce internal covariate shift. By normalizing the inputs of each layer, batch normalization helps ensure a more consistent data distribution across training epochs, leading to faster convergence. Additionally, it enhances the flow of gradients throughout the network, addressing common issues in deep networks such as vanishing and exploding gradients. Furthermore, batch normalization introduces a mild regularization effect by incorporating noise during the estimation of batch statistics, which in turn improves the model's ability to generalize.

In the context of the REAGLE framework, batch normalization is implemented in a way that optimizes the functionality of R-GCNs. Specifically, node embeddings are normalized immediately after being computed through the R-GCN layers. This step is crucial before passing the embeddings to subsequent layers or utilizing them in further compu-

tations. The normalization process also helps control the scale of activations, which is particularly important in graph-based models where variations in node degree and edge weights can significantly impact the network's behavior.

The inclusion of batch normalization within REAGLE significantly influences the learning dynamics of the model. It contributes to faster training by stabilizing the learning environment, allowing for the use of higher learning rates, and thereby speeding up the training process. Moreover, batch normalization helps maintain stability across the network's activations, reducing the likelihood of numerical instabilities. As a result, models trained with batch normalization tend to achieve higher performance and reach convergence more rapidly, making it a vital component for efficiently training sophisticated graph-based neural networks.

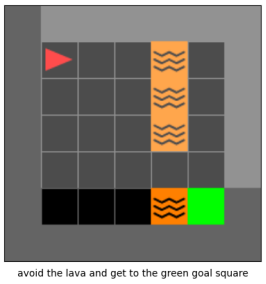

FIG. 3.1. Initial Environment State at t=0

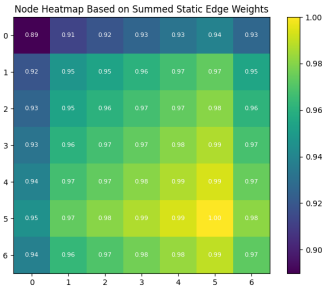

FIG. 3.2. Static Edge Weighting at t=0: Node heatmap illustrating traditional R-GCN static weights, highlighting consistent weight distribution across nodes regardless of the agent's interaction with the environment

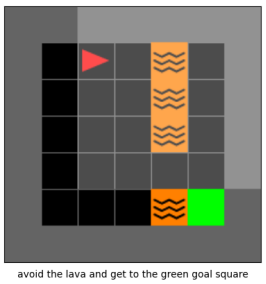

FIG. 3.3. Environment State at t=1

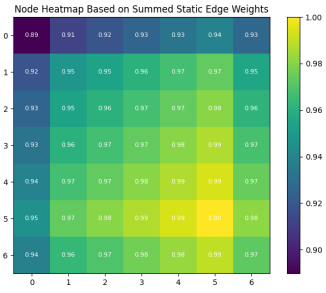

FIG. 3.4. Static Edge Weighting at t=1: displays static edge weights for the updated state.

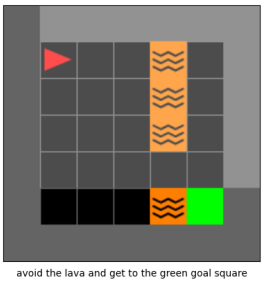

FIG. 3.5. Initial Environment State at t=0

Similar to Figure 3.1

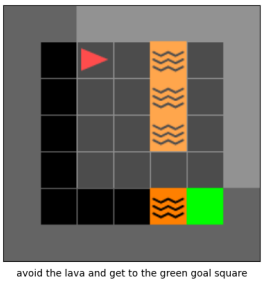

FIG. 3.7. Environment state at t=1

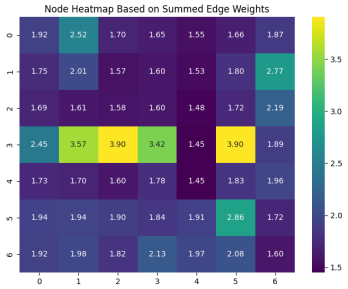

FIG. 3.6. Dynamic Edge Weighting at t=0: Similar to Figure 3.2, Heatmap showing the weighted significance of node relationships using the REAGLE method, which dynamically adjusts based on the current environmental context and the agent's position

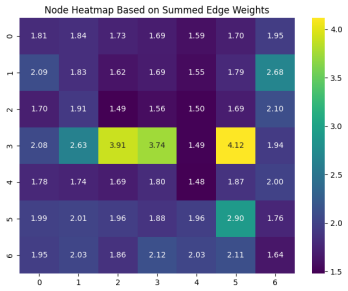

FIG. 3.8. Dynamic Edge Weighting at t=1: Similar to Figure 3.4 Continuation of dynamic weight adjustments shown through a heatmap, emphasizing changes in node relationships as the agent progresses and the environment evolves.

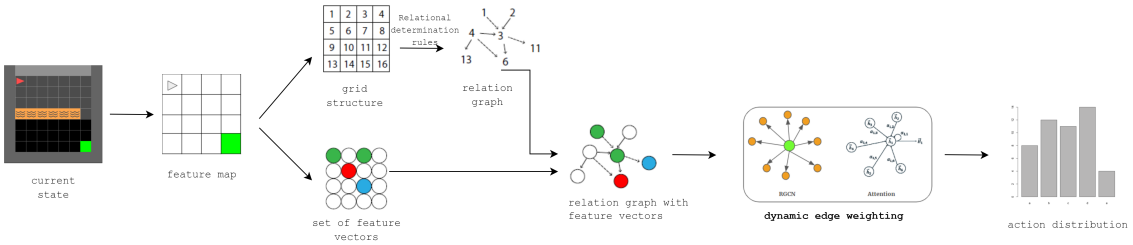

FIG. 3.9. A high-level overview of our approach, **REAGLE.** We abstract away the grid structure of the feature map and turn observations into a relational graph using relational determination rules. The vectors of the feature map are attached to nodes in the relational graph. We then use an R-GCN to reason over the relational graph and node features, applying dynamic edge weighting with the help of an attention mechanism. This mechanism focuses computation on the most relevant information, thus producing an action distribution.

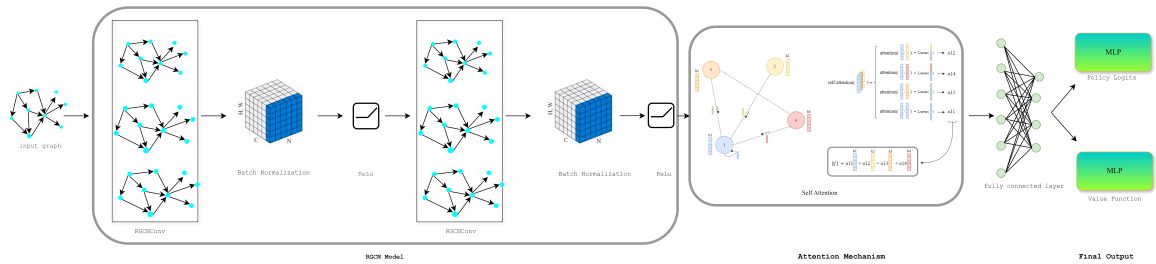

FIG. 3.10. Architecture of the edge wise attention-augmented Relational Graph Convolutional Network (R-GCN) for processing grid-based environmental data. This flow diagram illustrates the transformation of graph-based features into decision-making outputs in a reinforcement learning context.

Chapter 4

EXPERIMENTAL SETTING

In this chapter we outline the experimental settings and environments used to evaluate various reinforcement learning (RL) algorithms. Our research categorizes these environments into four levels of complexity to systematically assess the algorithms' performance under different conditions. These environments vary from simple, obstacle-free rooms to complex scenarios involving multiple tasks and distractors. The primary aim of this structured categorization is to explore how different RL models adapt to and navigate these varied settings, focusing particularly on their ability to handle sparse rewards, safe exploration, and combinatorial decision-making.

For the purpose of this research, the testing environments have been categorized into four distinct levels of complexity

- Easy
- Medium
- Hard
- Most Difficult

4.1 MiniGrid-Empty-6x6-v0 (Easy Environment)

The MiniGrid-Empty environment is an empty room with no obstacles involved and the goal of the agent is to reach the green goal square (as seen in figure 4.1), which provides a sparse reward. A small penalty is applied based on the number of steps taken to reach the goal and the reward is calculated at the end of each episode. This environment is partially observable, meaning the agent can only see a portion of the environment at any given time, which adds complexity to navigation tasks. This environment is particularly beneficial for validating your reinforcement learning algorithm in smaller rooms, and for experimenting with sparse rewards and exploration in larger rooms. In the random variants, the agent starts from a different position each episode, whereas in the regular variants, the agent always begins in the corner opposite the goal.

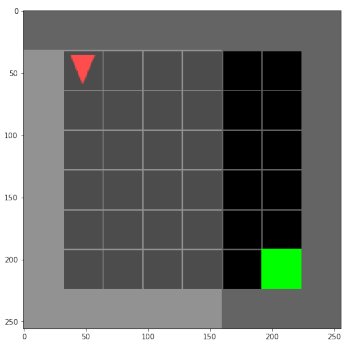

FIG. 4.1. Illustration of the MiniGrid-Empty environment where the green block is the goal and the Red triangle is the agent

4.2 MiniGrid-LavaGapS7-v0 (Medium Environment)

In the MiniGrid-LavaGapS7 environment (as seen in Figure 4.2), the agent's objective is to navigate to the green goal square located at the opposite corner of the room

while avoiding a deadly vertical strip of lava. Contact with the lava results in immediate termination of the episode with zero reward, emphasizing the importance of safe navigation strategies. This environment is partially observable, limiting the agent's field of view and increasing the challenge of safely reaching the goal. The penalty for each step taken encourages efficiency in pathfinding. This setup is particularly useful for studying safe exploration and handling dynamic obstacles in reinforcement learning.

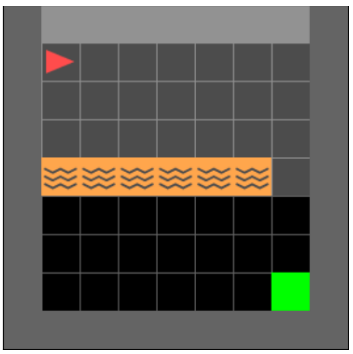

FIG. 4.2. MiniGrid-LavaGapS7 environment where the orange block corresponds to the lava river and the task of the agent is to avoid it and reach the green goal

4.3 Lavacrossing (Medium Environment)

This environment is similar to the Lavagap environment but the only difference is it has to cross multiple deadly lava rivers placed vertically and horizontally across the environment(as seen in Figure 4.3). For the purpose of thia thesis, we selected the environment which had 2 deadly rivers which randomise after every episode. This environment is useful for studying safety and safe exploration.

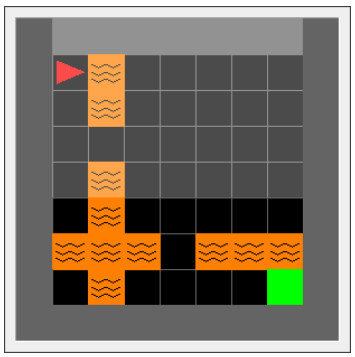

FIG. 4.3. Illustration of the MiniGrid-CrossingS9N2 environment where the rivers are placed horizontally and vertically across the environment and the task of the agent is to avoid them and reach the green goal

4.4 DoorKey Env (Difficult Environment)

In the MiniGrid DoorKey environment (as depicted in Figure 4.4), the agent faces the complex task of finding a key to unlock a door, which then allows access to the green goal square. This environment's partially observable nature adds to the difficulty, as the agent must explore to locate both the key and the door without prior knowledge of their placements. The reward is granted only upon successful completion of the task, with penalties for each step taken to encourage efficient exploration and problem-solving. This scenario is particularly useful for examining advanced learning strategies like curiosity-driven learning or curriculum learning in reinforcement learning frameworks.

4.5 Boxworld (Most Complex Environment)

Box-world, a grid-world navigation challenge introduced by Zambaldi et al. [Zambaldi *et al.*2018], requires an agent to collect a gem using the appropriate key. This key is secured within a locked box that can only be opened using another distinct key. The game

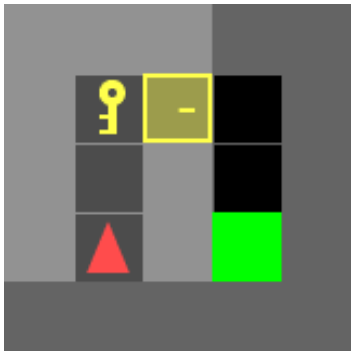

FIG. 4.4. MiniGrid DoorKey environment where the agent must navigate to find a key, unlock a door, and reach the green goal, illustrating the complexity of the task.

features distractor branches that can mislead the agent by consuming the current key and yielding another key, which does not unlock the gem box. Due to its combinatorial complexity, the probability of randomly stumbling upon the correct solution is minimal. Zambaldi et al. [Zambaldi *et al.*2018] reported that their reinforcement learning (RL) models needed between 200 million to 1.4 billion steps to achieve convergence in this environment.

Given our limited computational resources, our models are trained for only 1.8 million steps in each setting. We also simplify the **Box-World** environment to manage complexity: we reduce the field size to 10x10, limit the number of distractor branches to one, shorten the length of distractor branches to one, and set the goal length to two. While these modifications lower the game's difficulty, they maintain the fundamental elements of the Box-World environment, allowing us to still evaluate the relational reasoning capabilities of various models effectively while reducing the total training steps required for convergence.

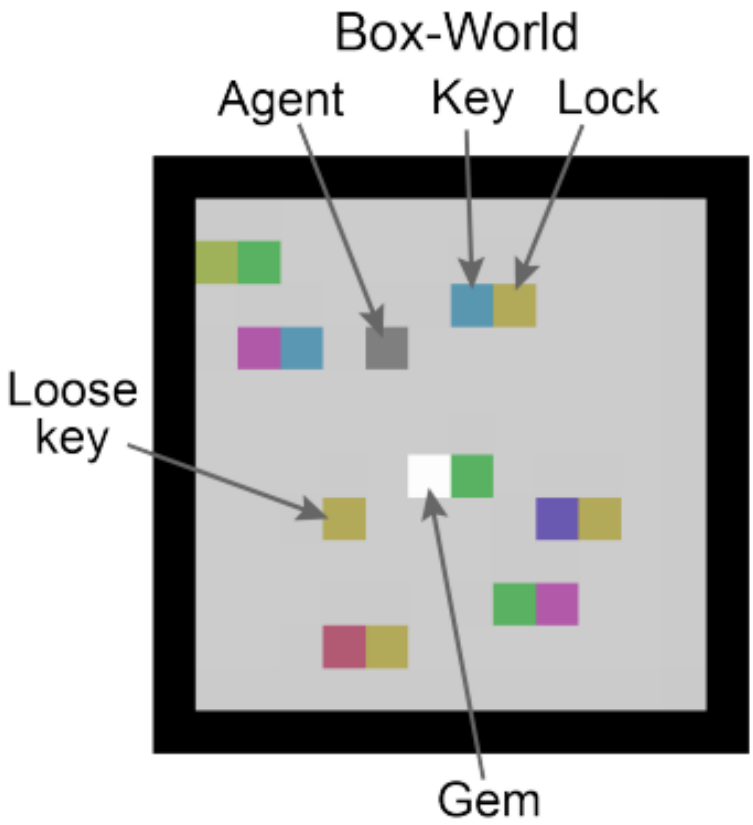

FIG. 4.5. Illustration of the Box-World environment

Chapter 5

EXPERIMENTAL RESULTS

This chapter explores how our proposed approach performs as compared to baseline models across different simulation environments.

5.1 Baselines

In our experiments, we make use of IMPALA [Espeholt *et al.*2018], a policy-gradient algorithm to train our RL models. We assess the effectiveness of our approach against the following well-established Graph Neural Network(GNN) models:

- **Graph Attention Network (GAT)**:GAT [Brody, Alon, & Yahav2022] Dynamically weighs the importance of nodes within a neighbourhood through attention mechanisms.
- **Graph Convolutional Network (GCN)**:GCN [Kipf & Welling2017] a foundational GNN model aggregating information across graph structures.
- **Vanilla Relational Graph Convolutional Network (RGCN)**: RGCN [Zhang *et al.*2019] handles multiple types of relationships in data, suitable for complex relational data.

5.2 Simulation Results

To investigate the training performance of our methodology, we conducted a series of experiments using a carefully designed setup. First, to evaluate the training performance concerning the mean episodic return, we performed training sessions using our method, REAGLE, along with other baseline methods. Each method was assessed across several simulation environments to ensure a comprehensive comparison.

To enhance the reliability of our results, each experiment was run five times. This practice is common in the reinforcement learning domain as it helps mitigate the effects of random initialization and other stochastic factors inherent in such experiments, ensuring that the reported outcomes are not unduly influenced by any single run. Following the completion of each experiment, the rewards from all episodes were averaged to compute the mean reward for that run. The final reported performance for each experiment is thus derived from the average of these mean rewards across all five runs, providing a robust measure of the training performance of the various methods evaluated.

5.2.1 Training Performance Evaluation

We designed three measurement metrics to access the sample complexity as follows :

- **Mean Episode Return:** The learning curve plots the performance of the algorithm over time or episodes. We plot the mean episode return of the models
- **Total Number of Episodes or Timesteps:** This metric helps us to count the total number of episodes or timesteps required for the algorithm to reach a certain level of performance or to converge to an optimal or near-optimal policy.
- **Asymptotic Performance:** This is the performance level achieved after a large number of samples, indicating the long-term effectiveness of the algorithm.

5.2.2 Minigrid Environment Results

For each model, we run 5 trials and then take the average of those 5 runs. The thin, opaque lines in the plots represent unsmoothed training curves corresponding to average runs, and the bolded lines represent smoothed mean episodic return averaged over all five runs.

As seen in Figure 5.1, for the **Minigrid-Lavacrossing** environment, our model consistently outperforms the vanilla method when trained for 100k steps. Both REAGLE and Vanilla_RGCN start with low initial performance but show rapid improvement in the early stages. As the number of steps increases beyond 40,000, both models begin to plateau, with REAGLE stabilizing at a higher mean return close to 1.0, while Vanilla_RGCN stabilizes slightly below that, around 0.9. REAGLE converges around 65,000 steps, and compared to Vanilla_RGCN, which converges around 85,000 steps, our method achieves higher performance in fewer steps. This indicates that our approach is more sample-efficient, making it more suitable for scenarios where training time is limited.

In the **MiniGrid-LavaGapS7-v0** environment, we trained our model **REAGLE** for 1 million steps and compared its performance against baseline models **Vanilla_RGCN**, **GAT**, and **GCN** (as seen in Figure 5.2). REAGLE showed better sample complexity, achieving a mean episodic return of 0.95 after approximately 250,000 steps. In contrast, **Vanilla_RGCN** required about 400,000 steps to reach the same performance level, making REAGLE more efficient. **GAT** and **GCN** performed poorly, plateauing at a much lower mean episodic return of around 0.2, indicating their struggles with the task complexity.

We further evaluated **REAGLE_GATv2**, which incorporates a more expressive attention mechanism. Despite the added complexity, **REAGLE_GATv2** did not significantly outperform the original **REAGLE** or **Vanilla_RGCN**. All three models converged to a high-performance level near a 1.0 mean episodic return by 1 million steps. The standard

deviations across these models were similar, indicating consistent performance with minor fluctuations during early training. This suggests that the Lavagap environment does not require the enhanced attention capabilities of GATv2 to achieve optimal results.

For the **MiniGrid-DoorKey-5x5-v0** environment (as seen in Figure 5.4), we observed that both REAGLE and the vanilla model respectively show rapid convergence, achieving approximately 85% of their maximum performance within the first 100,000 steps. Notably, the REAGLE model reached a mean episodic return of 0.85 at around 50,000 steps, whereas **Vanilla_RGCN** achieved similar performance at approximately 60,000 steps, indicating that REAGLE converges slightly faster with a 10,000-step advantage. As training continued, both models plateaued at a high-performance level close to a mean episodic return of 1.0, with both REAGLE and **Vanilla_RGCN** ultimately converging to approximately 0.98. This demonstrates that both models are equally effective in optimizing for the Lavagap environment. The stability of performance, particularly after 200,000 steps, was also noteworthy, with REAGLE exhibiting slightly lower variance and maintaining a more stable performance curve compared to **Vanilla_RGCN**. This enhanced stability suggests that REAGLE may offer more consistent results across different training runs, making it particularly valuable in scenarios where reliable agent behavior is crucial. Overall, while both models achieve comparable asymptotic performance, REAGLE's slight edge in convergence speed and performance stability could make it a more attractive option, especially in environments where early learning efficiency and consistency across runs are essential.

5.2.3 Boxworld Environment Results

For the Boxworld environment [Zambaldi *et al.*2018], which is the most difficult environment in terms of task complexity, our model outperformed the vanilla method by a significant margin(as seen in Figure 5.5). We trained our model for 1.8 million steps.

Sample Complexity and Convergence: The REAGLE model demonstrated superior

sample complexity by achieving a mean episodic return of approximately 6.0 within the first 100,000 steps, compared to Vanilla_RGCN, which only reached around 2.0 in the same number of steps. This early performance indicates that REAGLE is more sample-efficient, quickly adapting to the complexities of the Boxworld environment.

Mid-Training Performance: As the training continued beyond 1 million steps, REAGLE's performance consistently surpassed that of Vanilla_RGCN, with a mean episodic return fluctuating between 7.5 and 8.0. In contrast, Vanilla_RGCN stabilized at a lower mean return of around 5.0. The slight fluctuations in REAGLE's performance suggest that while it effectively handles the environment's challenges, the complexity of the tasks leads to some variability in returns.

Asymptotic Performance and Stability: After 1.5 million steps till the end of the training, REAGLE continued to demonstrate its robustness, maintaining a mean episodic return close to 10.5, whereas Vanilla_RGCN plateaued at approximately 8.0. The difference of about 2.5 in mean episodic return highlights REAGLE's superior capability to learn and adapt in the Boxworld environment, which involves intricate decision-making and exploration.

Table 5.1. Performance Metrics of Models in the Boxworld Environment

Model	Total Training Time (Steps)	Steps to Convergence (To reach a reward of 10+)	Total Mean Episodic Return	Standard Deviation
REAGLE [Ours]	1,800,000	1,400,000	10.70 ± 1.12	1.12
Vanilla_RGCN	1,800,000	Never Converged	8.30 ± 5.12	5.12
GCN	1,800,000	Never Converged	5.77 ± 5.04	5.04
GAT	1,800,000	Never Converged	5.20 ± 5.26	5.26

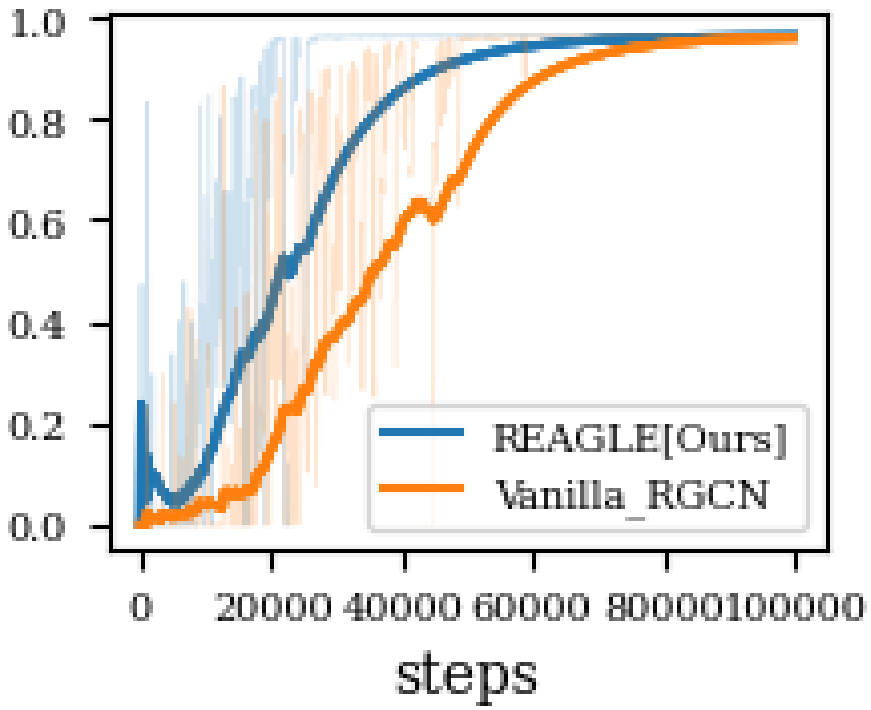

FIG. 5.1. Training curves of REAGLE and the baseline approach in the Minigrid-Lavacrossing environment.The thin, opaque lines in the plot represent unsmoothened training curves corresponding to average runs, and the bolded lines represent smoothed mean episodic return averaged over all five runs.

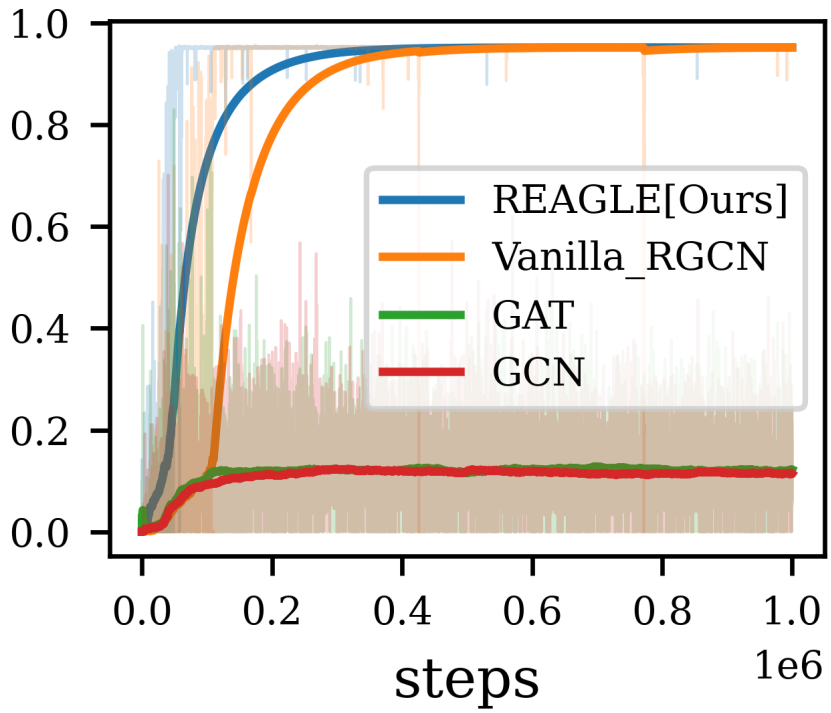

FIG. 5.2. Training curves of our method and the baseline approaches in the MiniGrid-LavaGapS7-v0 environment

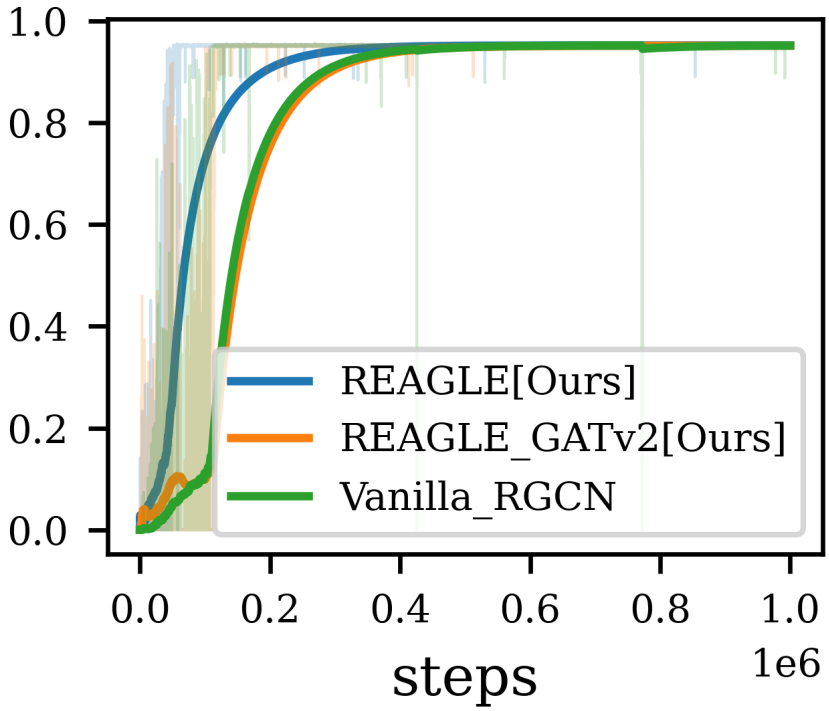

FIG. 5.3. Training curves of REAGLE, REAGLE with GATv2 Vanilla RGCN model in the MiniGrid-LavaGapS7-v0 environment. opaque lines in the plot represent unsmoothed training curves corresponding to average runs, and the bolded lines represent smoothed mean episodic return averaged over all five runs.

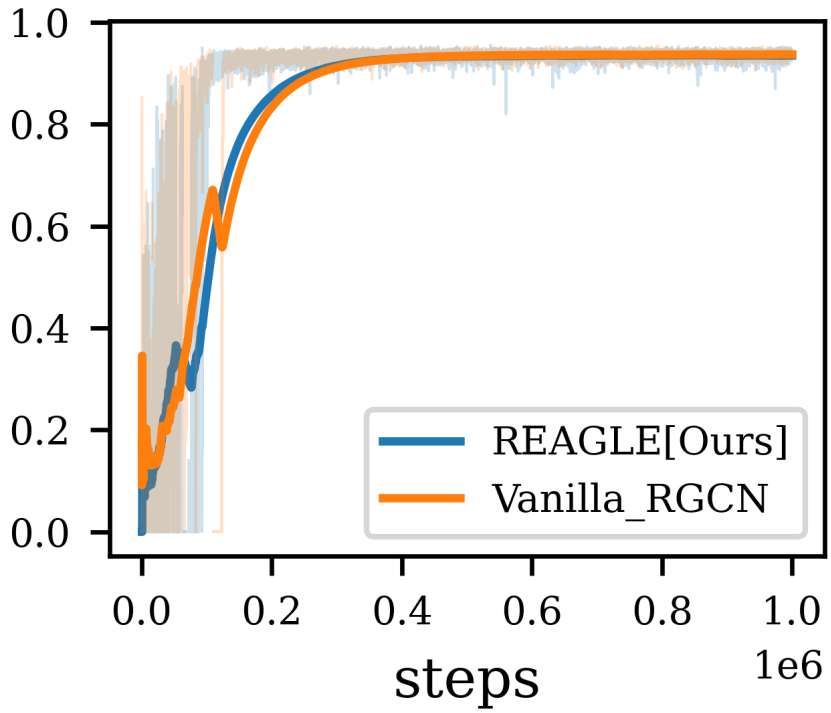

FIG. 5.4. Training curves of our method and the baseline approach in the MiniGrid-DoorKey-5x5-v0 environment

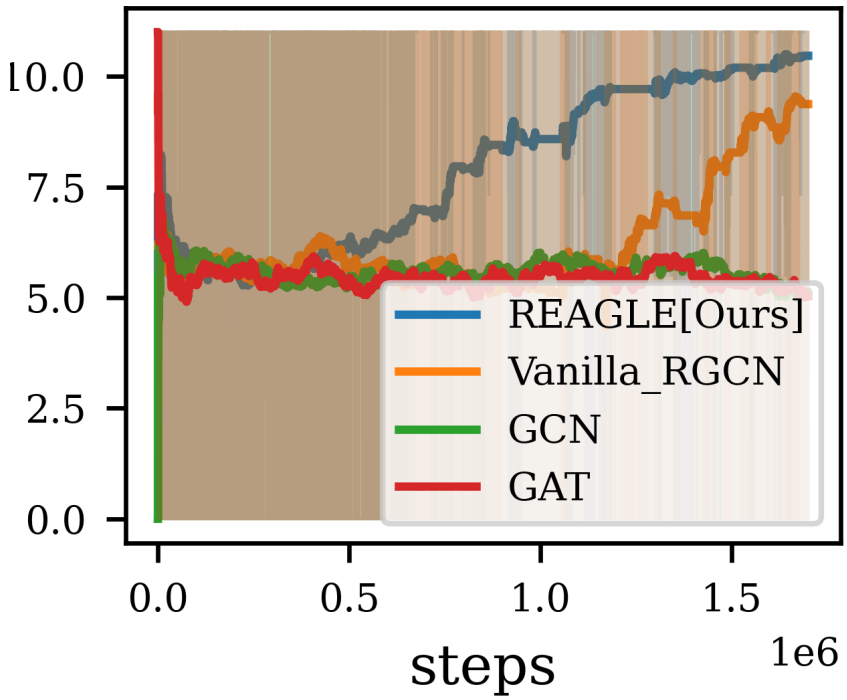

FIG. 5.5. Training curves of our method and the baseline approaches in the Boxworld environment

Chapter 6

CONCLUSION

6.1 Conclusion

This thesis presented a novel approach to enhance sample efficiency in reinforcement learning by dynamically weighting the edges in Relational Graph Convolutional Networks (R-GCN) using Graph Attention Networks (GAT). Our method, named REAGLE, demonstrated improved performance and efficiency in various complex environments compared to baseline models. The dynamic edge weighting allows the model to adapt more rapidly to the changing importance of relationships within the environment, focusing computational resources on the most relevant information and thus accelerating the learning process.

Our experiments across different levels of complexity in simulation environments have shown that REAGLE can achieve higher mean episodic returns and converge faster than traditional models. This indicates a significant reduction in the sample complexity required to train effective reinforcement learning agents. The integration of GAT within R-GCN provided the necessary flexibility and focus, enabling the agent to prioritize crucial information dynamically.

6.2 Future Work

While the results are promising, several avenues remain open for further research to expand the capabilities and understanding of dynamic edge weighting in graph-based reinforcement learning:

- **Extending to Other Environments:** Future work could explore the application of REAGLE to more diverse and even more complex environments, including those with continuous action spaces or real-world applications like robotic navigation and social network analysis.

- **Exploration of Other Attention Mechanisms:** Investigating other forms of attention mechanisms, such as multi-head attention or transformer-based models, could potentially yield even better performance by capturing more nuanced relationships within the data.

- **Scalability and Efficiency:** Further research could focus on improving the scalability and computational efficiency of our approach, making it feasible for larger-scale problems and reducing training times.

- **Integration with Other RL Techniques:** Combining REAGLE with other reinforcement learning strategies, such as hierarchical reinforcement learning or meta-learning, could enhance the agent's ability to handle complex tasks and improve transfer learning across different tasks.

- **Theoretical Analysis:** A deeper theoretical understanding of why and how dynamic edge weighting improves sample efficiency would be valuable. This could involve analyzing the impact of attention mechanisms on the stability and convergence of learning algorithms.

By addressing these areas, we can further enhance the adaptability, efficiency, and applicability of graph-based models in reinforcement learning, paving the way for more intelligent and efficient systems capable of learning in dynamic and complex environments.

REFERENCES

[1] Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35(8):1798–1828.

[2] Brody, S.; Alon, U.; and Yahav, E. 2022. How attentive are graph attention networks?

[3] Cho, M.; Alahari, K.; and Ponce, J. 2013. Learning graphs to match. In *Proceedings of the IEEE International Conference on Computer Vision*, 25–32.

[4] Cohen, M. X., and Ranganath, C. 2007. Reinforcement learning signals predict future decisions. *Journal of Neuroscience* 27(2):371–378.

[5] Dayan, P., and Balleine, B. W. 2002. Reward, motivation, and reinforcement learning. *Neuron* 36(2):285–298.

[6] Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; Legg, S.; and Kavukcuoglu, K. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures.

[7] Hunt, E. B. 2014. *Artificial intelligence*. Academic Press.

[8] Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift.

[9] Jiang, Z.; Minervini, P.; Jiang, M.; and Rocktaschel, T. 2021. Grid-to-graph: Flexible spatial relational inductive biases for reinforcement learning. *arXiv preprint arXiv:2102.04220*.

[10] Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4:237–285.

[11] Kakade, S. M. 2003. *On the sample complexity of reinforcement learning*. University of London, University College London (United Kingdom).

[12] Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

[13] Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks.

[14] Kober, J.; Bagnell, J. A.; and Peters, J. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32(11):1238–1274.

[15] Liu, Z.; Song, Y.; and Zhang, Y. 2023. Actor-director-critic: A novel deep reinforcement learning framework.

[16] Peng, Y.; Tan, G.; Si, H.; and Li, J. 2022. Drl-gat-sa: Deep reinforcement learning for autonomous driving planning based on graph attention networks and simplex architecture. *Journal of Systems Architecture* 126:102505.

[17] Peng, S. 1992. A generalized dynamic programming principle and hamilton-jacobi-bellman equation. *Stochastics: An International Journal of Probability and Stochastic Processes* 38(2):119–134.

[18] Puterman, M. L. 1990. Markov decision processes. *Handbooks in operations research and management science* 2:331–434.

[19] Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; van den Berg, R.; Titov, I.; and Welling, M. 2017. Modeling relational data with graph convolutional networks.

[20] Sharma, A. R., and Kaushik, P. 2017. Literature survey of statistical, deep and reinforcement learning in natural language processing. In *2017 International conference on computing, communication and automation (ICCCA)*, 350–354. IEEE.

[21] Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press.

[22] Torrado, R. R.; Bontrager, P.; Togelius, J.; Liu, J.; and Perez-Liebana, D. 2018. Deep reinforcement learning for general video game ai. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8. IEEE.

[23] Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

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

[25] Zambaldi, V.; Raposo, D.; Santoro, A.; Bapst, V.; Li, Y.; Babuschkin, I.; Tuyls, K.; Reichert, D.; Lillicrap, T.; Lockhart, E.; Shanahan, M.; Langston, V.; Pascanu, R.; Botvinick, M.; Vinyals, O.; and Battaglia, P. 2018. Relational deep reinforcement learning.

[26] Zhang, S.; Tong, H.; Xu, J.; and Maciejewski, R. 2019. Graph convolutional networks: a comprehensive review. *Computational Social Networks* 6(1):1–23.

[27] Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; and Sun, M. 2020. Graph neural networks: A review of methods and applications. *AI open* 1:57–81.