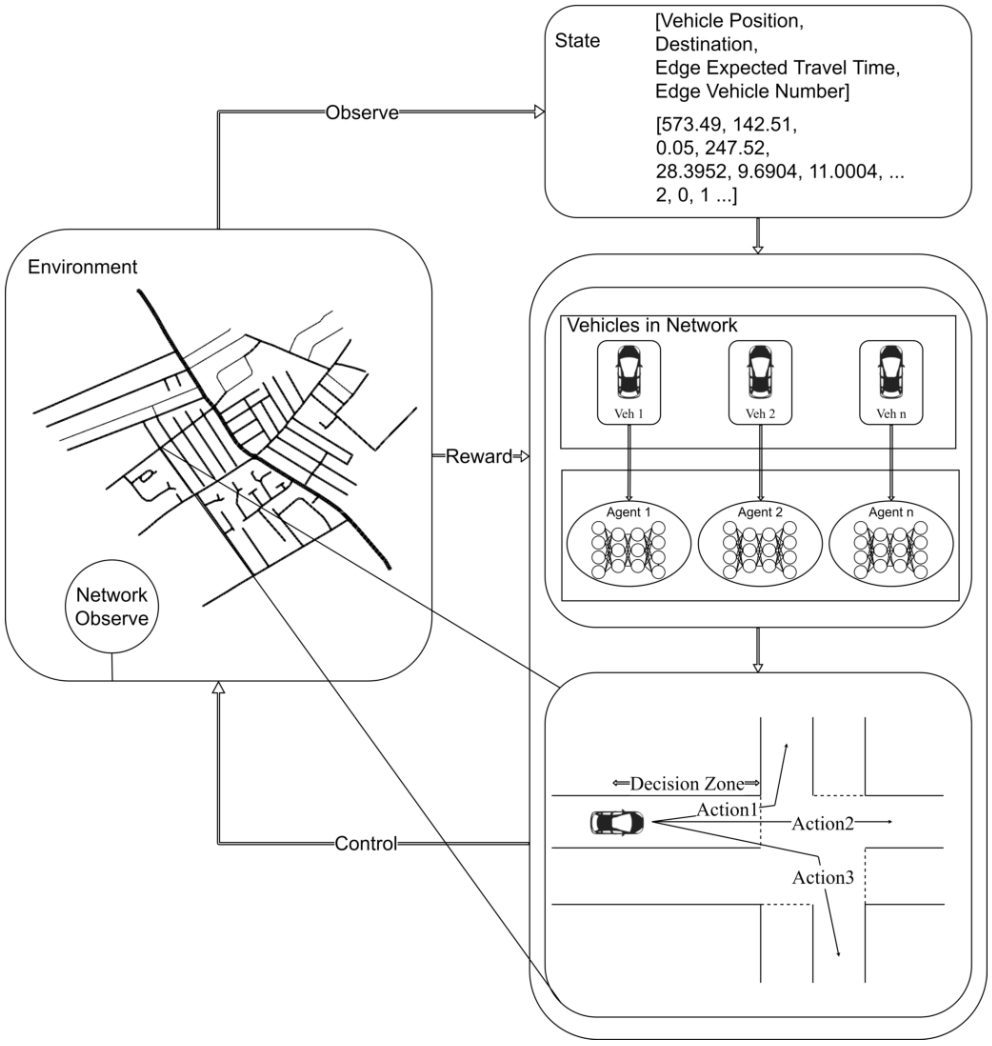
Graphical Abstract

Real-time Deep Reinforcement Learning based Vehicle Routing and Navigation

Songsang Koh, Bo Zhou, Hui Fang, Po Yang, Zaili Yang, Qiang Yang, Lin Guan, Zhigang Ji,



Highlights

Real-time Deep Reinforcement Learning based Vehicle Routing and Navigation

Songsang Koh, Bo Zhou, Hui Fang, Po Yang, Zaili Yang, Qiang Yang, Lin Guan, Zhigang Ji,

- A novel deep reinforcement learning method for real-time vehicle navigation.
- Smart agents are embedded into SUMO simulator in a dynamic transportation system.
- Performance is validated by nine realistic road and traffic combined conditions.

Real-time Deep Reinforcement Learning based Vehicle Routing and Navigation

Songsang Koh[a], Bo Zhou[a], Hui Fang[b,*], Po Yang[d], Zaili Yang[a], Qiang Yang[c], Lin Guan[b], Zhigang Ji[e,*],

[a]*School of Computer Science and Mathematics, Liverpool John Moores University, Liverpool, United Kingdom L3 3AF*
[b]*Department of Computer Science, Loughborough University, Loughborough, United Kindoms LE11 3TU*
[c]*College of Electrical Engineering, Zhejiang University Hangzhou, PRC 310027*
[d]*Department of Computer Science, Sheffield Unviversity, Sheffiled, United Kingdom S10 2TN*
[e]*National Key Laboratory of Science and Technology on Micro/Nano Fabrication, Shanghai Jiaotong University, Shanghai, PRC 200240*

Abstract

Traffic congestion has become one of the most serious contemporary city issues as it leads to unnecessary high energy consumption, air pollution and extra traveling time. During the past decade, many optimization algorithms have been designed to achieve the optimal usage of existing roadway capacity in cities to leverage the problem. However, it is still a challenging task for the vehicles to interact with the complex city environment in a real time manner. In this paper, we propose a deep reinforcement learning (DRL) method to build a real-time intelligent vehicle routing and navigation system by formulating the task as a sequence of decisions. In addition, an integrated framework is provided to facilitate the intelligent vehicle navigation research by embedding smart agents into the SUMO simulator. Nine realistic traffic scenarios are simulated to test the proposed navigation method. The experimental results have demonstrated the efficient convergence of the vehicle navigation agents and their effectiveness to make optimal decisions under the volatile traffic conditions. The results also show that the proposed method provides a better navigation solution comparing to the benchmark routing optimization algorithms. The performance has been further validated by using the Wilcoxon test. It is found that the achieved improvement of our proposed method becomes more significant under the maps with more

edges (roads) and more complicated traffics comparing to the state-of-the-art navigation methods.

Keywords: Routing and navigation optimization; Deep reinforcement learning; Deep-Q learning; SUMO; Intelligent vehicle.

1. Introduction

In recent years, traffic congestion in urban area has become a serious problem due to the rapid development of urbanization. It brings a major impact on urban transportation networks that lead to extra traveling hours, increased fuel consumption and air pollution. According to the study in McGroarty (2010), traffic congestion can be categorized into recurring congestion (RC) and non-recurring congestion (NRC). NRC is defined as the congestion made by unexpected events, such as construction work, inclement weather, accidents, and special events Hall (1993). Unsurprisingly, NRC accounts for a larger proportion of traffic delays in urban areas comparing to the RC due to its unpredictable nature Sun et al. (2017). There are three categories of methods proposed to tackle the NRC problem: (1) detecting and predicting traffic congestions by utilizing both the historical and real-time sensor data Zygouras et al. (2015); Ghafouri et al. (2017); (2) optimizing traffic signal control and management Wen (2008); Mousavi et al. (2017a); and (3) vehicle routing and navigation optimization Ritzinger et al. (2016); Jabbarpour et al. (2018); Okulewicz and Mańdziuk (2017); Abdulkader et al. (2015). Wherein, the vehicle routing and navigation, as the most promising solution, has been investigated extensively during the past decades.

Classical vehicle routing problem (VRP) is defined as finding the minimum cost of the combined routes of a given number of vehicles m to serve n customers. One of the typical examples is path planning for collecting and sending packages from a delivery company. Traditional VRP is formed and classified as an NP-hard problem Jabbarpour et al. (2018). Many optimization algorithms are proposed to find the sub-optimal solution under different

*Corresponding author

Email addresses: S.S.Koh@2014.ljmu.ac.uk (Songsang Koh), b.zhou@ljmu.ac.uk (Bo Zhou), H.Fang@lboro.ac.uk (Hui Fang), po.yang@sheffield.ac.uk (Po Yang), z.yang@ljmu.ac.uk (Zaili Yang), qyang@zju.edu.cn (Qiang Yang), L.Guan@lboro.ac.uk (Lin Guan), zhigangji@sjtu.edu.cn (Zhigang Ji)

constraints, e.g. genetic algorithm Ruiz et al. (2019), firefly algorithm Altabeeb et al. (2019), hybrid algorithm Hosseinabadi et al. (2019) or backbone algorithm Bertsimas et al. (2019). However, this type of routing problem definition assumes a relatively stable traffic condition, and target on the travel salesman problem. Thus, these proposed optimization algorithms search for optimal solution under a static context. Different from the classical VRP, this study targets the vehicle routing and navigation problem as controlling the vehicle to make path planning from a given start node to a destination node with shortest travel time under dynamic traffic conditions Guo et al. (2019). The objective is to find a path or a set of optimal actions to minimize travel time with real-time collected input of current traffic conditions. In the past few years, many heuristic and evolutionary optimization algorithms are proposed to solve the problem Gawron (1998); Lanning et al. (2014); Nahar and Hashim (2011); Zong et al. (2010); Kaparias and Bell (2010). Although recent research on vehicle routing and navigation has achieved reasonable results, the state-of-the-art methods suffer from several drawbacks: firstly, the shortest path type of methods, e.g., Lanning et al. (2014); Kaparias and Bell (2010); Guo et al. (2019), becomes less efficient to provide optimal solutions due to unpredictable nature of more complicated traffic conditions and cannot react instantly to NRC issues. Secondly, it is hard to solve the problem in a real-time manner when using optimization algorithms, e.g., Samaras et al. (2019); Zong et al. (2010); Anjum et al. (2019). The searching space grows exponentially when more routing edges are added into the map. Thirdly, optimization based vehicle routing and navigation algorithms, such as Dean et al. (2019); Nahar and Hashim (2011); Boesen (2017); Kponyo et al. (2012), cannot perform self-evolution and self-adaptation. To address the limitations of the methods, this paper proposes a deep reinforcement learning (DRL) method to achieve real-time intelligent vehicle navigation to alleviate the NRC issues.

By formulating vehicle routing and navigation task as a sequence of decisions, the DRL framework can be utilized to solve this automated control problem through taking real-time observations and evaluating the outcomes from actions under a complex environment. Inspired by recent success of deep reinforcement learning methods, the enhanced deep-Q network (DQN) method is adopted to deal with this real-world complexity given the fact that navigation task can be modeled as a Markov Decision Process (MDP). Comparing to the DQN in Mnih et al. (2015), our method enhances it in three aspects: (1) a suitable reward function is designed for the vehicle routing and

navigation task; (2) several advanced schemes, including double DQN, dueling DQN and priority experience replay, are integrated into the framework to achieve a more reliable convergence of the network; and (3) a distance ranking sampling strategy is proposed to speed up the convergence speed.

In our work, a traffic simulator, SUMO, is seamlessly connected to smart navigation agents to provide an integrated experimental framework. It could simulate real-world traffic conditions as well as embed agent decisions into the traffic simulator. Once agents are required to make navigation decisions, observations generated from simulated traffic environment are fed into these agents as current traffic state representations. The agents could, therefore, make automated real-time navigation decisions that minimize the travel time to reach their destination. The proposed DRL framework provides an appealing and innovative solution for vehicle routing and navigation problem as the learning process is fully automated which does not require any labeling or guidance. Furthermore, the agents can automatically adapt their policy networks by analyzing global environment as their context and eventually decrease travel time when new data is generated.

The main contributions of our research can be highlighted as follows: (1) this work proposes a novel DRL algorithm to achieve an effective real-time vehicle routing and navigation system; (2) the DRL agents are embedded into traffic simulator SUMO to achieve an integrated framework to facilitate the intelligent vehicle navigation research under the context of a dynamic urban transportation system; and (3) the potential practical usage is demonstrated in nine realistic road and traffic combined conditions. Further, its efficiency is validated by using the Wilcoxon test.

The remainder of the paper is organized as follows: The background of our research is described in Section 2 to clarify the motivation of our work. Section 3 provides an overview of the proposed framework and introduces the main components in our system. Section 4 and Section 5 explain the detail of two main components, i.e. the traffic simulator and DRL smart agents. In Section 6, the experimental results are presented to demonstrate the convergence of the DRL agents, how the agents navigate vehicles to make routing choice and their performance compared to the SUMO build-in route optimization algorithms. Finally, Section 7 presents the conclusive remarks and suggests future work.

2. Background

The original idea for solving the problem of traffic congestion was via traffic control and optimization in which a significant number of research works had conducted. In particular, many of the works focus on path planning and directing vehicles to their destination as soon as possible with considerations of static conditions, e.g., travel distance and speed limit.

The very first solution in early years for vehicle navigation problems was the shortest path algorithm, which aims to find a path between two nodes with minimum traveling distance. Dijkstra proposed a static algorithm to find the shortest path without considering any external factors such as congestion, accident, or average vehicle speed Dijkstra (1959). Therefore, this solution is no longer considered applicable to today's traffic congestion problem given the fast development of modern transportation networks and the constraints it brought into. Although in general common GPS applications, e.g. Google map or Waze still rely on shortest path algorithm Lanning et al. (2014), they also allow drivers to access certain real-time traffic information. e.g. accidents, construction, road blocking that may be biased and inaccurate due to the information rely heavily on human input.

Another direction for vehicle navigation path planning is to use the Ant-colony algorithm, which was first proposed by DorigoDorigo et al. (1996). It was inspired by the natural behavior performed by ants in finding food resources. Previous experiments have proven that ants can find the shortest route between two individual sections by leaving pheromone trails to allow other ants to track the path. Nahar and Hashim Nahar and Hashim (2011) proposed a traffic congestion control method based on different preferences to create an optimal traffic system and reduce the average traveling time by adjusting the ant colony variables. The experiment showed that the number of ants is directly correlated with the algorithm performance. Therefore, this method does not perform well when there are only a limited number of agents in the network. A multi-agent evacuation model was introduced by Zong et al. Zong et al. (2010) to minimize the total evacuation time for vehicles and balance the traffic load. Ants belong to one colony find their routes with the same properties. During the process of searching for routes, the two ant colonies will interact and share information. Experiments have shown that the Multi-Ant colony system is more effective than a single agent system. Kponyo et al. Kponyo et al. (2012) also proposed a distributed intelligent traffic system that uses vehicle average speed as a parameter to determine

the traffic condition. This system guides vehicles to paths with less traffics. Therefore, this system selects the best path more efficiently comparing with selecting the path in a random fashion. More recently, some other additional objectives such as work load balancing (in terms of time and distance) or minimize vehicles emission are considered in research. Galindres-Guancha et al. proposed a multi-objective problem of multi-depot vehicle routing (MOMD-VRP) with the aim to minimize the total distance travelled and the balance of routes. They developed a three-stage solution approach using constructive heuristic, iterated local search multi-objective meta-heuristics (ILSMO) and concepts of dominance Galindres-Guancha et al. (2018). Weiheng Zhang et al. presented a Multi-Depot Green Vehicle Routing Problem (MDGVRP) that applies a Two-stage Ant Colony System (TSACS)Zhang et al. (2020) to find a feasible and acceptable solution to minimize the total carbon emissions. Although these Ant-colony methods have achieved promising results, they did not perform well when it comes to a more realistic, complex and dynamic transportation system and cannot deal with unexpected events instantly.

With the rapid development and recent success of machine learning technologies lately, many researchers started to focus on solving the traffic congestion problem based on the deep learning based methods. Karlaftis et al. conducted an overview comparing statistical methods with neural networks in transportation-related research and it demonstrated that solutions based on deep reinforcement learning are very promising Karlaftis and Vlahogianni (2011). Most research projects in this area apply deep learning for traffic prediction Lv et al. (2015); Polson and Sokolov (2016) or accident prediction Ren et al. (2018); Sun et al. (2017) to detect traffic congestions in advance. For traffic prediction, Lv et al. proposed a deep learning-based traffic flow prediction method by using a stacked auto-encoder model to learn generic traffic flow features Lv et al. (2015). Polson et al. presented a deep learning predictor for spatial-temporal relations presented in traffic speed measurements. It focused on forecasting traffic flows that occur unexpectedly and hardly predictable such as special events or extreme weather Polson and Sokolov (2016). Both approaches aforementioned provide a relatively accurate traffic prediction and allow the user to foresee the potential upcoming traffic congestion. Nevertheless, these predictions lack of providing a decision-making method for users to plan their route to avoid traffic congestion. Ren et al. Ren et al. (2018) collected traffic accident data and analyzed the spatial and temporal patterns of the traffic accident frequency for the accident risk pre-

dictor. Sun et al. Sun et al. (2017) proposed a deep neural network DxNAT to identify non-recurring traffic congestion by converting traffic data in Traffic Message Channel (TMC) format to an image, and use a convolutional neural network (CNN) to identify non-recurring traffic anomalies. Similarly, although the papers above can identify the non-recurring traffic congestion, a decision-making method that aims to help the user to avoid those traffic congestions is missing. Furthermore, several works attempted to handle the traffic at the intersections such as controlling the traffic light signal Van der Pol and Oliehoek (2016); Mousavi et al. (2017b); Genders and Razavi (2016) and navigating vehicles at occluded intersections Isele et al., 2017, 2018). Although these approaches demonstrated promising result for reducing traffic congestion in certain ways, a real traffic network involves heavily in route planning and navigation, which are not covered by the study above.

Traffic optimization can potentially be more efficient if it combines with an intelligent vehicle navigation system in a complex traffic network via DRL methods. After identifying the challenges of traffic optimization and discussing the strengthes and weaknesses of the related works, our proposed framework that uses DRL method for intelligent vehicle navigation is presented in Section 3.

3. The Framework

Our framework encapsulates the use of SUMO (Simulation of Urban Mobility) with Traffic Control Interface (TraCI) that allows us to connect with the DRL agent. In this study, an improved Deep Q-Learning Network (DQN) methodMnih et al. (2015) is adopted to train intelligent agents to navigate the vehicles to their destinations and avoid congestions. As shown in Figure 1, the designed framework consists of three parts: The first part is the SUMO, which is the environment simulator for creating realistic traffic scenarios; the second part is the middleware that connects the SUMO environment with the DRL agents; and finally, the third part is the DRL agents which are capable of maintaining and updating the navigation policies and providing commands for the navigation simulation.

Our training framework coordinates the environment simulator SUMO with the observations, actions, and rewards needed and produced by the DRL agent. After the training is being initialized, The simulation in SUMO can be loaded, progressed and reset with required information such as the

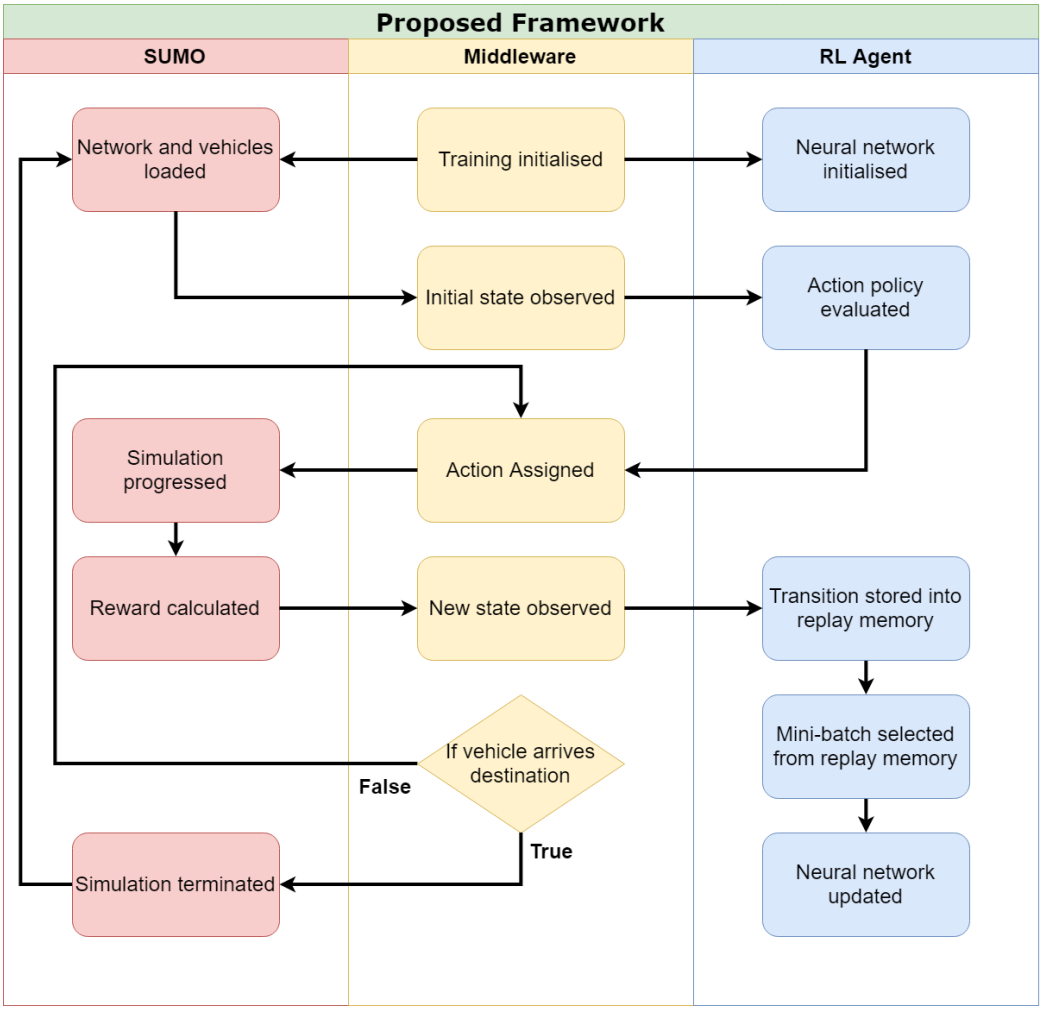


Figure 1: The Framework consists of SUMO simulator, Middleware and DRL Agent for the vehicle navigation task

transportation network and vehicles via TraCI. More details about the simulator and TraCI can be found in section 4.

Our objective is to train the policy network that navigates a vehicle to find the best route to its destination and avoid congestions. To justify the feasibility of the proposed training framework, several experiments with different transportation maps are carried out and the policy network is trained with different levels of traffics for performance comparison. In every training step, the training framework obtains the environmental observations in

SUMO and sends them to the DRL agent via the middleware. Based on the observations, the DRL agent evaluates the current traffic environment and assigns an action based on the policy neural network. The framework then sets the action accordingly to update the state and move to the next step in SUMO until the simulation is completed. The reward is calculated and passed to the DRL agent for optimization at the end of each simulation run.

4. Traffic Simulator

Simulation is considered an efficient approach in investigating different scientific problems. Facilitating the increasing processing power possessed by computers, simulation allows testing the complex scientific models in a reasonable time with minimum cost. Traffic simulator is especially widely used in transportation research as running experiments with vehicles in the real world is simply not practical Kotusevski and Hawick (2009). There are several widely used traffic simulators, including Quadstone Paramics Cameron and Duncan (1996), VISSIM Fellendorf (1994), AIMSUN Casas et al. (2010), MATSIM Axhausen et al. (2016) and SUMO Krajzewicz et al. (2012). These simulators provide different features and models for commercial and research purposes. Kotusevski et al. carried out a comprehensive comparison of these simulators with their features, characteristics and limitations Kotusevski and Hawick (2009). Among them, SUMO comes with an outstanding ability to simulate a very large and complex transportation network of up to 10,000 edges (roads).

As SUMO is an open-source, microscopic, multi-model traffic and extensible simulator, it has been widely used in research projects with worldwide community support. It allows the users to simulate specific traffic scenarios performing in given road maps. In our experiments, SUMO is used as the traffic simulator because: (i) it performs an optimized traffic distribution method based on vehicle types or driver behaviors to maximize the capacity of the urban transportation network; (ii) it provides flexibility and scalability to create the scenario maps; and (iii) it supports TraCI, a Python-based API to communicate the traffic simulation with the controls from the smart agents.

A SUMO map can be either generated manually with simple XML-data containing nodes and edges, or from third-party sources such as Open Street Map. To import external maps, SUMO provides an additional tool called "netconvert" to convert the map from other formats into a compatible SUMO

version for traffic simulation. There are two main sources for the map generation. Firstly, it could read maps from other traffic simulators such as VISUM, Vissim, or MATsim, and compute the needed input for SUMO to generate its maps in the compatible XML format. Secondly, it could also import common maps such as those from Open Street Map. Open Street Map is a valuable source for real-world map data which is free to be viewed and enhanced. Figure 2 shows an example of converting an Open Street Map to the SUMO map.

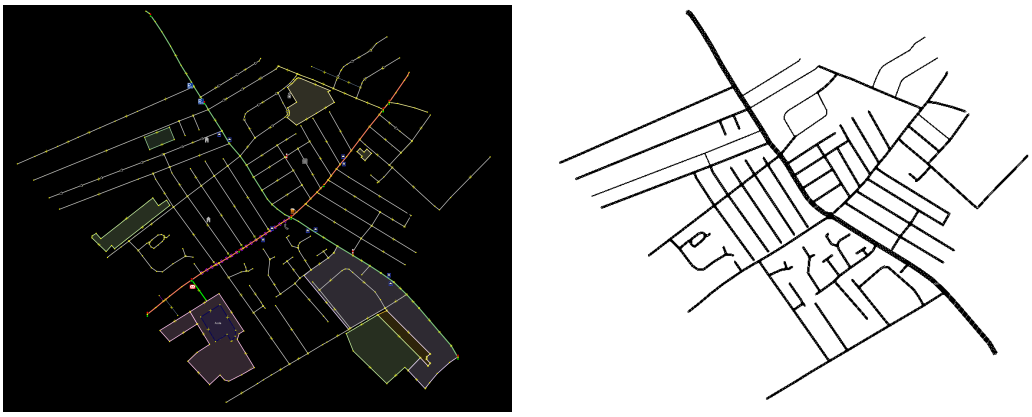


Figure 2: (a) openstreetmap (b) sumo map

Another distinctive feature of SUMO is TraCI. It is a Python-based API that treats the SUMO simulator as a server. The users gain real-time information from a running traffic simulation and update the simulation correspondingly. TraCI enables third party systems (or libraries) to integrate with the SUMO traffic simulation at runtime. In our training, TraCI plays the role of a communicator to link the SUMO and the DRL agent. It retrieves the information of vehicles on the road map in the simulation and provides useful features for the DRL agent to respond to the states of the environment.

5. Deep Reinforcement Learning for smart vehicle navigation

5.1. Problem statement

In our work, the graph theory is used to represent the traffic network in the SUMO simulator. The traffic network is represented as G ={N,E}, where N represents junctions and E represents roads in the map. Each intersection

in the network between roads is a node in our graph and an edge is defined if there is a road segment that connects the two corresponding intersections. As shown in Figure 3, the left sub-figure is a normal urban road traffic network that runs in SUMO simulator and the right sub-figure is its graph representation.

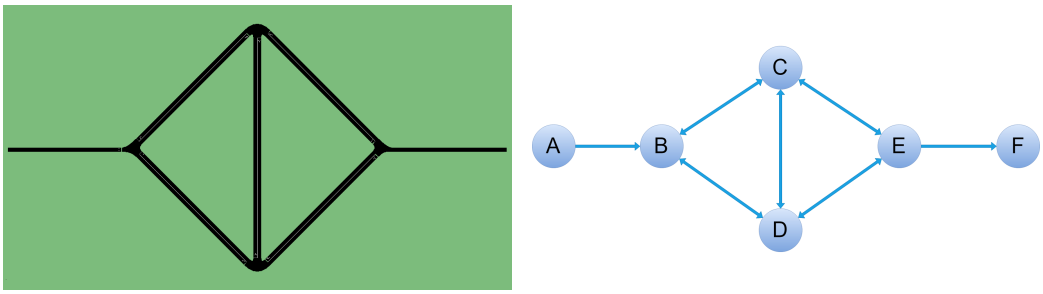


Figure 3: Sumo network and node map

Based on the graph representation, the problem statement of vehicle navigation is further defined as follows. As illustrated in Figure 4, assume that each vehicle has an acting agent defined as $v \in \{v_1, v_2, ..., v_n\}$ to navigate the vehicle to reach its destination. Once the vehicle approaches a junction node N, an observed state s_t is derived from the current traffic environment to form the state space S ($s_t \in S$). The state s_t is fed into the agent v_i as the representation of current traffic observation. Based on the s_t, the agent v_i requires to select a decision from an action space A, where $a = \{a_1, a_2, ..., a_m\} \in A$. Also, a decision zone that has a fixed distance towards each junction to make sure that the acting agent manages to change lane in our simulator to reach all possible actions in the action space is defined. After taking an action based on current state s_t, the agent receives a reward $r_t(s_t, a_t)$ from the traffic environment. The goal of each reinforcement learning agent is to drive its vehicle to reach its destination as quickly as possible to achieve a higher reward.

5.2. Key elements of DRL vehicle navigation

There are four key elements in the DRL system, named as vehicle agent, observation/state, action and reward scheme. The vehicle agent takes observation from the traffic environment as input and provides a recommended action as its output to maximize the final reward defined by reducing the traveling time to its destination. Their details are explained as follows:

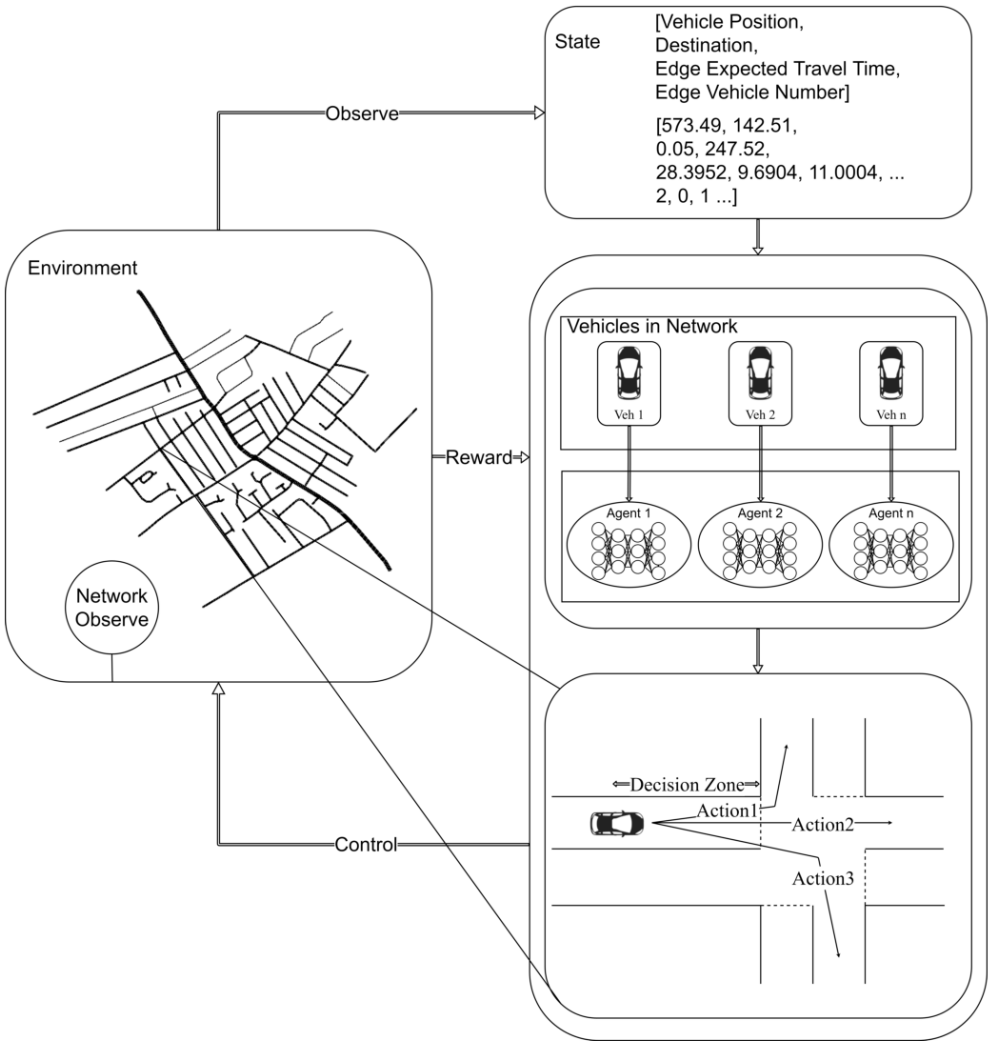


Figure 4: Problem statement of the DRL based multi-agents navigation

Vehicle Agent is a self-evolved neural network (NN) model, that takes traffic observations as its input and produces action decisions as to its output. The archietecture of the NN is a multi-layer perceptron network (MLP) with five layers (its details are explained in Section 6). At its early training stage, the agent has a high exploration rate to take more random actions to explore different routes to reach its destination. During the training, a so-called exploration rate decay (ERD) is set to make the agent learn a more

deterministic policy $\pi : S \rightarrow A$ so that the expectation of traveling time under similar travel conditions can be reduced significantly.

Action refers to the navigation decision made by a vehicle agent. The actions are discrete values corresponding to the decisions of navigating vehicle to m connected edges from the current edge. For example, the actions include left-turn, right-turn or go-straight as illustrated in Figure 4 where $m = 3$. In our experiments with realistic city maps, m is set to the maximum number of connected edges on the map.

State is an efficient representation of current traffic condition. The representation variables contain multiple parameters reflecting the circumstances in the global urban transportation network to precisely describe the complexity of its dynamics. Here, the state is defined as a vector with $[m_e, n_e, l_e, x_v, y_v]$ where the m_e is the numbers of vehicles in the edges, n_e is the average driving speeds in the edges, l_e is the road lengths of the edges, and the x_v and y_v represent the location of the agent and its destination.

Figure 5 illustrates an example of the state representation in a sample network and how the traffic conditions are observed and extracted. From the example, there are 8 roads in the network as $E \in \{AC, CA, BC, CB, CD, DC, CE, EC\}$. Road DC has three vehicles as $n_{DC} = 3$; road CA, EC have two vehicles as $n_{CA}, n_{EC} = 2$; road AC, BC, CD each has one vehicle as $n_{AC}, n_{BC}, n_{CD} = 1$ and road BC has none as $n_{BC} = 0$. Meanwhile, to reduce the dimensionality of the state space, feature construction is applied for the calculation of expected traveling time on the road, which is obtained from several features in the network. The calculation is shown by following:

$$t_e = \left\{ \begin{array}{ll} \frac{l_e}{v_e}, & \text{if } n_e > 0 \\ \frac{l_e}{m_e}, & \text{if } n_e = 0 \end{array} \right\}$$

where l_e is the length of the road, v_e is the average driving speed on the road, m_e is the speed limit on the road and n_e is the number of vehicles on the road. In the example, vehicle position x_v is the coordination of middle point of road AC where the vehicle agent (the red vehicle on the left) is at the current time stamp. While the destination y_v is the coordination of the road CE. As the vehicle agent only requires the latest state to make decision, a segment is compartmentalised in each edge named Decision Zone to indicate the best timing for obtaining a state. The decision zone d_v is expressed as:

$$d_v = min[l_{edge}, v_{max} + v_{max} b(v) \tau_v]$$

where l_{edge} is the length of the edge, v_{max} is the maximum speed of individual vehicle v, b_v is the deceleration function of vehicle v and τ is the driver's reaction time. The decision zone is determined by both vehicle type and driver's reaction time due to the safety consideration of the urban transportation network.

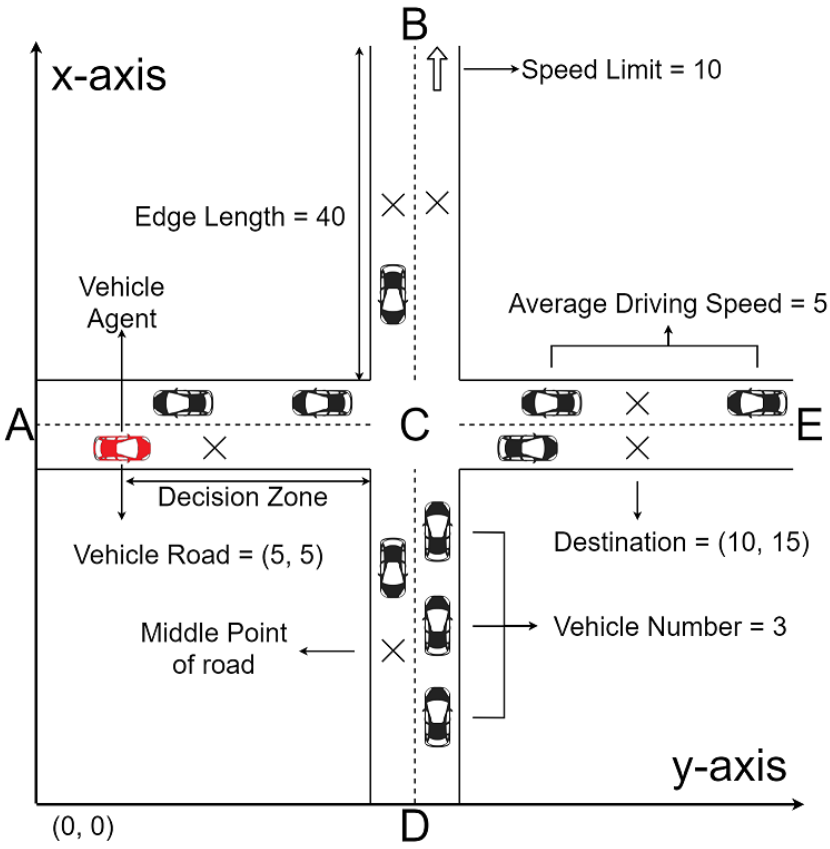


Figure 5: Observation that in a simple junction

Reward is the most important factor in the DRL system as it guides each agent to converge to an optimal policy π_θ by encouraging good actions taken from the function approximations. The overall principle of the reward setting in our work is to maximize the expected future discounted returns. The expected reward is described as:

$$R = \sum_{k=0}^{T} \gamma^k r_{s+k}$$

In the proposed system, the reward function r_s is set in an instant reward manner that can be formulated as $r_{s_t} = -(T_{s_{t+1}} - T_{s_t})$ where $T_{s_{t+1}}$ is the total traveling time to the state S_{t+1} and T_{s_t} is the total traveling time to the state S_t. The traditional way to set the reward of each action in a sequential decision making process suffers from the long delay issue. With the proposed reward scheme, the convergence performance of our DRL method is significantly better comparing to a discounted reward scheme.

5.3. DRL method for Realtime Intelligent Vehicle Navigation

In our work, an improved DQN architecture is designed for real-time intelligent vehicle navigation. DQN is an online training method to maximize an action-value function $Q(s, a)$, defined in Eqn.1, that is an estimation of expected cumulated return from a sequential decision-making process. In this method, multi-layer neural networks are utilised as function approximators that map from a state to Q-values $Q(s, a) \approx Q(s, a|\theta)$.

$$Q(s, a) = E[R_t|s_t = s, a_t = a] \tag{1}$$

According to the Bellman equation, if the Q values for all actions in the next state s_{t+1} are known in $Q^\pi(s_{t+1}, a_{t+1})$, the Q-value in current state is the summation of the immediate reward r_t and the maximum cumulated reward in the next step. Therefore, the target maximum expected reward for current stage could be set as $r_t + \gamma max Q^\pi(s_{t+1}, a)$, where $0 < \gamma \leq 1$ is a discount factor. By updating the Q value iteratively, the expected return is defined in Eqn.2:

$$Q_{t+1}(s, a) = Q_t(s_t, a_t) + \alpha(r_t + \gamma max Q_t(s_{t+1}, a) - Q_t(s_t, at)) \tag{2}$$

The parameter θ is trained by minimizing the error between the expected cumulated return and Q-value that predicted by the agent. Same as the work in Mnih et al. (2015), two neural networks, including a target network and an online trained network, are adopted in our work. The target network is used to estimate the Q values and being updated after a certain amount

of epochs. The loss of individual experience to train the online network is defined in Eqn.3:

$$L(\theta) = (r_t + \gamma maxQ(s_{t+1}, a|\theta^-) - Q(s_t, a|\theta))^2 \tag{3}$$

When sampling the experiences (s_t, a_t, r_t, s_{t+1}) from replay memory, prioritised experience replay algorithm Schaul et al. (2015) is used in our work to update the DQN network θ. In this method, the probability p_t defined in Eqn.4 is calculated to increase the possibility of sampling experiences that are new in the memory for faster convergence.

$$p_t = \frac{1}{rank(i)} \tag{4}$$

Here rank(i) is the rank of transition i when the replay memory is sorted according to new or old degree.

The techniques used in Double DQNs Van Hasselt et al. (2016) and Dueling DQN Wang et al. (2015) are also implemented in our DRL networks. The double DQNs method is integrated into our method for solving the Q value overestimation problem and the dueling DQN is for achieving better convergence when presenting many similar-valued actions.

In the proposed method, a two-stage exploration scheme is designed to improve the network convergence as well as the converging speed. In the first stage, the conventional ϵ-greedy policy is used to control the ratio between exploration and decisions made by the current neural network. In the second stage of the scheme, a distance-based method is used to replace the random selection for edge exploration. As illustrated in Figure 6, when the agent vehicle is in the decision zone (the vehicle in red colour), the edges with blue colour are those edges that link to the agent vehicle's current edge, known as possible actions of the agent vehicle in this particular case. The Euclidean distances between the end of blue edges and the end of destination edge can be calculated. After keeping all the Euclidean distances of each edge to the destination of the vehicle, where $\{d_1, d_2, ..., d_m\} \in D$ and m is the number of connected edges, the probability P, which is calculated from Eqn.5, is set for selecting the explored edge.

$$P = softmax\Big(\frac{D - \bar{D}}{\sigma}\Big) \tag{5}$$

where $\bar{D}$ is the average value of the distances in D, and σ is its standard deviation of distances in D.

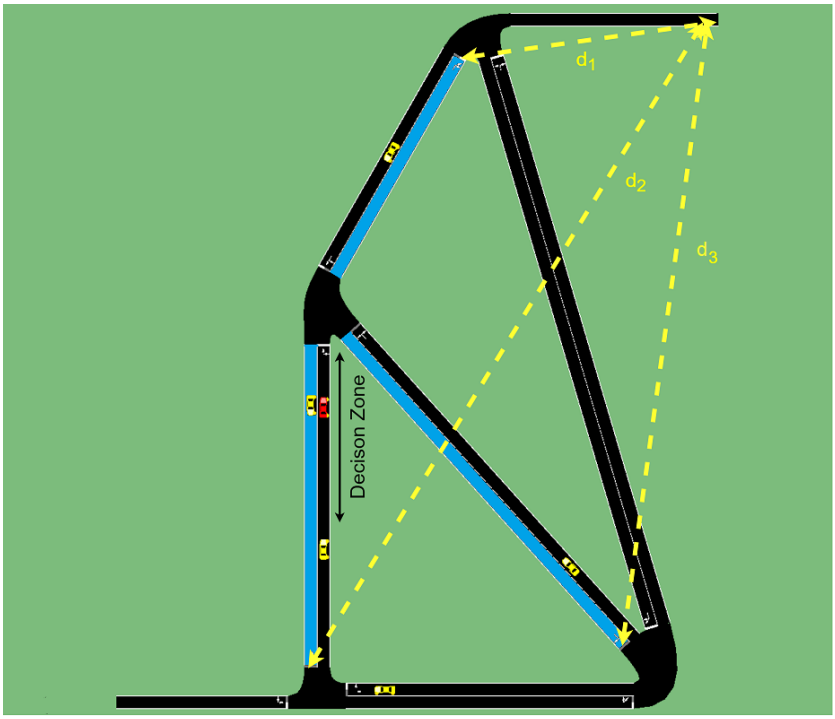


Figure 6: Comparison of different DQN methods

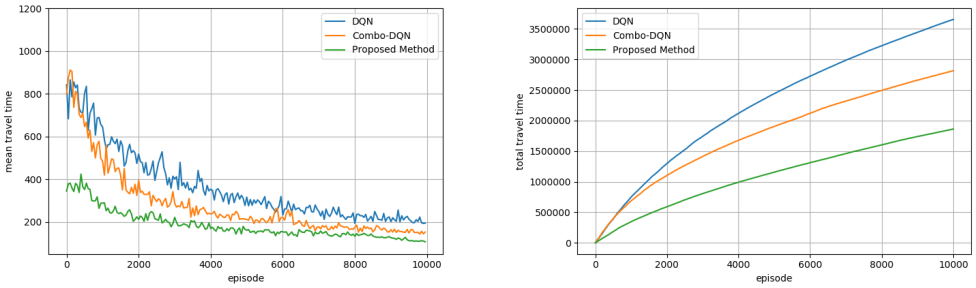


Figure 7: Comparison of different DQN methods

As illustrated in Figure 7, comparing to the traditional DQN and the combination of double DQN, dueling DQN method and priority experience replay (named as Combo-DQN here), our proposed DRL method has the best convergence performance on both the converged travel time and the converging speed. It shows that the average converged travel time by using our proposed method after 10,000 epochs is 100 time steps which is much

lower than the DQN and Combo-DQN. Furthermore, it shows that the cumulative travel time of the proposed method during the training is also the best among the three methods. The pseudocode of our algorithm is presented in Algorithm 1 to clarify the details of the method. In addition, the next section explains all the implementation details for its reproducibility.

Algorithm 1: Deep Q-Learning with memory replay

initialise replay memory D to capacity N; action-value function Q with θ; target-value function $\bar{Q}$ with $\theta^{-} = \theta$
for *episode=1, M* **do**
  repeat
    observe state s
    with probability ϵ select a random action a_t
    otherwise select $a_t = argmax_a Q(s, a; \theta)$
    execute action a, observe reward r and next state s'
    store transition (s, a, r, s') in D
    sample random minibatch of transitions from D
    if s'_j *is terminal* **then**
      $y_j \leftarrow r_j$
    else
      $y_j \leftarrow r_j + \gamma max_a \mathrm{Q}(s'_j, a'; \theta)$
    end
    if *episode* $>$ *9* **then**
      perform a gradient descent step on $(y_j\text{-}\mathrm{Q}(s_j, a_j; \theta))^2$ with respect to the network parameters θ
    end
  until *simulation is terminal*;
  Every C steps reset $\bar{Q} \leftarrow$ Q
end

6. Experiment Preparation

This section presents the preparation of the experiment for implementing our proposed method. It includes simulation environment building, demand traffic generation and smart agent training process.

6.1. Building a Simulation with SUMO

This subsection presents the implementation of how to build a simulation for experiment. Figure 8 illustrates the SUMO traffic simulation process diagram.

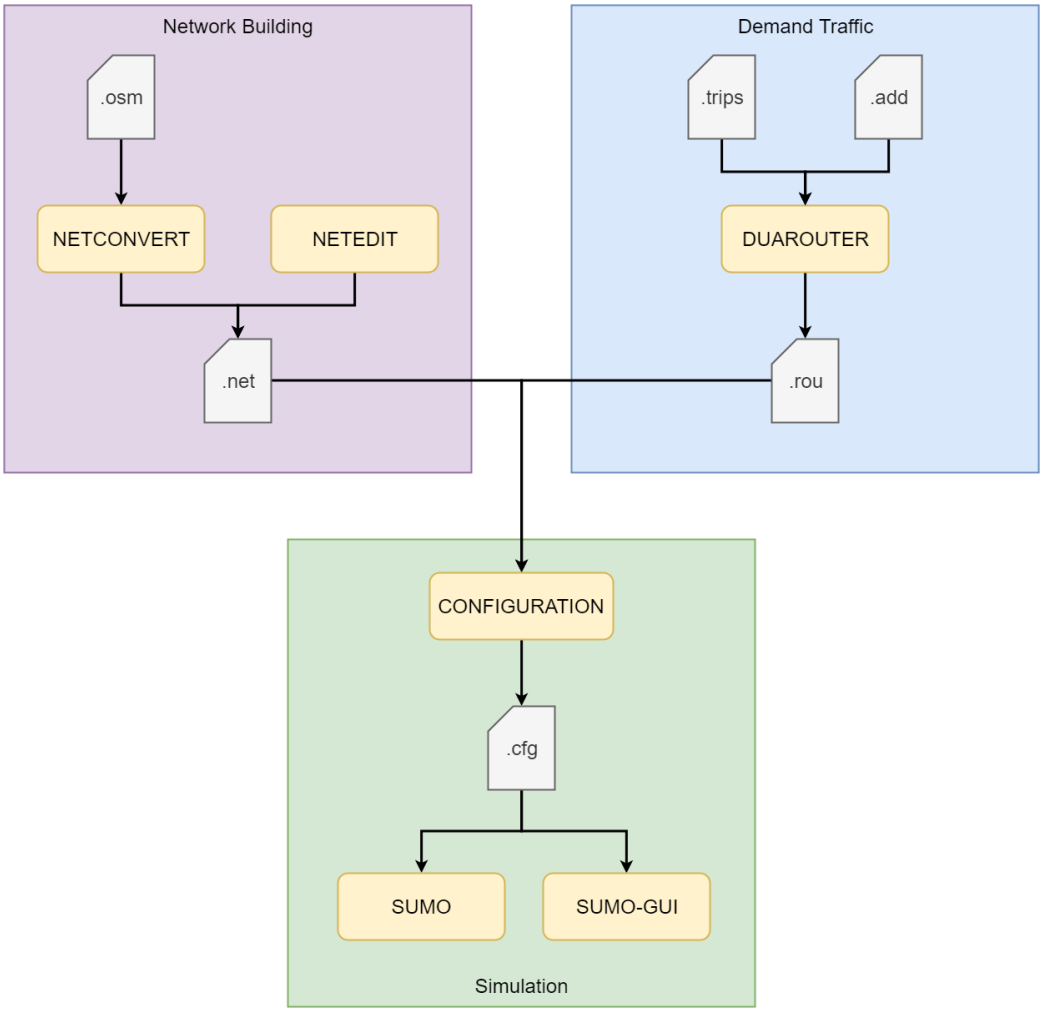


Figure 8: SUMO traffic simulation process diagram

Network building: The experiment targets on optimizing the vehicle route in a real-time manner in an urban transportation network, therefore all of the simulations are using real urban map that converted from OpenStreetMap via the SUMO method called NETCONVERT. For toy data simulation, the network is built by using a SUMO graphical network editor

NetEdit. Eventually, the SUMO networks are designed by microscopic road traffic simulations, which is ready for the routing navigation purpose.

Demand Traffic: Each vehicle in SUMO simulation is defined explicitly since SUMO is a microscopic traffic simulator. A unique identifier, its departure time, and the vehicle's route are provided via the SUMO network. Here, the route is the complete list of connected edges between the origin/destination pair. A trip is defined as the trajectory of a single vehicle that contains the origin/destination pair and the departure time. The trip data is stored in *.trips.xml* file.

Moreover, vehicle's properties can be further categorized based on vehicle type. The considered properties for the description of vehicle type in this experiment are described as follows:

- *id:* Unique identifier for this vehicle type.
- *accel:* The acceleration ability of vehicles of the corresponding type.
- *decel:* The deceleration ability of vehicles of the corresponding type.
- *sigma:* An evaluation of the imperfection of the driver which value is between 0 to 1.
- *maxspeed:* The maximum velocity of the vehicle.
- *color:* The color for this vehicle type (only applied in SUMO-GUI).
- *probability:* The probability of the distribution for this vehicle type.

Two vehicle types are defined in this experiment which is "normal_car" and "truck". The definition details of these two vehicle types are displayed in Table 1 and stored in *.add.xml* file. As shown in Table 1, each type of vehicle has its attributes, i.e. length, acceleration, deceleration, sigma, maximum speed, color and probability. Here the color is only for visualization purposes. These *.trips.xml* and *.add.xml* files are supplied to route generation method to generate the route file *.rou.xml* for traffic simulation.

6.2. Data Extraction and pre-processing

This subsection presents how the environment class extracts the data from SUMO simulation to compose the observation for state space and aggregate rewards. This experiment imports SUMO API TraCI to interact with SUMO

Vehicle Type	Length	Accel.	Decel.	Sigma	Max Speed	Color	Prob.
Normal Car	5.0	2.0	5.0	0.5	20.0	yellow	0.8
Truck	8.0	1.0	5.0	0.5	5.0	green	0.2

Table 1: Definition of vehicle type

Features	**TraCI method**
Number of vehicles in edge	traci.getLastStepVehicleNumber()
Expected travel time in edge	traci.getTravelTime()
Current Edge	traci.getShape()
Destination Edge	traci.getShape()

Table 2: Traci function for extracting features data

and extracts the data during the simulation. Table 2 shows the TraCI methods that used to retrieve features in the network, i.e. the number of vehicle on each road n_e, the expected travel time of each road t_e, the current position of the agent c_v and its destination d_v.

6.3. Deep Neural Network Architecture and Training

This subsection presents the deep neural network structures and the training parameters used in the proposed framework. The neural network contains five fully connected layers with its input layer, two hidden layers and dueling structure, and its output layer. The first hidden layer has 150 neurons and the second hidden layer has 100 neurons. Both hidden layers use RELU Mnih et al. (2015) as the activation function. A dueling network splits into two streams of fully connected layers which are the advantage stream and value stream. The value stream only has one output and the advantage stream has the output as many as the number of actions, which is the maximum number of connected roads in the urban network. The neural network is created by using the Tensorflow library Abadi et al. (2016).

As shown in Table.3, 10000 episodes are executed in SUMO with 0.001 learning rate to train the vehicle agent. The exploration rate, also known as the greedy rate, decreased from 1 to 0.05. The target network parameters will be replaced every 3000 learning steps. The discount factor for the reward is 0.99. The total replay memory size for storing the transactions is 10000, and the mini-batch for training is 32. The prioritization exponent is set to

0.6 and the prioritization important sampling is increased from 0.4 to 1.

Parameter	Value
Episodes	10000
Learning Rate	0.001
Exploration	$1.0 \rightarrow 0.05$
Target Network Update per learning step	3000
Discount Factor	0.99
Replay Memory Size	10000
Mini Batch for Update	32
Prioritisation Exponent	0.6
Prioritisation Important Sampling	$0.4 \rightarrow 1.0$

Table 3: Vehicle agent hyper parameters for intelligent navigation

7. Experiment Evaluation

There are two subsections in the experimental evaluation: Firstly, two toy data maps are generated for testing the convergence of intelligent navigation agents. Additionally, the toy data simulation can further provide a tool to gain insight of the decisions made by the intelligent agent during the navigation. Secondly, nine traffic conditions based on three regions in Liverpool city center are simulated to demonstrate the efficiency of the DRLs.

To further demonstrate the performance of the proposed method, the proposed method is compared with five algorithms, which are GDUE-Dijkstra, GDUE-A*, Dynamic-Dijkstra, Dynamic-A* and Ant-Colony. GDUE-Dijkstra and GDUE-A* are the default traffic assignment algorithm in SUMO, where GDUE stands for Gawron's Dynamic User Equilibrium Gawron (1998). GDUE uses Dynamic Traffic Assignment (DTA) to model the traffics via a discrete time-dependent network. It assigns routes for all trips using some shortest path algorithms (e.g., Dijkstra's algorithm and A* algorithm) as an initialization step by taking the edge length as edge cost. After running the traffic simulation, it records the actual traveling time on each edge, and uses the same shortest path algorithms to re-assign the routes. This step is done iteratively until the edge cost for all roads is relatively converged. Dynamic-Dijkstra and Dynamic-A*Kaparias and Bell (2010) uses a dynamic vehicle route planning method. These routing approaches re-compute their route

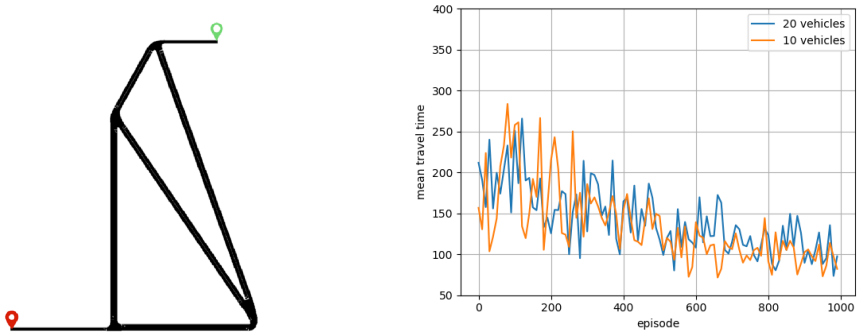
periodically, or at a specific time, dynamically by using one of the shortest path methods. The routing takes into account the current and recent state of traffic in the network and thus adapts to traffic jams and other changes in the network. Based on these methods, vehicles can be re-routed dynamically while a simulation is running. Furthermore, the Ant Colony algorithm in Kponyo et al. (2012) is also implemented to find the shortest traveling time for a vehicle to its destination. The core idea is to let the vehicle to choose, in probability, the path marked by stronger pheromone concentrations.

7.1. Toy Data

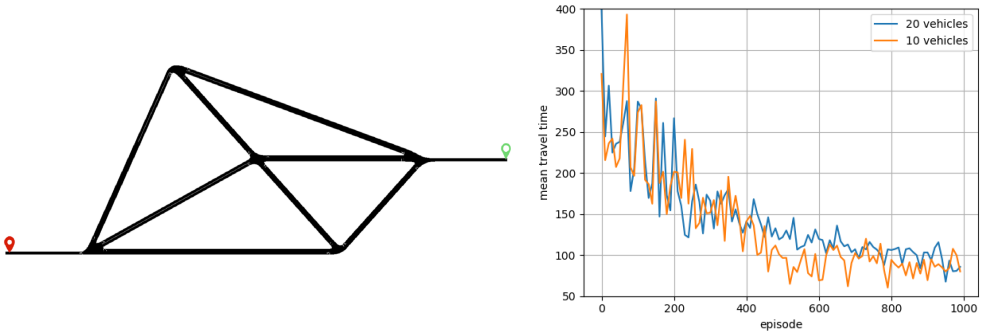
In our toy data experiments, two simple maps are built to train and test the intelligent vehicle agent. The first map has 12 edges and each edge connects with two other edges (m=2) as illustrated in Figure 9a. The second map has 18 edges and each edge connects with three other edges (m=3) as illustrated in Figure 9b. In these maps, two location icons (red and green icons) are used to show the starting point and the destination of the navigation tasks. In the experiments, two types of vehicles, normal_car and truck, are injected into the maps to simulate realistic traffic conditions by using SUMO tools randomTrip and duarouter. The two types of vehicles have a different maximum speed, acceleration and deceleration. The number of vehicles is set as 10 and 20 respectively and these vehicles are added into the map randomly during the time stamp between 0 and 30.

In Figure 9a and 9b, the right subfigures illustrate the convergence of the intelligent agent during the training process. It demonstrates that all the cases converged within 1000 episodes from random explorations. The converged average traveling time is about 100 time steps. Comparing to the simple map one, the average traveling time of the agent trained in the simple map two is much longer as the agent has more action selections at the start of the training stage. However, both finally converged to similar traveling time levels due to the similar routing distance and traffic complexity. In the experiments, it is also found that the convergence of the run with 10 vehicles in the maps is slightly faster than the run with 20 vehicles.

When the decision network converges, it is capable of selecting optimal decisions to navigate its vehicle to the destination based on its observations of the current traffic states. The average decision-making process timing is 2.7 millisecond and it means that the agent is suitable to make real-time decisions for the navigation task. The routing selected by using Dijkstra and A* methods is shown in Figure 10 (a). It is static and not able to be adapted



(a) Simple map 1



(b) Simple map 2

Figure 9: Simple map structure and mean step in 100 episodes

to the volatile traffic states. However, the DRL based agent (illustrated from Figure 10 (b) to Figure 10 (e)) makes flexible routing decision based on its observation when it approaches each decision zone. In Figure 10 (b), the Q value of the decision to travel straight is much higher than the Q value of the decision to turn left. It is consistent with an intuitive observation that a truck with lower speed is on the left edge. In Figure 10 (c), the selection of the left edge is reasonable as the vehicle number on the right edge is much more than the number on the left edge. The route selected by the intelligent agent is illustrated in Figure 10 (f). In this demonstration, it takes 39 time steps to reach the destination by using the proposed algorithm while it takes 56 time steps when using the routing algorithm of Dijkstra and A*. In other words, the proposed navigation method improves 30.4% in terms of traveling

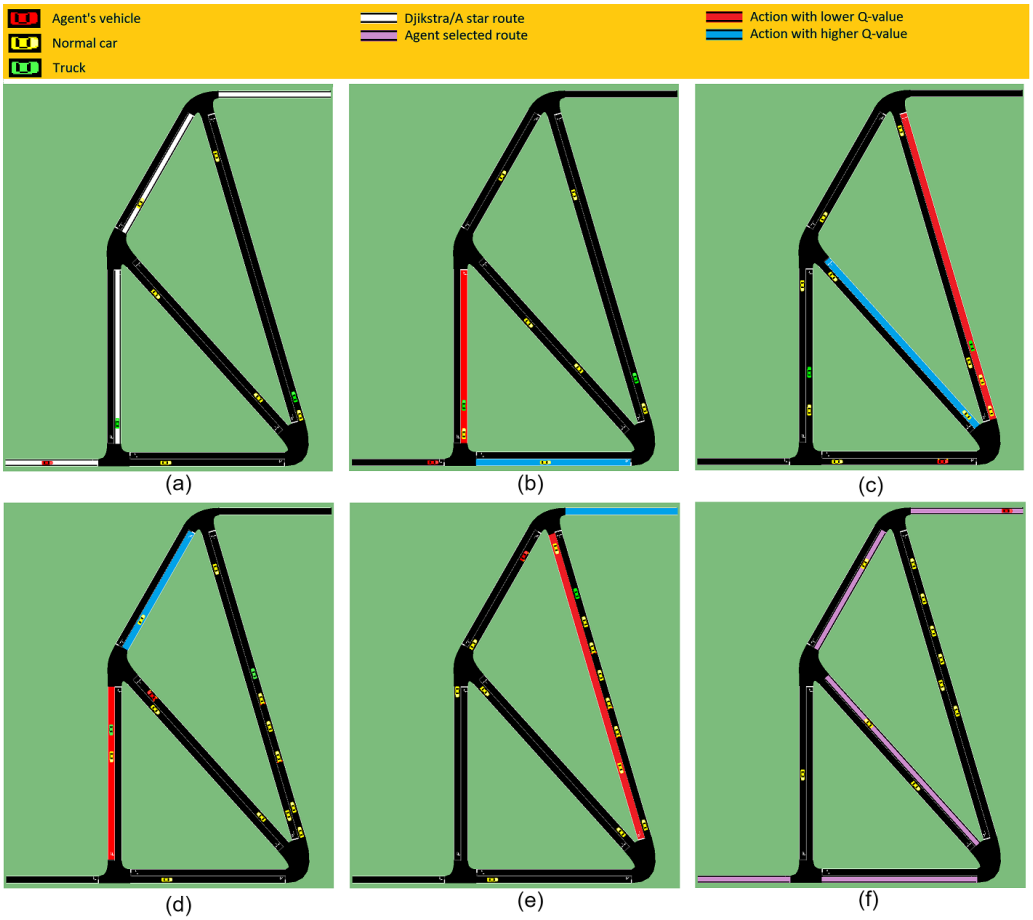
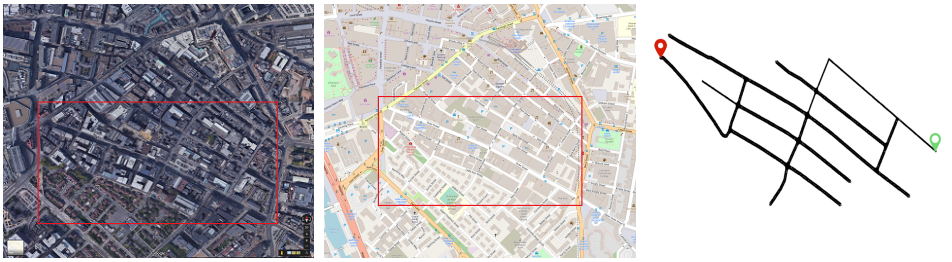


Figure 10: Simulation Result

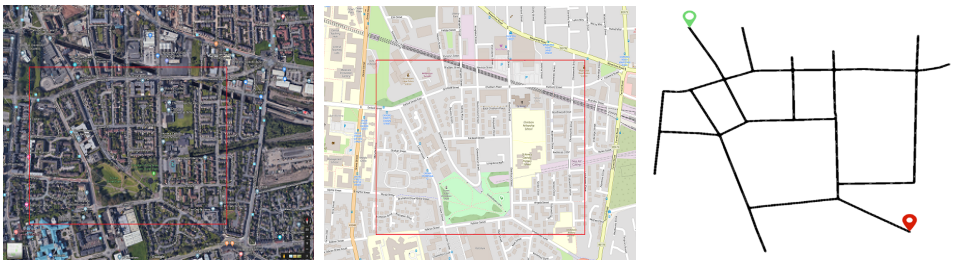
time in this case.

7.2. Realistic scenario analysis

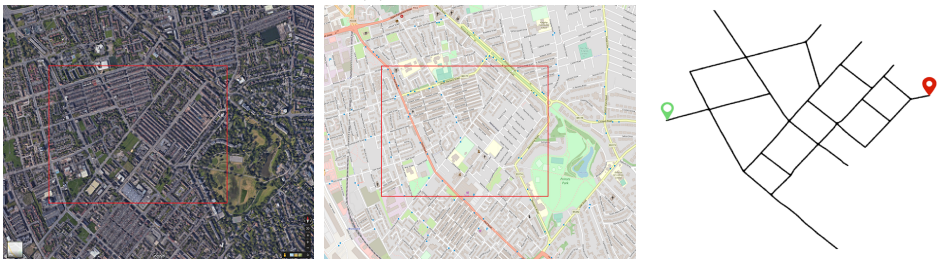
Tthe effectiveness of our framework is further tested in a more realistic scenario. In this experiment, three busy traffic regions in Liverpool city center are selected on OpenStreetmap and converted into the SUMO maps in our system. The highlighted regions on GoogleMap, OpenStreetMap and SUMO generated maps are illustrated in Figure 11. In each map, three demand traffics with different number of vehicles are made accordingly to test the proposed DRL method in the integrated environment. The details of the map information are presented in Table 4.



(a) Liverpool city map 1



(b) Liverpool city map 2



(c) Liverpool city map 3

Figure 11: Real world city map for SUMO environment

	City Map 1	City Map 2	City Map 3
Total edges	40	60	80
Avg edge length (m)	107.79	143.23	173.95
Edge max speed range (mph)	20-30	30-50	30-50

Table 4: Maps information

The convergence under these traffic conditions is illustrated in Figure 12. In these three maps, all the cases converged to certain levels ranging from 78 to 160 time steps. This is highly correlated to the complexity of the maps as the converged average traveling time in the third map is the longest while the

time in the first map is the shortest. Another finding is that the converging speed is relatively slower when there are more vehicles in the simulation due to the more volatile road conditions.

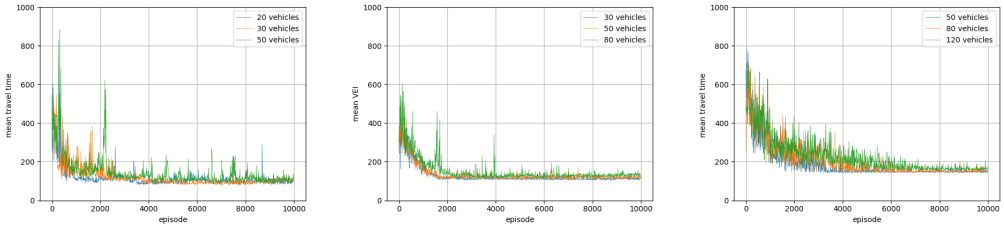


Figure 12: Convergence of the smart agent in (a) City Map 1 (b) City Map 2 (c) City Map 3

For vehicle route navigation, it is very important to recognize the potential traffic congestion to arrive at destination as fast as possible. From our experiments, it is illustrated in Figure 12 that under the same city map, when the demand traffic is higher, it is more likely to get a sharp peak while training. This is because of the higher possibility that certain roads on the map are suffering from traffic congestion. In the city map 3, instability was observed at the beginning stage of the training as illustrated in Figure 12. The reason is that city map 3 contains the highest number of edges which represents more exploration options during the training. While our proposed scheme solves the slow convergence problem so that it takes similar time stamps to achieve the convergence.

When the RL agent converges, we compared the proposed DRL agent with the GDUE Dijkstra, GDUE A*, Dynamic Dijkstra, Dynamic A* and Ant Colony methods objectively. As aforementioned, the 5 comparing methods are the state of art algorithms for vehicle navigation and are generally applied in modern world for decade. The purpose of this comparison is to prove that our purposed method can make a better performance by improving the travel time for individual vehicle. In the interested of fairness, we assigned 3 different traffic conditions in city map 1, 2 and 3 to present different level of traffic. 100 runs are made on each traffic condition and each traffic condition has exact vehicle number and vehicle type. We then navigated a vehicle to its destination by using different methods and observed the travel time it needs to finish a trip. The average traveling time of the runs and the standard deviation is presented in Table 5. The results show that the proposed method

outperforms other algorithms in all of the traffic conditions. According to the comparison table, in the smallest map, i.e. city map 1, our proposed method reduces at most 5.3%, 5.1% and 16.4% traveling time with different demand traffic. Further, in the largest map city, i.e. city map 3, our proposed method reduces at most 4.9%, 12.4% and 22.5% traveling time in different demand traffic. The results also showed that the improvement of the performance becomes much better when the road condition is more complex and volatile.

To further prove the effectiveness of the proposed method, Wilcoxon test McDonald (2009) is used to run the significance analysis to compare with the benchmark methods. The Wilcoxon test is a nonparametric statistical test where the data population does not require to follow a normal distribution assumption. According to the significance level analysis presented in Table 6, all the significant level comparisons between the proposed method and individual benchmark algorithms are smaller than 0.05. It shows that the performance of our proposed method is superior over the state-of-the-art methods. More precisely, from our experiments dynamic Dijkstra and Ant Colony had slightly better performance than GDUE Dijkstra, GDUE A* and dynamic A*. From each map, the significant level raised when the demand traffic is higher. In Table 6, we can see in each map, higher demand traffic has higher significant level where in Map 1 is $3.34e^{-}9$ (50 vehicles), Map 2 is $3.27e^{-}10$ (80 vehicles) and Map 3 is $7.58e^{-}11$ (120 vehicles). This is mainly because with less complicated demand traffic, conventional methods are still able to find the best route for vehicle to reach destination. However while the demand traffic increases, they struggle to maintain the same performance. Another reason for this is, if there are more than one path to destination with similar distance, making bad decision in vehicle navigation does not bring significant impact in travel time as both paths are more likely to have similar traffic when demand traffic is lower. Furthermore, the improvement also becomes more significant when the city map is larger. According to Table 6, biggest map (Map 3) has greatest significant level among 3 maps, with $7.58e^{-}11$, $3.73e^{-}11$ and $1.26e^{-}9$ in demand traffic 120 vehicles, 80 vehicles and 50 vehicles respectively. This is because in bigger map, the travel time difference between different paths is bigger than smaller map as the average length of the road is longer. Therefore, making wrong navigation in bigger map needs to take longer time to reach destination. Due to these reasons, our propose methods performed significantly better in bigger map and more complicated demand traffic.

Methods	Map1			Map 2			Map 3		
	20 vehicles	30 vehicles	50 vehicles	30 vehicles	50 vehicles	80 vehicles	50 vehicles	80 vehicles	120 vehicles
GDUE Dijkstra	81.43 (±3.31)	81.78 (±3.61)	92.66 (±14.47)	109.48 (±5.74)	118.23 (±11.59)	131.26 (±19.03)	162.94 (±12.75)	180.04 (±16.72)	206.82 (±25.37)
GDUE A*	82.48 (±4.34)	82.96 (±3.87)	95.54 (±13.82)	110.78 (±8.62)	119.14 (±11.76)	131.77 (±19.48)	163.86 (±13.06)	181.70 (±17.44)	208.51 (±28.26)
Dynamic Dijkstra	79.26 (±4.94)	79.70 (±2.98)	90.84 (±11.99)	105.74 (±4.68)	110.96 (±8.82)	126.42 (±9.46)	148.98 (±11.50)	161.34 (±11.07)	194.32 (±9.80)
Dynamic A*	81.25 (±6.40)	80.26 (±4.01)	91.18 (±13.36)	107.98 (±5.90)	111.36 (±8.62)	127.76 (±9.82)	149.16 (±11.65)	162.12 (±9.91)	195.96 (±8.92)
Ant Colony	79.56 (±4.79)	79.84 (±4.72)	90.98 (±12.64)	106.90 (±4.63)	113.36 (±11.82)	127.98 (±12.44)	151.08 (±9.07)	176.80 (±18.00)	205.12 (±30.97)
RL Agent	78.10 (±2.66)	78.72 (±4.19)	84.06 (±5.29)	103.44 (±2.48)	107.66 (±4.43)	117.00 (±6.85)	143.64 (±4.84)	147.82 (±5.60)	159.02 (±8.73)

Table 5: The objective performance comparisons under a various of traffic conditions

RL Agent vs Previous Works	Map 1			Map 2			Map 3		
	20 vehicles	30 vehicles	50 vehicles	30 vehicles	50 vehicles	80 vehicles	50 vehicles	80 vehicles	120 vehicles
GDUE-Dijkstra	$7.11e^{-3}$	$4.55e^{-7}$	$3.34e^{-9}$	$2.48e^{-7}$	$1.56e^{-8}$	$3.27e^{-10}$	$1.26e^{-9}$	$3.73e^{-11}$	$7.58e^{-11}$
GDUE-A*	$3.33e^{-7}$	$5.12e^{-8}$	$6.27e^{-11}$	$4.65e^{-7}$	$6.77e^{-8}$	$3.55e^{-10}$	$2.31e^{-9}$	$4.12e^{-11}$	$7.79e^{-11}$
Dynamic-Dijkstra	$1.90e^{-3}$	$2.86e^{-2}$	$4.70e^{-7}$	$2.52e^{-4}$	$9.71e^{-5}$	$8.20e^{-7}$	$2.15e^{-5}$	$3.25e^{-9}$	$5.98e^{-10}$
Dynamic-A*	$7.21e^{-7}$	$3.90e^{-7}$	$2.63e^{-9}$	$1.23e^{-6}$	$5.74e^{-6}$	$1.02e^{-8}$	$2.21e^{-5}$	$7.06e^{-10}$	$6.86e^{-10}$
Ant-Colony	$1.87e^{-3}$	$2.30e^{-3}$	$3.24e^{-7}$	$2.71e^{-5}$	$4.53e^{-6}$	$3.24e^{-8}$	$2.01e^{-7}$	$8.24e^{-10}$	$6.99e^{-11}$

Table 6: The significance analysis by using Wilcoxon test

8. Discussion and conclusion

Traffic congestion is a major contemporary issue in many densely populated cities. Thus, during the past several decades, many vehicle navigation systems are designed for vehicles to reach their destinations as quickly as possible when traffic is busy. However, it is not a trivial task to find an optimal solution under a complex city environment. In this paper, a novel DRL based vehicle routing optimization method is proposed to re-route vehicles to their destinations in complex urban transportation networks. In specific, we simulate nine realistic traffic scenarios, using SUMO and train deep neural network model, to navigate vehicle in real-time. The improved design of a DQN architecture makes our solution best suited for real-time smart vehicle navigation. Our DRL model has been proved effectively when compared with five benchmark methods.

Nonetheless, there are several limitations in our work: (i) although this paper shows a significant reduction of the vehicle traveling time when applying the DRL method to train the model for vehicle agent, the performance is expected to be further improved when more efficient features can be extracted

from the environment; (ii) the proposed framework needs to not just focus on traveling time, but also vehicle emissions to achieve a more sustainable transportation network; and (iii) the model is trained based on simulated traffic data, thus a fine-tuned stage is required when deployed on real traffic data.

To further improve the proposed vehicle route optimization for urban transportation networks, several further research topics need to be considered. First, optimization of the proposed framework towards a more sustainable urban transportation network. A method that can index the impact of vehicle emission for vehicle navigation is needed in the next stage. Besides abnormal driving behaviors for emergency, vehicles should be considered by the DRL agent, e.g., exceeding the speed limit, overtaking on the right, or driving in opposite direction. Secondly, more efficient features could be investigated to represent a more realistic urban traffic condition and vehicle behavior. The features in the proposed approach have confirmed the effectiveness of optimizing vehicle routes in the urban transportation network. However, in practice, more factors could affect the urban traffic condition. These need to be investigated in the future. Thirdly, collections and analyses of real traffic data are needed in order to narrow the gap between real-data and simulation data generated by the simulator. Another potential future work is to use transfer learning and domain adaptation techniques to fill the gap so that the concept can be commercialized.