NETWORK-AWARE MULTI-AGENT REINFORCEMENT LEARNING FOR ADAPTIVE NAVIGATION OF VEHICLES IN A DYNAMIC ROAD NETWORK

FAZEL ARASTEH

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTERS OF APPLIED SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING & COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO

SEPTEMBER 2021

Abstract

Traffic congestion in urban road networks is a condition characterized by slower speeds, longer trip times, increased air pollution, and driver frustration. Traffic congestion can be attributed to a volume of traffic that generates demand for space greater than the available street capacity. A number of other specific circumstances can also cause or aggravate congestion, including traffic incidents, road maintenance work and bad weather conditions. While construction of new road infrastructure is an expensive solution, traffic flow optimization using route planning algorithms is considered a more economical and sustainable alternative. Currently, well-known publicly available car navigation services, such as Google Maps and Waze, help people with route planning. These systems mainly rely on variants of the popular Shortest Path First (SPF) algorithm to suggest a route, assuming a static network. However, road network conditions are dynamic, rendering the SPF route planning algorithms to perform sub-optimally at times. In addition, SPF is a greedy algorithm. So, while it can yield locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time, it does not always produce an optimal solution. For example, in a limited road capacity, the SPF routing algorithm can cause congestion by greedily routing all vehicles to the same road

(towards the shortest path). To address limitations and challenges of the current approach to solve the traffic congestion problem, we propose *a network-aware multi-agent reinforcement learning* (MARL) model for the navigation of a fleet of vehicles in the road network. The proposed model is *adaptive* to the current traffic conditions of the road network. The main idea is that a Reinforcement Learning (RL) agent is assigned to every road intersection and operates as a *router agent*, responsible for providing routing instructions to a vehicle in the network. The vehicle traveling in the road network is aware of its final destination but not its exact full route/path to it. When it reaches an intersection, it generates *a routing query* to the RL agent assigned to that intersection, consisting of its final destination. The RL agent generates *a routing response* based on (i) the vehicle's destination, (ii) the current state of the network in the neighborhood of the agent aggregated with a shared graph attention network (GAT) model, and (iii) routing policies learned by cooperating with other RL agents assigned to neighboring intersections. The vehicle follows the routing response from the router agents until it reaches its destination. Through an extensive experimental evaluation on both synthetic and realistic road networks we demonstrate that the proposed MARL model can outperform the SPF algorithm by (up to) 17.3% in average travel time.

Acknowledgements

I would like to give my special thanks to my supervisor, Dr. Manos Papagelis, for his consistent support, encouragement, and deep technical insight that has guided me through this project.

I would like to extend my special thanks to the committee members of my thesis oral exam, professor Aijun An, and professor Costas Armenakis, for their constructive technical reviews of the current work.

Furthermore, I would like to extend my thanks to my friend and room-mate Mr, Soroush Sheikh GarGar, for his assistance in the implementation of the project, constructive technical discussions about the project, and more importantly, being a great friend :) during the Covid-19 lock-down.

I would also want to thank my loving parents and siblings for keeping me in their hearts and for their emotional support for me from a long distance away.

Finally, I would like to have my thanks to my friends, Hodjat Asghari Esfeden (Babaye Alex), Amir Nourbakhsh Rezaie, Alireza Naeiji, Eli and Ali Mosleh, Farnaz Bidokhti, Behnam Asadi, Mahta Shafiee, Parsa Farshadfar and all other friends and colleagues at York University,

whom I have spent joy-full times with.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

An effective transportation system is essential for the development of large cities. Traffic congestion in urban road networks is a condition that can adversely affect the transportation system. Traffic congestion can be attributed to a volume of traffic that generates demand for space greater than the available street capacity. A number of other specific circumstances can also cause or aggravate congestion, including traffic incidents, road maintenance work and bad weather conditions. Traffic congestion is characterised by slower speeds, longer trip times, increased air pollution, and driver frustration.

While construction of new road infrastructure is an expensive solution, traffic flow optimization using route planning algorithms is considered a more economical and sustainable alternative. [32]. Currently, well-known publicly available car navigation services like Google Maps and Waze help people with route planning. These systems mainly rely on a variant of the popular Shortest Path First (SPF) [9] algorithm to suggest a route assuming a static

network.

In a static network, the SPF algorithm is optimal. However, road networks are not always static. In a dynamic road network, the SPF path is calculated based on the estimated travel time of roads when the vehicle requests a route. The more accurate the estimated travel times are, the better the suggested route. Predicting the estimated travel times is part of the Traffic Prediction Problem. Classical methods such as Historical Average and ARIMA, machine learning methods such as Support Vector Regression and Random Forest, and more recently, deep learning methods have been applied to the Traffic Prediction Problem to capture more accurate estimated travel times [55, 44]. Nevertheless, the estimated travel times, specifically long-term predictions of the travel time in a road, may be inaccurate. Hence, the SPF working based on the inaccurately estimated travel times proves sub-optimal.

Since the long-term prediction of estimated travel time is not accurate, other methods such as probabilistic dynamic programming and ant colony have been proposed to directly route the vehicles in the dynamic network [50, 43]. More recently, deep reinforcement learning has also been proposed for direct routing without travel time prediction [39, 24, 16]. Also, graph convolution networks are proposed to embed the structure of the network to use with reinforcement learning for routing in large dynamic networks [54].

Moreover, the SPF routing algorithm greedily sends every vehicle to the currently available shortest path. If the road infrastructure is limited, the current shortest path gets congested quickly. In other words, the routing algorithm causes congestion by greedily sending every vehicle to the shortest path.

To address limitations and challenges of the current approach, we turn to navigate a fleet of vehicles in a dynamic road network so that not only adapt to the current traffic condition but also to create collaboration in using the road infrastructure to avoid creating congestion. The goal is that the fleet of the vehicles together experiences an improved average travel time. We denote this problem as the Vehicle Navigation Problem and differentiate it from the Shortest Path Problem. The Vehicle Navigation Problem reduces to the Shortest Path Problem if the graph is static and the size of the fleet is 1.

The equivalent problem to the Vehicle Navigation Problem in the IP network is called the Packet Routing Problem. Packet Routing Problem is the problem of routing information packets through optimal paths from a source IP to a destination IP in a dynamic IP network. The commonly accepted algorithm for routing in IP networks is OSPF which is a distributed version of the SPF [36]. To adapt to the dynamics of the network, Boyan and Litman first introduced Q-routing [3]. In Q-routing, a reinforcement learning agent is assigned per router node. The agent in each router node is responsible for forwarding the packets to the next nodes based on their destination. Other similar methods were developed later based on the work of Boyan and Litman [6, 40]. More recently, deep learning and deep reinforcement learning are also proposed for adaptive routing in the IP network [17, 51, 57].

From the routing perspective, road networks and IP networks are similar. If all the vehicles in the road follow a central router server, we can treat vehicles in the road network as packets in the IP network. One way that we can be sure that vehicles will follow a central router is with the emergence of connected autonomous vehicles. Autonomous vehicles (AVs)

have a promising future because they can alleviate many of the problems in the current transportation systems [34]. Based on the current trends of technology development, AVs are expected to constitute 50% of all the traffic by 2040 [4]. Hence, it is reasonable to assume that in the imminent future, many vehicles can follow a central router agent.

We propose a *network-aware multi-agent reinforcement learning (MARL)* model for routing the AVs in the road network. We use the analogy between road networks and IP networks; An RL agent is assigned to every intersection as a router agent, and an AV is like a packet in the IP network. The AV doesn't previously know its exact route to its destination. When an AV reaches an intersection, it generates a routing query to the agent at that intersection. The agent generates a routing response based on the vehicle's destination and the current state of the network. The AV follows the routing response from the router agents until it reaches its destination.

Although we propose our method for navigation of autonomous vehicles, as long as all the vehicles, including human drivers, can communicate with the router servers, and follow their routing responses, we can relax the autonomous vehicle constraint. Vehicles may use the available V2I technologies to communicate with the router servers. Through the V2I communication, data can be delivered to the vehicles over an ad-hoc-network and vice versa. V2I uses dedicated short-range communication (DSRC) [22] frequencies to transfer data. One advantage of using the DSRC technologies is that it does not need GPS location data which can be inaccurate from time to time.

To the best of our knowledge, we are the first to proposes a network-aware multi-agent

deep reinforcement learning model for navigation in the road network that can adapt to the dynamics of the traffic and lead to the collaboration in using the road network.

1.1 Research Questions

The SPF algorithm is a widely accepted routing algorithm both in road networks and IP networks that can provide approximations of the optimal solution by assuming the network state to be static. The major research questions of this work are the following:

- What is the gap between the performance of the SPF algorithm and an optimal solution for routing in a dynamic road network?
- For path planning in a geographical environment, humans use the intuition that geographically near destinations must have similar paths. We call this characteristic of a geographical environment *the locality of access*. How can we utilize the locality of access in the road network to help the routing algorithm find the optimal path?
- How can we model the adaptive navigation problem as a multi-agent deep reinforcement learning problem that can approximate an optimal routing solution?
- How can we model the routing algorithm to be aware of the dynamic state of the network (e.g., due to traffic congestion) and learn to adapt to it?

1.2 Contributions

We have systematically studied these research questions and we list below the major contributions of the thesis:

- We motivated and formally introduced the problem of *adaptive navigation of a fleet of vehicles in a dynamic road network.*
- We proposed *a network-aware multi-agent reinforcement learning method* for addressing the problem.
- We designed a method for using *the locality of access* in the road network that enables efficient exploration of the action space for the RL agents.
- We conducted comprehensive empirical studies on both synthetic and realistic road networks to evaluate the proposed adaptive navigation method.

1.3 Thesis Organization

The remainder of the thesis is organized as follows: Chapter 2 discusses the related work, specifically, the Traffic Prediction Problem, the Vehicle Navigation Problem, the Vehicle Routing Problem, the Packet Routing Problem, and the makes a connections of our problem to the benefits of deploying Autonomous Vehicles. Chapter 3 provides background information on reinforcement learning and graph attention networks. Chapter 4 provides preliminaries and formally defines the Adaptive Navigation Problem. Chapter 5 presents our network-aware

multi-agent reinforcement learning method for addressing the Adaptive Navigation Problem. Chapter 6 discusses the experimental evaluation of the proposed method. Chapter 7 concludes the work, and provides directions of future work.

Chapter 2

Related Work

In this chapter, we first review the Traffic Prediction problem that deals with predicting various traits of the traffic such as travel time, and speed of a road. We next present another problem called the Vehicle Navigation Problem, and discuss why traffic prediction is important for vehicle navigation. We also introduce a well-known problem in the operation research named Vehicle Routing Problem (VRP) and we discuss how it is different form the Vehicle Navigation Problem. We next introduce a closely related problem to the Vehicle Navigation Problem but in the IP network, named Packet Routing Problem. Finally, we discuss the future of autonomous vehicles and why we think that autonomous vehicles have a promising future. We then explain how connected autonomous vehicles can provide an analogy between road networks and IP networks.

2.1 Traffic Prediction Problem

Traffic Prediction Problem is the problem of prediction various features of the traffic such as speed, flow, and travel time in a rode network. Traffic prediction can improve route planning, vehicle dispatching, and traffic congestion. Due to the dynamic spatio-temporal dependencies between different regions in the road network, accurate traffic prediction problem is challenging [55].

Classically, statistical methods and traditional machine learning methods were used for solving the traffic prediction problem. The most representative classic statistical algorithms are Historical Average (HA), Auto-Regressive Integrated Moving Average (ARIMA) [49] and Vector Auto-Regressive (VAR) [61]. Also, traditional machine learning methods like Support Vector Regression (SVR) [5] and Random Forest Regression(RFR) [20], are proposed for traffic prediction problem. More recently, deep learning has been proposed for travel time prediction in a dynamic road network. Deep learning models can achieve higher performance since they can build complex nonlinear models capable of capturing the spatial-temporal dependencies in the road network.

We next discuss deep learning methods for spatial dependency modeling, temporal dependency modeling, and the joint spatio-temporal dependency modeling for traffic prediction.

2.1.1 Spatial Dependency Modeling

There are two major methods for capturing the spatial dependency, Convolutional Neural Network (CNN), and Graph Convolutional Network (GCN). We next discuss each of these

methods.

Convolutional Neural Network Convolutional Neural Network (CNN) is a special Neural Network architecture capable of capturing local correlations and hence is commonly used in computer vision for feature extraction [25]. Li Y, and C Shahabi, survey a number of studies that apply CNN to capture spatial dependencies in traffic networks from two-dimensional spatio-temporal traffic data [27]. Since 2D matrices can not fully describe the traffic network, one method is to convert the traffic network structure at different times into images and divide the images into standard grids, with each grid representing a region. In this way, CNNs can be used to learn spatial features among different regions [55].

Graph Convolution Network Graph Convolution Network (GCN) is a Neural Network architecture that can capture graph-structured data correlations [23]. Due to the structure of the CNN architecture, CNN is limited to modeling Euclidean data. However, road network data structure has non-Euclidean dependencies that CNNs can not capture. Hence, the GCN is used to model non-Euclidean spatial structure data of traffic networks [55].

Modeling the traffic network as a general graph rather than a simple grid can allow full utilization of the spatial information [58]. In this graph, the nodes are the monitoring stations in the traffic network, the edges are the connections between stations, and the adjacency matrix is the distances between stations. In [58], graph convolution approximation strategies based on spectral methods are used to extract patterns and features in the spatial domain.

Geng et al. encode the non-Euclidean pair-wise correlations among regions into multiple graphs and then explicitly model these correlations using multi-graph convolution [15]. Then

they uses ChebNet [8] based GCN to model spatial correlations. Li et al. model the traffic network as a directed graph and define a new process named Diffusion Convolution Operation which can capture the influence of upstream and downstream traffic in the dynamics of the traffic [28]. Song et al. are the first to introduce an aggregation function as a linear combination of the features of the neighboring nodes and use this function for graph convolution. The weights of the aggregation function are equal to the weights of the edges between the node and its neighbors [41].

However, the impacts of the traffic of the neighboring nodes on the traffic of a road are not all the same. Moreover, these impacts change over time. The spatial attention mechanism is utilized to capture these dependencies. This mechanism learns to assign appropriate weights to different regions at every time step [38, 29, 60, 56].

2.1.2 Temporal Dependency Modeling

The well-known architecture for temporal dependency modeling is the Recurrent Neural Network (RNN). RNN predicts the traffic as a time series depending on the order of the data in the sequence. Moreover, a particular type of CNN is also used for temporal dependency modeling. We next review RNN and CNN for temporal dependency modeling.

Recurrent Neural Network Recurrent Neural Network (RNN) is commonly used for learning the sequential data dependencies, e.g., in NLP or time series prediction applications [31]. Li Y, and Shahabi C review some RNN based methods for traffic prediction [27]. In these methods, RNN is used for modeling the non-linear temporal dependency of traffic data.

These models depend on the order of the data in the sequence.

In RNN sequence learning for traffic prediction, an encoder-decoder structure is used to encode the source sequence as a fixed-length vector and use the decoder to generate the prediction [28, 38, 26, 60].

The major drawback of the RNN methods is that the ability of the model to remember the captured older patterns in the data decreases as the model sees newer data in the sequence. Hence, these models fail when applied to long sequences [55]. An attention mechanism on the previous time states can be utilized similar to the attention in the spatial methods. This mechanism allows for adaptive selection of the relevant hidden states of the encoder to produce the output sequence [60].

Convolutional Neural Network In temporal dependency modeling with CNN methods, dilated causal convolution is adopted to capture the temporal trends of a node [13, 53]. Dilated causal convolution is a one-dimensional convolution that adjusts the size of the receptive field by changing the dilation rate. This convolution allows for capturing the long-term periodic dependence in the traffic data.

One benefit of CNN compared to RNN is that CNN does not rely on the previous time steps for training. As a result, the CNN computations can be done in parallel, while RNN needs sequential computation to maintain a valid hidden state of the past [55].

2.1.3 Spatio-temporal Joint Dependency Modeling

Both spatial and temporal dependencies are usually considered in traffic prediction models. However, these models assume that spatial and temporal information are independent; hence, they do not consider mutual relations.

To solve this limitation, Yin et al. have attempted to integrate spatial and temporal information into an adjacency graph matrix [55]. Song et al. first construct a localized spatio-temporal graph that includes both temporal and spatial attributes and use the proposed spatial-based GCN method to model the spatio-temporal correlations simultaneously [41]. More recently, Fu et al. use GCN to capture the spatio-temporal dependencies in the road network and create a rich feature vector which is then used for predicting the travel time from the origin to the destination along a given travel path [14].

2.2 Vehicle Navigation Problem

Vehicle Navigation Problem is the problem of finding the optimal path between a given source and destination for a fleet of vehicles in a dynamic road network. In traffic navigation, the optimal path is equivalent to the fastest path with the shortest travel time. The goal for Vehicle Navigation Problem is that the whole fleet experiences a better average travel time.

In a deterministic scenario, when the traffic condition does not change, e.g., at 2 a.m, when the roads are empty, we can consider the road network to be static. The Vehicle Navigation Problem reduces to the Shortest Path Problem if the graph is static, and the size

of the fleet is smaller than it can affect the travel time in the road network. Conventionally, Dijkstra Algorithm [9], or shortest path first (SPF), is used to find the shortest path. SPF algorithm is a greedy algorithm that is granted to find the optimal path in a static network. Also, heuristic aided algorithms like A* [19] are proposed to improve the performance of the SPF. A* uses a heuristic function to prune the search space. Under certain conditions, A* is also granted to find the optimal path. However, road networks are not always static. The SPF path is calculated based on the estimated travel time of roads when the vehicle requests a route. The more accurate the estimated travel times are, the better the suggested route.

Although there is a vast literature on traffic prediction and estimating the travel time, since the traffic prediction problem is intrinsically complex, the resulting traffic predictions, specifically the long-term predictions are not accurate. As a result the suggested routes of the SPF algorithm prove sub-optimal in a dynamic road network. Hence, other methods have been proposed to directly route the vehicles in the dynamic network. Lin X and Hong K developed a probabilistic dynamic programming problem and solved it through a backward recursive procedure for the adaptive vehicle navigation with stochastic traffic information [50]. Tatomir et al. propose an end-to-end travel time prediction and adaptive routing using the Ant Colony algorithm [43]. More recently, Panov et al. show some preliminary results on the problem of path planning in a grid using deep reinforcement learning [39]. Koh et al. assign a separate RL agent to every vehicle for routing it according to the dynamic traffic without predicting the travel times [24]. Geng Y et al. develop a route planning algorithm based on DRL for pedestrians using travel time consumption as the metric to plan the route

by predicting the pedestrian flow in the road network [16].

2.3 Vehicle Routing Problem

The Vehicle Routing Problem (VRP) [7] is an important problem in transportation, distribution, manufacturing, and many other real-world applications. In its basic version, the problem of Capacitated VRP (CVRP) is to find the set of routes for a fleet of similar vehicles (trucks) to service all the locations that demand the service with the minimum route cost. All the vehicles begin and terminate at a central location, e.g., a warehouse. Each customer that demands a service should be visited exactly once and by exactly one route. The loading capacity and traveling distance for each vehicle are limited to a constant number. We can think of VRP as a Multi-Agent TSP problem. VRP is an NP-hard problem, and its real-life applications are relatively large in scale [11]. There are many metaheuristic approaches for solving the VRP [11]. Among these methods, there are several articles that try solving the VRP using reinforcement learning [21, 37].

While VRP tries to find an optimal sequence of locations to visit, Vehicle Navigation Problem tries to find the optimal path between an origin and a destination for a fleet of vehicles in a dynamic network. We further discuss the Vehicle Navigation Problem in section 2.2, and we formally define it later in Chapter 4.

2.4 Packet Routing Problem

Packet Routing Problem is the problem of routing information packets through optimal paths from a source IP to a destination IP in an IP network. Like a road network, the IP network is also considered a dynamic network since the network conditions change in time. The optimal path in the IP network can have various goals, e.g., the path with the shortest response time or the path with the least traffic load. For efficient communication, it is crucial to send the packets through the optimal path. Hence, the routing algorithm is a critical part of the IP network. In the following, we review different approaches for routing packets in the IP network.

2.4.1 Classic Approaches

The standard widely accepted algorithms for Packet routing in IP networks are usually a variant of the SPF algorithm. However, since there is no central moderator in the IP network, the algorithm must be distributed so that every router can apply independently. One of the most common implementations of a distributed SPF is called OSPF, Open Shortest Path First [36]. In OSPF, each router keeps the whole network state in a database called Link State Data Base or LSDB. Each router uses LSDB to find the shortest path to all the other routers.

2.4.2 Adaptive Approaches

The OSPF algorithm does not adapt to the dynamic loads in the IP network; hence other works have tried to address this problem. Boyan and Litman are the first to propose reinforcement learning for packet routing. They proposed Q-routing, a Q-learning based method that could decide for a router where to forward a packet based on its destination [3]. One big drawback of Q-routing is the hysteresis problem. Hysteresis problem happens when the network's traffic load increases for a while to an overload state and then decreases to the normal. In this condition, Q-routing fails to adapt to the new situation and follows the previous overload policies. Choi and Yeung proposed a modified version of Q-Routing with a more detailed model to address the hysteresis problem [6]. While Q-routing is a deterministic value-search algorithm, Peshkin and Savova propose a stochastic algorithm with gradient ascent policy search [40].

More recently, with the advances in deep learning, new approaches have emerged for the same problem. In [17], Geyer and Carle propose Graph Neural Networks for capturing the dynamics of the IP Network and use a multi layer perceptron (MLP) to learn the routing tables of the OSPF algorithm. However, they can not address the reliability problem. A reliable routing algorithm must not create black holes or infinite loops, which are unacceptable for any routing algorithm. Xiao et al. resolve the reliability problem using a directed acyclic graph (DAG) structure [51]. You X et al. propose an end-to-end Multi-Agent reinforcement learning algorithm for adaptive routing in the IP network [57]. They use the history of the previous routing decisions in their recurrent model architecture. However, they do not

consider the network state and its dynamics.

2.5 Autonomous Vehicles

In this section, we first discuss why autonomous vehicles have a promising future in the industry. We then discuss how connected autonomous vehicles can allow further optimization in traffic management. Finally, we discuss how autonomous vehicles can be seen as a bridge between routing in the IP network and navigation in the road network.

2.5.1 Motivations and Trends

The continuous increase in vehicles in urban areas has led to serious problems such as accidents, traffic jams, and air pollution. Human error is a primary reason for all of the mentioned problems. For example, about 90 % of road crashes root in human errors like attention drifting and drunk driving [32]. Connected autonomous vehicles (CAVs) are a potential solution for all of the above problems [34].

Recently Autonomous Vehicles have been a hot topic of research in both academia and industry [32]. Industry giants like Cruise AV from General Motors (GM), Waymo from Google, Daimler-Bosch, and Ford have invested in the AVs. The accomplishments are not limited to research, and the AVs from companies like Tesla, Google, and Volvo are now taking their baby steps in the streets [2].

Although the industry is growing exponentially, the problem is still too challenging to want to expect the AVs to replace human drivers anytime soon [4]. Based on the current

trends of technology development, AVs are expected to constitute 50% of all the traffic by 2040 [4].

2.5.2 Traffic Optimizations

Connected autonomous vehicles can allow various traffic optimization. For example, one of the main reasons for traffic congestion in metropolitan areas is the intersections and traffic lights. Dresner and Stone propose a reservation-based intersection management for Connected Autonomous Vehicles (CAVs) [10]. They simulate the movement of the CAVs in the intersection and reserve their place in the expected moment. Consequently, they allow simultaneous use of the intersection for all the vehicles that approach the intersection without the need for a traffic signal. Moreover, AVs can operate alongside human drivers. As the portion of the AVs in the road increases, the benefits also increase [12].

Moreover, connected autonomous vehicles (CAVs) enable Vehicle Platooning, improve Lane Change, and improve vehicle energy management [34]. CAVs also can alleviate car-pooling costs, which decrease and even eliminates the need for personal vehicles [4].

2.5.3 Autonomous Vehicles as Packets

With autonomous vehicles on the roads, the details of the path can be abstracted away from the passengers. A passenger requests a pick up at one location and a drop off at another location, and the AVs take care of the rest. A vehicle in the road network is like a packet in the IP network. An intersection in the road network acts as a router in the IP network.

We propose a method that enables a fleet of autonomous vehicles to adapt to the dynamics of the traffic and cooperate to change the traffic dynamics to their benefit so that all the vehicles reach a better travel time. We more formally define the problem in Chapter 4.

Chapter 3

Background

In this chapter, we first review the essential background of reinforcement learning. More specifically, we provide the background needed for understanding how the Q-learning algorithm works. Next, we dive into Graph Attention Networks as an instance of Graph Neural Networks.

3.1 Reinforcement Learning (RL)

From a broader perspective, machine learning can be categorized into three major groups: supervised learning, unsupervised learning, and reinforcement learning (RL). Sutton and Barto define reinforcement learning as finding suitable actions for an agent to take in a given situation in order to maximize a reward [42]. One major difference between RL and supervised or unsupervised learning is the training data. In supervised and unsupervised learning, the datasets are static and independent of the learning algorithm. However, in RL, we generate the data by exploring an environment based on a policy; as a result, the training

data is dependent on the learning algorithm [52].

3.1.1 Reinforcement Learning Objective

At any time step t, the agent observes an observation o_t from the state $s_t \in S$, and takes the action $a_t \in A$ according to its current policy π_t. At time step $t+1$, the environment transitions to the state s_{t+1} based on the action a_t of the agent and its underlying dynamics, which are unknown to the agent. The environment creates a reward $r_{t+1} \in R$ and sends it back to the agent. The agent uses this reward to update its policy to π_{t+1}. The goal of reinforcement learning is to maximize its cumulative discounted reward. A policy that generates the maximum reward is called π^*, which is expressed by equation (1).

$$\pi^* = argmax\ \mathbb{E}_\pi \left[\Sigma_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] (1)$$

In (1) γ is the discount factor, where:

$$0 \leq \gamma \leq 1\ (2)$$

3.1.2 Model-free RL

Many variants of reinforcement learning have been proposed in the literature, and each serves certain situations. Reinforcement Learning algorithms can be categorized into two major categories based on how they model the environment: model-based and model-free. A model-free agent does not need to generate a model for the environment. It can either learn

a policy (on-policy) or the value of the actions directly (off-policy) from its experiences of acting in the environment. On the other hand, a model-based agent creates a model of the dynamics of the environment during its interaction with the environment. In this work, we only focus on model-free reinforcement learning.

3.1.3 Temporal-difference (TD) Learning

An episodic problem(task) is a problem that finishes within finite number of time steps. In an episodic problem, the experiences are partitioned into episodes. Independent of the actions taken, each episode ends up in a terminal state. Monte Carlo methods can be applied to learn from episodic experiences. However, in Monte Carlo approaches, the learning is delayed until the end of each episode, leading to a slow learning rate. To address this issue, temporal-difference (TD) learning approaches are proposed. TD approaches update the estimates of the values of the states in every time step using the immediate reward from the environment without waiting for the end of the episode. This process is called bootstrapping, which leads to a faster convergence for TD learning.

3.1.4 Q-Learning

One of the most well-known TD learning algorithms is Q-Learning [46]. Q-Learning is an off-policy algorithm meaning that the agent tries to learn the action-values for every state without constructing a policy. In Q-Learning, the value of an action in a state is called Quality of the action ($Q(a, s)$). The Q-values are stored in a table named Q-table. The table

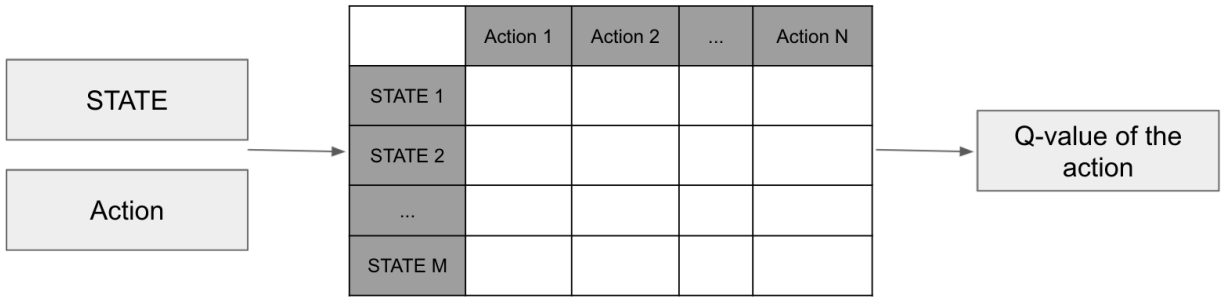


Figure 3.1: Q-learning

updates every time step with the update rule 3.1:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)) \tag{3.1}$$

where α is the learning rate and γ is the discount factor. Figure 3.1, illustrates the schema of the Q-learning algorithm with a Q-table containing Q-values for N action in M states.

3.1.5 Deep Q-network (DQN)

The number of (state, action) pairs in an RL problem can grow exponentially as the problem gets more complicated, leading to memory issues. For example, in continuous domain problems, the number of states and actions can be uncountable. To overcome this problem [33] proposes deep Q-networks (DQN). In DQN, a neural network acts as a function approximator that replaces the Q-table. This network gets the representation of an state as its input and predicts the action values in that state. The update rule 3.1 for Q-learning is equivalent to minimizing

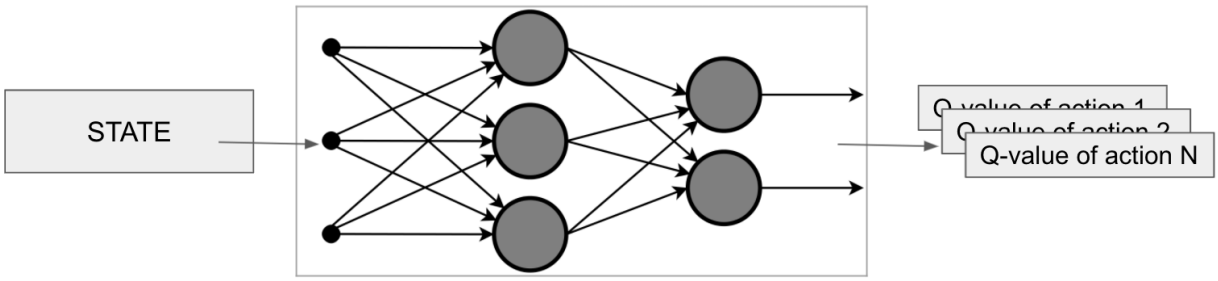


Figure 3.2: DQN

the loss for DQN defined in equation3.2:

$$L(\theta) = \mathbb{E}[(R_{t+1} + \gamma \max_{a' \in A_{t+1}, ns \in S_{t+1}} Q(ns, a' : \theta) - Q(s \in S_t, a \in A_t : \theta))^2] \tag{3.2}$$

where θ stands for all the network parameters. Figure 3.2 illustrates the schema for DQN algorithm. The Neural Network predicts the Q-values for all the possible actions.

3.2 Graph Attention Networks (GAT)

This section provides the background necessary for understanding how graph attention networks help summarize the network state. Graph Attention Network (GAT) is a type of Graph Convolution Networks (GCNs) that brings the attention mechanism to the convolution over the graph-structured data. GAT first learns a set of attention weights to each node in a neighborhood. GAT works based on message passing between neighbors and does not require any costly matrix operation such as inversion. It also doesn't need to know the graph

structure in advance [45]. Here we provide a brief review of the method presented in the original paper [45]. GAT works based on graph attentional layers and can have as many graph attentional layers as needed. The definition of all the graph attentional layers is the same.

3.2.1 Graph Attentional Layer

For a graph with N nodes, the input to the graph attentional layer is a set of N node features, Graph Signal $G_s = \{h_1, ..., h_N\}$, $h_i \in \mathbb{R}^F$. The output of the layer is a new set of node features, Graph Embedding $G_e = \{h'_1, ..., h'_N\}$, $h'_i \in \mathbb{R}^{F'}$. The dimension of G_s can be different from the dimension of G_e.

At least one layer of a learnable linear transformation is needed for transforming the input features into higher-level features. Hence, a shared linear transformation $W \in \mathbb{R}^{F' \times F}$ is applied to the input features G_s:

$$H_i = W.h_i$$

Then a shared layer of self-attention $a : \mathbb{R}^{F'} \times \mathbb{R}^F \to \mathbb{R}$ is applied between node features to compute attention coefficients. For example, $a : \mathbb{R}^{F'} \times \mathbb{R}^F \to \mathbb{R}$ can be a single-layer of feed-forward neural network:

$$e_{ij} = a(H_i, H_j)$$

The attention coefficient e_{ij} indicates the importance of node j to node i. The graph structure is taken into account by computing e_{ij} only for nodes $j \in N_i$, where N_i is the one-hop away neighborhood of i. Note that $i \in N_i$. The attention coefficients are then normalized over the

neighbors:

$$\alpha_{ij} = softmax(e_{ij}) = \frac{exp(e_{ij})}{\Sigma_{k \in N_i} exp(e_{ik})}$$

Figure 3.3 summarizes how attention scores between nodes is calculated:

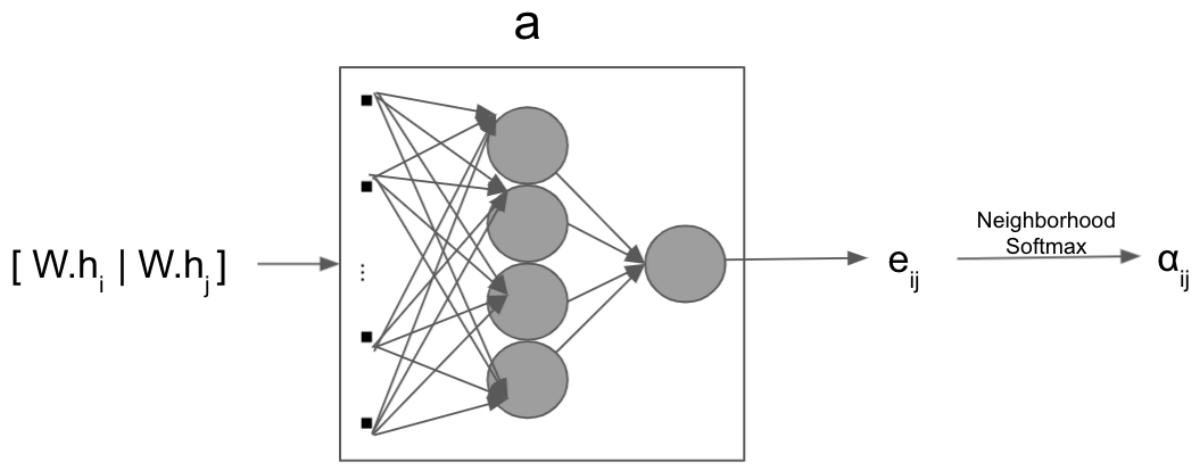


Figure 3.3: Attention Mechanism

The normalized attention coefficients are then used to compute a linear combination of the neighboring features as the output feature vector:

$$h'_i = \sigma(\Sigma_{j \in N_i} \alpha_{ij} H_j) \tag{3.3}$$

3.2.2 Attentional Heads

GAT uses multi-head-attentions to make the learning process stable. Specifically, K independent attention head mechanisms compute the transformation of equation 3.3. The K

set of embedded features are then aggregated together as the final output of the layer. The aggregation function for the intermediate layers is concatenation, and for the final layer is averaging.

Chapter 4

Problem Definition

In this chapter, we introduce some preliminary definitions and further formally define the adaptive navigation problem.

Consider a road network represented as a directed graph $W = \{I, R\}$, where $I = \{i_1, ..., i_N\}$ is a set of vertices that represent the intersections, and $R = \{r_1, ..., r_M\}$ is a set of edges that represent the roads. A road $r_j \in R$ is a directed edge from $r_{j_1} \in I$ to $r_{j_2} \in I$. Note that this assumption means that every road connects two intersections. Also, consider a set of L autonomous vehicles $AVs = \{av_1, ..., av_L\}$, and N router agents $U = \{u_1, ..., u_N\}$ each corresponding to an intersection. We first define a routing query at simulation time t as following:

Definition 1 *q: **Routing Query***

$$q =< t, av, u, r_c, i_d, t_{max} >$$

A routing query is a query for routing at the simulation time t from autonomous vehicle av

currently driving in the road r_c, to the router u, with the destination intersection i_d. Router u is the router agent assigned to the intersection r_{c_2}. t_{max} is the maximum simulation time allowed for this query.

The adaptive routing server receives a routing query, and produces a response. To define a routing response, we first define next-hop roads:

Definition 2 NH***: Next-Hop Road Set***

$$NH_{r_j} = \{r_k \in R | r_{k_1} == r_{j_2}, r_k \textit{ is connected to } r_j\}$$

The next-hop road set for the road r_j, is the set of all the outgoing roads from intersection r_{j_2} that are connected to r_j. We say that r_k is connected to r_j if the road network structure at intersection r_{j_2} allows the flow of traffic from r_j to r_k.

In figure 4.1, the next-hop road set of road r_2, $NH_{r_2} = \{r_{-1}, r_{-3}, r_{-4}\}$ is represented in blue. Note that road $r_{-2} \notin NH_2$ since U-turns are not allowed in the next intersection.

Definition 3 r_q ***: Routing Response***

$$r_q = \begin{cases} < success >, & if\ q(r_{c_2}) == q(i_d) \\ < fail >, & otherwise\ if\ q(t_{max}) < q(t) \\ < r_n \in NH_{r_c} >, & otherwise \end{cases}$$

The routing response r_q is the response to the routing query q. If the next intersection of the vehicle sending the query is equal to the destination intersection, the response is

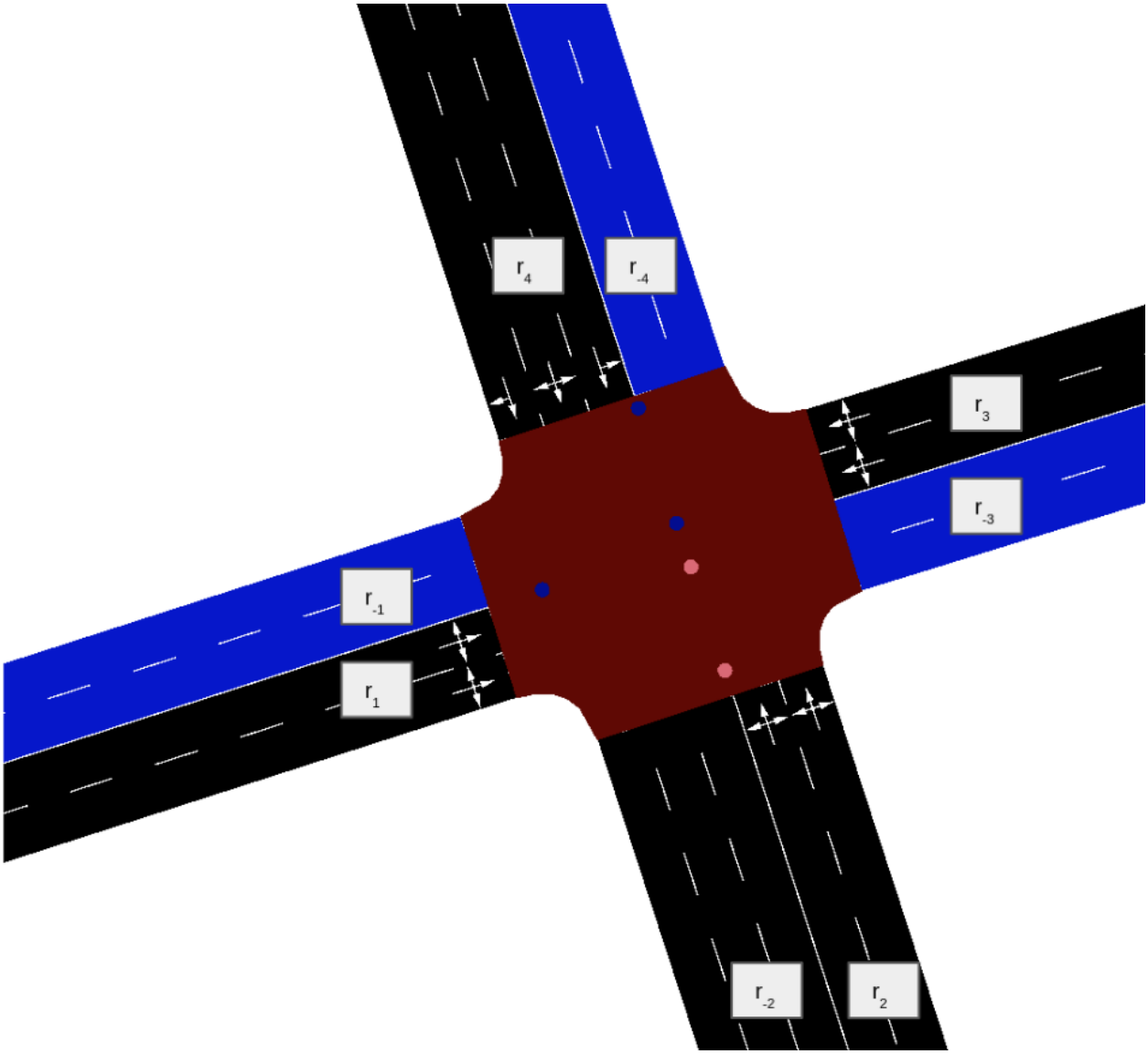


Figure 4.1: Next-Hop Road Set: the next-hop road set of road r_2, $NH_{r_2} = \{r_{-1}, r_{-3}, r_{-4}\}$ is represented in blue.

<success>, and the vehicle exits the simulation. Otherwise, if the current time exceeds the t_{max}, the response is <fail>, and the vehicle exits the simulation. Otherwise, the router server returns one road $r_n \in NH_{r_c}$ as the query response, and the vehicle follows the routing response.

For an autonomous vehicle av that comes to the simulation at time t, we define a trip as a tuple containing its origin and destination. The path of the trip is the trajectory of av following the adaptive navigation server responses.

Definition 4 T_{av}***: Trip of*** av

$$T_{av} = (av, t, r, i, t_{max})$$

is the trip of autonomous vehicle av that comes to the simulation at time step t from road r and wants to go to the destination intersection i. We denote the set of all the trips as T_{AV_s}*:*

$$T_{AVs} = \{T_{av} | av \in AVs\}$$

Definition 5 P_T***: Path of*** T

$$P_T = (r_{q_1}, r_{q_2}, ..., < success > / < fail >)$$

Path of T is a sequence of routing responses. Specifically

$$q_j = \begin{cases} < t = t_j, av = T(av), u = r_{c_2}, r_c = r_{q_{j-1}}, i_d = T(i), t_{max} = T(t_{max}) >, & j > 1 \\ < t = t_1, av = T(av), u = r_{c_2}, r_c = T(r), i_d = T(i), t_{max} = T(t_{max}) >, & j == 1 \end{cases}$$

t_j *is the simulation time for the* j_{th} *time that av reaches a routing decision point, e.g. a predefined distance to the next intersection* r_{c_2}*. We denote the length of path P as* $|P|$*, and the tail of path P as* $P(|P|)$*. We also denote the set of all paths as* $P_{AVs} = \{P_T | T \in T_{AVs}\}$*.*

Definition 6 t_P***: Travel Time of*** P *The travel time of path* P *is the simulation time for the tail of* P*,* $(P(|P|))(t)$*.*

We next define the routing failure of an episode of simulation to know about the number of *av* that fail to reach their destination. Also, we define the average travel time for all the *av*s that reach their destination:

Definition 7 RF ***and*** RS***: Routing Failure and Routing Success***

$$RF = \{P \in P_{AVs} | P(|P|) == < fail >\}$$

The Routing Failure of an episode of the simulation is the set of paths that end up in a $< fail >$ *routing response.*

$$RS = \{P \in P_{AVs} | P(|P|) == < success >\}$$

The Routing Success of an episode of the simulation is the the set of paths that end up in a $< success >$ *routing response.*

Definition 8 ATT ***: Average Travel Time***

$$ATT = \Sigma_{p \in RS} t_p / |RS|$$

The average travel time of an episode of the simulation is the average travel time of all the paths that end up in <success> routing response.

Moreover, we formally define a specific characteristic for road networks named locality of access.

Definition 9 ***Locality of Access*** *Assume that $D(i, j)$ represents the geographical (euclidean) distance between intersection i and j, $E(T(i, j))$ represents the expected travel time between i and j. A network has locality of access if for intersections $i_1, i_2, i_3 \in R$ with $D(i_1, i_2) > D(i_1, i_3)$, we have $E(T(i_1, i_2)) > E(T(i_1, i_3))$.*

Locality of access means that on average the travel time to a nearer destination should be smaller. We interchangeably call a network with locality of access as a locally accessible network.

We now formally define the ADAPTIVE NAVIGATION PROBLEM:

Problem 1 ***Adaptive Navigation*** *Consider a locally accessible road network W, and a set of routing queries Q in an episode (e.g. a day), our problem is to generate a suitable response r_q for each $q \in Q$, with the two following long-term goals:*

1. *Completeness: Minimizing $|RF|$ (Maximizing $|RS|$)*
2. *Optimality: Minimizing ATT*

Chapter 5

Methodology

In this chapter, we present our methodology for solving the Adaptive Navigation problem. To optimally solve the Adaptive Navigation problem, we have to generate collaborative routing responses that can adapt to the dynamic environment of road networks. Multi-agent reinforcement learning (MARL) is a good candidate for solving the problems that deal with collaborative decision-making in dynamic environments [59]. To model the problem with MARL formulation, we assign an RL agent to each router agent at every intersection. The RL agents keep interacting with the environment and making routing responses. When a vehicle following the routing response of an RL agent arrives at the next routing decision point in its trajectory (a predefined distance to the next intersection), it generates a new routing query. At this point, the environment generates a reward equal to the negative of the vehicle's travel time since the last time it generated a routing query. The routing query is forwarded to the next router agent, and the reward is sent back to the previous router

agent. The next agent receives the routing query and generates a new routing response. The previous agent receives the reward and uses it to update its policy.

This chapter formally presents the multi-agent reinforcement learning (MARL) formulation for the Adaptive Navigation problem. Next, we discuss an example of the collaborative routing policies. Furthermore, we discuss how the RL agents learn the end-to-end travel time estimations. We then discuss how we can enable router agents to use the locality of access in the road network to effectively explore the action space. Moreover, the router agents should be aware of the network state to adapt to its dynamics. We next discuss how we use Graph Attention Networks for aggregating the network state. Finally, we present the model architecture and the pseudo-code of the proposed algorithm.

5.1 MARL Formulation

We first present our MARL formulation for the adaptive navigation problem.

- u_i, **Router Agent at intersection** i**:** we assign a unique agent u_i to intersection $i \in I$. Agent u_i only responds to the queries $q \in Q$ with $q(r_{c_2}) == i$.

- s_q, **State of Query** q**:** the state of query q, is the unique representation of the destination intersection $q(i_d)$. We refer to this representation as $[q(i_d)]$. The choice of a suitable representation for $[q(i_d)]$, is crucial to the performance of the proposed reinforcement learning algorithm. A poor representation of $[q(i_d)]$, e.g, an N dimensional one-hot embedding or even worse a 1-dimensional unique identifier integer value, can

increase the search space dimension greatly. We will further discuss more efficient representations for $[q(i_d)]$ in section 5.4.

$$s_q = [q(i_d)]$$

- s_i^t**, State of intersection** i**:** State of the intersection i at time t consists of its one-hot id and the traffic congestion condition in its outgoing roads at time t. We denote the one-hot id of intersection i as $[i]$. A road is considered congested if the latest travel time in that road is smaller than a fixed proportion of the free-flow travel time in the road:

$$C(r) = \begin{cases} True, & travel - time(r) > \frac{length(r)}{\texttt{congestion-ratio} * free - flow - speed(r)} \\ False, & else \end{cases}$$

Where, `congestion-ratio` is a hyper-parameter defined in table 6.2, $travel - time(r)$ is the travel time of the latest vehicle that has entered the road r, $length(r)$ is the length of the road r and $free - flow - speed(r)$ is the free-flow speed in the road r. If a road is congested, its corresponding value in s_i^t is set to 1 otherwise 0:

$$s_i^t = [[i] \mid [1 \; if \;\; C(r) == True \; else \; 0]] \; r \in R, r_1 == i$$

- s_W^t**, State of road network** W **at time** t**:** The state of the road network W at time t is the concatenation of the state of all the intersections:

$$s_W^t = [s_1^t | ... | s_N^t]$$

- $S^t_{u_i}$**, Agent** u_i **state set at time** t**:** At time step t, there can be more than one routing query for agent u_i. We concatenate the state of each query with the current state of the network, and add it to the agent u_i state set:

$$S^t_{u_i} = \{s^t_q = [s_q|s^t_W]|q \in Q, q(u) == u_i\}$$

- $a(s^t_q)$**, Action of agent** u_i **for state** $s^t_q \in S^t_{u_i}$**:** selecting one of the outgoing road-segments of the intersection i.

$$a(s^t_q) = r_q =< r_n >, r_n \in NH_{q(r_c)}$$

- $ns(s^t_q)$**, Next State for** $s^t_q \in S^t_{u_i}$**(Transition from** s**):** Assume that at time t an autonomous vehicle av generates the routing query q, and receives the routing response r_q. Assume that $r_{q'}$ is the next routing response in the path $P_{T_{av}}$ after r_q. Then, $s^{q'(t)}_{q'} \in S^{q'(t)}_{q'(u)}$ is the next state for s^t_q:

$$ns(s^t_q) = s^{q'(t)}_{q'}$$

- $r(a(s^t_q))$**, Reward:** Assume that at time t an autonomous vehicle av generates the routing query q, and receives the routing response r_q. Assume that $r_{q'}$ is the next routing response in the path $P_{T_{av}}$ after r_q, and $t' = q'(t)$:

$$\Delta T = t' - t$$

$$r(a(s^t_q)) = -\Delta T \tag{5.1}$$

The definition of the reward function is counter-intuitive since it does not explicitly take into account the distance to the destination. However, in combination with the dynamic programming nature of Q-learning, this reward function guides the Q-learning to learn Q-values that are end-to-end estimations for the travel time to the specified destination in the given network state. We will further discuss why this reward function leads to end-to-end travel time estimations in section 5.3.

- $(s_q^t, a(s_q^t), ns(s_q^t), r(a([s_q^t]))$**, Experience tuple:** For the completeness of our formulation, we introduce an experience memory buffer for every agent. Every agent holds an experience replay memory buffer which contains tuples of previous experiences of the agent routing decisions in different situations. The agents sample a batch from this memory and learn from the batch of experiences.

5.2 Collaborative Policies

The Adaptive Navigation algorithm is able to explore collaborative policies. Specifically, in update rule 3.1, the q-network of an agent updates with the value $Q(S_{t+1}, a)$ which comes from the q-network of the neighboring agent. As a result, the training of the agents is intertwined, which allows finding collaborative routing policies.

As an example of an effective collaborative policy, consider the network represented in figure 5.1. Assume that the green nodes (1,2,3) are sources and the red nodes are the destinations (4,5,6). In this scenario, a flow of vehicles may come from any of the sources and

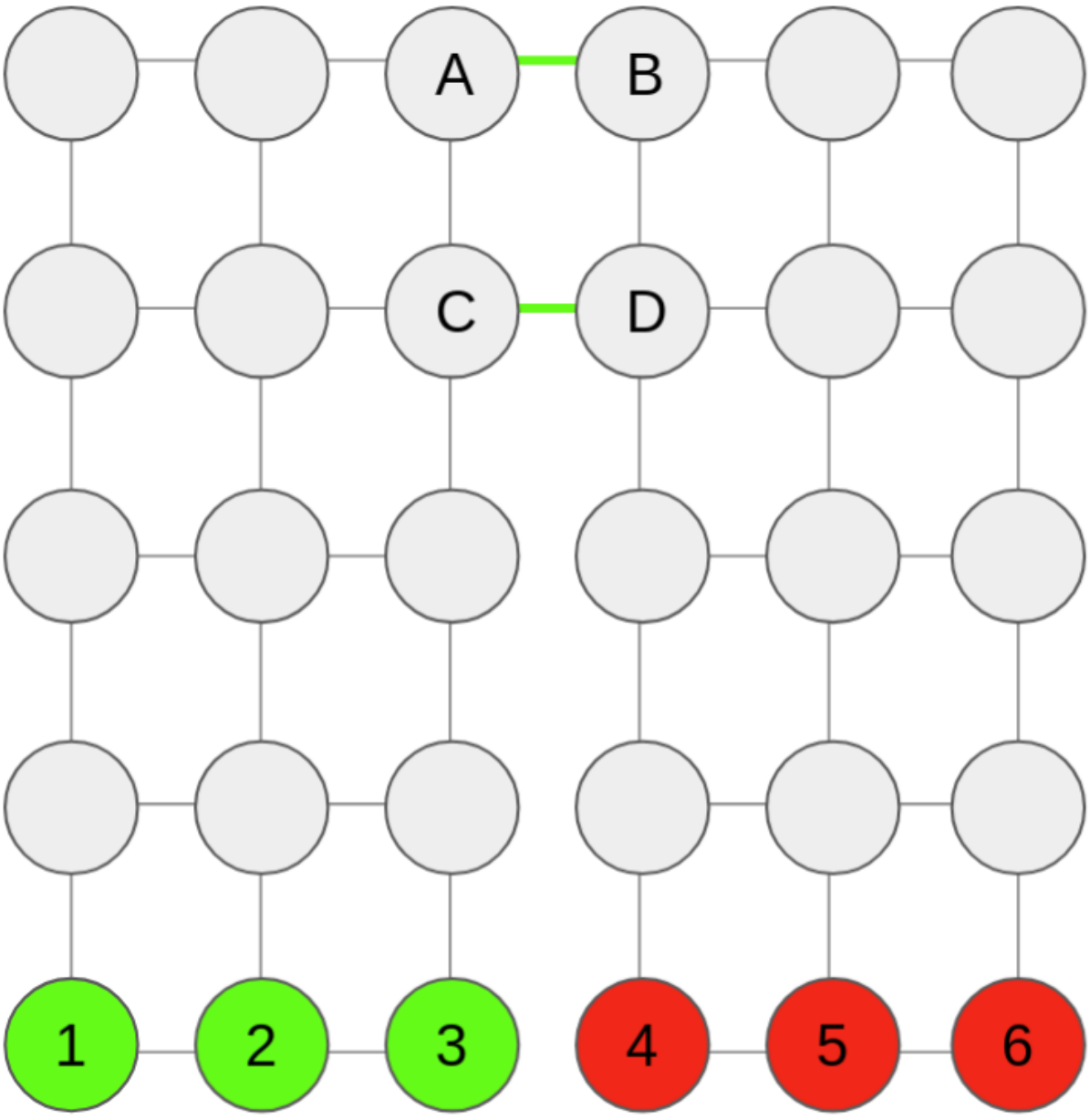


Figure 5.1: Collaborative Policies: (1,2,3) are the source nodes while (4,5,6) are the destination nodes. In a high-load scenario, the SPF algorithm oscillates between using the bridge AB and CD. Adaptive Navigation can divide the traffic between the bridges and avoid congestion.

go to any of the destinations (other nodes are not either a source or a destination). There are two bridges in the network that connect the two sides of the network (AB,CD).

As we discussed in chapter 1, the SPF routing algorithm can cause congestion by greedily sending every vehicle to the shortest path. When the network is in the under-load state, the SPF algorithm sends all the vehicles through the CD bridge because it has a shorter travel time. When the network load goes up, the CD bridge gets congested, and the SPF algorithm sends all the vehicles through the new shortest path AB, freeing CD and congesting AB. In this scenario, the SPF algorithm oscillates between the two paths. However, on the other hand, Adaptive Navigation explores all the possible policies. An optimal policy may choose to divide the vehicles between the bridges, e.g., all the vehicles with destination 6 may go through the AB bridge, and all the vehicles with destination 4 or 5 may go through the CD bridge. Boyan and Litman [3] were the first to observe this collaboration in their Q-routing algorithm in the IP network.

5.3 End-to-End Travel Time Estimations

Equation 3.1 shows the update rule for the RL agent for the finite horizon case ($\gamma \neq 0$). The finite horizon assumption allows the agent to care more about the earlier rewards in time, which leads to the convergence of the algorithm. However, to get an insight into the Q-values, we can consider the infinite horizon ($\gamma = 1$) with a learning rate=1. The equation

3.1 simplifies to equation 5.2:

$$Q(S_t, A_t) = R_{t+1} + \max_a Q(S_{t+1}, a) \tag{5.2}$$

RL uses dynamic programming for learning the Q-values. In the beginning of the training, agents start learning the Q-values for the terminal states. Then, the Q-values for the non-terminal states are learned based on the Q-values learned for terminal states.

Assuming that S_{t+N} is a terminal state, we can expand equation 5.2 as 5.3:

$$\begin{aligned} Q(S_t, A_t) &= R_{t+1} + R_{t+2} + \max_a Q(S_{t+2}, a) \\ &= R_{t+1} + R_{t+2} + ... + R_{t+N} \end{aligned} \tag{5.3}$$

Equation 5.4 is the result of substituting the reward function 5.1 in 5.3:

$$Q(S_t, A_t) = -\Delta T_1 - \Delta T_2 - ... - \Delta T_N \tag{5.4}$$

Equation 5.4 shows why with the infinite horizon assumption, Q-values are an end-to-end estimation of the travel time to a given destination in a given network state. Moreover, equation 5.4 shows that a next-state (S_{t+1}) which is further away from the destination, has smaller Q-values in expectation than a next-state which is nearer to the destination. Hence, a fast step away from the destination is not preferred to a slow step toward the destination, although the immediate reward for the former is bigger.

5.4 Preserving the Locality of Access

Intuitively, we know that road networks are locally accessible. Assume that $D(i, j)$ denotes the euclidean distance between intersection i and j, $E(T(i, j))$ denotes the expected travel time between i and j, and $D_e(i, j)$ represents the distance between the representation of i and j in the embedding space. Assume that for intersections $i_1, i_2, i_3 \in I$ we have $D(i_1, i_2) > D(i_1, i_3)$, then, local accessibility of the road network requires that $E(T(i_1, i_2)) > E(T(i_1, i_3))$. A representation for $[q_t(i_d)]$ can preserve the locality of access if $D_e(i_1, i_2) > D_e(i_1, i_3)$ in the embedding space.

With a representation that preserves the locality, the intersections that fall into the same geographical location get similar IDs and are embedded together in the embedding space. As a result, a cluster of intersections in the road network should be clustered together in the embedding space. Similar destination intersection IDs activate similar nodes in the neural network of the agents leading to similar output actions. This functionality matches human intuition toward navigation in a road network, and hence, can help the agents to search the action space more efficiently.

We propose to use the Z-order curve, or Morton space ordering curve [35] for preserving the locality of access. The Z-order curve maps multi-dimensional data to a one-dimensional array while trying to preserve the locality of the data points. The Z-order curve first computes a Z-value for every multi-dimensional point. The algorithm interleaves the binary representations of the coordinate values of the multi-dimensional point to compute the Z-value. Figure 5.2 shows the Z-values for the two dimensional case with integer coordinates [48]. The Z-order

curve algorithm then sorts the points according to their z-value and creates a one-dimensional array. Note that the Z-order algorithm looses part of the locality information, however, it can still work well enough for our training purposes.

For example, figure 5.3 shows five intersections and how the Z-order curve works for these intersections. The array of these five intersections in order of the computed Z-values is i_4, i_5, i_1, i_2, i_3. We use the binary representation of the intersection index in the Z-order array as the ID representation of that intersection. For example, in the previous Z-order array the ID of intersection i_1 is $binary(2) = [0, 1, 0]$.

Note that we need $log_2(N)$ bits to represent the intersection ID, where N is the number of the intersections. This representation while preserves the locality of access to some extent it also substantially reduces the state dimension.

5.5 Aggregating the Network State

Since the travel time in the road network is a function of the network state s_W^t, the router agents should be aware of the network state to be able to make informed routing decisions. However, the routing decision of the agent u_i is more dependent on its intersection state at the current time s_i^t and probably its neighboring intersections states. The whole network state s_W^t, contains a lot of unrelated information that is irrelevant to a single agent. Moreover, even for an average size network, the representation of the whole network's state might be of a prohibitively high dimension (i.e., very large). As a result, using the whole network state increases the model complexity of the agents, which makes the training harder and more

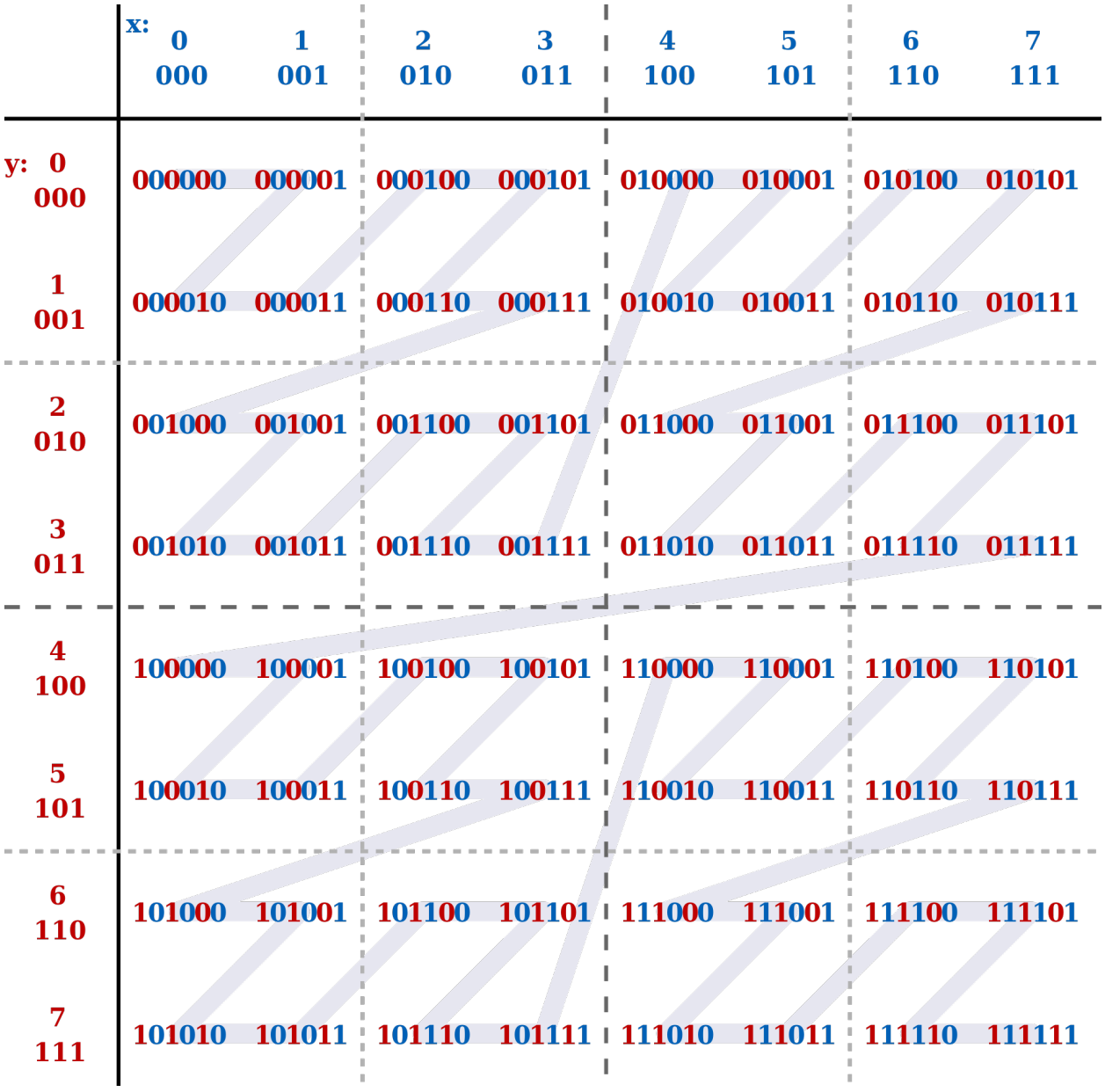


Figure 5.2: Z-values for 2D case with integer coordinates [48]

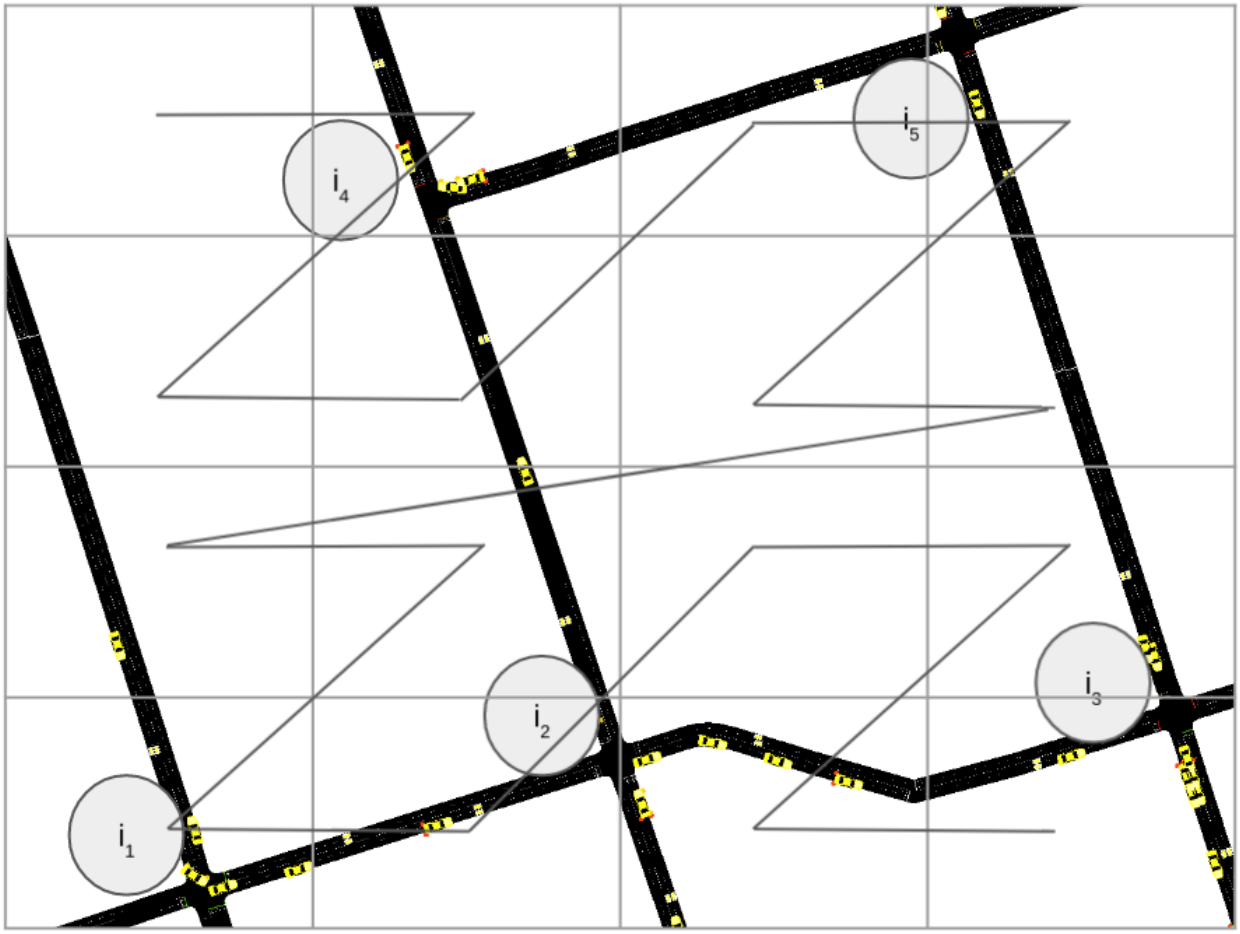


Figure 5.3: Z-order Curve: The Z-order for intersections i_{1-5} is i_4, i_5, i_1, i_2, i_3. The representation for intersection i_1 is $[i_1]$=binary(2)=[0,1,0]

time-consuming. Hence, passing the whole network state s_W^t directly to the agents is not a good choice.

On the other hand, the state of the neighboring intersections can have an important role in the routing decisions of an agent. Hence, we want the agent to be aware of the state of its neighboring intersections. The importance of the state of different neighbors in the routing decision of the agent are not all the same. Moreover, the importance of different neighbors varies in different situations.

With the mentioned goals in mind, we use Graph Attention Networks (GAT) to capture the network state embedding for the neighborhood of an agent. Graph Attention Networks use an attention mechanism to capture the importance of the neighboring intersections states to the agent for making the routing decision. Based on the attention values, GAT aggregates the state of an intersection neighborhood.

The number of GAT layers indicates the range of the neighborhood that GAT aggregates. One layer of GAT aggregates the immediate neighbors of an intersection and the second GAT layer is capable of capturing information from two-hop-away neighbors of an intersection.

We pass the network state s_W^t as a graph signal through the GAT model. The network state embedding for the agent u_i neighborhood at time t is the corresponding embedded representation of the intersection state s_i^t in the embedded network state :

$$[s_{W,i}^t] = (GAT(S_W^t))[i]$$

5.6 Model Architecture

Figure 5.4 shows the architecture of the Adaptive Navigation algorithm. The number of GAT layers is a hyper-parameter we can tune. For example figure 5.4, assumes two GAT layers.

The network state S_W^t is passed to the GAT model as a graph signal and produces intersection state embeddings:

$$[S_{W,i}^t] = (GAT(S_W^t))[i]$$

On the other hand, the state of the routing queries S_q, goes through a linear layer (one linear layer per agent), and produces embeddings of the destination IDs ($[S_q]$):

$$[S_q] = ReLU(Linear_i(S_q))$$

The concatenation of $[S_q]$ and $[S_{W,i}^t]$ is passed to the Q-network of agent u_i to get the Q-values of the actions:

$$Q\text{-}values_{u_i} = Q_i([[S_q]|[S_{W,i}^t]])$$

The routing response of the agent is the action with the highest Q-value.

To train the neural networks, we follow the conventional MSE loss in the Q-learning algorithm per agent:

$$L(\theta) = \mathbb{E}[(r + \gamma \max_{a'} Q(ns, a' : \theta) - Q(s, a : \theta))^2]$$

Back-propagating the loss leads to end-to-end optimization of all the network parameters θ (Q-networks, and GAT learnable parameters). Note that all agents contribute to the training of the Graph Attention Network model while only optimizing their own Q-network.

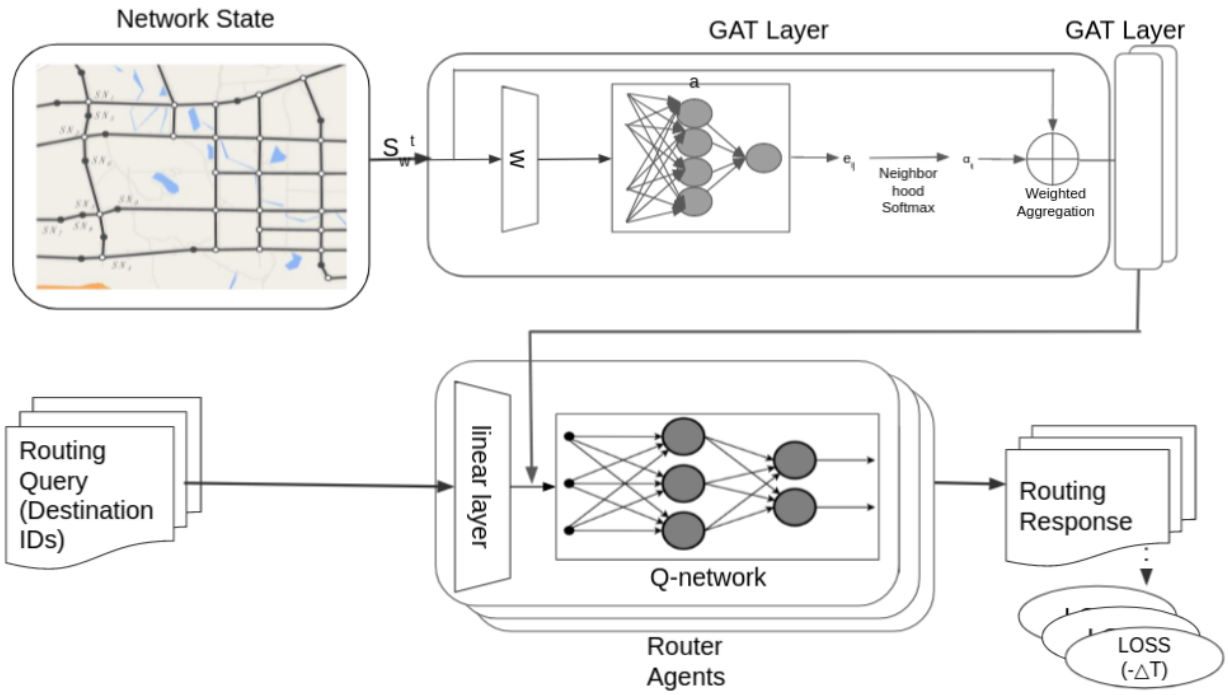


Figure 5.4: Adaptive Navigation Model Architecture

5.7 Algorithm Sketch

Algorithm 1 shows the procedure for the proposed methodology. The inputs of the algorithm 1 are the network state (s_W^t) and the set of all the routing queries at the current time step (Q). The algorithm generates appropriate routing responses r_q for $q \in Q$.

At every simulation time step, the algorithm 1 is called once. In line 4 of algorithm 1, the previous experience tuple is pushed into the experience replay memory buffer of the agent. The contents of the memory are later used for the training of the agent.

In line 5 of the algorithm 1, a linear transformation is applied to the state of the query q to create intersection ID embedding $[s_q]$. In line 6, the GAT module aggregates the network state for the neighborhood of the agent u to create the network state embedding $[s_{W,i}^t]$. In line 7, the aggregation of the network state embedding and the intersection ID embedding is passed to the Q-network of agent u. The routing response for query q is the action with maximum Q-value. Line 10 calls the training procedure, which is shown in the algorithm 2.

Algorithm 2 is responsible for the training of all the agents. In line 2, it checks if it is time for the training of the agent u. The agent has to have enough new experiences and also the total number of its experiences should be more than the batch size. Line 3, uniformly samples a training batch from the experience memory of the agent u. Lines 4 and 5 do the optimization for agent u. Note that the optimization of the GAT module happens during the optimization of the agents.

Algorithm 1 Adaptive Routing at time t

Input: state of the road network s_W^t, set of all the routing queries Q at time t,
Ensure: optimal r_q for $q \in Q$

for all $q \in$ Q **do**
  $s_q \leftarrow$ state of query q
  $u \leftarrow q(u)$, the router agent
  $u.memory$.push(previous experience tuple of agent u)
  $[s_q] \leftarrow ReLU(Linear_u(s_q))$
  $[s_{W,i}^t] \leftarrow GAT((s_W^t))[i]$, the embedded network state for the agent u with index i
  $r_q \leftarrow$ argmax $Q - net_u([[s_q]|[s_{W,i}^t]])$
end for
$U = \{q(u)|q \in Q\}$
train(U)

Algorithm 2 Training

Input: set of all router agents U
Ensure: training of RL agents

for all $u \in U$ **do**
  if time-to-learn(u) **then**
    training-batch=u.memory.sample()
    loss=MSE-loss(training-batch)
    loss.backward()
  end if
end for

Chapter 6

Experimental Evaluation

In this chapter, we present our experimental evaluation of the Adaptive Navigation algorithm. We first overview the experimental setup, including, baselines, datasets, and the evaluation metric. Then we evaluate the performance of the algorithm by following the online-training and offline-testing paradigm for reinforcement learning. We also investigate how effective our method is in preserving the locality of access. Next, we take a deeper look into how the Graph Attention Network aggregates the network state of the neighborhoods.

6.1 Experimental Setup

6.1.1 Simulator

The RL agents learn by interacting with an environment. To create an environment, we have to simulate the road network and the traffic. For traffic simulation purposes, we used Simulation of Urban Mobility (SUMO), an open-source, portable, microscopic, and continuous multi-modal traffic simulation package designed to handle large networks [30]. We used the SUMO-provided python API, TraCI, for interacting with the simulation.

6.1.2 System Specifications

All the experiments were conducted on the Data Mining Lab Tiger GPU server at Lassonde School of Engineering, York University. The specification of the server is presented in table 6.1

System Specifications			
System	**CPU**	**Memory**	**GPUs**
Tiger	2 x Intel Xeon E5-2687W v4 3.0 GHz 12-Core Processor, 30 MB L3 Cache	8 x 64 GB RAM	8 x NVIDIA MSI GeForce GTX 1080Ti 11 GB Aero OC

Table 6.1: List of systems used for the experiments

6.1.3 Datasets

We use both synthetic and realistic network data with synthetic traffic demands for our evaluation purposes.

Networks

Grid Network: We used NetEdit which is a part of the SUMO package to generate a 5x6 grid network. Figure 6.1 shows the schema of the network. In this network all the roads are identical. The 4-way intersections are set to have traffic signals. This network is controlled by 26 router agents (the 4 corners don't need agents). The small yellow squares, are the routing zones, where the autonomous vehicles get their next routing response (a predefined distance to the next intersection).

Downtown Toronto: Since a grid network is a regular network, it is not comprehensive for showcasing the potentials of the Adaptive Navigation algorithm. To have a more realistic

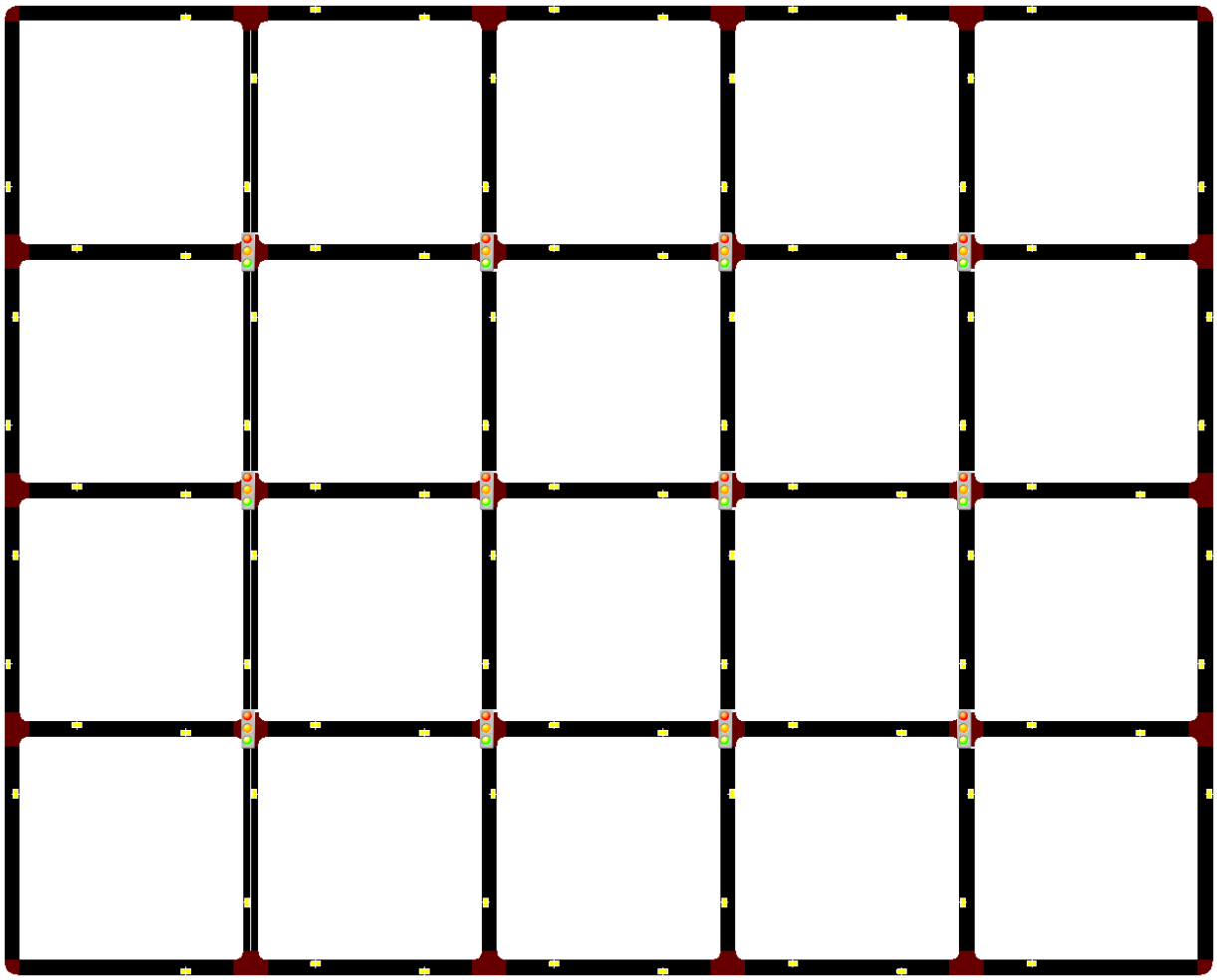


Figure 6.1: Synthetic network, 5x6 grid

experiment, we used WebWizard which is also a part of the SUMO package to extract the Downtown Toronto network structure (from Bathurst Street (west) to Don Valley Highway (east) and from Gardiner Expressway (south) to Bloor Street (north)) from Open Street Maps [1]. We abstracted the network by removing the less important roads (the roads with one lane only, the alleys,...), and their associated network structure. Figure 6.2 shows the abstracted network plotted over the real network from Google Maps. We are left with 52 intersections that need to be assigned a routing agent. The connecting edges after abstracting the network, include different types of roads, e.g. one highway.

Traffic Demand, and Network State

Since we don't have access to the detailed real-world traffic demands, we turn to use synthetic traffic demands. The hyper-parameters used for creating the synthetic traffic demands and network state are presented in table 6.2.

To further approximate the real-world traffic demands, the synthetic demand consists of a uniform demand and a biased demand.

Uniform Demand: the Uniform Demand is a set of origin-destination tuples distributed uniformly in the road network. More formally:

$$Uniform\ Demand = \{(r, i) | r \in R, i \in I\}$$

Where r is selected uniformly at random from set of all roads R, and i is selected uniformly at random from set of all intersections I. With the fixed period `uniform-demand-period`, we select from the Biased Demand set and create a trip for the selected demand at the current

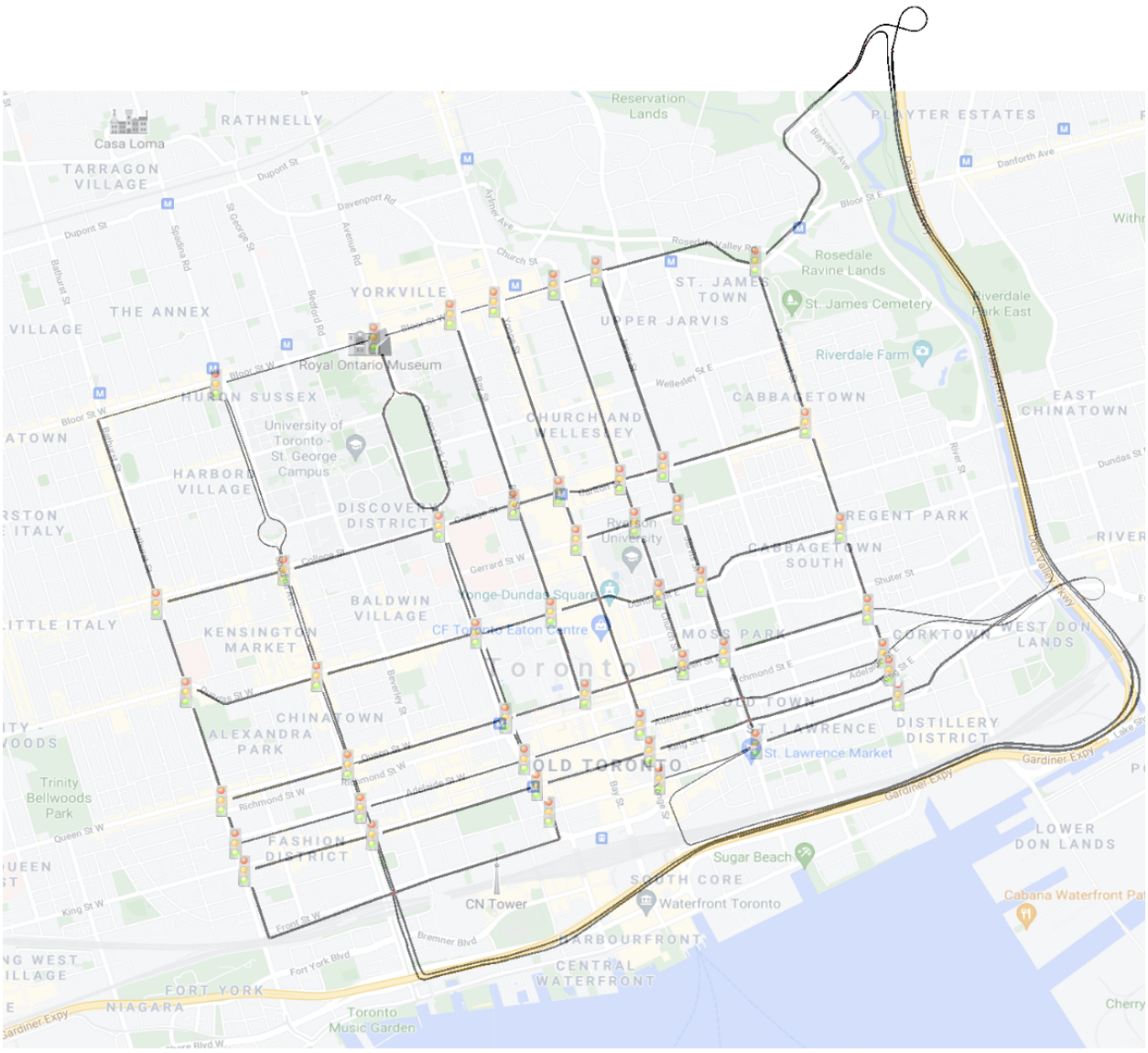


Figure 6.2: Abstracted network, Downtown Toronto, The black roads are the abstracted roads. The abstracted network is plotted over the real network from Google Maps.

simulation time. We denote the total number of trips created from the Uniform Demand set as *uniform-demand-dispatched*.

Biased Demand: the Biased Demand is a set of user-defined origin-destination tuples. More formally:

$$Biased\ Demand = \{(r^*, i^*) | r^* \in R, i^* \in I\}$$

Where r^* and i^* are predefined user-selected roads and intersections. We randomly select five tuples for the Biased Demand ($|Biased\ Demand| = 5$). We denote the total number of trips created from the Biased Demand set as *biased-demand-dispatched*, and the ratio of the total number of trips created from the Biased Demand set to the total number of trips created from the Uniform Demand set as *current-b2u*:

$$current - b2u = \frac{biased - demand - dispatched}{uniform - demand - dispatched}$$

Whenever we create a trip from a tuple selected from the Uniform Demand set, if $current - b2u$ is smaller than `biased-2-uniform-ratio` we select a tuple randomly from the Biased Demand set and create a trip for that tuple. Hence, the $current - b2u$ remains constantly equal to `biased-2-uniform-ratio` during the simulation.

Network State To further simulate the dynamic nature of the road network, we simulate congestion in the road network by reducing the maximum allowed speed in a road.

Specifically, with the fixed period `traffic-state-change-period`, we apply a network state change. In a network state change, every out-going road of an intersection is prone to congestion with the predefined probability `congestion-epsilon`. If a road is selected for congestion, its maximum allowed speed is set to the fraction of its initial maximum allowed

speed. This fraction is a hyper-parameter named `congestion-speed-factor`.

With `sim-step-per-episode=5000` and `traffic-state-change-period=500`, the traffic state changes 10 times in an episode. In a traffic state change, every road is prone to congestion with a relatively high probability of congestion-epsilon=0.25. In event of congestion, the congestion is set to be heavy with `congestion-speed-factor=0.1`, so that a sub-optimal routing response has a substantially worse travel time. The more dynamic the network state, the more room for Adaptive Navigation algorithm to further improve the performance. Changing these hyper-parameters can lead to various scenarios, however, without the loss of generality we set the hyper-parameters in a way to allow a highly dynamic scenario. For simplicity, we keep the `congestion-epsilon` and `congestion-speed-factor` similar among all the roads, however, in a more realistic scenario, these hyper-parameters can be various.

6.1.4 Baselines

We compare our method with the two well-known baselines in path planning, and one RL based path planning in the IP network:

- **Travel Time Shortest Path First (SPF)**: a path is calculated based on the current travel time in each road segment when the vehicle starts its trip from the origin to the destination road segment. This method finds the path with the shortest travel time in the current situation of the network. This baseline is available in SUMO as predefined path planning algorithms [30]

- **Travel Time Shortest Path First with Rerouting (SPFWR)**: this method is

similar to the previous method, however, every time that the vehicle changes road, a new route based on the new situation of the network is calculated. We use the shortest path planning algorithm of SUMO every time a vehicle reaches an intersection and update the path.

- **Q-routing (QR) [3]** As another baseline, we consider the work in [3] which uses reinforcement learning for packet routing in the IP network. We implemented a deep learning version of their work and applied it to the road network. This baseline is not aware of the network state.

We also compare different versions of our Adaptive Navigation algorithm:

- **AN(0hop)** In this version we short circuit the graph attention module and directly feed the intersection state to the router agent.
- **AN(1hop)** In this version we apply one layer of message passing in the graph attention network and create a layer-1 embedding for the network state. We pass the layer-1 embedding to the router agent.
- **AN(2hop)** In this version we apply two-layer of message passing in the graph attention network and create a layer-2 embedding for the network state. We pass the layer-2 embedding to the router agent.

6.1.5 Evaluation Metric

Following the previous works in traffic optimizations, we use the average travel time as an evaluation metric. The average travel time is frequently used as an evaluation metric in the

transportation field. For a given period and a selected area, it calculates the average travel time of all the vehicles between their origin and destination in the selected area [47]. In our experiments the average travel time is defined as definition 8 in chapter 4:

$$ATT = \Sigma_{p \in RS} t_p / |RS|$$

6.1.6 Hyper-parameters Settings

Tables 6.2, 6.3, and 6.4 summarize the Hyper-parameter Settings.

Parameter	**Value**	**Parameter**	**Value**
Max-number-vc	200	uniform-demand-period	5
biased-2-uniform-ratio	0.1	traffic-state-change-period	500
sim-step-per-episode	5000	congestion-epsilon	0.25
congestion-speed-factor	0.1		

Table 6.2: Simulation Hyper-parameters

Parameter	**Value**	**Parameter**	**Value**
Optimizer	Adam	Optimizer eps	1e-4
learning rate	0.01	batch-size	64
batch-norm	False	gradient-clipping-norm	5
buffer-size	10000	num-new-exp-to-learn	1
tau	0.01	discount rate	0.99
epsilon-decay-rate-denom	num episodes/100	stop-exploration-episode	num-eps-10
linear-hidden-units-size AN(0hop)	[8,6]	linear-hidden-units-size AN(1hop)	[10,6]
linear-hidden-units-size AN(2hop)	[12,9,6]		

Table 6.3: Q-Learning Agents Hyper-parameters

Parameter	Value	Parameter	Value
Optimizer	Adam	num-heads-per-layer	3
Optimizer eps	1e-4	learning rate	0.01
add-skip-connection	False	bias	True
dropout	0.6	layer-0 output dimension	7
intersection state dimension	4	layer-1 output dimension	10

Table 6.4: Graph Attention Network Hyper-parameters

6.2 Performance

6.2.1 Training

In the training phase, we define an episode as a fixed number of simulation time steps specified as the hyper-parameter `sim-step-per-period`. We don't set a deadline for arrival ($t_{max} = \infty$). Also, to avoid over-populating the network during the training which can cause a deadlock, we limit the number of vehicles in the simulation to a fixed number specified as the hyper-parameter `Max-num-vehicles`. For every episode we report, the Routing Success (number of vehicles that reached their destination during this episode), and Average Travel Time (for vehicles that reached their destination).

Figures 6.3 and 6.4, show the training results for 800 episodes of training for Downtown Toronto and 5x6 network respectively. In figures 6.3a,6.3b,6.4a,6.4b the Y-axis shows the average travel time in seconds, and the X-axis shows the episode number. Figures 6.3b, 6.4b show the average travel time in the final episodes. In figures 6.3c, and 6.4c the Y-axis shows the number of vehicles that have successfully arrived at their destination in that episode (|Routing Success|), and the X-axis shows the episode number.

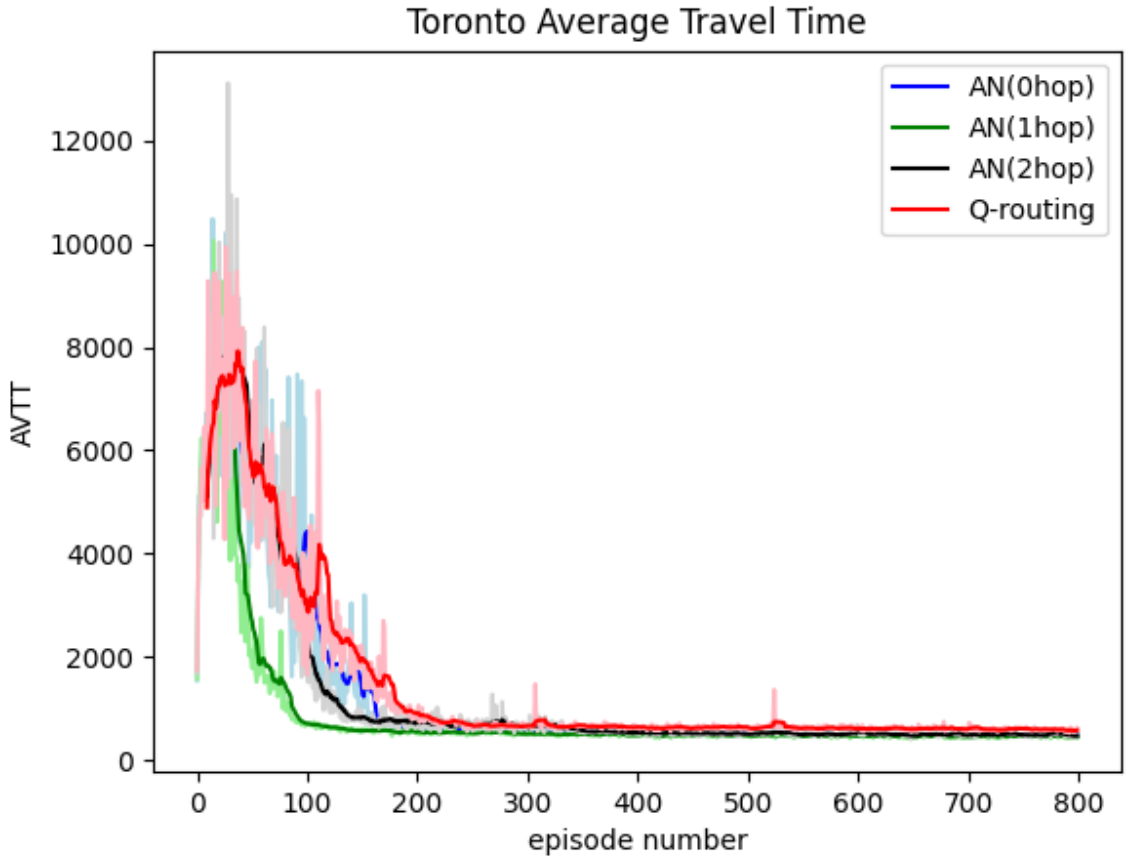
All the versions of the algorithm and the Q-routing baseline converge in 800 episodes.

Note that SPF and SPFWR baselines do not need training hence are not plotted in the figures. The AN(2hop) and AN(1hop), converge in fewer episodes, however, their training time is higher due to the higher complexity of their models. The training curves suggest that the Q-routing baseline is performing sub-optimally as expected. AN(1hop) performs better than the other versions of the algorithm. The gap in Routing Successes between different baselines is not comparatively big. However, these curves are not conclusive about how the algorithms truly perform. Note that this experiment setup allows infinite loops for the routing. If a vehicle turns around in an infinite loop it never reaches a point to affect the average travel time (AVTT). Hence, we further need to evaluate the algorithms in a setting that detects if infinite loops exist.

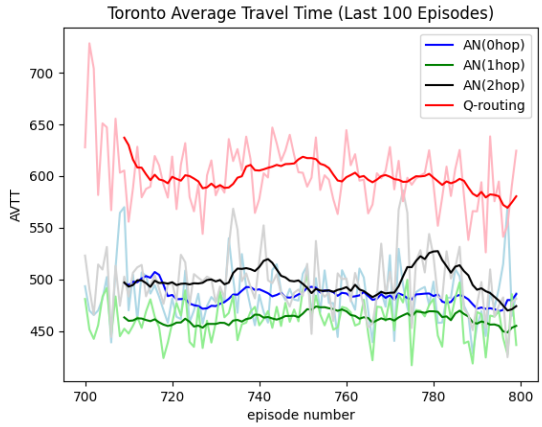
6.2.2 Testing

For testing purposes, to avoid the random noises, instead of having a flow, we predefined 2000 uniform trips. We also set a similar random seed for network state changes so that the congestion patterns are identical between the baselines. An episode in the testing phase lasts as long as all the vehicles reach their destination ($t_{max} = \infty$). In the testing phase, the Routing Success is equal between episodes = 2200 vehicles with `biased-2-uniform-ratio==0.1`. Table 6.5 shows the average results of running 5 episodes in the testing phase.

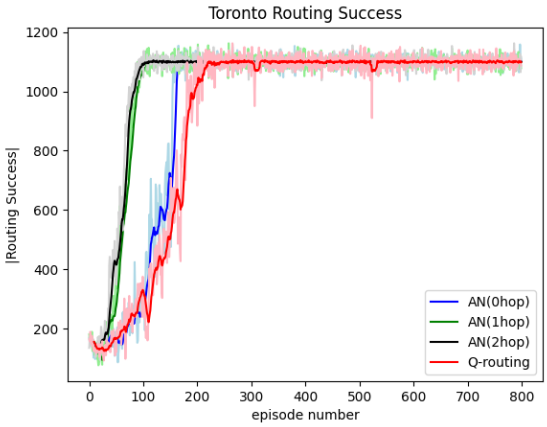
In the Downtown Toronto experiment, SPFWR has the best performance, and AN(1hop) marginally stands in second place. However, SPFWR is computationally heavy since it needs to compute all-pairs shortest path in every time-step. For example, 5 episodes of testing with SPFWR takes 23 minutes while with AN(1hop) only takes 5 minutes. AN(1hop) is fast since



(a) Average Travel Time

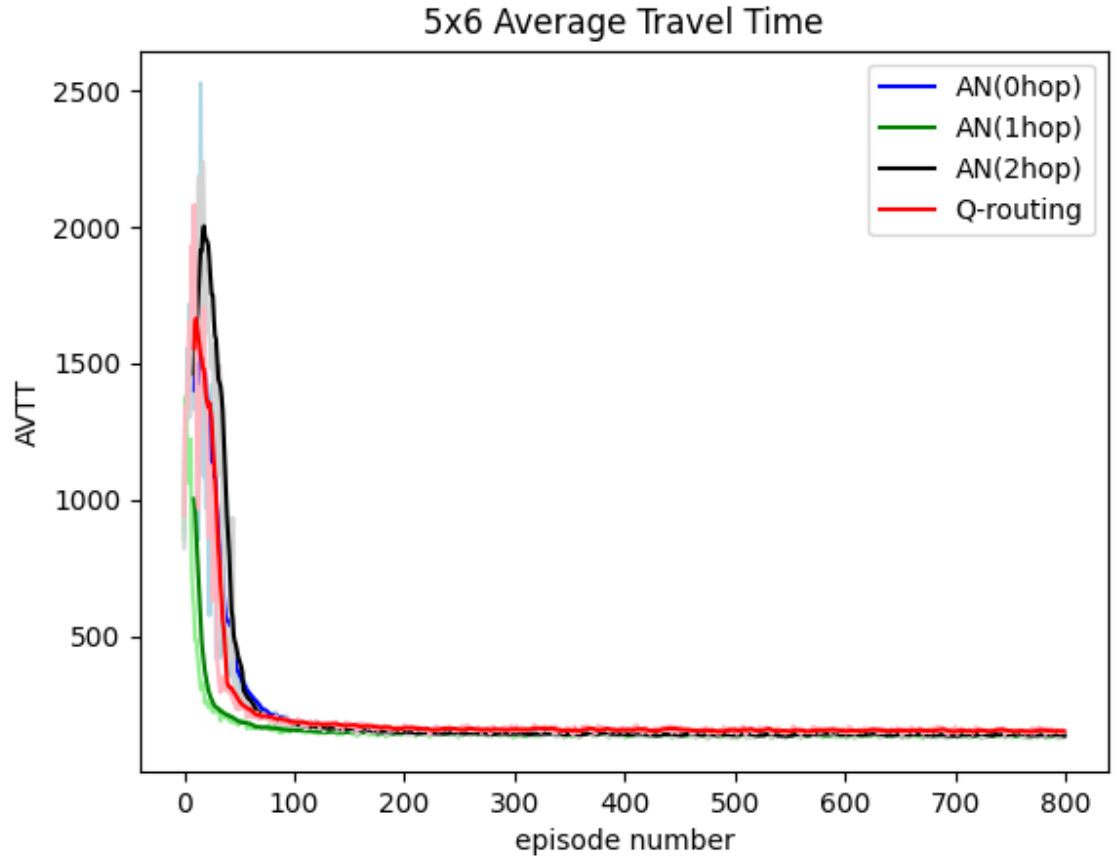


(b) Average Travel Time (Last 100 Episodes)

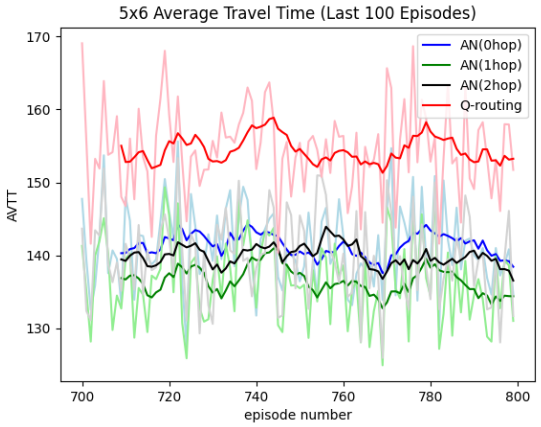


(c) Routing Success

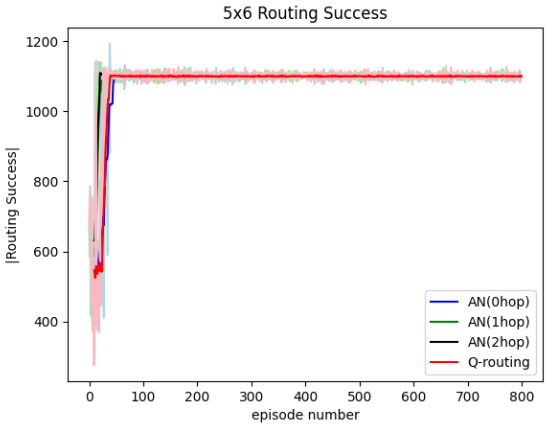
Figure 6.3: Training Results for Downtown Toronto: Average Travel Time (AVTT) and Routing Success during 800 training episodes



(a) Average Travel Time



(b) Average Travel Time (Last 100 Episodes)



(c) Routing Success

Figure 6.4: Training Results 5x6 Network: Average Travel Time (AVTT) and Routing Success during 800 training episodes

the forward pass of neural networks is quick. In this experiment, Q-routing fails to route all the vehicles successfully due to the infinite loops. The reason for the existence of infinite loops in Q-routing is that the algorithm does not have a way to capture the dynamics of the network, hence the estimated Q-values are expected values for all the network states. As a result the algorithm can get confused and generate infinite loops.

In the 5x6 experiment, AN(1hop) outperforms the other algorithms. Note that, counter-intuitively, SPFWR has the worst performance in this case. There can be several reasons for this observation. The network structure and its traffic capacity, e.g. number of lanes per road, number of intersections, number of traffic lights, and length of all the network roads, play an important role in the performance of the SPFWR algorithm. In this case, every road has only one lane, hence, a road can easily get congested. SPFWR greedily sends all the vehicles to the current shortest path and congests it so that it is no longer the shortest path. Moreover, SPFWR does not consider the waiting times in the traffic lights queues. Hence, the more the number of intersections and traffic lights the worse the performance of the SPFWR. On the other hand, all the versions of Adaptive Navigation, and also Q-routing, take into account all the waiting times that contribute to the final travel time.

Current experiments show that adding an extra layer of GAT does not necessarily improve the results. The reason for this observation is that AN(1hop) already hits a performance wall, hence adding another GAT layer only increases the model parameters. As a result, the performance decreases.

Experiment	AN(2hop)	AN(1hop)	AN(0hop)	QR	SPF	SPFWR
D.T. Toronto	479.3	476.4	477.6	∞	551.7	**475.6**
5x6 Network	145.4	**138.4**	143.7	159.6	173.4	205.1

Table 6.5: Testing Results: Average Travel Time in Seconds(AVTT)

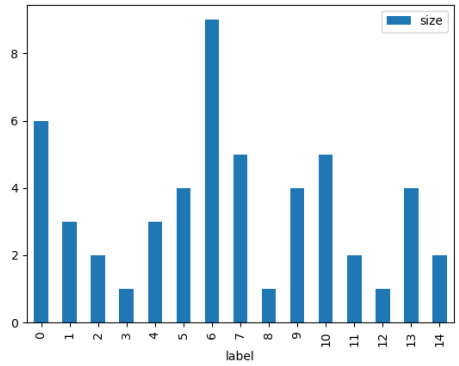
6.3 Locality of Access

As we discussed in section 5.4, the representation for intersection IDs is important in preserving the locality of access information and the effective search of the action space. If a representation preserves the locality of access, a cluster of intersections in the road network should be clustered together in the embedding space. In this section, to evaluate the effectiveness of the proposed intersection id representation in preserving the locality of access we turn to investigate how clusters of intersection in the road network are distributed in the embedding space.

We take the abstracted network of Downtown Toronto and apply a 4x4 grid upon it to create clusters of intersections. Note that, the portions of the map that reside outside of the grid do not contain any agents. We denote all the intersections that fall into a grid cell as a cluster of intersections, and give them the same label. Figure 6.5a shows how the deployed grid creates 16 clusters of intersections in Downtown Toronto network. Figure 6.5b shows the histogram of the number of intersections that reside in each cluster (cardinality of clusters) in figure 6.5b.

To investigate how clusters of intersections in the road network are distributed in the embedding space we need to visualize the intersection ID embeddings. Since the intersection ID embeddings have more than 3 dimension, we have to use diminsionality reduction methods

(a) 16 clusters in Downtown Toronto: 4x4 grid upon Downtown Toronto network



(b) Histogram of the number of intersections in each cluster (cardinality of clusters)

Figure 6.5: Clustering the intersections

to visualize them. Dimensionality reduction enables us to visualize the data in a lower dimension space. We use Principal Component Analysis (PCA), a linear algebra technique for dimensionality reduction with minimum information loss.

Specifically, we want to investigate how the output embeddings of the Z-order-curve method presented in section 5.4 alongside the intersection ID embedding layer of the agents (the linear layer) is distributed in the embedding space. For instance, we take the linear transform layer of the agent residing in the top right corner of grid cell 14 and apply it to all the intersection-id representations. We then pass the output of the linear layer through the ReLU activation function to capture the output embeddings for intersection IDs. We then apply PCA to the output embeddings alongside the label of the intersection. The explained variation per principal component for the first two components of PCA is 0.987 and 0.005 accordingly. The first two components together account for 0.992 of the explained variation.

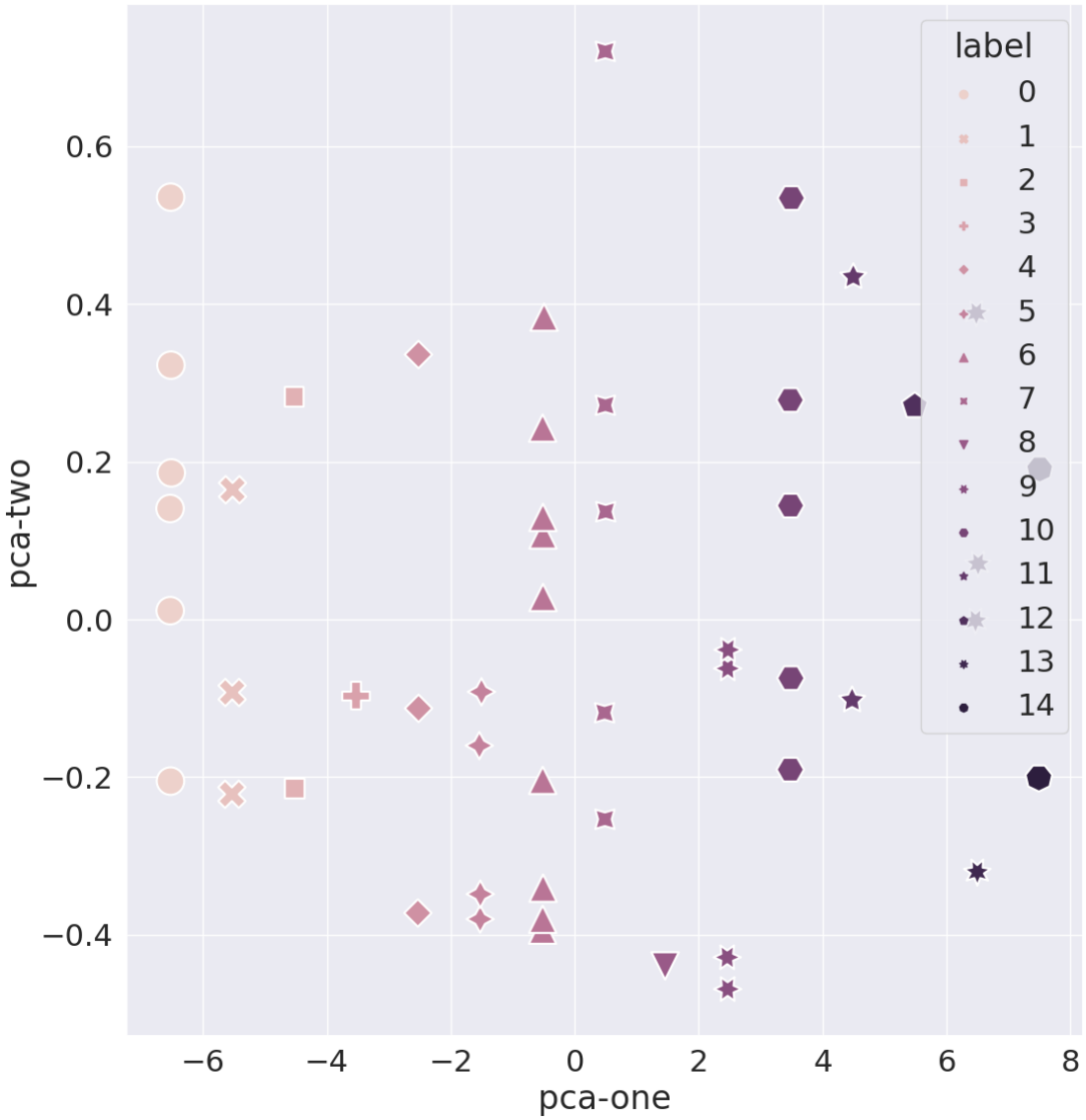


Figure 6.6: Locality of Access Evaluation: first two components of PCA on intersection-id-embeddings

Figure 6.6 shows the first two components of PCA applied to the output embeddings. As the figure 6.6 suggests, the clusters of intersections in the road network are clustered together in the embedding space with a similar first principal component.

Moreover, since the Z-order curve is a lossy algorithm, our method for preserving the locality of access can not be perfect. However, our evaluations show that the Z-order-representation and the intersection ID embedding layer together have well preserved the locality of access information. For example, using this embedding, the router agent at the top right corner of grid cell 14 has an inductive bias toward making similar routing responses for destination intersections that reside in grid cell 0. Similar results achieved on the 5x6 network are not presented to avoid redundancy.

6.4 Attention

In this section we want to evaluate how well the GAT model is capturing the network state. After training the GAT, we want to investigate how the attention weights are distributed for a given network state.

Figure 6.7 shows the input network state to the GAT. Figure 6.8 illustrate the attention scores for some agents. We can see that GAT model has learned non-trivial attention score patterns. For instance, at intersection `gneJ21`, most of the attention is paid to the intersection `gneJ22`. Looking at the network state we can see that the intersection `gneJ22` has no congestion at any of its out going roads. It seems that the GAT model finds such an intersection state more important than the other neighbors of `gneJ21` for making a routing decision. Interesting attention scores for some other intersections are also plotted in 6.8. To

avoid redundancy, similar results achieved on Downtown Toronto network are not presented.

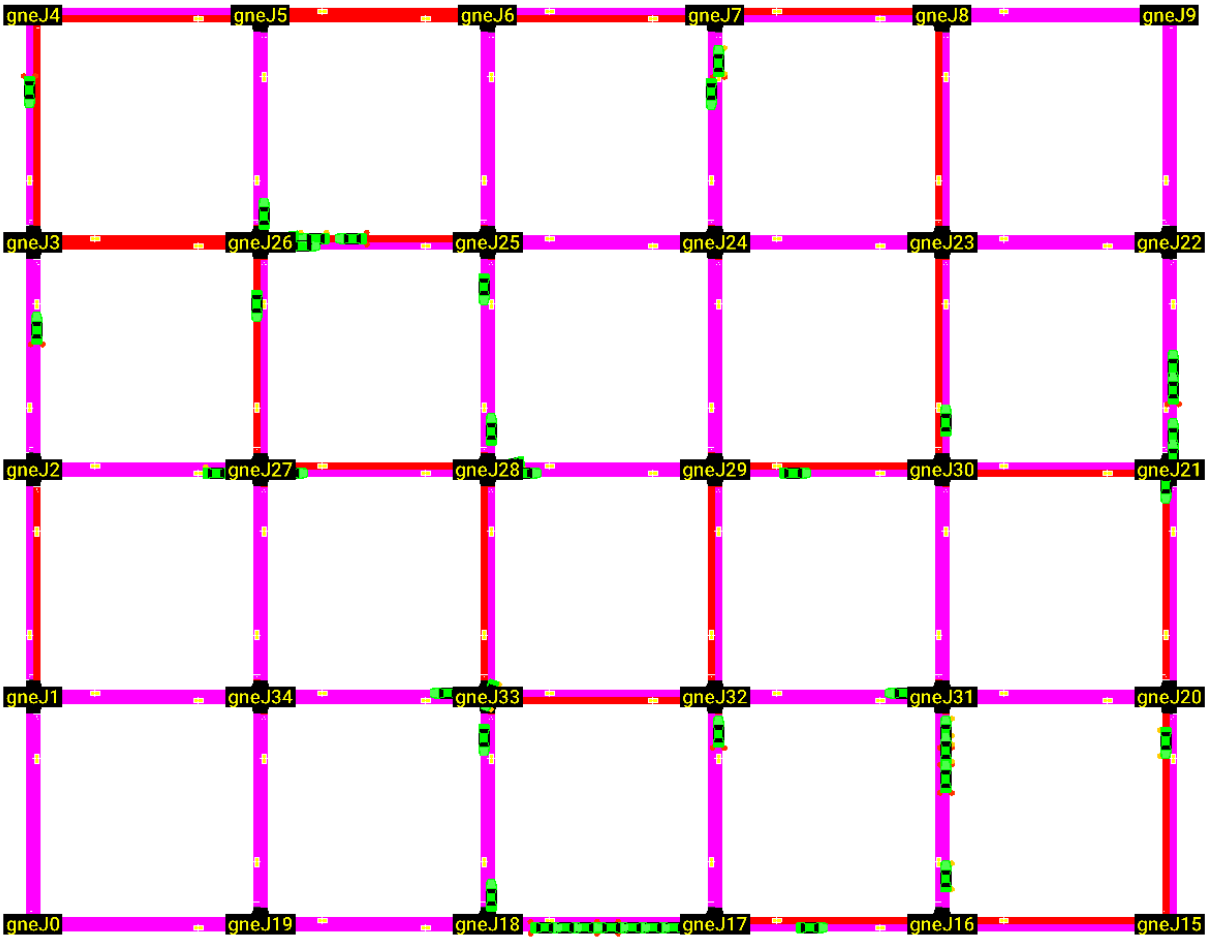


Figure 6.7: GAT Layer-0 evaluation, 5x6 network: red=congested, purple=free

Furthermore, to evaluate how much information the GAT model is capturing from the network state, we use entropy histograms [18]. We can think of the attention weights in the neighborhood of a node as a probability distribution. The null model probability distribution is uniform (equal attention). Figure 6.9 shows entropy histograms for 5x6 and Toronto networks. The orange boxes show the null model distribution and blue boxes show the

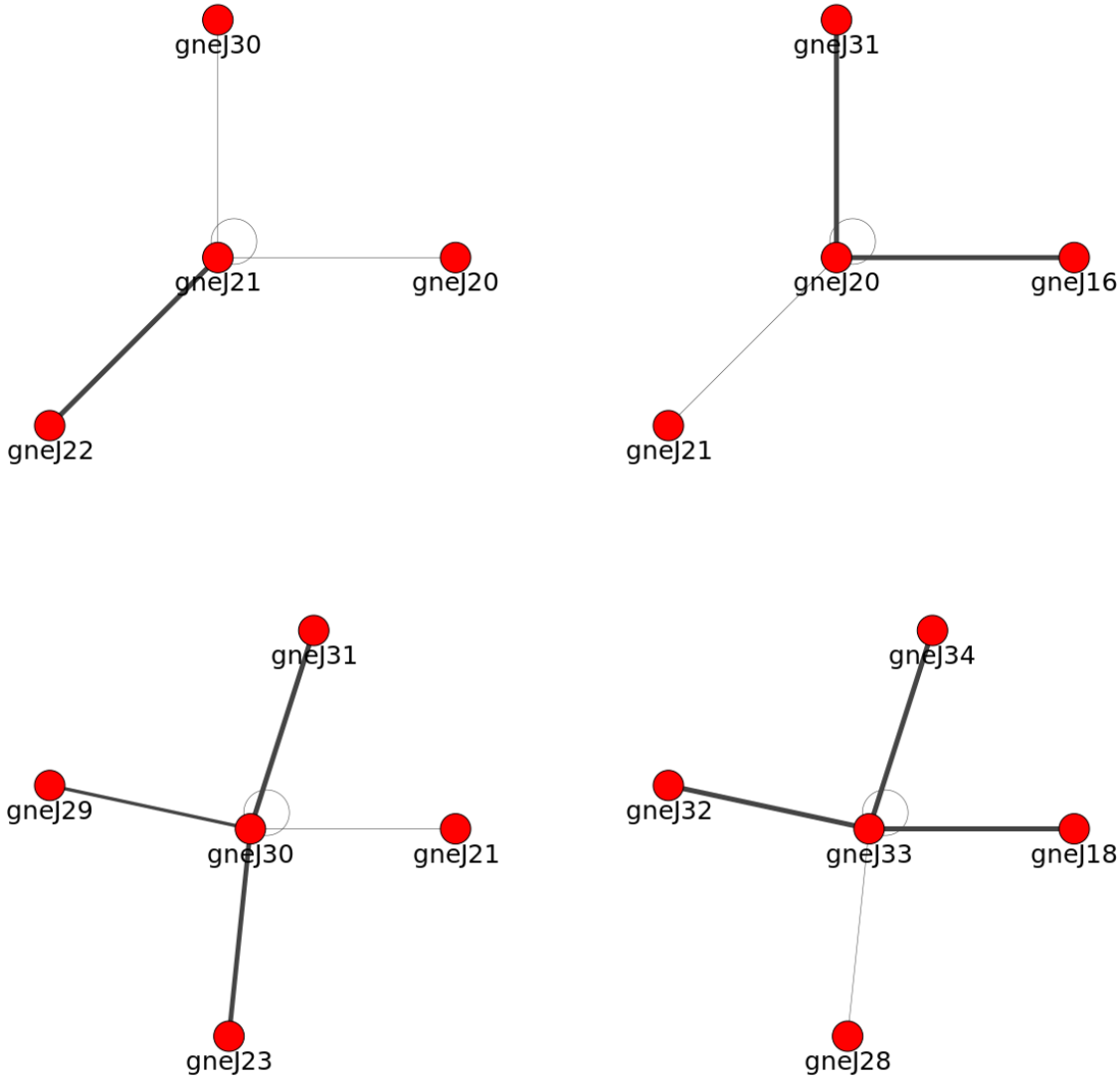
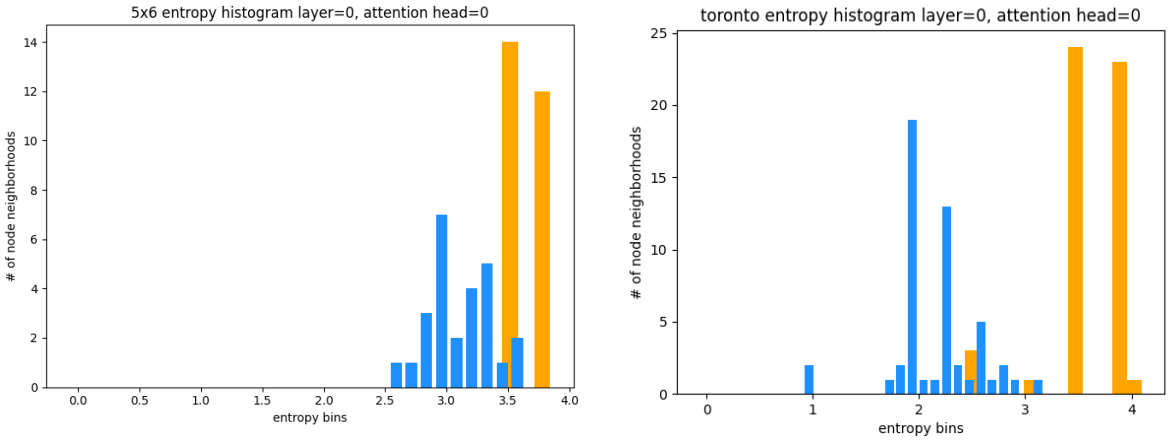


Figure 6.8: GAT Layer-0 evaluation, 5x6 network: attention scores, the score is proportional to the lane width



(a) 5x6 network

(b) Downtown Toronto network

Figure 6.9: Entropy Histograms for Attention Weights

GAT model distribution. As the figure 6.9 suggests, GAT model has well learned non-trivial attention patterns that allows router agents to make informed decisions. We don't present the similar results for the layer-1 attentional head to avoid redundancy.

Chapter 7

Conclusion and Future Works

7.1 Conclusion

Finding the optimal path between a source and a destination in a network is an important problem with many applications, such as vehicle route planning in the road network and packet routing in the IP network. The shortest path first (SPF) algorithm is the widely accepted algorithm for routing, providing an optimal solution in a static network. In a static network, the travel time of the edges remains constant. However, considering networks to be always static is an unrealistic assumption.

In this thesis, we introduced the Adaptive Navigation Problem, the problem of navigating a fleet of vehicles in a dynamic road network with the objective of minimizing the average travel time of the whole fleet.

We used the analogy between road networks and IP networks to propose a multi-agent reinforcement learning (MARL) method for the navigation of the vehicles in the road network. Specifically, we assigned a Q-learning agent to every intersection. When a vehicle approaches an intersection, it generates a routing query. The routing query is forwarded to the agent

assigned to the intersection. The router agent at the intersection is responsible for generating suitable routing responses forwarding its approaching vehicles to a neighboring intersection. The action space size of each agent is the size of the outgoing roads of the intersection they are assigned to. The RL router agent uses the destination ID of the approaching vehicle for predicting Q-values for each action. The reward for each action is set to be equal to the negative of the travel time to the next intersection. With this reward function, the predicted Q-values are an end-to-end estimation of the travel time from the intersection to the given destination. The agent chooses the best-predicted travel time as its routing response.

The locality of access is one important characteristic of road networks. The locality of access means that, on expectation, the travel time to a geographically near destination is shorter than the travel time to a geographically further destination. We proposed a method based on a Z-order-curve space-filling filter for creating intersection IDs that can inform the Q-learning agents of the locality of access information. This method allows the agents to explore the action space more effectively.

Since in a dynamic network, the traffic state of the network changes continuously, the router agents should be aware of the network state to generate informed routing responses. However, the whole network state is both prohibitively large and mainly unrelated to a single agent. The routing response of an agent depends more on the traffic state of its intersection and its neighborhood. Since the importance of different neighbors is different from time to time, we used Graph Attention Networks to adaptively assign different attention scores to each neighbor. Graph Attention Network uses the attention scores to aggregate the network

state of the neighborhood of the agent into a highly dense feature space. The router agent uses the aggregated network state to make informed adaptive routing responses.

We conducted comprehensive empirical evaluations of the Adaptive Navigation algorithm in both synthetic and realistic network structures. The results show substantial improvements of up to 17.3% in average travel time in comparison with the SPF baseline. We used Principal Component Analysis to investigate how the Z-order-curve method, together with the linear layer, are able to inform the agents of the locality of access in the road network. Furthermore, we investigated how the Graph Attention model generates non-trivial attention patterns that enable the agents to make informed routing decisions.

7.2 Limitations

The major limitations of the current work can be discussed as, reliability issues, scalability issues, and network state capturing issues.

7.2.1 Reliability

In general, one major problem with machine learning models is the reliability problem. Specifically, for the task of routing, a reliable solution must avoid generating infinite loops. An infinite loop happens when a vehicle following the routing decisions is stuck in a never-ending cycle in the graph. Although our model has proved reliable in the empirical experiments, there is no analytical guarantee on the reliability of the method.

7.2.2 Scalability

Although the current version of the algorithm, due to its multi-agent nature and the locality of access preservation mechanism, is functional on relatively large network scales, scaling to real-world networks is a potential limitation to this work. Every intersection requires a new agent, and every agent requires GPU-hungry neural networks. Hence, in a realistic network, we soon run out of GPU memory. We further discuss methods to address this limitation in section 7.3.

7.2.3 Network State Capturing

Knowing the accurate traffic state of the network is critical for the optimal performance of the Adaptive Navigation algorithm. In its current version, the Adaptive Navigation algorithm assumes that at every moment, the traffic congestion state in all the roads is known. However, in a realistic scenario capturing such information, although possible, is not straightforward. Furthermore, for simplicity, we assume every road is either congested or not congested. In a realistic scenario, the congestion in a road can have more than two states such as heavily congested, congested, and free. The more accurate the traffic modeling, the better the performance of the Adaptive Navigation algorithm.

7.3 Future Work

As discussed in section 7.2, there are too many intersections in a real-world road network, and it is impractical to train a separate agent for every intersection. Potentially, we can suggest two orthogonal directions for addressing this issue, Shared Policies between agents and

Hierarchical Routing. Moreover, as another orthogonal future work, the Adaptive Navigation algorithm can be coupled with other smart traffic optimizations, e.g., Traffic Signal Control.

7.3.1 Shared Policies

Although there are many intersections in the road network, the policy for neighboring intersections that reside in similar geographical locations is probably similar. One potential direction is to explore other multi-agent reinforcement learning algorithms that allow policy sharing, e.g., Actor-Critic algorithms.

7.3.2 Hierarchical Routing

It is not a reasonable design to assign router agents for every intersection when the scale of the network increases. A reasonable design is to divide the network into hierarchical levels. For example, consider Down Town Toronto network in 6.5a. In the hierarchical level zero, we assign agents to every intersection, and in level one, we assign agents to the grid cells. If a vehicle is driving in grid cell 0 and has a destination in grid cell 5, the grid cell 0 agents decide which of the outgoing roads of the grid cell should be the next destination of the vehicle. Then the routing to the intermediate destination is passed to level 0 routers (intersection routers).

7.3.3 Traffic Signal Control

One of the major causes of congestion in road networks is traffic signals. Currently, the adaptive router agents consider traffic signals as a part of the environment, and they simply adapt to their cycles. Another parallel research question in the literature of smart transportation

systems and traffic optimization is adaptive traffic signal control. As future work, one can consider adaptive routers and adaptive signal controllers to work alongside each other to achieve further optimal utilization of the road network infrastructure.

Bibliography

[1] Open street maps, www.openstreetmap.org/copyright.

[2] Top 5 self driving car companies to watch out in 2021. `https://www.analyticsinsight.net/top-5-self-driving-car-companies-to-watch-out-in-2021`. Accessed: 2021-09-13.

[3] Boyan, J. A., and Littman, M. L. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in neural information processing systems* (1994), pp. 671–678.

[4] Chen, B., Sun, D., Zhou, J., Wong, W., and Ding, Z. A future intelligent traffic system with mixed autonomous vehicles and human-driven vehicles. *Information Sciences 529* (8 2020), 59–72.

[5] Chen, R., Liang, C.-Y., Hong, W.-C., and Gu, D.-X. Forecasting holiday daily tourist flow based on seasonal support vector regression with adaptive genetic algorithm. *Applied Soft Computing 26* (2015), 435–443.

[6] Choi, S. P., and Yeung, D.-Y. Predictive q-routing: A memory-based reinforcement learning approach to adaptive traffic control. In *Advances in Neural Information Processing Systems* (1996), pp. 945–951.

[7] Dantzig, G. B., and Ramser, J. H. The truck dispatching problem. *Management Science 6* (10 1959), 80–91.

[8] Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems 29* (2016), 3844–3852.

[9] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische mathematik 1*, 1 (1959), 269–271.

[10] Dresner, K., and Stone, P. Multiagent traffic management: a reservation-based intersection control mechanism. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004.* (2004), pp. 530–537.

[11] Elshaer, R., and Awad, H. A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. *Computers and Industrial Engineering 140* (2 2020), 106242.

[12] Fajardo, D., Au, T.-C., Waller, S., Stone, P., and Yang, D. Automated intersection control. *Transportation Research Record: Journal of the Transportation Research Board 2259* (12 2011), 223–232.

[13] Fang, S., Zhang, Q., Meng, G., Xiang, S., and Pan, C. Gstnet: Global spatial-temporal network for traffic flow prediction. In *IJCAI* (2019), pp. 2286–2293.

[14] Fu, K., Meng, F., Ye, J., and Wang, Z. Compacteta: A fast inference system for travel time prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (New York, NY, USA, 2020), KDD '20, Association for Computing Machinery, p. 3337–3345.

[15] Geng, X., Li, Y., Wang, L., Zhang, L., Yang, Q., Ye, J., and Liu, Y. Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting. In *Proceedings of the AAAI conference on artificial intelligence* (2019), vol. 33, pp. 3656–3663.

[16] Geng, Y., Liu, E., Wang, R., Liu, Y., Rao, W., Feng, S., Dong, Z., Fu, Z., and Chen, Y. Deep reinforcement learning based dynamic route planning for minimizing travel time. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)* (2021), IEEE, pp. 1–6.

[17] Geyer, F., and Carle, G. Learning and generating distributed routing protocols using graph-based deep learning. In *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks* (2018), pp. 40–45.

[18] Gordić, A. pytorch-gat. `https://github.com/gordicaleksa/pytorch-GAT`, 2020. Accessed: 2021-09-13.

[19] Hart, P., Nilsson, N., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics 4*, 2 (1968), 100–107.

[20] Johansson, U., Boström, H., Löfström, T., and Linusson, H. Regression conformal prediction with random forests. *Machine Learning 97*, 1-2 (2014), 155–176.

[21] Kalakanti, A. K., Verma, S., Paul, T., and Yoshida, T. Rl solver pro: Reinforcement learning for solving vehicle routing problem. In *2019 1st International Conference on Artificial Intelligence and Data Sciences (AiDAS)* (2019), IEEE, pp. 94–99.

[22] Khan, A. R., Jamlos, M. F., Osman, N., Ishak, M. I., Dzaharudin, F., Yeow, Y. K., and Khairi, K. A. Dsrc technology in vehicle-to-vehicle (v2v) and vehicle-to-infrastructure (v2i) iot system for intelligent transportation system (its): A review. *Recent Trends in Mechatronics Towards Industry 4.0* (2022), 97–106.

[23] Kipf, T. N., and Welling, M. Semi-supervised classification with graph convolutional networks. *CoRR abs/1609.02907* (2016).

[24] Koh, S., Zhou, B., Fang, H., Yang, P., Yang, Z., Yang, Q., Guan, L., and Ji, Z. Real-time deep reinforcement learning based vehicle navigation. *Applied Soft Computing Journal 96* (11 2020), 106694.

[25] LeCun, Y., Bengio, Y., et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks 3361*, 10 (1995), 1995.

[26] Li, Y., and Moura, J. M. Forecaster: A graph transformer for forecasting spatial and time-dependent data. *arXiv preprint arXiv:1909.04019* (2019).

[27] Li, Y., and Shahabi, C. A brief overview of machine learning methods for short-term traffic forecasting and future directions. *SIGSPATIAL Special 10*, 1 (2018), 3–9.

[28] Li, Y., Yu, R., Shahabi, C., and Liu, Y. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926* (2017).

[29] Li, Y., Zhu, Z., Kong, D., Xu, M., and Zhao, Y. Learning heterogeneous spatial-temporal representation for bike-sharing demand prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 1004–1011.

[30] Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., and Wiessner, E. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems* (2018), IEEE.

[31] Medsker, L., and Jain, L. C. *Recurrent neural networks: design and applications*. CRC press, 1999.

[32] Miglani, A., and Kumar, N. Deep learning models for traffic flow prediction in autonomous vehicles: A review, solutions, and challenges. *Vehicular Communications 20* (2019), 100184.

[33] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[34] Montanaro, U., Dixit, S., Fallah, S., Dianati, M., Stevens, A., Oxtoby, D., and Mouzakitis, A. Towards connected autonomous driving: review of use-cases. *Vehicle system dynamics 57*, 6 (2019), 779–814.

[35] Morton, G. M. A computer oriented geodetic data base and a new technique in file sequencing.

[36] Moy, J. T. *OSPF: anatomy of an Internet routing protocol.* Addison-Wesley Professional, 1998.

[37] Nazari, M., Oroojlooy, A., Snyder, L. V., and Takáč, M. Reinforcement learning for solving the vehicle routing problem. *arXiv preprint arXiv:1802.04240* (2018).

[38] Pan, Z., Liang, Y., Wang, W., Yu, Y., Zheng, Y., and Zhang, J. Urban traffic prediction from spatio-temporal data using deep meta learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019), pp. 1720–1730.

[39] Panov, A. I., Yakovlev, K. S., and Suvorov, R. Grid path planning with deep reinforcement learning: Preliminary results. *Procedia computer science 123* (2018), 347–353.

[40] Peshkin, L., and Savova, V. Reinforcement learning for adaptive routing. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)* (2002), vol. 2, IEEE, pp. 1825–1830.

[41] Song, C., Lin, Y., Guo, S., and Wan, H. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2020), vol. 34, pp. 914–921.

[42] Sutton, R. S., and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

[43] Tatomir, B., Rothkrantz, L. J., and Suson, A. C. Travel time prediction for dynamic routing using ant based control. In *Proceedings of the 2009 Winter Simulation Conference (WSC)* (2009), IEEE, pp. 1069–1078.

[44] Tedjopurnomo, D. A., Bao, Z., Zheng, B., Choudhury, F., and Qin, A. A survey on modern deep neural network for traffic prediction: Trends, methods and challenges. *IEEE Transactions on Knowledge and Data Engineering* (2020).

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

[46] Watkins, C. J. C. H. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, 1989.

[47] Wei, H., Zheng, G., Gayah, V., and Li, Z. A survey on traffic signal control methods. *arXiv preprint arXiv:1904.08117* (2019).

[48] Wikipedia contributors. Z-order curve. `https://en.wikipedia.org/wiki/Z-order_curve#cite_note-1`. Accessed: 2021-09-13.

[49] Williams, B. M., and Hoel, L. A. Modeling and forecasting vehicular traffic flow as a seasonal arima process: Theoretical basis and empirical results. *Journal of transportation engineering 129*, 6 (2003), 664–672.

[50] Xiao, L., and Lo, H. K. Adaptive vehicle navigation with en route stochastic traffic information. *IEEE Transactions on Intelligent Transportation Systems 15* (10 2014), 1900–1912.

[51] Xiao, S., Mao, H., Wu, B., Liu, W., and Li, F. Neural packet routing. In *Proceedings of the Workshop on Network Meets AI & ML* (2020), pp. 28–34.

[52] Yaghmaie, F. A., and Ljung, L. A crash course on reinforcement learning. *arXiv preprint arXiv:2103.04910* (2021).

[53] Yao, H., Liu, Y., Wei, Y., Tang, X., and Li, Z. Learning from multiple cities: A meta-learning approach for spatial-temporal prediction. In *The World Wide Web Conference* (2019), pp. 2181–2191.

[54] Yin, J., Rao, W., and Zhang, C. Learning shortest paths on large dynamic graphs. In *2021 22nd IEEE International Conference on Mobile Data Management (MDM)* (2021), pp. 201–208.

[55] Yin, X., Wu, G., Wei, J., Shen, Y., Qi, H., and Yin, B. Deep learning on traffic prediction: Methods, analysis and future directions. *IEEE Transactions on Intelligent Transportation Systems* (2021).

[56] Yin, X., Wu, G., Wei, J., Shen, Y., Qi, H., and Yin, B. Multi-stage attention spatial-temporal graph networks for traffic prediction. *Neurocomputing 428* (2021), 42–53.

[57] You, X., Li, X., Xu, Y., Feng, H., Zhao, J., and Yan, H. Toward packet routing with fully distributed multiagent deep reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (8 2020), 1–14.

[58] Yu, B., Yin, H., and Zhu, Z. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875* (2017).

[59] Zhang, K., Yang, Z., and Başar, T. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of Reinforcement Learning and Control* (2021), 321–384.

[60] Zheng, C., Fan, X., Wang, C., and Qi, J. Gman: A graph multi-attention network for traffic prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2020), vol. 34, pp. 1234–1241.

[61] Zivot, E., and Wang, J. Vector autoregressive models for multivariate time series. *Modeling Financial Time Series with S-Plus®* (2006), 385–429.