

Artificial Intelligence in Traffic Signal Control for Large-Scale Urban Networks

Shimon Komarovsky

Artificial Intelligence in Traffic Signal Control for Large-Scale Urban Networks

Research Thesis

In Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Shimon Komarovsky

Submitted to the Senate of the Technion - Israel Institute of
Technology
Tevet, 5784 Haifa December, 2023

The Research Thesis Was Done in the Faculty of Civil and Environmental Engineering Under The Supervision of Assoc. Prof. Jack Haddad.

List of Publications

Some results in this thesis have been published as an article in a conference during the course of the author's research period:

- Shimon Komarovsky and Jack Haddad, "Spatio-Temporal Graph Convolutional Neural Network for Traffic Signal Control in Large-Scale Urban Networks," in *Transportation Research Board Annual Meeting (TRB'22)*, 2022.

The generous financial support of the Technion is gratefully acknowledged.

The author of this thesis states that the research, including the collection, processing and presentation of data, addressing and comparing to previous research, etc., was done entirely in an honest way, as expected from scientific research that is conducted according to the ethical standards of the academic world. Also, reporting the research and its results in this thesis was done in an honest and complete manner, according to the same standards.

Contents

List of Symbols and Abbreviations

1	Introduction	3
1.1	Research vision	5
1.2	Research motivation	5
1.3	Research goal	6
1.4	Scientific background	8
1.4.1	Machine-Learning-based works in TSCP research	8
1.4.2	Other AI works in TSCP research	11
1.4.3	Traffic event detection via text data	11
1.5	Research contributions	13
1.6	Methodology	13
1.6.1	Step 1	14
1.6.2	Step 2	16
1.6.3	Step 3	16
1.6.4	Step 4	17
1.7	Thesis outline	17
2	Spatio-temporal Graph Convolutional Neural Networks for traffic signal control in large-scale urban networks	19
2.1	Problem definition	19
2.2	Spatio-temporal graphical representation for transportation networks	20
2.3	Graph Convolutional Networks	22
2.3.1	Spatial GCNs	24
2.3.2	Spectral GCNs	24
2.4	Implementation	27
2.4.1	GNN setup	27
2.4.2	Spatio-Temporal Graph Convolutional Network (STGCN) models	28
2.5	Results	30

2.5.1	Evaluation metrics	31
2.5.2	Experiment 1: spatio-temporal model	32
2.5.3	Experiment 2: Spatio-temporal model including commands and executions	34
2.6	Discussion	40
2.7	Conclusions	41
3	Preference commands in traffic signal control via Double Deep Q-Network	42
3.1	Motivation	42
3.1.1	Implemented RL versus SL comparison	43
3.2	Introduction	46
3.3	Related works on RL in Traffic signal control problem (TSCP)	50
3.3.1	Graph Neural Networks (GNNs) in RL	53
3.4	Preference Commands in TSCP	55
3.4.1	Problem Definition	55
3.4.2	Optimization methods	55
3.5	Implementation	61
3.5.1	Neural Network Structure	61
3.5.2	Constraints	64
3.5.3	The implemented algorithm	67
3.5.4	Networks and models	69
3.6	Results	71
3.6.1	No-preference preliminary results	71
3.6.2	Issues observed during training	73
3.6.3	Deciding upon most appropriate (input, output) for preference command	75
3.6.4	Issue with preference deployment	77
3.6.5	Hybrid reward	78
3.6.6	Relative versus extreme preference	79
3.6.7	Generalization from 2×2 network to 5×5	80
3.6.8	Fixed versus changing preference weights	81
3.6.9	Green constraints	87
3.6.10	STGCN	87
3.7	Discussion	93
3.8	Conclusions	95
3.9	Plans for next research	97
4	Converting textual instructions into preference commands for traffic signal control	98
4.1	Relevant literature review	99

4.1.1	Language model (LM) in TSCP	99
4.1.2	LM basics	100
4.1.3	Shortest-path problem via deep learning	101
4.2	Problem definition	103
4.3	Implementation	104
4.4	Results	105
4.4.1	Fine-tuning step 1 - Dataset Construction	106
4.4.2	Fine-tuning step 2 - Simple Commands	108
4.4.3	Fine-tuning step 3 - Shortest Path Problem	111
4.4.4	Fine-tuning step 3 - Ablation Study	116
4.5	Discussion	118
4.6	Conclusions	122
5	A proof-of-concept case study	124
6	Conclusions, contributions and future work	131
6.1	Conclusions	131
6.2	Contributions	133
6.3	Future work	133
6.3.1	Proposed encoder-decoder DLM structure	134
6.3.2	A detailed model of the encoder-decoder structure	135
6.3.3	Connection to other chapters	137
A		141
A.1	Explicit approach	141
A.1.1	Prompting ChatGPT	141
A.1.2	Prompting ChatGPT for converting our problem into a programming code	144
8	References	167

List of Figures

1.1	Automatic traffic management system.	6
1.2	Deep graphical models for a simple automatic management system.	15
1.3	The different research steps and their corresponding chapters.	18
2.1	GNN for prediction (Y) given the inputs A and X , in road sections.	28
2.2	An enhanced STGCN model, for each road section i	30
2.3	Tel Aviv BT sensors map (left) and Even-Gvirol arterial with its graph (right).	31
2.4	Predicted vs real speed data for a given sensor within specific time range.	33
2.5	Methods comparison by their validation, test, and other metrics.	34
2.6	Experiment 1 versus Experiment 2 in time steps.	36
3.1	The TSCP problem tackled in this chapter (the system and its objective).	48
3.2	State evolution representation of $\text{TD}(\lambda)$, illustrated as a combination Q and MC learning, taken from [161].	60
3.3	Different control structures in DQN (separated versus unified outputs).	62
3.4	Each intersection has four phases of traffic light.	63
3.5	The RL system incorporating external logic to enforce constraints.	66
3.6	Networks used in this chapter.	70
3.7	Different demands in paired intersections network, fixed weights.	72
3.8	Comparison between three different policies.	73
3.9	Different issues preventing a proper optimal policy learning.	75
3.10	Different state and reward combinations, each with un-weighted or weighted features (legend is on the top-left of each intersection).	77

3.11	Preferred movement 1 is in grid lock due to preferred movement 4. The preferred trajectory comprises of four preferred movements.	78
3.12	Comparing different hybrid rewards, in training the FCN model.	79
3.13	Comparison between relative and extreme preferences.	80
3.14	Comparison of DQNs trained on 2×2 and 5×5 networks, in different scenarios.	80
3.15	Some of the preference weight scenarios for testing, in 3×3 network.	83
3.16	Different model configurations of DQN (r =reward).	84
3.17	STGCN setup: a single and shared FCN model is attached to STGCN for every intersection (r =reward). Similar to Fig. 3.16(a).	90
3.18	The preference weight scenarios for testing, in 5×5 network.	91
4.1	General LM-based model for preference prediction, with an input and output example.	103
4.2	Standard token-based LM, adapted to our preference prediction problem.	108
4.3	Training of graph-based LM for preference prediction.	109
4.4	Examples of model predictions via graphs for the shortest path commands.	110
4.5	Shortest-path-command models.	111
4.6	Shortest-path iterative model.	112
4.7	Shortest-path iterative DNN model, via LSTMs.	113
4.8	Examples of shortest-path iterative model predictions via graphs, for the shortest path commands.	115
4.9	Examples of paths with different ambiguity levels.	118
5.1	The different research steps and their corresponding chapters, based on Fig. 1.3.	125
5.2	Converting GPT-road to RL-lane preferences.	126
5.3	An example of GPT’s trajectory solution, inserted next to the realtime RL traffic signal control system.	129
5.4	Simulation of traffic signal control via RL, given the preferences illustrated in Fig. 5.3.	130
6.1	Sensory data and commands in encoder-decoder DLM structure.	134
6.2	Hierarchical learning of tasks.	136
6.3	Hierarchy-temporal DLM.	137
6.4	The different research steps in the view of the proposed DLM.	140
A.1	Full description of our problem to ChatGPT.	142

A.2	Dialogue with ChatGPT.	143
A.3	Initial prompt and a dialogue with ChatGPT.	145
A.4	General code for our task, produced by ChatGPT.	146
A.5	Specific code solution produced by ChatGPT, given several examples.	147

List of Tables

2.1	Experiments comparison under different categories.	36
2.2	Different evaluations for a constant cycle time step.	39
3.1	SL versus RL comparison over different aspects.	46
3.2	Fixed and changing preference weights, via multiple model configuration, see Fig. 3.16(b).	85
3.3	Comparison of different models in different scenarios, for states with and without downstream features.	86
3.4	Comparing different models in different scenarios, with and without applying minimum and maximum green duration constraints for every phase.	88
3.5	Comparing different models (mainly STGCN) with different parameters and scenarios in 3×3 network.	89
3.6	Comparing different models with different parameters and scenarios in 5×5 network.	92
3.7	Different preference setups for states and rewards.	94
4.1	Comparison of different DNNs to solve the shortest-path problem.	113
4.2	Comparison of LSTM models with different iteration lengths. .	116
4.3	Comparing different configurations of iterative models. . . .	117
4.4	Comparison of models trained and tested on different datasets, with different levels of ambiguity.	118

List of Symbols and Abbreviations

α	target Q-Network update coefficient	59
ϵ	Exponentially decaying coefficient	57
γ	The discount factor (in Reinforcement Learning).....	56
Λ	Diagonal matrix with Laplacian eigenvalues.....	25
λ	Trace decay parameter	59
λ_t	Laplacian eigenvalues.....	25
π	Policy in Reinforcement Learning.....	50
θ	Parameters.....	26
A	Action space (in Reinforcement Learning)	44
A	Adjacency matrix (in Graph Neural Networks)	22
a	Action (in Reinforcement Learning)	49
B	Batch size.....	71
D	Diagonal node degree matrix	24
d	Number of features.....	22
d_i	Vehicle delay	47
H	Feature matrix.....	22
L	Graph Laplacian	25
lr	Learning Rate.....	36

n	Number of nodes or intersections (in Graph Neural Networks)	22
O	Output priority matrix	102
OD	Origin-Destination	71
Q	Q function or Q-values	56
Q_{target}	Q-values according to learned policy	58
R	Reward in Reinforcement Learning	50
S	State space (in Reinforcement Learning)	44
s	State (in Reinforcement Learning)	49
T	Number of time steps in state	68
u_t	Laplacian eigen-vectors	25
w_i	Preference relative values	46
A2C	Advantage Actor-Critic	53
AC	Actor-Critic	50
AI	Artificial Intelligence	4
BERT	Bidirectional Encoder Representations from Transformers	100
BT	BlueTooth	1
CNN	Convolutional Neural Network	9
COD	curse of dimensionality	9
CoT	Chain-of-Thought prompting	143
DBN	Deep Belief Network	8
DDQN	Double Deep Q-Network	55
DL	Deep Learning	4
DLM	Deep Learning model	5
DNN	Deep Neural Network	7
DQN	Deep Q-Network	1

DRL	Deep Reinforcement Learning	8
FCN	Fully-Connected Neural network	49
GAT	Graph Attention Network	20
GCN	Graph Convolution Network	20
GNN	Graph Neural Network	13
GPT	Generative Pre-trained Transformer	2
GRU	Gated Recurrent Unit	20
GT	Green-Time duration	1
GTN	Graph Transformer Network	20
i.i.d.	independently identically distributed	53
IC	Interpolation-based Control	3
LDA	Latent Dirichlet Allocation	12
LLM	Large Language Model	99
LM	Language Model	98
LSTM	Long Short-Term Memory	9
LSUN	Large-Scale Urban Networks	3
MAE	Mean Absolute Error	31
MAPE	Mean Absolute Percentage Error	31
MARL	Multi-Agent Reinforcement Learning	52
MFD	Macroscopic Fundamental Diagram	4
ML	Machine Learning	6
MPC	Model Predictive Control	3
MSE	Mean Square Error	31
NLP	Natural Language Processing	5
NN	Neural Network	8

POI	Places-Of-Interest	22
PSI	Paired Signalized Intersections network.....	69
r.o.w	right-of-way	47
RL	Reinforcement Learning	8
RMSE	Root Mean Square Error	32
RNN	Recurrent Neural Network.....	9
SL	Supervised Learning	8
SP	Signal Plan.....	1
ST	Spatial-Temporal.....	14
STGCN	Spatio-Temporal Graph Convolutional Network	28
SVM	Support-Vector-Machine.....	10
TCN	Temporal Convolutional Network.....	20
TD	Temporal Difference	57
TSCP	Traffic Signal Control Problem.....	3
TT	Travel Time	28
UL	Unsupervised Learning	8

Abstract

This research aims at developing an automatic traffic management system, which can realize traffic operator orders via traffic signal actuations. The system should handle three main issues in large-scale urban networks: (i) scalability of the network, (ii) non-linearity in traffic dynamics, and (iii) availability of heterogeneous big data.

To achieve the research goal, a deep learning approach is developed. The research was conducted in three steps. Step 1's purpose was to simulate execution of commands. This is carried out on a given dataset, which has different outcomes: green-time duration (GT) based on signal plan (SP) command which was designed for the objective of minimum delay. In step 2, the basic objective was extended to include priority for different movements. While in step 3, we produced preference weights from text instructions.

The final system should implement the desired goal of this research. It does so by concatenating steps 3 and 2 in a sequence. Meaning, given some text command entering step 3, the output is weighting preference in every edge in the graph representing the transportation network. Then, this output becomes the input to the system of step 2, which in its turn produces the appropriate control signal in all the intersections in the transportation network.

Step 1 is trained via supervising learning, by a spatio-temporal graph neural network model, utilizing the large-scale urban network of Tel Aviv. It consists of SPs and Bluetooth (BT) extracted features such as velocities.

Step 1 was our first attempt to implement our main goal of applying commands in a transportation network. However, the commands were too simplistic. Hence, a more general family of commands was proposed for step 2: preference commands. In step 2 we use deep Q-network (DQN) and implement it in SUMO transportation simulation. DQN's input is the features from step 1, and its output is the expected total reward for each of the intersections. Then we select an action based on the outputs from the DQN, to be applied in the transportation network.

In the third step we decided to remove the hard manual weight tuning for

every edge in the transportation network, and represent the command more naturally: by text. In this step we used both ChatGPT dialog interface and mainly the *Generative Pre-trained Transformer* (GPT) model for fine-tuning to our specific task.

Finally, results from the steps above showed several points: (i) proof-of-concept for preferences in urban transportation network simulation. (ii) Additional step for combining steps 3 and 2 demonstrated a fully operative automatic management system. (iii) Over all the steps, the principle of communication among agents in a multi-agent structure proved to be beneficial compared to the lack of it. (iv) We showed over the chapters that our system handled the three essential issues mentioned above. (v) The different steps showed that there is compatibility between the type of data and the deep neural architecture to process it. (vi) The importance of data cleaning in deep learning and consistency within the data, especially in step 2.

Chapter 1

Introduction

Traffic congestion is a serious nuisance in large scale urban road networks [1]. It has a large effect on various aspects, such as public safety, pollution, and economy. Therefore, an appropriate traffic control strategy, which operates the signal lights of the network in real-time, should be implemented [2].

Over the decades, several studies were conducted to solve optimally the large-scale traffic signal control problem (TSCP), e.g. SCOOT [3], SCATS [4], and TUC [5,6], and many others. In general the TSCP is usually defined as manipulating the decision variables, e.g. green durations, cycle times, and offsets, in real time to optimize given criteria. Based on control theory, and its various approaches, different traffic signal controllers were designed for closed-loop traffic systems with real time feedback, e.g. linear quadratic regulator [7], model predictive control (MPC) [8], and interpolation-based control (IC) approach [9].

There are basically three main elements associated with the TSCP for *large-scale urban networks* (LSUN): (i) Scalability, (ii) Nonlinearity, and (iii) Big Data.

Scalability. While many works reported successful control implementation of isolated intersections or arterials with a few number of intersections [10–12], only a few works developed applicable control algorithms for large number of traffic signal lights for LSUN. Dealing with large number of signalized intersections increases the computational burden, which raises real-time calculation and implementation issues. To address these issues, recent works proposed distributed traffic controllers, where each traffic control law is based only on local information at isolated intersections, see e.g. [13–15]. On the one hand, the distributed control is computationally fast, but on the other hand it might produce suboptimal solutions given only local information. Another option, which was proposed to address scalability, is to utilize macroscopic aggregated models, e.g. based on macroscopic fundamental di-

agram (MFD), which can introduce less complex dynamics. However, MFD is mainly used to manipulate part of the signal lights, i.e. the ones on the perimeter of the urban region [16].

Nonlinearity. While dealing with LSUN, the traffic flow modeling is a more complex task if it is compared with isolated intersections, due to spatial and temporal correlation with various constraints, see e.g. [12]. Until now, most studies that implemented control theory approaches utilize simplified linear models or nonlinear models, where usually the nonlinearities in the model are treated by linearization techniques, rendering nonlinear models to linear ones, e.g. the store-and-forward model in [5, 12]. Therefore, a more realistic and enhanced model should be developed. Such a model should also account for unexpected events such as accidents, jams, queue spillbacks or partial gridlocks, weather effects, and long-term variation such as seasons and reconstructions.

Big Data. Previous studies were developed based on the fact that a limited number of sensors and usually one type of data, such as inductive loops, can be utilized in real time control. However, nowadays there are more sources of data [17], e.g. radars, Bluetooth sensors, probes, GPS and WiFi trackers, video cameras, smart cards on subways and buses, online ride-hailing system, and more. This data consists of complex non-linear spatio-temporal dependencies of traffic events, e.g. different vehicle types, different road types, and time-varying demands (such as accidents, traffic jams). The various sources can be utilized to operate the traffic network system, thus might improve the control performance [18]. An interesting research direction is to integrate textual information from different data sources, as e.g. from social media. The availability of big data in urban transportation systems opens the possibility to apply new data-oriented approaches in TSCP.

Addressing the above issues *together* is not a trivial task. Previous works in control theory tried to tackle these issues, e.g. network-wide traffic control systems via MPC approach [19–22], by a centralized subnetwork controllers or by a hierarchical control structure. Still, they assume fixed pre-defined objective function. Nevertheless, in the past decades a promising approach for extracting accurate information, Deep Learning (DL) which is a part of the general concept of Artificial Intelligence (AI) has been emerged. DL is used to solve different tasks, such as traffic forecasting and traffic control. Recent works that utilized DL in solving TSCP have introduced preliminary results [23, 24], aiming to handle the three main issues mentioned above. However, these works appear to be tailored to fit a given problem, thus being task-specific, i.e. the problem is defined by its objective(s), hence its learning process is limited and specified for this (these) objective(s) only. Moreover, these works do not include textual inputs.

1.1 Research vision

The current research strives to develop an advanced DL model (DLM), which should be integrated in the closed loop of the traffic signal control for LSUN, to handle the three main elements: scalability, nonlinearity, and big data. The developed DLM should merge sensory data from various network sources. Text input data may be processed via Natural Language Processing (NLP) techniques [25], such that communication with the DLM will include consequential learning of concepts and operations, enabling cognitive activities such as abstracting simple concepts into a more general ones. Moreover, a dictating form of communication will be introduced in the DLM, requesting it to perform different tasks, after accumulating enough knowledge. Tasks such as detecting a forming congestion, maximizing total throughput in the network, detecting incidents and other objectives.

The designed model will hopefully yield highly-cognitive spatio-temporal pattern recognition, via hierarchical abstracting mechanism, see Chapter 6.3. Thus, allowing the developed model to be general and more autonomous, e.g. predicting the forming of congestion by its early stage and regulating network flow accordingly. This approach will enhance the network performance, thanks to a long-term up to short-term foreseeing control.

1.2 Research motivation

The general motivation is to explore recent advances in data-driven theories and approaches, and to analyze the possible improvements which would be gained from applying these to the TSCP for LSUN.

The idea is to develop an automatic urban traffic management system, which will be based on integrating data-driven methodology components within a control theory framework, having the ability to *learn* the data. We aim to seize the futuristic potential to develop a system which would understand the data at a high human-like level. Consequently, it would process the data beyond the mere prediction tasks. It will be able also to analyze, plan, derive conclusions, and control.

One can roughly distinguish between two different solution approaches based on control theory: (i) general developed tools (e.g. MPC, IC), which can be applied for the considered application, and (ii) developing application-oriented algorithms, i.e. considering and studying some specific data sources for the considered application problem and tailoring solution algorithms by adaptation and renovation.

Usually, the first approach lacks the familiarity with traffic data, hence

solution is searched blindly and the applied method is mostly effective only for limited cases (e.g. linearized models with uncertainties), while the second approach might be more effective, as it should be oriented to the application, but usually only for specific considered traffic scenarios. Therefore, in this research, the AI is advocated for its potential both to take into account the data, and also its promise to generalize, i.e. first learning the data for some specific tasks, and then perhaps at an advance stage of AI, using it for new tasks, via cognitive processing such as analyze, planning, etc.

Currently, most Machine-Learning (ML) models of urban traffic control are designed for single tasks, or multi-tasking with specific tasks. Any new task or type of data will usually require retraining the model. However, DL has the ability to generalize learned patterns for unseen data, and perhaps also over the new tasks. E.g. transfer learning in DL is about applying the same neural network that was trained over some tasks on new tasks by training over these new tasks but updating only the classification layers, while freezing all lower layers. Or Meta-learning, which is an approach where learning from previous tasks assists in learning and adapting to new unseen tasks.

1.3 Research goal

The main goal of this research is to develop an advanced deep learning model for TSCP in LSUN, which can construct an automatic traffic management system. In such system, the human operator can supply task orders via textual commands, and the system realizes them, based on big data, via its traffic control measures. See demonstration of such system in Fig. 1.1.

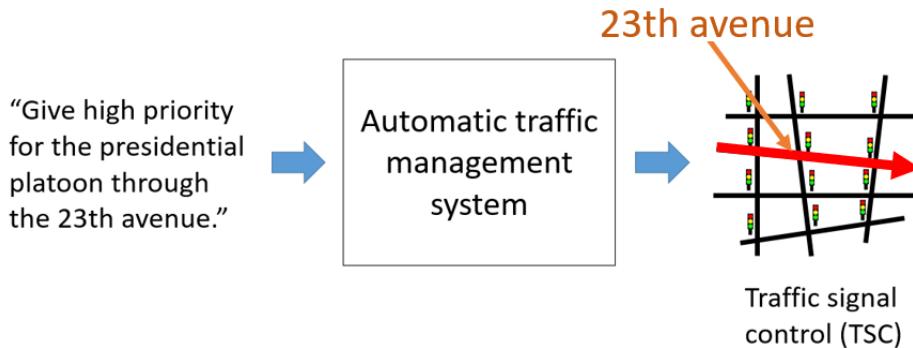


Figure 1.1: Automatic traffic management system.

Several of the current control approaches [12] are limited by not being dependent on data, especially online data, hence, the models used to repre-

sent such data are less accurate in describing them dynamically. In addition, most of these approaches are limited by the scalability issue, as they usually require enormous computational efforts, resulting in the need to simplify the models, many times by linearization, which naturally cannot handle non-linearity. The DLM, which will be developed in this research, should deal with these issues via using Deep Neural Networks (DNNs) [26] as building blocks, which in turn will allow non-linearity representation of the traffic network system.

Moreover, unlike previous works that introduce AI methods in TSCP, the developed Deep Learning model will be integrated within the closed loop of the traffic signal control for LSUN.

If we observe the traffic system in Haifa, Israel, for example, it operates using actuated signal plans, which are independently managed for each intersection. ‘Actuated’ means that each intersection is equipped with sensors, such as loop detectors, and the signal plans are dynamically adjusted based on data from these sensors. Each intersection has a set of signal plans to choose from. Certain plans are selected at specific pre-scheduled times, such as during the morning peak, evening peak, holidays, weekends, etc. This routine mode of operation can be regarded as the system’s regular functioning. The main control system continuously monitors the traffic flow across the network. In the event of a breakdown or anomaly, the operator or supervisor can override the actuated signal plan and activate a different signal plan from a pre-designed set of special plans, such as those for police, emergency situations, roadwork, etc.

Other systems operate in different cities and regions around the world. Our proposed automatic traffic management system can be viewed as an extension of the Haifa system, for example. On the one hand, it can still monitor traffic flow and respond to anomalies accordingly. On the other hand, during regular operation, it offers the operator significantly more flexibility to automatically manage different scenarios that Haifa’s system cannot handle without proper re-planning (i.e. redesigning and recalibrating). In Haifa’s system, when the operator does not intervene, the signal plans at each intersection are typically designed for specific demand levels, with each intersection optimized for a common objective, such as minimizing total delay, equally weighted across all lanes. However, in our system, intersections are managed with specific prioritization, allowing the system to handle a variety of possible scenarios, each representing different objectives. Further details on such scenarios are provided in Section 3.1.1.

1.4 Scientific background

In the following, traffic-related studies are reviewed. Mainly, ML works and other AI works in TSCP are discussed, then a brief review on the textual processing field is presented.

ML has three types of learning: Supervised Learning (SL), Unsupervised Learning (UL), and Reinforcement Learning (RL). SL is learning from a given dataset of examples, each containing both input and output, where the output is a label corresponding to a specific input. UL on the other hand learns the distribution of inputs, because the only given in the dataset are examples with inputs only. Similarly, RL holds examples of inputs. But it also includes feedback from applying the output in the environment. The input in the RL is referred to as state, and the different outputs are different possible actions that can be applied given some input/state, in some environment.

Note that AI is an enormous field, while ML is only a sub-topic in it. And in ML among other methods and topics, Deep Learning is one of ML methods.

1.4.1 Machine-Learning-based works in TSCP research

First studies of AI in traffic control utilized rule-based methods [27], such as Knowledge-Based Systems [28, 29], heuristics and constraint satisfaction programming [30], all of which are human-based design, e.g. experts are required for feature selection. For the purpose of replacing rule-based design [31] with an automated design, an ML was introduced [32]. ML is implemented by many methods [33], e.g. Neural Network (NN), regression, decision trees, and multi-agent systems [34]. ML also includes deep learning of features, e.g. Deep NN (DNN) [26], Deep Reinforcement Learning (DRL) [35], and Deep Belief Network (DBN) [36].

DNNs comprise of layers of simple computational units (neurons). Each of those units receives as input some feature or a sum of weighted features from previous layer, and applies some non-linear function upon this sum.

The ML-based research in TSCP focuses on traffic prediction, e.g. [37, 38], traffic regulation, e.g. [39, 40], and traffic modeling, e.g. [41].

Reinforcement Learning in TSCP research

More recent studies in TSCP were based on RL approach since labeled data about the performances of the signal timing plans, e.g. how poor or good in decreasing delays, is unavailable for supervised learning. Hence, most papers [42] take actions to change the signal plans and then learn from the

outcomes. Other studies, as reviewed in [40], use Deep RL via some DNN, where the inputs are the state measured by sensors (such as speed, queue length, etc.) and the outputs are the actions, i.e. traffic signal decisions. Different rewards were presented in the literature, e.g. a cumulative waiting time difference between two cycles, travel times, queue lengths, throughput, etc. In [41,43], the Q-learning RL method, which uses intersection signal control and vehicles as multiple agents that inter-connect between themselves, was implemented. The agents also learn independent behavior through a trial-and-error experimentation combined with reward feedback. Results show that communication between agents can be limited to the neighboring ones [44], if a faster convergence policy is sought. However, most studies implemented the Q-learning method on isolated intersections. Moreover, one of the limitations of RL is the very strict set of actions and states [45], which results from the curse of dimensionality (COD) problem [41], i.e. it requires categorization of continuous actions into discrete and finite action ranges.

Supervised learning

Other works, [46,47], applied supervised learning NN and its variants, such as Convolutional NN (CNN), Recurrent NN (RNN), and Neuro-Fuzzy Inference System (NFIS) for traffic prediction. There are studies which use NN for traffic congestion prediction [48], or to replace traffic simulator [49] by training on simulation parameters (as inputs) and their outcomes (as outputs).

The CNN is built as a cascade of two successive types of layers: (i) convolution layers acting as filters on a 2D input, and (ii) pooling layers that reduce previous convolution layer's dimensions [31], while the RNN has also backward connections in the network to hold the history of past processing, which enable it to learn time-dependent data. Due to CNN and RNN properties, i.e. CNN is powerful for spatial data and RNN is best for temporal data [23], various combinations were suggested for dealing with spatio-temporal data.

In [24], traffic forecasting was performed by CNN and RNN connected in series, where the input is an image converted from speed data, inserted to CNN for spatial feature learning, and then their output is inserted into long short-term memory (LSTM) neural networks for temporal feature learning. Results show better performance both for short-term and long-term traffic predictions. The LSTM networks are special case of RNNs, which are used often since they are capable to handle long-term time series prediction. LSTM was used for speed prediction in [50] and for traffic flow prediction in [51], while CNN was used for speed prediction in [52] and for various parameter predictions in [53]. In [23], the CNN and RNN are used in parallel, where their inputs retrieved from attention model which determines the importance

of past traffic flow data and its effects on the future traffic flow.

As shown from the above studies [23,24,50,51,53], CNN and RNN present better prediction results in accuracy and stability compared to other methods, such as support-vector-machines (SVMs) and random forest. However, this might be due to being tailored to the specific problem, which makes them best due to specificity and particular planning. Apparently, CNN and RNN are examples of simplification of the general NN, since they are more intuitive and understandable, with the price of being task-specific [23]. Therefore, to yield a more general and flexible model, designing DLM is preferred.

Even though NNs are powerful for prediction, there are various ways to utilize it for control. E.g., using an approximation of the loss functions and/or labels via supervised learning, such as in [39,54]. The simultaneous perturbation stochastic approximation (SPSA) algorithm is used, where the output is given by an objective function which is system-wide measure of effectiveness (MOE). In [39] MOE is used for full system control, while in [54] it is implemented for each agent in a multi-agent system. Note that utilizing single-output modeling for control is problematic, since a full retraining is needed if the objective (e.g. MOE) is changed. Therefore, a multiple output NN should be used, as it demonstrates better results, see [55].

Unsupervised learning

Different learning models, apart from most famous back-propagating DNNs, are Deep Belief Network (DBN) and Stacked Auto-Encoder (SAE). DBN is a deep NN composed of a stack of Restricted Boltzmann Machine (RBM) networks, while SAE consists of multiple layers of autoencoders. Both DBNs and SAEs use unsupervised training of the first couple of layers, then they continue with the same training for the second couple of layers, and so forth. Finally, a predictor, a fully connected layer, on the top layer is used for prediction. The system can be trained for high performance, by using a small amount of examples. In [38], the DBN was used for traffic flow prediction with spatial and temporal data as its input. In [37], the SAE was used to learn a hierarchical feature representation of human mobility, for the purpose of efficient prediction of traffic accident risk level. In [56], an automatic incident detection is performed over a spatio-temporal data extracted from Bluetooth (BT) sensors on the Ayalon highway, using regular DNNs with attention mechanism and via encoder-decoder structure.

1.4.2 Other AI works in TSCP research

Additional studies in the AI field with cognitive architectures in TSCP research were reviewed.

Previous studies [10, 11] tried to implement AI based on distributed *cognitive* architectures [57]. The study in [10] is based on Global Workspace Theory (GWT). In GWT, the agents are local controllers behaving reactively, and compete with each other over access to a working memory (WM), in order to define which of them is experiencing the most critical traffic situation. GWT contains agents imitating four types of brain functions: (i) sensory type, which holds positions and velocities of vehicles; (ii) behavioral type, which determines the light of the traffic signal in a particular intersection; (iii) consciousness type, which represents the working memory and interacts with other brain functions; and (iv) motor type, which executes the chosen signal phase for each intersection.

In [11], an AI was implemented on a single intersection using a Multi-purpose Enhanced Cognitive Architecture (MECA) [58], which was adapted for TSCP. The design consists of cognitive manager, a special kind of agent managing a set of physical objects, providing information about themselves and receive commands, e.g. car and intersection agents. MECA is composed out of two independent systems that communicate with each other: (i) a fast reactive system that holds the input sensors and output actuators, which is suited for normal situations, and (ii) a motivational system which is goal-oriented and suited for unexpected situations.

Both papers [10, 11] show improvements in the overall mean travel time of vehicles in the network, compared to fixed timing plan controller, under different conditions. It is thanks to automatic reactive and the conscious elements, which together imply an intelligent behavior. The papers use simple cases of small scale networks, therefore they provide only a proof-of-concept.

1.4.3 Traffic event detection via text data

In the following, different traffic-related works performed in textual domain, mostly related to event classification, are presented.

There are several definitions for an “event”, one of them [59–61] defines event besides being a significant thing that happened at some specific time and place, also as the magnitude of people’s conversation about the matter, and as a dynamical process in time.

Event detection can be either learned via supervised or unsupervised methods [62], e.g. clustering can be determined either by topic [63], or by spatial and temporal attributes [61] where dynamic clustering is used, as if a

cluster is consistent for a period of time it is referred to as an event. Clustering also requires the definition of similarity metric, e.g. content and distance similarities. Most papers use ML [61, 63, 64, 64–68] and not DL, hence require manual feature engineering.

Most frequent ML techniques for classification in these papers are: SVM, Latent Dirichlet Allocation (LDA) and Radial basis function (RBF). Other techniques were also used, such as Decision Tree, Naive Bayes classifier, Multilayer Perceptron (MLP), Logistic Regression and others, [62]. These studies use NLP to process the text and convert it into machine understandable language, in order to use it for a specific task, e.g. anomaly detection. Classification can be either binary, e.g. classifying whether the tweet is traffic-related or not [67], or it can be multi-class classification [64, 65], i.e. classifying into different event categories.

It should be noted that besides extracting features from tweets, there are also lots of pre- and post-data processings, such as filtering and constructing of data [66], or fetching the tweets by tokenization, i.e. transforming a stream of characters into a stream of meaningful tokens [64, 65]. Tasks other than event classification include for example: (i) extracting tweet locations [64, 65, 69]; (ii) sentiment analysis to allocate a positive (“good” traffic) or negative (e.g. due to complaints about traffic and roadworks) polarity and its strength, representing the level of stress [65]; (iii) predicting clearance time [63] - the period between incident reporting and road clearance; (iv) predicting a road status [68] (continuous value from free-flow up to jam), to help drivers reroute their trajectories to be more efficient.

DL research in text classification is a bit richer comparing to ML. The first step in feature extraction [70] is word embedding, i.e. converting the words into some numerical representation. Also, sometimes instead of text being classified by single words, it uses combinations of words, specifically pairs of them [71, 72], to establish a better context. Different DL methods are implemented, e.g. DNN and RNN in [70], or DBN and LSTM in [72], which show better performances compared to non-DL methods: SVM, LDA. Moreover, [72] also discusses how detector solely dependence for data retrieval is problematic due to frequent malfunctioning, hence the need for other data resources as complementary is required, e.g. text from social media.

In conclusion, the above mentioned papers solve classification problems, classifying textual input into events. Note that in our research, the goal from utilizing text data is different than previous works. While the inputs in mentioned works are informational texts, in this research we aim to use instructional task texts as inputs, and the outputs will be used as actions, rather than a classification result. Therefore, the developed model must be different, since we do not merely classify, which is a process of encoding the

input text features into higher features for classification, but we also attach a decoder to the end of this encoder, to use these higher features to eventually execute actions.

1.5 Research contributions

As described in the scientific background Section 1.4, ML research in TSCP has replaced rule-based systems for traffic prediction, regulation and modeling tasks. Specialized NNs, such as CNNs and RNNs, have been developed to excel in particular tasks. Additionally, AI cognitive architectures and various other ML and deep learning frameworks have been developed, primarily for event classification tasks in NLP.

While these studies have shown success in solving specific tasks, our approach takes a more comprehensive view. We aim for a broader, less specialized perspective in tackling TSCP and strive to incorporate continual learning wherever possible. To this end, we explore diverse AI concepts that have not yet been applied to TSCP. Subsequently, we have devised an overall system that is general enough to encompasses a wide range of tasks, which are typically addressed individually.

In this research, an automatic traffic urban management system is developed. The new system is designed based on AI, specifically on DL.

First of all, a simple version of such system is developed via Graph Neural Network (GNN), and it is trained through supervised learning. Then, a more general version of such system is combined by a textual processing module (a GPT) and a closed-loop RL module, to convert textual commands to a traffic signal control.

Secondly, within the developed framework, multiple sources of data are utilized, to obtain different features representing the state of the network, e.g. travel times, speed, density, queue length, and more.

Finally, the developed automatic traffic urban management system is expected to relieve congestion, and serve as a support tool for traffic operators, or perhaps some day even replace them. It can for example prefer also some regions over others, e.g. as it is implemented in perimeter control.

1.6 Methodology

In this section, we describe the processes leading us to the different parts of our research. Specifically, we split our research to four steps, and in the following we describe each of these steps.

1.6.1 Step 1

We start from developing a simple model demonstrating the automatic management system. It uses a GNN model. Next, we proceed with planning of how to extend this simple model to represent a broader scope of commands, thus ending up with RL method applied in a traffic simulation. We continue with connecting this system to our original goal: supplying the commands in their natural form, i.e. text. Consequently, we develop another system to convert text commands to the input of the previous RL system: preference values. Naturally, the next step is to combine these two systems, to realize the original goal of this research: an automatic management system.

We first describe the implementation of our problem using GNNs. Note that it is important to represent data in its most efficient and natural way. E.g. one should not input to NN a 1D flatten data of an image to be classified, because then it will ruin the spatial relationships information embedded in the data. Similarly, graphical data should keep its structure.

Because there are no studies with our specific traffic problem to compare with, we construct a benchmark model, a simplified one, to compare with other advanced models in the future. The proposed simplified model is shown in Fig. 1.2(a). It is a simple encoder-decoder system, with end-to-end training. However, even with such a simplified model, we have a problem of fusing external non-graphical input, the numerical commands, to a graphical model without ruining the graphical decoder to the same graph structure.

Therefore, as a basic model to start with, we propose to embed the numerical commands as an additional feature in the graph input, which also can be treated as additional layer in CNN. In other words, we embed node-related-commands only into existing graph data to yield also node-related-executions, in our case green times for the corresponding approach in the specific road section, see Fig. 1.2(b). This model has input features, in our case are the velocity at each road section. In addition, we include commands in the form of signal plan IDs, and the output features are average green times for some period.

Our data is spatio-temporal, since it involves spatial distribution of intersections and road sections in a transportation network, and the data features are changing with time. Spatial-temporal (ST) GNNs handle this type of ST graph data, where node input features change dynamically over time, and enter the model as multiple graphs, each representing a single time step, where graphs may have varying sizes.

Graphical information in our case is retrieved from different sensors spread in road intersections in the transportation network. This information consists of nodes and links, and expressed via two elements: adjacency matrix,

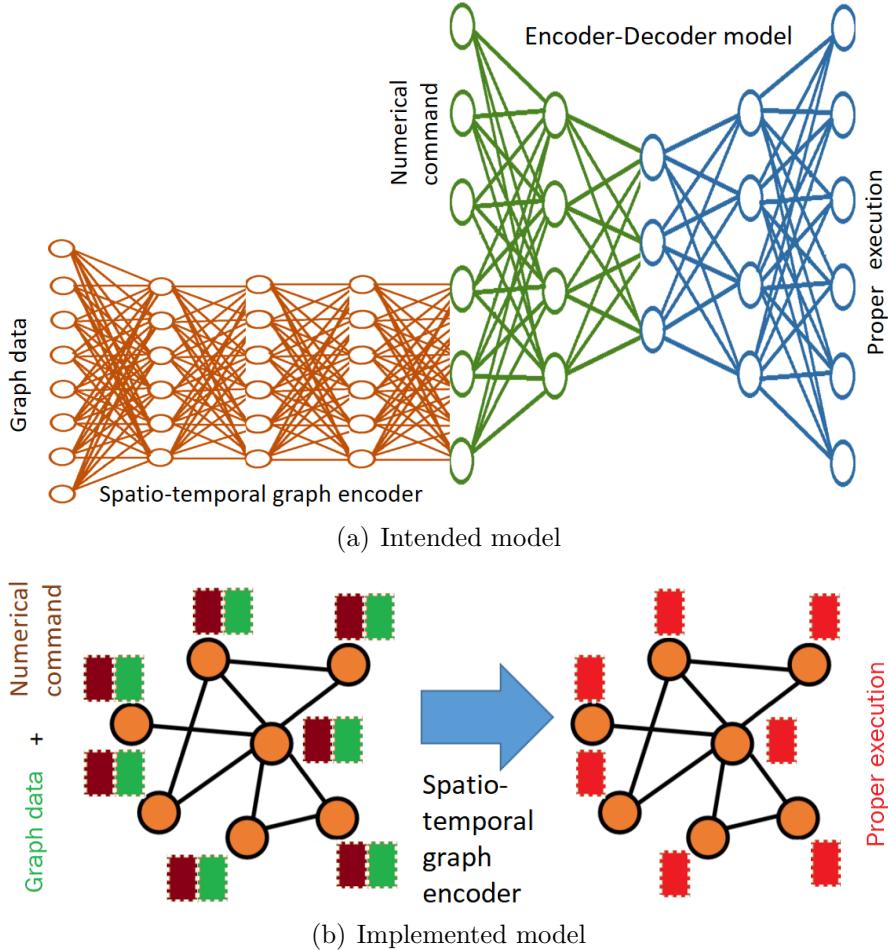


Figure 1.2: Deep graphical models for a simple automatic management system.

sized $\# \text{nodes} \times \# \text{nodes}$ ¹, that contains information about which links are connected to which nodes, and feature matrix, sized $\# \text{nodes} \times \# \text{attributes}$, that consists of feature vectors for every node in the graph, describing it.

Next, we utilize supervised learning on a dataset in which each example composed of (input, output)=(command+additional information, execution). Specifically, we use BT data, instead of satellite/drone intersection images as sensory input in our DLM. It is much easier and lighter computationally. We decided to process BT data via GNN. Eventually, the used dataset was comprised of inputs = (sensory input=BT node features, commands=SPs) and output=(GTs). In other words, this step imple-

¹# denotes “number of”

ments the simplest task of producing green time durations in all intersections in the network, given a pre-defined signal plan IDs. The implementation and investigation of the model depicted in Fig. 1.2(b) is discussed thoroughly in Chapter 2.

1.6.2 Step 2

However, the dataset, which is used in Step 1, contains low level commands, i.e. SPs that are determined via the minimum delay objective in an intersection, as different SPs are determined for different set of demands. This command type can be extended to a broader type, i.e. preference commands, where the minimum delay objective is weighted differently according to some preferences defined by the user.

Due to lack of dataset for this purpose, we use DRL as an ML tool to handle preference commands. We start by simulation training of different DLMs as DRL models, till we accomplished this preference task. See Chapter 3. It is done via RL utilizing a DDQN model, which converts the preference values for the movements to a traffic phase signal in each intersection in the network. The RL is implemented via multi-agent structure, where each intersection is considered an agent. Most basic DLM we use is the fully-connected NN, which is a dense type of NN, where all neurons at consecutive layers are connected between layers. Later, we improve the performance of the RL, by including cooperation among agents. We try different methods of cooperation, where the best one turns to be cooperation via ST GNN as the one used in Chapter 2.

1.6.3 Step 3

In the next step, we concentrate over textual form of instructions, a more natural form to express commands. Hence, while considering the RL process in Chapter 3, our task is to convert textual instructions to preference weights, which the RL model from Chapter 3 can handle and realize via traffic signal control. More specifically, a given text instruction is converted to preference weights over all edges in a graphical representation of network.

We experiment with ChatGPT directly, by describing the problem and handing it few examples (few-shot learning). Then, we improve the performance by designing synthetic dataset specially for our task. This dataset is used to fine-tune an existing language model, e.g. GPT (which the ChatGPT is based on). The difference here, is that we learn from many more examples in GPT than the few ones given in ChatGPT. GPT is a DNN based on

attention mechanism and its task is to predict the next word given an input sequence.

Later, we concentrate on specific type of commands: priority over shortest-path trajectory, which should be deducted by our model from given source and destination nodes in the textual instruction. For that purpose we use different DNNs and different types of GNN. Another option that we examine is iterative type of models, such as Recurrent NNs (RNNs).

1.6.4 Step 4

Eventually, after developing the systems in Chapter 3 and Chapter 4, we combine them into a fully operating traffic management system. This system receives a textual instruction as input, converts it into preference weights over edges, by the system from Chapter 4. Then, these preference weights over edges are converted to preference weights over lanes via set of matching rules, to form the input to the RL system from Chapter 4. This RL system then executes a real-time traffic signal control based on the preference weights, assuming them to be fixed as long as the instruction is not changed.

1.7 Thesis outline

The thesis outline is presented as follows, and in a diagram form, shown in Fig. 1.3. Note that Fig. 1.3 shows only the main inputs in the models developed in all steps. A more detailed view of these steps is shown later on in Fig. 5.1.

Chapter 2 deals with simple types of command of producing GT given SP ID (Fig. 1.3(a)), and then we proceed to more general type of commands - preference commands (Fig. 1.3(b)), in Chapter 3.

Chapter 4 deals with automating the process in Chapter 3, by replacing the tedious manual process of tuning preference weights for each movements in a transportation network, with a system that converts textual instruction into preference weights in their appropriate movements (Fig. 1.3(c)).

Subsequently, we could combine the systems developed in Chapter 3 and Chapter 4. The final system is the desired automatic traffic management system, depicted in Fig. 1.1, that receives instructions in their native form, e.g. text, and execute them in the transportation network, via traffic signal control (Fig. 1.3(d)).

Finally, a novel DLM is proposed in Chapter 6.3 as a future research work, involving hierarchical temporal processing, which enables handling commands with different levels of complexity.

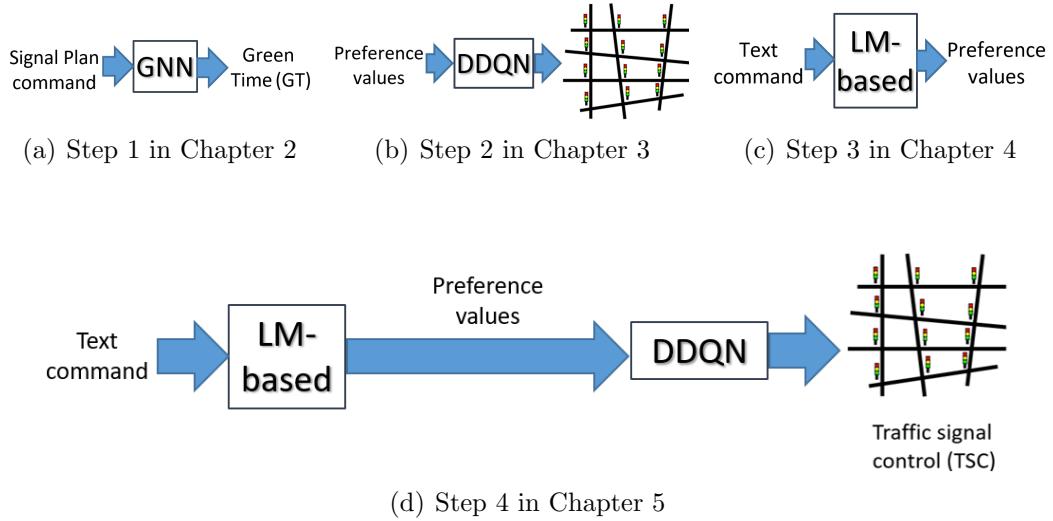


Figure 1.3: The different research steps and their corresponding chapters.

Chapter 2

Spatio-temporal Graph Convolutional Neural Networks for traffic signal control in large-scale urban networks

As was described in Chapter 1, this thesis aims at tackling the traffic signal problem for large-scale networks via a deep learning approach. Our ultimate goal is to construct an automatic traffic management system, where human operators supply commands, and the system realizes them via executing appropriate signal plans (SPs) or green durations in the intersections. The current chapter considers the first step to achieve this goal.

In this chapter, two models that can handle spatio-temporal graphical data are developed based on Graph Convolutional Neural Network. The developed models can be utilized either for traffic prediction tasks or for decision-making, e.g. of green times in intersections, given fixed cycle time steps.

2.1 Problem definition

Let us consider a large urban transportation network which consists of many signalized intersections with short links, experiencing high demand flows with (high probabilities of) queue spillbacks. The network is deployed by (at least one type of) sensors gathering traffic and other information about this network. The sensors can extract information of different types, such as: travel time, speed, number of vehicles. In addition, it is assumed that some data on SP of the signalized intersections in the network is available. Such

data can be used for *command inputs or/and actuators output*. Given a set of features for every link, the goal is to predict some SP features, e.g. SP ID or green time. One can use SP IDs as an explicit type of command in the DL’s *input*, while other features, e.g. speed, as implicit or auxiliary commands. On the other hand, one can use green time durations as the DL’s *output* which is the realization and actuation of the commands.

2.2 Spatio-temporal graphical representation for transportation networks

This chapter is deals with graphical structure of the transportation network. There are various models that handle graph data. Graph Convolution Network (GCN) [73] and Graph Attention Networks (GAT) [74] are strictly spatial models, while RNN, LSTM, temporal convolutional network (TCN), gated recurrent unit (GRU), and Transformer [75] are strictly represent the temporal models. Transformer is the only model that perceives all time steps simultaneously, while others perceive them sequentially, step-by-step. TCN use dilated causal convolution, where each layer in the neural network receives data in different time scales. GAT and Transformer both use attention mechanism to have adaptive (graph) weights depending on the input data. Special case of Transformer is Graph Transformer Network (GTN) which combines the graph structure through adjacency matrix in its layers.

There are vast amount of studies in the topic of traffic signal forecasting by graph representation [76, 77]. Most studies deal with spatio-temporal data, and they process spatial information and temporal information separately [78–81]. Likewise we do in our current research. Eventually, all studies perform fusion of these two channels, either in the mid of their model or at its end, before the prediction unit.

Recent studies utilized the famous Transformer model for different uses, e.g. to process spatial information only [80, 82, 83], temporal information only [79, 81, 84, 85], or both [86, 87]. Other studies used a variation of Transformer model, the GTN [79, 82]. Sometimes, Transformer was not used but the attention mechanism was [88]. [81] also implements a spatial-temporal embedding method. This method integrates spatial positional information (where nodes may be close in Euclidean distance but far in graph), and temporal positional information to know the exact time slot of different inputs. [84] adapts the Transformer model specifically to the traffic forecasting problem, by including 1D causal convolutional sequence embedding and relative position encoding. The former is to insure future data not affecting past

data in the process of prediction, while the latter is to learn automatically the relative position information of time nodes.

Although many studies involve Transformers, some also combine other modules in their DL models, such as CNNs [78, 86, 87], gated temporal convolution layers [82, 88], TCN [83, 89], LSTM [78]. Others combine GNNs, such as GCN [82] and GAT [81, 88]. For example, [78] structure the graph data in a 2D matrix form to use regular Euclidean convolution, instead of the usual graph convolution, as it is applied in this chapter. Others include less famous modules such as SOM [79] and DCRNN [90].

There are different structures of models. Ours and some studies use sequence of spatio-temporal blocks [83, 87]. Others use encoder-decoder structure [81, 84, 85, 90]. Others use seq2seq structure [81, 82, 86, 87] (where the input and the output are sequences), while others implement single output [79, 80, 88].

Some studies claim that the static graph representation do not represent adequately the complexity of the problem. Therefore, many studies try to model a dynamic graph structure. For example, [86] uses multiple graphs while [82] produces multiple graphs. [88] produces dynamic graph structure by a dynamic variation of GAT model (ASTGAT), while [83, 87] do it by Transformer, and [78] produces it by traffic correlation statistics based on historical traffic passenger flows. We, on the other hand, implement static graph, but a multivariate one (multiple features for each node in the graph).

Since as mentioned previously, the given problem is complex, hence studies tried to learn local and global dependencies, both for the spatial and the temporal domains [86]. [80] learns spatio-temporal dependencies jointly, while [81, 88] learns only local spatio-temporal dependencies, and [78] learns local and global dependencies but only temporal ones.

Some studies were set to predict for the short-term [80, 89] while others for the long-term [83, 85, 87, 88]. Our study is mainly built for short-term prediction, however long-term prediction was also tested, but it was not reported since it had worse performance, as expected.

In this chapter, an urban system is realized in a graph form with spatio-temporal (ST) multivariate data. The node input features change dynamically over time, and enter the model as multiple graphs, each representing a single time step. By multivariate we mean multiple traffic variables are considered simultaneously. The graph data examined in this chapter is considered static, i.e. the traffic infrastructure assumed to be fixed. There are two types of approach for dealing with sequenced graphs [91]: (i) recurrent neural network (RNN) solutions that generally apply graph convolutions to aggregate node level temporal information, (ii) convolutional approaches that apply graph convolutions to aggregate nodes, and then 1D convolutions for

temporal level information. There are plenty of ST models in general and many specifically for graphical inputs, [92]. ST models [93, 94] can represent different features of the graph with different relationships, i.e. different adjacency matrices for each layer of information, thus inputting each such sub-graph to separate but identical models, and after some processing, fusing them all together. The graphs can be different e.g. in : (i) spatial proximity or the neighborhood, (ii) functional similarity which encodes the similarity of surrounding places-of-interest (POIs), and (iii) transportation connectivity which encodes the connectivity via roads or between distant regions. Similarly, [95] introduces directional and positional relationships, which are also described through multi-graphs. The model in [96] also captures the non-local spatial correlations besides the usual local ones. [90] has a time-series input, and the model captures the spatial dependency using bidirectional random walks on the graph, and the temporal dependency using the encoder-decoder architecture with scheduled sampling. [97] includes more features for a prediction task, e.g. spatial-only features and external features (weather and date).

2.3 Graph Convolutional Networks

In this section, a general overview on Graph Neural Networks (GNNs) is first presented, followed by description on Graph Convolutional Networks (GCNs) and their algorithms.

GNNs are responsible for feature extraction for graphical input. Graph data in general has irregular structure, compared to special cases of graph data with regular structures: (i) images that have 2D grid structure, and (ii) time series that have 1D sequential structure. Both of these cases are in the Euclidean domain.

Graphs consist of nodes and edges, and their graphical information can be expressed via two elements: (i) *an adjacency matrix* $A \in \mathbb{R}^{n \times n}$ that contains information about which nodes are connected to each other, where n is the number of nodes; (ii) *a feature matrix* $H \in \mathbb{R}^{n \times d}$ that consists of feature vectors for every node in the graph, where d is the number of features. GNNs use these two elements to process data for any prediction task, i.e. node prediction, link prediction, and graph prediction.

Adjacency matrix in general represents the strength of relationships between the nodes in the graph, hence due to occasionally a lack of this prior knowledge it is hard to decide upon the best one. It can be unweighted, e.g. “1” for every pair of nodes having link connecting them and “0” otherwise. It can be weighted or normalized, e.g. normalized by the distance between

nodes [96].

Besides direct connections between nearby nodes, adjacency matrix can also represent connections between indirect nodes, some k nodes farther away, i.e. k -neighbor nodes are represented by A^k [98]. Relationships can be also captured by the similarity among nodes, which is a function of the node features. Features such as nearby POIs, speed limits, number of link lanes, length, one or two ways, is it a bridge, different road types, road conditions, vehicle density, population density, etc, [96, 99, 100]. It is not a trivial task to a priori determine appropriate adjacency matrix, due to a lack of prior knowledge. Adjacency matrix can also change with time, hence some methods use adaptive adjacency matrices [92, 101, 102].

The GNNs that will be considered are equivalent to the CNNs for images, only on graphs - GCNs, which aggregate information from neighbors of each node. There are several options of aggregation: sum, average (simple or weighted), maximum, etc. This process is the 1st order. Then, the 2nd order is aggregating from neighbors of neighbors, and so on. The GCNs can aggregate information at two levels: (i) “local” information from neighbors of each node, and (ii) “deep” information from neighbors of neighbors. It should not be too deep for a given graph network, just enough to reach most nodes. The depth notion in convolutional networks, either in images or graphs, is different from the regular depth in NNs, since here it is about aggregating from neighbors (like field of view), so its effectiveness is limited (it should not need to be too deep [100]). Also, the aggregating may mix the features of vertices from different clusters and make them indistinguishable. I.e. the features of vertices within each connected component of the graph might converge to the same values. Additionally, unlike 2D images input to CNNs, graph input to GCNs is not dependent on the size or the structure of the graph. The input can be changed in structure, while keeping the convolutional operator unchanged. This property is useful for practical use in urban traffic systems.

However, note that the scalability mentioned in Chapter 1 applies only to inference in DL, not to the training phase. During training, we encounter other well-known scalability issues common to all DL models. First, the vanishing gradient problem occurs when using a large number of layers (deeper NN). Second, the COD arrises: as the input dimension increases, a larger dataset is required to achieve the same performance.

GCNs can be implemented by one of the two approaches: (i) *spatial*, where convolution is defined as propagating messages from nodes along graph edges; and (ii) *spectral*, where convolution is defined as filters in Fourier space. In the following, we briefly describe spatial and spectral GCNs. The reader can refer to [103, 104], and others, for more detailed information.

2.3.1 Spatial GCNs

The spatial GCNs perform aggregation of neighboring nodes, see [103]. Let us denote the input feature matrix by $X \in \mathbb{R}^{n \times d}$ and the weight matrix for the l -th GCN layer by $W^{(l)} \in \mathbb{R}^{d \times d}$. Then, the feature matrix for layer $l + 1$, i.e. $H^{(l+1)} \in \mathbb{R}^{n \times d}$, is computed by a non-linear activation function $\sigma(\cdot)$ over the feature matrix of the previous layer l , i.e. $H^{(l)}$, the adjacency matrix A , and the weight matrix $W^{(l)}$, as follows

$$H^{(l+1)} = \sigma(AH^{(l)}W^{(l)}) , \quad (2.1)$$

with $H^{(0)} = X$, where $\sigma(\cdot)$ can be e.g. ReLU or max functions [31, 103]. The aggregation executed on node features $H^{(l)}$ is simply the sum of their neighbors' features $AH^{(l)}$, see (2.1).

This is probably the simplest aggregation form, while a few other forms exist in the literature. E.g., one can include the features of the node itself in the aggregation, by adding an identity matrix to A , $\hat{A} = A + I_n$. Then, one can even normalize \hat{A} to prevent scaling issues between layers, via multiplying \hat{A} by the inverse of \hat{D} , where \hat{D} is a diagonal matrix and defined as the number of connections each node has, i.e. $\hat{D}_{i,i} = \sum_j \hat{A}_{i,j}$. Finally, to make the normalization symmetric, one can multiply \hat{A} by the square root of \hat{D} from both its sides, which ends with

$$H^{(l+1)} = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) . \quad (2.2)$$

2.3.2 Spectral GCNs

As in the spatial case, our purpose is to aggregate information from neighboring nodes. The general way to do so is by convolution. Convolution operation is defined as $H^{(l+1)} = H^{(l)} * W^{(l)}$. Generally, the convolution can be defined as transforming the feature matrix $H^{(l)}$ and the filter $W^{(l)}$ into Fourier domain, and performing element-wise multiplication between them in the Fourier space, then finally applying inverse Fourier transform to convert the result back to spatial domain:

$$H^{(l+1)} = H^{(l)} * W^{(l)} = \mathcal{F}^{-1}(\mathcal{F}(W^{(l)}) \odot \mathcal{F}(H^{(l)})) , \quad (2.3)$$

where \mathcal{F} is the Fourier transform, \mathcal{F}^{-1} is the inverse Fourier transform, and \odot is the element-wise Hadamard product. In applying Fourier Transform, we transform the signal H into frequency domain, then the element-wise product becomes matrix multiplication of the signal H with a special matrix

W (basis). This basis assumes a regular grid or the signal being in Euclidean domain, e.g. 1D sequential data (audio, text) or 2D data (images). Or in other words, it assumes that the order of the filter between nodes is kept fixed. Thus we cannot use it for irregular graphs, that resides in non-Euclidean domain, where there is no ordering of the nodes. Instead, we use a more general basis, which is eigenvectors derived from eigen-decomposition of the Laplacian, that is described next. For this purpose, we use a core operator from spectral graph theory, to calculate this element-wise product in the form of matrix multiplication. This operator is called graph Laplacian L , and it can be interpreted as a measure of the graph's smoothness (or the difference between the local value of a node and its neighborhood average value). Intuitively, the graph Laplacian shows in what directions and how smoothly the “energy” will diffuse over a graph if we put some “potential” in node i . The graph Laplacian L is:

$$L = I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}, \quad D_{i,i} = \sum_j A_{i,j}, \quad (2.4)$$

while the unnormalized graph Laplacian is $D - A$, and turns to normalized by multiplying with $D^{-\frac{1}{2}}$ from right and left sides. L is the representation of a graph, and can be regarded as an adjacency matrix A normalized in a special way.

The algebraic transformation from element-wise product in (2.3) to matrix multiplication is done by eigen-decomposition of the graph Laplacian L over Fourier functions, where the eigenvalues represent the spectrum of the graph.

In general, spectral analysis means decomposing a signal/audio/image/-graph into a combination (usually a sum) of simple elements (wavelets for audio, graphlets for graphs). These simple elements are usually orthogonal, i.e. mutually linearly independent, and therefore form a basis. When the decomposition is done by implying Fourier transform, it restricts the basis to consist of elementary sine and cosine waves of different frequencies, so that we can represent our signal/image as a linear combination of these waves¹.

We start with formalizing the eigen-decomposition of graph Laplacian:

$$L = I_n - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^T, \quad (2.5)$$

where the Λ is a diagonal matrix with Laplacian eigenvalues λ_t ; and U consists of the appropriate eigen-vectors u_t , which satisfy $L u_t = \lambda_t u_t \forall t$, and are

¹This method is used for compression of files, e.g. audio files such as MP3 and image files such as JPEG, by storing only the dominant frequencies in the data.

a set of Fourier functions basis onto which we project the graph Laplacian. Then, one can develop (2.3) further as follows

$$\begin{aligned}
H^{(l+1)} &= \mathcal{F}^{-1} (\mathcal{F}(W^{(l)}) \odot \mathcal{F}(H^{(l)})) = U \left(\widehat{W^{(l)}} \odot U^T H^{(l)} \right) = \\
&= U \left(\widehat{W^{(l)}}(\Lambda) U^T H^{(l)} \right) = U \widehat{W^{(l)}}(\Lambda) U^T H^{(l)} = \\
&= \widehat{W^{(l)}}(U \Lambda U^T) H^{(l)} = \widehat{W^{(l)}}(L) H^{(l)},
\end{aligned} \tag{2.6}$$

where for any signal x the graph Fourier transform is defined as $\widehat{x} = \mathcal{F}(x) = U^T x$ and the inverse Fourier transform is defined as $x = \mathcal{F}^{-1}(\widehat{x}) = U \widehat{x}$, hence we denote $\widehat{W^{(l)}} = U^T W^{(l)}$, where $\widehat{W^{(l)}}(\cdot)$ is the transfer function of the filter $W^{(l)}$. As seen, this spectral function is applied on L and then multiplied by $H^{(l)}$. Note that $\widehat{W^{(l)}}$ is shift-invariant in Euclidean domain but not in graphs (it has no circulant structure). Also, the filter coefficients depend on the basis, hence changing the graph requires recalculation of the decomposition.

The spectral filter $\widehat{W^{(l)}}$ has θ parameters, and defined as $\widehat{W^{(l)}}(\Lambda) = \text{diag}(W^{(l)}(\lambda_t)) = \text{diag}(\theta)$, where the parameters here are the Fourier coefficients.

In conclusion, the signal is transformed via $U^T H^{(l)}$ to spectral domain, then adjust its amplitude by a diagonal filter $\text{diag}(\theta) U^T H^{(l)}$, and finally perform inverse Fourier transform to return to spatial domain:

$$H^{(l+1)} = U \text{diag}(\theta) U^T H^{(l)}. \tag{2.7}$$

For more information see e.g. [104].

In [104], it is shown that even though the spectral formulation uses different tools, specifically via a first-order approximation of localized spectral filters on graphs, it ends up to the same equation (2.2) as in spatial development, refer also to [91]. However, spectral eigen-decomposition is computationally heavy, hence a 1st order approximation or a Chebyshev polynomials expansion up to some order can be preformed, to finally derive (2.2). Note that (2.2) is derived after applying non-linearity over the defined spectral convolution.

It is shown in [104] that the computational complexity of the $\text{diag}(\theta)$ filter is exponential $O(n^2)$, hence it is inefficient. It is because computations are done over entire graph. Hence, we should make the filters smaller or more local. Therefore, we replace the spectral filter by polynomial parametrization of degree $K \ll n$, where K is the kernel size of graph convolution, which determines the maximum radius of the convolution from central nodes. The replacement results with: $\widehat{W^{(l)}}(\Lambda) \approx \sum_{k=0}^{K-1} \theta_k \Lambda^k$, or in our case Chebyshev

polynomial $\widehat{W^{(l)}}(\Lambda) \approx \sum_{k=0}^{K-1} \theta_k T_k(\Lambda)$, reduces the computational order to a linear one: $O(|\epsilon|K)$, where $|\epsilon|$ is the number of edges in a graph, which holds $|\epsilon| << n^2$, since most of the graphs are sparse. Hence, polynomial filters are exactly K-localized, i.e. $K - 1$ -order neighborhood of each node, e.g. $K = 1$ is equivalent to 3×3 filter in images. Consequently, these filters have the same $O(n)$ complexity per feature map as CNNs, [105].

The final spectral filtering operation as derived from (2.6) is:

$$H^{(l+1)} = \sum_{k=0}^{K-1} \theta_k T_k(\Lambda) H^{(l)}. \quad (2.8)$$

As we will see in Section 2.4.2 we combine spectral convolutions and temporal 1D convolutions, which both have $O(n)$ cost, as we showed above. Hence, this guarantees the scalability property of the algorithm.

2.4 Implementation

In the following, the GNN basic setup is described. Then, two GCN models for large-scale urban traffic networks with ST data are introduced.

2.4.1 GNN setup

Given that the data used in this chapter is extracted from BT detectors deployed in the roads of an urban network, the data structure is graphical. Hence, developing a GNN model seems to be the most appropriate for this kind of data. There are various methods to construct graph from datasets [77]. In this chapter, the nodes of the graph are the road sections and the edges connecting these nodes are the intersections. An illustration of the model inputs $H^{(0)} = X, A$, and the output Y graphical information are shown in Fig. 2.1. The numbers in the matrices represent the following: A matrix contains the values: 1 for edge, and 0 for no edge; H contains two features for every node: signal plan ID and number of vehicles; and Y contains green time duration feature for every node. A and X represent the input to the GNN, visualized as a graph with a structure defined by A and node features defined by X . Y represents the output features, which are also visualized on the same graph structure with node features defined by Y .

The spectral GCN is used in our experiments, as our main model, since (i) the Chebyshev expansion is efficient and (ii) our graph network is fixed during training and testing.

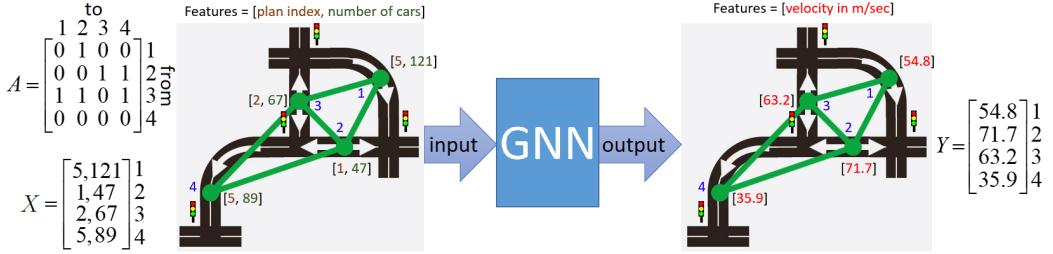


Figure 2.1: GNN for prediction (Y) given the inputs A and X , in road sections.

2.4.2 Spatio-Temporal Graph Convolutional Network (STGCN) models

The STGCN model is a spectral type of GCN that can handle and process ST graph data.

In the following, two STGCN models are introduced. The first model, called STGCN model, is taken from [106], and it is implemented and tested for the Tel Aviv large urban network, see Experiment 1 in Section 2.5. The second model, called an *enhanced* STGCN model, is developed based on the first model, and is implemented and tested on Even Gvirol arterial, see Experiment 2 in Section 2.5. The features that are used in both experiments (and their abbreviations) are as follows. *SP features*: P=signal plan ID, GT=green time; *BT features*: V=speed, nV = number of vehicles, F=Flow, D=Density, TT=Travel Time; and *External features*: m=month, d=day, w=weekday, h=hour.

All SP and external features are measured as integer values. Note that we use continuous regression to predict the GT output; hence, for real application, the output should be rounded. On the other hand, BT features are mostly real values, except for nV, which is an integer. Moreover, the full information extracted from BT sensors consists of the TT and nV features. The remaining features mentioned above are derived from these two features and are provided for the operator's convenience. Specifically, speed is defined as $V = \frac{l}{TT}$, flow as $F = \frac{nV}{TT}$, and density as $D = \frac{nV}{l}$, where l is the road section length. The SP data is stored in an SQL database and contains various data tables. We used the table where each row includes the intersection ID, cycle timestamp (start time), movement ID within the intersection, start time of the green duration for the given movement, green duration, and program number.

We observed that the program number and green durations are completely correlated, making one of them redundant. Subsequently, we used

another table from the SQL file, which contains the different Signal Plans. The table consists of the columns: intersection ID, program number, target destination, and cycle time. The target destination is a valuable alternative to the program number because it consolidates multiple program numbers under one value and represents different objectives, which can correspond to different commands. As a result, we used the target destination column as the SP ID feature. Additionally, the program number has no meaningful significance beyond being an ordinal value, whereas the target destination numeric values represent different SP goals, which are shared among different intersections. Finally, we combined the SP data with the BT data, averaging it in 5-minute intervals.

In the original paper [106], the STGCN model is based on spectral GCNs, and implemented in PyTorch [107]. The STGCN was used for self-supervised short-term prediction of speed feature estimated from loop detector sensors in an urban network. The speed for each sensor was predicted few steps forward based on speeds of a few steps back. Each step is defined as an average speed over 5 minutes. The PeMS traffic dataset [108] from 228 sensors over two months, only on weekdays, in District 7 of California, was used.

Similarly, in our first experiment, we used an ST graph model [106] for the same self-supervision task with speed features as input and output, aggregated over 5 minutes for each step.

Only it was based on a different transportation network with different source (BT detectors). Therefore, calculated average speeds are not time-averaged [109] but spatially-averaged across the whole road section. Note that double loop detectors extract full real-time information from all vehicles passing the detector, while BT detectors can detect only part of the vehicles. In addition, there is missing data from BT detectors, due to malfunctioning or errors. The self-supervision can be considered as a good unsupervised type of learning, unlike clustering which is ambiguous with unknown appropriate number of clusters. Prediction is performed for each node, hence the task is node-classification.

In the second experiment, an enhanced STGCN model, compared with the one in [106], is proposed, see Fig. 2.2. First, while in [106] the feature was only the speed, in the current experiment, more features can be included in the developed model (any combination of them): travel times, number of vehicles, density, flow, green durations and SP IDs. Second, an embedding layer in the input for the P feature is added, since it is a categorical feature which requires transforming it into a continuous feature. It is defined as a one-hot vector (1 for the SP index, and 0 for the rest), before the embedding processing. Lastly, external features are combined with the output ST feature, by first processing them through an embedding layer, then concate-

nating and passing them through a fully-connected layer to yield the final output. This external processing is based on [97].

We used time features (i.e. month, weekday, day and hour) as external features. Several ST models [96, 110, 111] use the same graph only split the input data into separate channels: recent, daily-periodic, and weekly-periodic. However, we utilized time features instead of applying this direct prior of separating data inputs into different time-scales.

In the enhanced STGCN model, the P feature represents the commands. This feature is unique for each node in the graph, hence we embed it in the graph input, see Fig. 2.1. In other words, we embed node-related-commands only into existing graph data to yield also node-related-executions, in our case GTs for the corresponding approach in the specific road section, see Fig. 2.1.

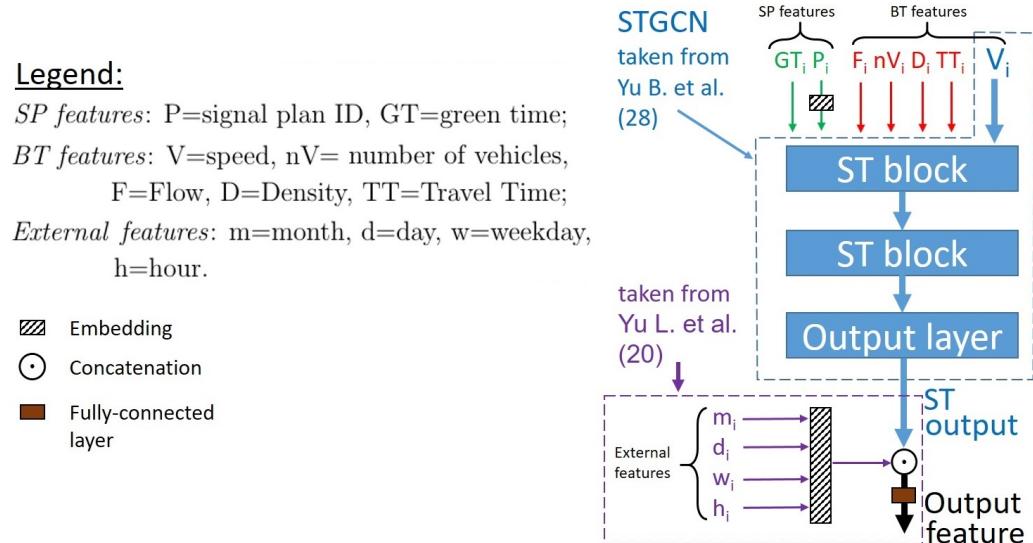


Figure 2.2: An enhanced STGCN model, for each road section i .

2.5 Results

The urban network of Tel Aviv city is considered to conduct two experiments, see Fig. 2.3. The network has 83 intersections, with 171 links. The network is deployed by 83 BT sensors. In addition, the traffic control center of Tel Aviv provided information on SP of the signalized intersections in the network. The SPs are packets of different types of features: their IDs, green time durations, and cycle durations (the two latter ones may be fixed or dynamic).

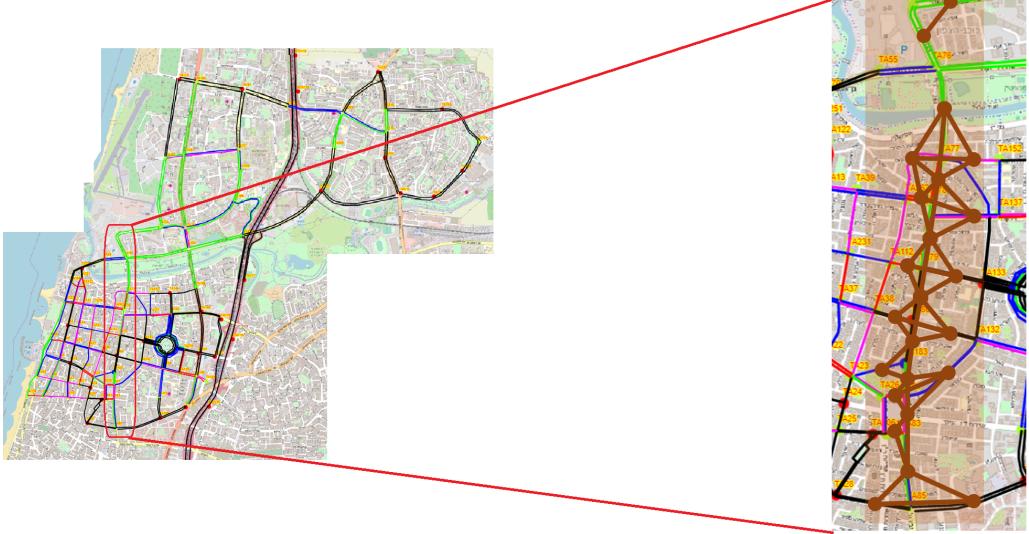


Figure 2.3: Tel Aviv BT sensors map (left) and Even-Gvirol arterial with its graph (right).

In this section, the results of two experiments, Experiment 1 and Experiment 2, are presented and analyzed. Experiment 1 is similar to the one presented in [106], as the time step is a 5-minute interval, and the task is to predict future speed based on past speed values. This speed is averaged over the 5-minute time step period. It is conducted on the full urban network as shown in Fig. 2.3 left. In Experiment 2, the time step is defined as a cycle time, and the task is to predict green duration or speed based on multiple features. This experiment uses the enhanced model developed in this chapter, as described in Section 2.4.2. It is conducted on a fraction of the full urban network, as shown in Fig. 2.3 right.

2.5.1 Evaluation metrics

Dealing with a regression task, the loss function which can be used is the Mean Square Error (MSE) or L2 norm: $L = \frac{1}{n \cdot m} \sum_{i=1}^m \sum_{j=1}^n (Y_{i,j} - \hat{Y}_{i,j})^2$, where $Y, \hat{Y} \in \mathbb{R}^{1 \times n}$ are the predicted and actual speed vectors, respectively, and n is the number of road sections. The mean is first calculated over all elements in Y, \hat{Y} of a given sample, and then averaged over all m samples. Additionally, the following evaluation metrics are used: Mean Absolute Errors (MAE), Mean Absolute Percentage Errors (MAPE), and Root Mean

Squared Errors (RMSE), which are respectively given as follows

$$\begin{aligned}
 MAE &= \frac{1}{n \cdot m} \sum_{i=1}^m \sum_{j=1}^n |Y_{i,j} - \hat{Y}_{i,j}|, \\
 MAPE &= \frac{1}{n \cdot m} \sum_{i=1}^m \sum_{j=1}^n \frac{|Y_{i,j} - \hat{Y}_{i,j}|}{Y_{i,j}}, \\
 RMSE &= \sqrt{\frac{1}{n \cdot m} \sum_{i=1}^m \sum_{j=1}^n (Y_{i,j} - \hat{Y}_{i,j})^2}.
 \end{aligned} \tag{2.9}$$

Note that the MSE loss function is used for all data sets evaluation, while the evaluation metrics (2.9) are performed on the test set only.

2.5.2 Experiment 1: spatio-temporal model

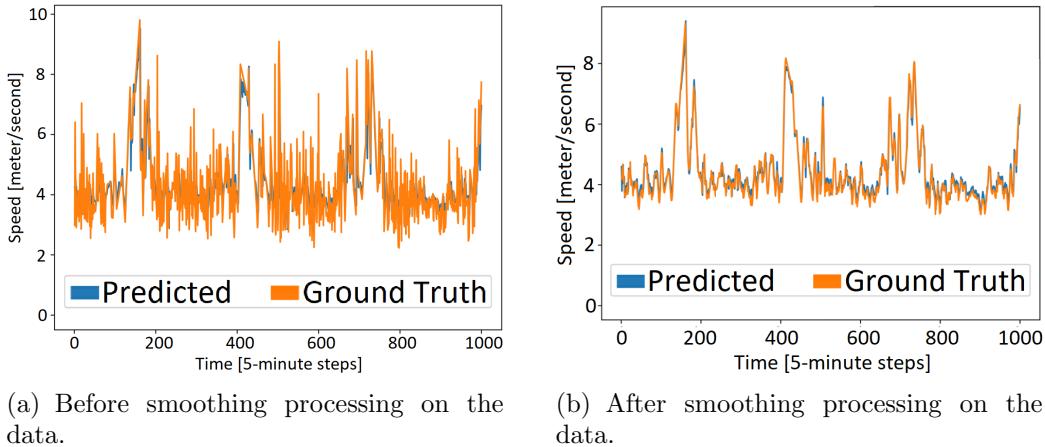
The implemented model [106] is referred to as STGCN (spatio-temporal GCN) and includes two spatio-temporal convolutional (ST) blocks, see the blue dashed box in Fig. 2.2, as each block consists of three convolutional layers with the following sequence: temporal \rightarrow spatial \rightarrow temporal, where the temporal is a 1D convolution layer. Each ST block includes also a layer normalization and a dropout layer. The output layer located in the end is responsible for the classification.

Data pre-processing and prediction results

The input and output data are the average speed of each road section, over few time steps before and after each given step. The data propagated in the model is constructed from four dimensions: batch size or #samples at each training iteration, #channels which is also #features (which are all per specific time step), #time steps, and #nodes in the graph. Different types of adjacency matrix were tested, and though there is no conclusive best type, the symmetrical matrix with inverse distances of the connected nodes only was chosen. The distances were the sum of the half of the connected road sections length. Similar adjacency matrix is used in Experiment 2. The data is partitioned into train, validation, and test sets. Then, we normalized it by the Z-Score method (mean = 0, standard deviation = 1).

In this experiment, we start from similar pre-processing as in the original paper [106], i.e. linear interpolation for filling the missing data and smoothing the data via a sliding window that averages each data point with its nearest neighbors on the time axis, to remove high-frequency noise. Then,

we examined the task of speed prediction based on previous speed values. In Fig. 2.4, plots of predicted versus real speed data are shown for a given sensor within specific time range. Fig. 2.4(a) shows results before the pre-processing, while the results after the pre-processing are shown in Fig. 2.4(b). As shown, a better performance is achieved.



(a) Before smoothing processing on the data.
 (b) After smoothing processing on the data.

Figure 2.4: Predicted vs real speed data for a given sensor within specific time range.

However, comparing the speed prediction performance with the prediction performance in [106], one can conclude that the latter reports better results. This might be due to a few factors: (i) in [106] the data was collected from double loop detectors, which can extract exact flows, while we have BT detectors that have only partial (not 100%) representative data. In addition, there is missing data from BT detectors, due to malfunctioning or errors; (ii) the collected data in [106] was over a period of two months only, while our data is from August to November 2019, which includes different seasons and different social conditions (summer break and school); and (iii) weekends were excluded from the collected data in [106], hence we have much more complex patterns to be learned.

Comparison results with other models

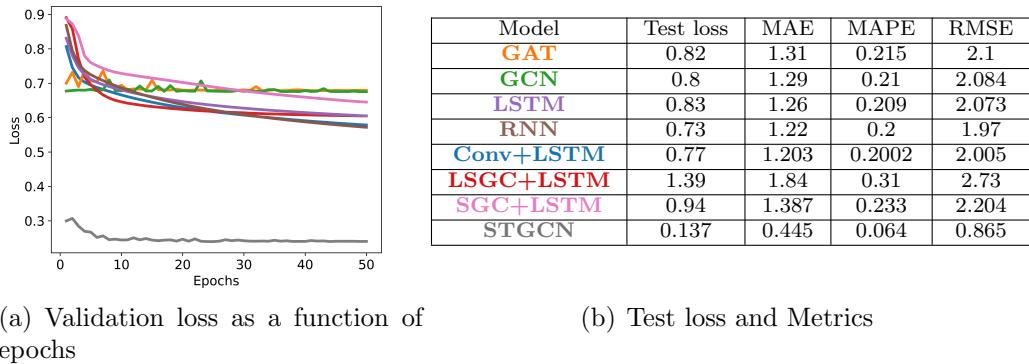
Several models [112] were implemented in PyTorch [113, 114] for comparison with the STGCN model: (1) GCN, (2) Graph Attention Networks (GAT), (3) LSTM, (4) RNN, (5) Conv+LSTM [115]: a 1D convolution layer with two channels followed by an LSTM layer, (6) SGC+LSTM: stacking a one-layer spectral graph convolution (SGC) layer [116] with an LSTM layer,

(7) LSGC+LSTM: stacking a one-layer localized spectral graph convolution (LSGC) layer [104] and an LSTM layer. All the LSTM/RNN layers have the same weight dimensions.

The comparison results are shown in Fig. 2.5. The results show that the STGCN model performs significantly better than both spatial-only and temporal-only models, in all chosen metrics.

GCN and GAT are strictly spatial models, for which the time steps are represented as features, so that we can compare these models with the other models. We can see that they both preform the worst compared to all other methods, since they do not learn the temporal dimension. On the other hand, RNN and LSTM strictly represent the temporal models. All above models that were included for comparison do not handle the whole time steps together as one collection, as STGCN does, but as any recurrent NN does: treating each incoming time step data sequentially, one after the other.

We can also see that LSGC+LSTM performs poorly, perhaps since it utilizes one-layer LSGC, i.e. it includes too few parameters, which is not enough to represent the network features [112].



(a) Validation loss as a function of epochs

(b) Test loss and Metrics

Figure 2.5: Methods comparison by their validation, test, and other metrics.

2.5.3 Experiment 2: Spatio-temporal model including commands and executions

In Experiment 2, the time step is defined as a cycle time, and multiple prediction tasks, i.e. different sets of input and output features, are tested. Experiment 2 uses the enhanced STGCN model described in Section 2.4.2, and shown in Fig. 2.2. It should be noted that a small fraction of the whole network is considered in Experiment 2, i.e. the Even Gvirol arterial, see Fig. 2.3 on the right, since we had to track intersections which have similar

cycle durations. Even gvirol arterial contains 9 intersections and 30 road sections.

Data pre-processing

Unlike the first experiment, here we have longer data pre-processing pipeline. (1) We start from SP features extraction, i.e. green durations and SP IDs. We filter out irregular cycles, e.g. all cycles below 60 seconds and above 130 seconds. These values are beyond the 1st and 3rd quartiles of the cycle duration distribution. After filtering, more than 95% of the data remains. (2) Since SP IDs are categorical features, we represent them as a dictionary or one-hot vector. We do so by either turning SP IDs to a semantically similar groups, e.g. police group, morning rush hour group, etc, or representing SP IDs with a big dictionary containing all different SP IDs of all intersections in the network. Then, we complete missing cycles (due to filtering in the 1st step) according to some criterion. (3) Extract features from BT data files. We also construct the adjacency matrix, according to BT detected passes between road sections. We must extract first the SP features, because BT features are calculated in the specific cycles defined by SP data file. Hence, every BT feature is calculated for the cycles they occur at. We calculate the sum of speeds and travel times at all cycles, and eventually divide each of these features by the #occurrences in the given cycle, i.e. divide them by the #vehicles passed at this cycle. Then, the flow and density are calculated by dividing the #vehicles by the cycle duration and road section length, respectively. Even though these are not the actual flow and density, since #vehicles is only partial information, we assumed a constant BT penetration rate [13, 117]. I.e. the actual #vehicles is merely a multiplication by a fixed number with the BT #vehicles value. Given the fact that we normalize all continuous input features anyway, then the BT values can be considered equivalent to the actual values. Therefore, we refer these features with similar names: flow and density. (4) Next we choose whether to interpolate missing data in some of the features, and whether to smooth away fast-changing data by an averaging sliding window. After the cleaning and filtering the data, gaps emerged in the sequential data. To address this, an interpolation method was used between cycles to fill these gaps. (5) Finally, we create data sets. First we find all sequences of cycles with the same duration, consecutive cycles, i.e. without missing data. Then, we normalize each feature by the Z-Score method (mean = 0, standard deviation = 1). Finally, we split the data (input, output) tuples into train, validate, and test sets. We do it either by random shuffling or by their original order.

Finally, after the described data cleaning and filtering, we can illustrate

the difference of how steps were computed in both experiments, in Fig. 2.6.

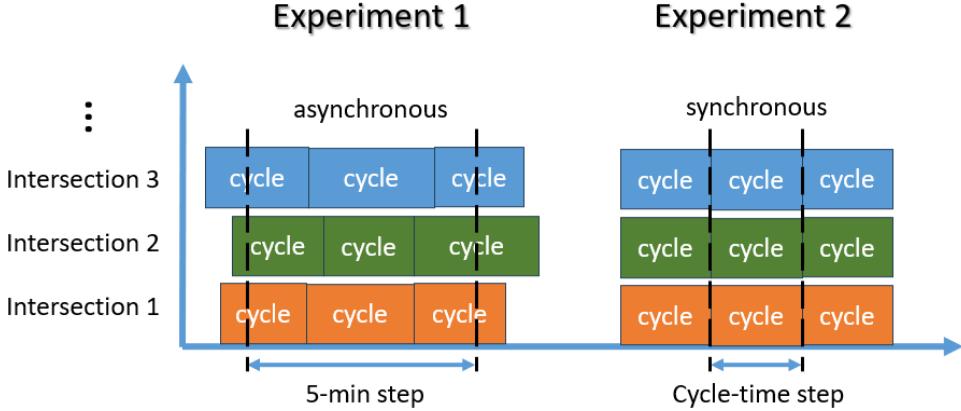


Figure 2.6: Experiment 1 versus Experiment 2 in time steps.

We could also summarize and compare both experiments under different categories, see Table 2.1.

Category	Experiment 1	Experiment 2
Tasks	$V \rightarrow V$	Many \rightarrow Many
Data size	$\sim 20,000$	thousands
Data cleaning	Less cleaner data	Very clean data
Model	Simple STGCN	Extended STGCN
Time step	5-minutes (coarse)	Cycle time ($\sim 1\text{-}2\text{min}$)
Transportation network	Full Tel-Aviv BT network	Even Gvirol (part of BT network)

Table 2.1: Experiments comparison under different categories.

Results analysis

First mission was general hyper-parameter tuning. Although different tasks (sets of input and output features) resulted with different best values for hyper-parameters, we have chosen average values: $K_t = 2$, $K_s = 5$, $\text{drop_prob} = 0.2$, $lr = 0.005$, where K_t and K_s are the temporal and spatial convolution sizes, respectively. drop_prob is the dropout layer probability value and lr is the learning rate. The reader can refer to [31] for more information on these parameters. We also decided to represent SP IDs via a big dictionary containing all different SP IDs of all intersections in the network, since in the grouping representation we had over 98% similar values, implying no effect.

Then, different configurations for predicting GT are tested for two data sets, as shown in Tables 2.2(a) and 2.2(b). The data set in Table 2.2(b) has 881 data sequences, much more than the 114 data sequences in Table 2.2(a). This is because Table 2.2(a) gathers data sequences of all SP and BT features, while Table 2.2(b) considers only for GT, P, and V features. From Tables 2.2(a) and 2.2(b), and other additional tests, we concluded that the best performance is obtained for input features that include GT and P features. It should be noted that including BT features resulted in a poorer performance.

We hypothesize that GT+P features can obtain better performance in GT prediction compared with SP+BT features, since in the generation of SP data it was not planned to account for BT features, e.g. nV, F, D. Moreover, GT is the result of two factors: human operator decision-making (mostly on P) and loop detectors on road sections, which are very rough in estimating BT features, e.g. loop detectors do not distinguish one vehicle or many vehicles for extending GTs, nor the actual demand based on speed or TT. Additionally, loop detectors use instantaneous features while BT features are spatial, i.e. over the whole road section.

Consequently, assuming that the GT feature is weakly correlated with BT features, it may explain the degraded performance in GT prediction task while including BT features. Nonetheless, it is perhaps would be better if GTs have been controlled based on BT-data in the first place.

From Tables 2.2(a) and 2.2(c) it is shown that P contributes significantly to prediction of either speed or GT. It may indicate that although previous steps of the same feature we are trying to predict may be sufficient, but including the P feature adds useful and complementary information.

Moreover, the task of speed prediction is analyzed, as a representative feature of the BT features. As shown in Table 2.2(c), the best performance is obtained for the combination V, P with other BT or SP features.

It should be noted that the effect of data sequence length on the performance was tested. For the same number of prediction steps, different number of previous steps were considered. As the size increases, the results improves but up to some size. For both the GT prediction and speed prediction tasks, the optimal previous steps size is 7 steps.

We also examined additional external features, as shown in Fig. 2.2, which are fixed spatially and temporarily, hence combined only after ST extraction of the ST features. Including these time external features to the speed and SP ID features can improve the performance of predicting GT, as shown in Table 2.2(d).

Finally, we tested how the external features enter the GCN, either via the enhanced model, externally, or via the original STGCN model as additional

input features. Results showed slight improvement in all evaluation metrics in favor of the enhanced model.

(a) Comparing different features in predicting GT

Input features	test loss	MAE	MAPE	RMSE
GT,P	0.0144	1.469	0.414	2.095
GT,V	0.0187	1.9	0.065	2.386
GT,P,V	0.0188	1.981	0.075	2.39
P	0.286	7.3	0.267	9.316
V	0.321	7.153	0.236	9.874
P,V	0.279	6.935	0.238	9.21
GT	0.486	9.184	0.34	12.15
GT,P,nV	0.02	2.024	0.071	2.463
GT,P,D	0.021	2.013	0.07	2.516
GT,P,F	0.213	2.064	0.073	2.542

(b) Comparing different GT, P, and V features in predicting GT

Input features	test loss	MAE	MAPE	RMSE
GT,P	0.0048	0.93	0.03	1.24
GT,V	0.0056	1.027	0.035	1.33
GT,P,V	0.0062	1.1	0.04	1.4
P	0.039	2.69	0.087	3.524
V	0.05	2.9	0.1	3.98
P,V	0.04	2.88	0.1	3.56
GT	0.097	3.47	0.14	5.54

(c) Comparing different features in predicting speed

Input features	test loss	MAE	MAPE	RMSE
P,V,TT,D,F	0.081	0.545	0.12	0.71
GT,P,V,nV,TT	0.09	0.564	0.122	0.748
GT,P,V	0.094	0.597	0.136	0.763
GT,V,nV	0.0943	0.61	0.142	0.764
P,V	0.1	0.6	0.132	0.786
P,V,nV,TT	0.108	0.605	0.128	0.817
all SP+BT	0.108	0.6	0.133	0.82
V	0.108	0.64	0.144	0.817

(d) Adding external time features in predicting GT

Input features	test loss	MAE	MAPE	RMSE
V,P,m,d,w,h	0.0042	0.814	0.0262	1.159
V,P,m,d	0.0051	0.826	0.0265	1.27
V,P	0.0089	1.051	0.034	1.676

Table 2.2: Different evaluations for a constant cycle time step.

2.6 Discussion

In this chapter, the traffic signal problem for the large-scale urban network of Tel Aviv is considered via a deep learning approach. The utilized data is spatio-temporal data that is represented via graph, therefore, a spatio-temporal graph convolutional network model is implemented.

Two experiments were conducted in this chapter. The first one's purpose was to compare different data sets, both with the speed feature but calculated differently. It was also compared to other models. The second one was for the main purpose of this chapter: to implement an initial concept of autonomous traffic management system, specifically using SP IDs as commands entering the system and green time durations as the system's output being considered as the executions of the commands. For that matter we tested different sets of possible input features, acted as explicit and implicit commands. We also tested different input features for the speed task, compared to the literature, to examine if additional information may improve the performance.

Results showed that in speed prediction task, additional features improve performance, specifically the inclusion of SP IDs with other SP/BT features, i.e. V + P + SP/BT features.

Other results showed a few orders of magnitude better performance for Experiment 2 (constant cycle time steps) compared to Experiment 1 (5-minute steps), even though we had $\sim 20,000$ fixed number of data sequences in Experiment 1 and only thousands in Experiment 2. Note that number of data sequences is dependent on many factors, e.g. size of data sequence and which features participate in input and output. In addition, a much more data cleaning and pre-processing were done in Experiment 2. Several possible reasons may explain the difference in performance between the two experiments: the steps in Experiment 1 were too coarse, i.e. 5 minutes compared to 1-2 minutes; and 5-minute aggregation is asynchronous, i.e. each step can contain partial cycles with different durations, see Fig. 2.6.

It is shown that although predicting GT is not efficient with some BT features, using BT features for GT prediction affects only mildly the performance, as shown in Table 2.2(a). Additionally, we have seen that including SP IDs and/or time external features in speed or green time duration prediction tasks can improve the performance.

Moreover, in GT prediction, it is shown that GT + P as input features perform better than SP + BT features. In our opinion, it is since the BT and SP are different datasets, and based on different measurements.

Finally, while we predicted execution as the green duration for each movement in the network, it is not actual traffic control. However, we also did P prediction, but did not show it since it was similar to GT in performance.

This way it can be utilized as a full traffic control system. Nevertheless, both P or GT can be considered as executions in the original model presented in Chapter 6.3.

2.7 Conclusions

Different datasets and features were considered. In the first model, prediction of speed data is examined, while in the second model green times and speed are predicted. The large-scale urban network of Tel Aviv is considered, where data features such as speed are extracted from an array of Bluetooth sensors located at the network signalized intersections, while its signal plans represent the traffic operators' commands. The obtained results showed that:

- including signal plan IDs and/or temporal features (month, year, day, etc) in speed or green time duration prediction tasks can improve the performance;
- considering fixed cycle time steps enhances the prediction compared with non-cycle-time steps; and
- including Bluetooth features in green times prediction task resulted with a slight degradation in performance.

Chapter 3

Preference commands in traffic signal control via Double Deep Q-Network

In this chapter, we propose to implement traffic signal control, by modifying the traffic signal via a reinforcement learning (RL) model, where the inputs contain explicit commands, and the output is a realization of these commands in the form of a phase. Also, we restrict the commands to preference commands. These commands represent relative priority among different movements in the network. Thus, the problem is formulated as a dynamic objective, a function of both the preference values and some pre-defined traffic control objective, such as minimum delay in an intersection. We follow the control method of a double deep Q-network (DQN) model-free reinforcement learning, with state for each intersection, action selected from a pre-defined set of phases for each intersection, and the reward representing our dynamic objective.

3.1 Motivation

We can compare this chapter to our previous Chapter 2, to emphasize its importance and novelty. In the previous Chapter 2, a supervising learning (SL) method was utilized to simulate a simplistic form of our goal management system, by providing a command (in the input) and producing its correct execution (in the output). The command in this case represents a fixed plan id, taken from a set of fixed plans, also referred to as Signal Plans (SPs), while the output is Green Times (GTs). We also assume that SPs in the given dataset, were designed via Webster technique, as an optimal solution

for single objective (minimum delay), see 3.2. Each SP was designed for a specific set of demands. Therefore, these designed SPs are considered as a very limited set of commands. Although other methods can design more conservative SPs, such as robust control design.

Therefore, to extend the set of commands, this chapter considers both time-varying demands and time-variable preference over movements in a transportation network, dictated by an operator in central control.

At first we thought to plan an algorithm (rule-based approach) for the desired commands, and then teach the DL model according to this algorithm via supervised learning (input-output pairs created by the rule-based algorithm). But the complexity of such task is high, since we have to plan for different sensory input scenarios, such as breakdown or accidents. Also, all the point of DL is to replace rule-based approach.

Hence instead, we will use simulation to replace the heavy design of commands to execution, by evaluating possible actions tested by a simulation. The actions are some pre-defined phases, and the method is RL where the evaluation is via a reward function.

Currently, there are many RL models that implement a single objective (e.g. minimum delay) for time-varying demands, which are represented in the state. Though, no work yet studied also time-varying preferences in the network.

Hence, we could state that Chapter 2 implemented open-loop system due to supervised learning (where only the input determine the output), while the current chapter implements closed-loop system, since RL retrieves a reward from the environment (in our case a simulation) which influences the input, which is the state in RL. The reward function is our designed evaluation algorithm.

3.1.1 Implemented RL versus SL comparison

ML can be implemented either via SL or RL, or unsupervised learning which is not an option for us. Here are some differences between SL and RL:

1. In training: the SL has the correct label, while RL has only partial answer - reward.
2. SL is constrained to the set of classes it was trained on. New classes are definitely mis-classified. Similarly if some classes are rarely appear during training.
3. Too much RL training can lead to an overload of states, which can diminish the results.

4. RL assumes the world is Markovian, i.e. the current state depends only on the previous state and not on past states, which is usually not true.
5. A well known phenomenon in RL, is the curse-of-dimensionality (COD), see [45], which is a scalability issue. It occurs due to high-dimensional space which the RL searches in, for optimal policy. A space of $|S| \times |A|$ size, which in our case $S, A \in \mathbb{R}^{d,m}$, where S, A are state and action respectively and d, m are their dimensions.
6. Finally, it is well known that RL requires much more samples (due to COD mentioned above), and is much heavier computationally than SL, see about its computation complexity in [118]. In our case, the task is to find best control action policy only for a finite set of commands, which reduces the dimension of the problem.

Additionally, there are other advantages for preference commands in the current chapter over SP commands from the previous Chapter 2. First, changing demands can be dealt via actuated SPs, such as being derived from the extension of Webster design. In our case, we utilize RL as another actuated control. Second, non-Webster SPs, e.g. emergency, work, and police - are specific/discrete set of solutions. Preferences, on the other hand, represent also intermediate cases, e.g. conflict cases (conflict between the limited set of SPs, e.g. police and holiday simultaneously).

In regarding to the previous SL from Chapter 2, several additional advantages support the use of RL. First, usually the action space A in RL is discrete, unlike the continuous space we had in our SL problem in Chapter 2. The SL problem was a regression task predicting Green Times (GTs). Here, instead we use phases as actions, while the time steps are fixed. It also means, that there is no constant cycle time in RL, because we do not have a fixed sequence of phases, as we had in our SL problem in Chapter 2. Both the freedom of phase order and unfixed cycle reduce the constraining in the optimization problem. Also, since the SL problem outcome were GTs, it required additional computation step to generate a legitimate and feasible phase.

Next, due to the comparison above and since SL requires a dataset, which is absent in our specific problem, the method of RL is considered a better option.

Moreover, Chapter 2 used two separate datasets to construct the features in the state. Results showed that they were interfering each other at prediction tasks. Hence the need for combining those features, in a new dataset, in this case generated by RL.

Additionally, the GT dataset consisted of asynchronous cycle times, i.e. heterogeneous cycles for different intersections, which required aggregation for a centralized control, that makes the data less accurate (more rough). But RL can produce synchronous data, with fixed step.

Finally, we did not compare the AI methods to other classical methods in prediction in SL or in traffic signal control here in RL, for several reasons. First, our goal is to build a comprehensive, holistic system that continually learns and adapts to new requests and appropriate executions. Even if some model-based or control-based methods outperform our AI presented methods in specific tasks, they cannot replace the AI methods within the overall system. These classical methods are often specific and tailored for pre-designed tasks and can be considered rule-based or logic-based. For example, in the prediction task, the store-and-forward model [5] is a human-designed model, constructed entirely from inductive bias or prior knowledge, without relying on data. In the traffic signal control task, several optimal control methods exist, such as LQR [7], MPC [8, 19], IC, SIC [9, 12], and their adaptive and robust versions. All of these methods are designed for specific, pre-defined objectives and are based on models. In contrast, our aim to construct a system using only machine learning tools that do not require prior design.

Second, before selecting the methods for SL and RL [119, 120], we reviewed numerous papers that had already compared AI methods with classical methods, and we chose the best-performing AI methods.

The comparison between SL and RL discussed in this section is summarized in Table 3.1.

Category	Previous step (SL)	This step (RL)
Commands type	Simple	Preference
Objective	Static (minimum delay)	Dynamic
SP design	for specific demands	for varying demands
Event handling	Specific SPs: emergency, work, police, ...	Intermediate cases (conflicts)
Training method	Given dataset	No dataset (simulation)
Dataset composition	Separate datasets (SP and BT) interfere each other at prediction tasks	Combined different features
Data units	Mostly asynchronous cycle times	Synchronous fixed steps
Traffic signal control	Not necessary	Yes
State definition	node = road section, edge = intersection	node = intersection edge = road section

Table 3.1: SL versus RL comparison over different aspects.

3.2 Introduction

In this chapter, we restrict the general commands to commands from specific category - preference commands. These commands represent relative priority among different movements in the network, see example in Fig. 1.1, where the red line represents prioritized movements, while the rest of the movements (in black) in the network are neutral.

More precisely, the problem can be described as follows. Given a traffic network, a set of preference relative values w_i for each of the movements i in the network, and a fixed pre-defined traffic control objective, e.g. minimum delay or maximum throughput in an intersection, the problem is defined as determining phase according to green time constraints and a dynamic objective, which is a function of both the preference values and the pre-defined traffic control objective. It is dynamic, since the preference values can be determined at any time instance t , i.e. $w_i = w_i(t)$.

In other words, the objective is a weighted minimum delay with *dynamic weights* objective, as follows

$$J(t) = \min \sum_{i=1}^n w_i(t)d_i(t), \quad (3.1)$$

while the typical weighted minimum delay with *static weights* objective J :

$$J(t) = \min \sum_{i=1}^n w_i d_i(t), \quad (3.2)$$

is a special case of the *dynamic weights* objective in (3.1), where w_i are constant.

The n is the #movements¹ in a given intersection in the network, $w_i(t)$ is the weight of movement i or its “preference” relative value, and $d_i(t)$ is the vehicle delay in movement i , which is defined as the total accumulative waiting time of all vehicles of movement i at current time t . Again, we assume here an example of the fixed objective to be minimum delay, but in general it could be any fixed objective.

Note, that the intersection delay J is the sum of the accumulative waiting time of all vehicles at a given intersection, until the current time step. While the network delay is defined as average delay for all intersections.

The problem, as it is defined in (3.2), is a multi-objective problem, since the objective is a joint sum of separate objectives, where each objective, e.g. total delay, representing a single movement in a given intersection in a network. However, our problem, as defined in (3.1), is a dynamic multi-objective problem, since it has a different solution for a given set of preference weights.

“Preference” means to give high or low priority for movements in a given network. For example, when a platoon of the president travels to some place in a special route, or when there is a tournament, the municipality usually blocks roads to allow smooth and un-interfered movement. Alternatively to physical changes in the network, e.g. blocking, the movement could be controlled via the same actuators: by signal lights. Each movement is assigned with a relative preference value (relative to other movements), and the signal traffic control is adjusted to give more right-of-way (r.o.w) to the more preferred movements, and less r.o.w for the less preferred movements. Thus, instead of controlling the signals to merely reduce congestion, they could be also utilized for any general objective, such as described in the examples above.

Note that the preference weights w_i are independent from the state of the network, such as flow demands. Since it is a top-down management scheme, hence the term ”commands”. The commands affect the state, but not the other way around. Otherwise, unwanted feedback to the system will be introduced, where the intention of the operator is not defined by him,

¹# denotes ”number of”

but also from the network. For example, if some accident or new demands occur in the network, the command still applies, until the operator decides to change it. Nevertheless, further discussion about how some feedback could be possible is in Section 3.7.

This chapter considers both time-varying demands and time-variable preferences over movements in a transportation network, dictated by an operator in a control center. Currently, there are many Reinforcement Learning (RL) models that implement a single objective (e.g. minimum delay) for time-varying demands, which are represented in the state. Though, no work yet studied also time-varying preferences in the network.

Additionally, the state representing the current network situation can be comprised of different features, such as travel time, speed, flow, density, etc. These features are gathered for all the movements in the network.

In summary, the problem is defined as RL, with state composed of features, including the preference weights w_i , and the result is an action in the form of a phase for each intersection in the transportation network in the next step. It is illustrated in Fig. 3.1.

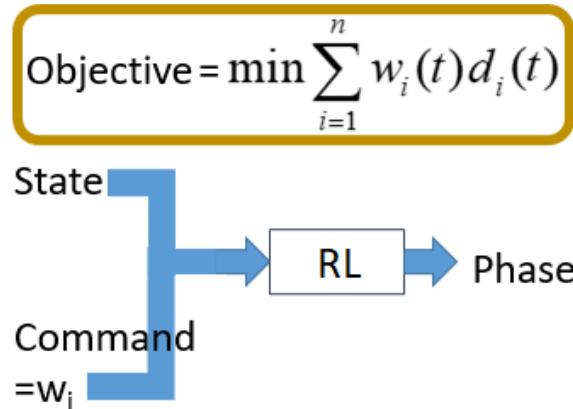


Figure 3.1: The TSCP problem tackled in this chapter (the system and its objective).

Several methodologies can be used to solve our problem, denoted in (3.1). One possible solution is to use linear programming (LP), where the objective is a convex combination as depicted in (3.1). However, each time a new set of preference coefficients is introduced - a new (online) optimization algorithm is needed to be executed. Subsequently, since (3.1) is a dynamic multi-objective problem, LP optimization generates a Pareto Front of optimal solutions for different sets of preference weights.

Another solution could utilize the classical control theory, however, the Machine Learning (ML) method is chosen. This is due to several reasons:

(i) system identification is avoided; (ii) simplified models are usually used, e.g. linear ones are too simplistic and perform poor in non-linear type of model as in our case; and (iii) usually control systems assume a model to represent the data and not to learn it, hence depend only partially on data (as in Adaptive control schemes).

ML can be implemented either via Supervised-Learning (SL) or RL. SL is about learning direct mapping from given inputs to given and desired outputs, while RL is about learning to select an output given an input, but as a function of an engineered and pre-defined reward, a signal from the environment, evaluating the correctness of the applied output on the environment. Therefore, it can be also considered as an implicit mapping from inputs to outputs, i.e. not a direct one, but rather through some inter-mediating reward from a given environment.

Finally, the RL setup in this chapter is the following: the state space is of low resolution type, i.e., it consists of aggregated/averaged features, hence ignoring the small details. Features such as average speed, average queue length, average density, average travel time, and last applied action. The reward is queue length or total waiting time of all movements or lanes for an intersection. The action is a phase from a pre-defined set, and it is determined every fixed time step. The control structure is decentralized multi-agent structure. The RL strategy used is Q-learning. However, Q-learning can be implemented through two methods. One is called Q-table, where the Q-values are learned in a table. In the table, each Q-value corresponds to a pair of discrete state and action. Alternatively, each Q-value could be for range of state or action, in case either of them is continuous.

The other is called Deep-Q-learning, where a DNN is learning Q-values at its output, for a given state s_t in its input, with each Q-value for a specific action $a_t \in A$. In this research, the Deep-Q-learning method is utilized via DQN.

The DQN models used are two types of Deep Neural Networks (DNNs): a Fully-Connected Neural network (FCN) and a Spatio-Temporal Graph Convolutional Network (STGCN) for communication among agents.

The contribution of this chapter is the introduction of preference weights with time-varying demands in the TSCP. It is implemented via RL, in a SUMO simulation software. And we demonstrate robustness, and the benefit of preference in complex situations, e.g. when a phase is decided upon complex relation of both demand and preference weights.

3.3 Related works on RL in Traffic signal control problem (TSCP)

RL in TSCP [42, 121] was implemented via different methods, e.g. in [122] the TD(λ) is used, while in [123, 124] a discrete state Q-learning(λ), SARSA(λ), actor-critic(λ); continuous state Q-learning(λ), SARSA(λ), actor-critic(λ), and residual actor-critic(λ) are used. In [124, 125] the Actor-Critic (AC) is used, while in [126] Asynchronous Advantage Actor-Critic (A3C) is used. These methods will be explained more thoroughly in Section 3.4.2.

These different methods are implemented via different simulation programs. Many studies use SUMO for simulation [125–130], while some use AIMSUN (Advanced Interactive Microscopic Simulator for Urban and Non-Urban Networks) [123] or other simulators.

In RL we have to define the state $s \in S = \mathbb{R}^d$, where $d = \#\text{features}$, the action $a \in A$ with the set $A = \{A_i, i = 1 \dots m\}$ where $m = \#\text{actions}$, the reward $R \in \mathbb{R}$, and the policy that the RL learns: $\pi : S \rightarrow A$. The state describes everything we need to know for performing best decisions to achieve some objective. The policy represents how to choose (preferably the optimal) next action. And the reward is our measurement or evaluation of our objective in the optimization problem.

A brief literature review of RL in TSCP is given according to the following categories: actions, states, rewards, and RL methods.

Actions: In the following several action definitions are introduced, then different constraints over actions are discussed.

Actions could be defined in several ways. Only few studies [122, 131] utilize dynamically generated phases, constructed dynamically from non-conflicting movements. Also, [130] utilizes a step of full cycle, where the action space is continuous, and every step the controller decides the cycle-length within some range and the relative durations of ordered phases to be implemented during this cycle.

However, most of the studies define a set of phases, where there are usually two types of action selection:

1. **#actions=#phases:** at each step, a phase is selected out of the pre-defined set of phases [132, 133].
2. **#actions=2:** at each step, it is decided whether to continue the current phase or to change it [44, 120, 122, 127, 134–136].

In [133] instead of immediately transitioning from the current signal phase to the next one, a sequence of intermediate traffic signal phases is applied.

Next, actions in TSCP include several constraints. Mainly over the green duration, the cycle, and the phase order.

As in classical control theory the action is applied throughout cycles, in RL the action is applied usually at fixed steps. In either case, appropriate constraints are defined. Most of the studies restrict their action to be physically possible, e.g. utilizing minimum and maximum green times [41, 127, 137, 138]. However, there are those that reduce these restrictions to minimum, e.g. [120, 126, 134] utilize minimum green time only, which leads to starvation of other vehicle movements and restrict pedestrian movement [139].

Some studies are using non fixed-time cycles with fixed phase order [40, 120, 127, 135, 136, 139], while other studies do not have cycles, e.g. [44], rather they have steps. One issue with the random phase order is that it is not safe nor comfortable for drivers, and it confuses them [136, 139]. Though changing green duration in fixed phase order is also problematic and may lead to instability [40]. Hence, smoothness is applied to reduce this issue. On the other hand, [139] applies its action to be a discrete set of possible green duration for the next phase.

However, these two cases [40, 139] represent the most restricted optimization problem of all the problems described above. For example, [40] cannot respond to dramatic traffic changes, while [139] does not optimize with immediate feedback.

State: State is composed of features. Possible features could be speed, travel time, queue length, density, and more. Given the set of selected features, a space of the state is defined. Some works implemented high-resolution space, e.g. a 2D cell grid, with each cell includes some feature, such as occupancy or speed of a vehicle [40, 128, 133, 140]. While others implement low-resolution space, representing the whole road section or its lanes, via aggregation functions such as sum or average. One issue with aggregated features, is that they do not distinguish queue or density location in the road. For example, queue may be in the mid of the road, and the policy thus creates a wasted green time. Here [126, 135] argued that the different resolutions perform similarly, while in [133] the claim is that more and complementary features can be beneficial, e.g. queue length alone could regard all non-queued vehicles as irrelevant, thus additional features such as density are needed.

Some additional features are included in special cases. For example, there are cases where the state includes also the last action applied in the system. In other cases, studies that implement minimum phase duration constraint also include the elapsed phase time, the time passed since the minimum period finished for the current phase [41, 44, 126, 132]. Finally, there are cases

in which the state can be extended even further, to allow communication between agents, e.g. [41, 138] by including information about neighboring intersection.

Reward: There are different reward functions for TSCP in the literature [42]: queue length, waiting time, speed, number of vehicle stops, throughput (number of vehicles exit the network/intersection), frequency of signal change during a certain time period, accident avoidance, and pressure [141] (difference of some feature between the downstream and upstream movements).

Some papers use a single reward, one from the list above, while others solve a multi-objective problem [142] by using multiple rewards either separately, i.e. each reward is active under different circumstances, or jointly, in a single reward via some combination of different reward functions, e.g. in [128] the reward combines penalties for crashes, jams, emergency stops, and phase switching. On the other hand, [138] utilized dynamic reward, or more accurately multiple separate rewards. This dynamic reward depends on the current congestion level. I.e., during free flow conditions the reward is determined by the queue lengths, and during medium level of congestion the reward is determined by the average delays; while it is determined by a Normalized-Queuing-index (NQ-index) during saturated conditions, which represents the available capacity of each lane, only normalized by the length of the lane.

Additionally, many multi-agent RL (MARL) studies use also global reward besides the usual local reward, to promote cooperation among the agents and to avoid conflicting self-interest [41, 120].

Moreover, the reward is associated to each intersection in the network. Hence, each of those metrics are aggregated from the lanes or the roads approaching the intersection, e.g. by sum or average.

Besides, the reward could be also a relative value, i.e. the difference between the instantaneous values of consequent steps, e.g. the difference between the current and previous cumulative waiting times (or delays) [40, 44, 126, 127, 133], or the sum of squared maximum queue lengths [132]. [143] defines a reward as the difference between previous and current intersection total delays, and normalized by their maximum value, thus limiting it to be in the range [-1,1]. [140] defines reward as the average flow per lane (i.e. total flow divided by #lanes) or its incremental value: the average flow through the intersection during the step minus the flow predicted by the network Macroscopic Fundamental Diagram (MFD) at the prevailing density.

RL methods: Some research works implement DQN, such as in [40]: double DQN, dueling DQN, and Double Dueling Deep Q Network (3DQN). While other studies implement non-deep or regular Q-learning, i.e. via Q-

table or Q-matrix [41, 137]: a table assigning each pair of (state,action) with their Q-value. DQN is usually utilized via fully-connected or convolutional neural networks, while some studies [134] use deep Sparse Auto-Encoder (SAE) neural networks.

Note, that while Q-table can guarantee convergence at infinity, DQN does not [128]. This is because the DNN of DQN method assumes i.i.d. data distribution, while RL is sequential, hence highly correlated.

In addition, while the basic implementation of MARL is of a separate model for an intersection, most of the works in literature evolve cooperation or communication among agents [41, 44, 128]. Some works limit themselves to neighboring agents only, due to curse-of-dimensionality (COD) (state-space grows exponentially with number of agents) [44]. For example, by incorporating neighboring agents' rewards with some relative weight [143], or [144] by providing congestion information from neighbor agents. In [128] the authors use *transfer planning*, which finds solutions, denoted by Q_e , from a smaller problem (involving few agents) and uses this solution to select actions in a coordinated fashion in the larger target problem. The authors factorize the global Q-function as a linear combination of local sub-problems: $Q(s, a) = \sum_e Q_e(s_e, a_e)$, where e corresponds to a subset of neighboring agents.

Similarly, in [120] the network is split into small regions, each representing an agent and solved locally, which stabilizes the agent's behavior (the low level). At a higher level, cooperation is allowed to seek a globally optimal solution via a global reward. Thus, resulting with a hybrid reward of convex combination: $\beta \cdot \text{reward}_{\text{global}} + (1 - \beta) \cdot \text{reward}_{\text{local}}$, where $0 \leq \beta \leq 1$. The authors utilize clipped Proximal Policy Optimization (PPO) algorithm as a policy update algorithm, for the actor model in Advantage Actor-Critic (A2C) method.

3.3.1 Graph Neural Networks (GNNs) in RL

When a multi-agent system is considered, as in our case, the issue of control structure should be addressed.

Different control structures have been considered in AI-based TSCP studies [42, 145]: (i) centralized or single-agent, which is better in performance for small systems due to computational load, (ii) multi-agent single-layer, which has less computational load on expense of lower performance, with either dis-communicative agents (decentralized) or communicative agents (distributed), and (iii) hierarchical or multi-layer, where some of the agents have authority over other agents. A multi-layer control structure can combine the advantage of the single-agent structure with the multi-agent single-layer structure, i.e.

overall system performance with tractability.

For instance, three studies used a hierarchical control. One [146] used distributed control via regulators, and the others [21, 147] used decentralized control. Results in [21, 147] show that communication between agents in a hierarchical structure has better results comparing to the system without communication, as in [146], due to inter-dependency between the agents. This fact encourages the use of highly connected NNs, such as Graph NNs (GNNs).

GNNs belong to one type of NNs that can handle graphical inputs [73]. They are responsible for feature extraction to solve three types of task: node prediction, link prediction, and graph prediction. Many studies implement GNN in RL for different applications. Applications such as circuit designing [148], chip implementation [149], robot mobility [150, 151], internet network planning [152], routing optimization use case in optical networks [153], combinatorial optimization [154], urban mobility [155], infection spreading [156], and more.

There are also studies concentrated specifically on the topic of traffic signal control. [157] uses Graph Convolutional Network (GCN) to directly extract geometric road network features and adaptively learns a policy for each of the agents (intersections) in the network. It solves the TSCP of a road network with multiple intersections. [158] shows how training on an arbitrary set of road networks, represented by a GCN, via parameter sharing, can be generalized to new road networks and thus enables also scalability. [158] also captures the demand both in the lane level and in the vehicle level. [119] proposes Inductive Heterogeneous GNN Multi-agent Actor-critic (IHG-MA) algorithm, which conducts both representation learning via IHG and policy learning via decentralized MA. The IHG-MA is a decentralized cooperative framework, which optimizes the whole algorithm to learn transferable traffic-signal policies (to new networks). [129] proposes a decentralized graph-based multi-agent A2C method, and uses a GCN with a weight factor λ to adjust the weight of the self connections (similar to attention mechanism). It tackles the adaptive TSCP with learning cooperation among agents. [130] proposes a Multi-Agent RL based on the Deep Spatio-Temporal Attentive NN and AC structure to determine the traffic signal timing in a large-scale road network. The authors use GCN and RNN to constrain the action search space instead of fully random exploration.

All the studies above solve the TSCP for a multi-intersection road network with dynamic traffic demand. Also, these studies emphasize the benefit of communication and cooperation among agents. This fact encourages the use of GNN in reinforcement learning for the TSCP, e.g. Spatio-Temporal Graph Neural Network (STGCN) that was used in our previous work [159].

3.4 Preference Commands in TSCP

3.4.1 Problem Definition

In this chapter, our specific problem is defined as follows. Let us consider a transportation network with n movements and some number of intersections. The problem is to select the optimal phase for each intersection in the network, given a pre-defined set of phases, different intersection and lane features, and a set of preference weights for each lane. The phase selection should be optimal with respect to the dynamic weights objective, defined in (3.1).

This problem will be solved via RL, specifically via Double DQN (DDQN) method. In RL we have to define the state $s \in S$, the action $a \in A$, the reward R , and the policy that the RL learns: $\pi : S \rightarrow A$.

As presented in Section 3.2, the RL state is composed of several features and the preference weights w_i for each movement i for a specific intersection. The RL action is discrete and selected from a pre-defined set of phases for each intersection in the transportation network. In addition, the TSCP here is unique since the state and the reward are formulated differently, as discussed in Section 3.6.3.

In general, RL is based on the Markov Decision Process (MDP) assumption, in which any transition from the current state s_t to the next one s_{t+1} is sampled from some probability distribution, i.e. $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$. This assumption states that next state depends only on the current state and not the previous states, i.e.: $P(s_{t+1}|s_t, s_{t-1}, \dots, s_0, a_t) = P(s_{t+1}|s_t, a_t)$. RL learns the optimal policy, which yields the long-term optimal action at a given state. Action which will maximize the current reward and all the following rewards also.

Finally, DQN is a model-free approach, i.e. it is not needed to know the MDP model or the transition function, e.g. for deterministic decision process: $s_{t+1} = f(s_t, a_t)$. Instead, it learns the optimal policy from experience.

3.4.2 Optimization methods

In the following, several optimization methods are described: (i) the DQN, as the fundamental method, (ii) the Double DQN, which is implemented in this chapter, and (iii) other related methods from the literature.

DQN

Here, the optimization problem that DQN solves is presented. The optimization problem starts from a random initialization, or random policy, and searches for the optimal policy. Hence, we define Q-function for any policy. This Q-function, $Q : S \times A \rightarrow \mathbb{R}$, also referred to as Q-values, is the accumulative expected reward in a given state s_t , for each of the actions in the A set, or: $Q(s_t, a_t)$ where $s_t \in S$ and $a_t \in A$. The defined Q-function for some given policy π is:

$$Q^\pi(s_t, a_t) = \mathbb{E} \left[\sum_{\tau=t}^{\tau=\infty} \gamma^{(\tau-t)} R_\tau | s_\tau = s_t, a_\tau = a_t, \pi \right] \quad (3.3)$$

and Q-function for the optimal one as follows,

$$Q^*(s_t, a_t) = \max_{\pi} \mathbb{E} \left[\sum_{\tau=t}^{\tau=\infty} \gamma^{(\tau-t)} R_\tau | s_\tau = s_t, a_\tau = a_t, \pi \right] \quad (3.4)$$

where $a_t = \pi(s_t)$ is a policy by which an action is selected, while $\gamma \in [0, 1]$ is the discount factor that determines the importance of future rewards compared to the current reward. That is, $\gamma = 0$ is to account only for the immediate reward, while $\gamma \approx 1$ (e.g. $\gamma = 0.99$) is to account mostly for the future rewards (long-term planning). Then, Q^* encodes the optimal policy:

$$\pi^*(s_t) = a_t = \arg \max_{a_i \in A} Q(s_t, a_i). \quad (3.5)$$

Moreover, it is known [160] that Q^* satisfies the Bellman equation, i.e.

$$Q^*(s_t, a_t) = \mathbb{E} \left[R_t + \gamma \max_{a_i \in A} Q^*(s_{t+1}, a_i) \right]. \quad (3.6)$$

Hence, the optimal Q-function, Q^* should satisfy this Bellman equation. Thus, we start with a random Q , and use the Bellman equation as an update rule:

$$Q_{k+1}(s_t, a_t) = \mathbb{E} \left[R_t + \gamma \max_{a_i \in A} Q_k(s_{t+1}, a_i) \right]. \quad (3.7)$$

Then, it should converge to Q^* , i.e. $Q_i \xrightarrow{i \rightarrow \infty} Q^*$. Hence, ideally

$$\mathbb{E} \left[R_t + \gamma \max_{a_i \in A} Q^*(s_{t+1}, a_i) \right] - Q^*(s_t, a_t) = 0. \quad (3.8)$$

Since we use NN with parameters, denoted as θ , we try to approximate $Q^*(s_t, a_t)$ as $Q^*(s_t, a_t) \approx Q(s_t, a_t, \theta)$. We define a loss function to be minimized and strive to be (3.8), i.e.

$$\text{Loss}(s_t, a_t) = \mathbb{E} \left[R_t + \gamma \max_{a_i \in A} Q(s_{t+1}, a_i, \theta) \right] - Q(s_t, a_t, \theta). \quad (3.9)$$

It is also referred as temporal difference (TD) error, between the estimated TD target and the current Q value (estimation of Q).

The loss (3.9) is updating the Q-values according to the following rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(R_t + \gamma \max_{a_i \in A} Q(s_{t+1}, a_i) - Q(s_t, a_t) \right), \quad (3.10)$$

which updates the Q-value according to the given (s_t, a_t, R_t, s_{t+1}) tuple.

Training algorithm

After a loss is developed in (3.9), and with its appropriate updating rule (3.10), we can explain the overall DQN training algorithm. Each step of training consists of five subsequent steps:

1. Selecting an action a_t given the current state s_t .
2. Extracting the reward R_t and the next state s_{t+1} , after applying the chosen action a_t .
3. Adding the tuple (s_t, a_t, R_t, s_{t+1}) to a memory batch or experience replay.
4. Selecting a batch of (s_t, a_t, R_t, s_{t+1}) tuples from the memory batch.
5. And optimizing the model over the batch accordingly to the loss (3.9) and the updating rule (3.10).

The algorithm starts with action selection (1), which in our case is implemented via ϵ -greedy policy:

$$\pi(s_t) = a_t = \begin{cases} \arg \max_{a_i \in A} Q(s_t, a_i) & \text{with probability } 1 - \epsilon, \\ \text{random action} & \text{with probability } \epsilon. \end{cases} \quad (3.11)$$

Specifically, an exponentially decaying $\epsilon = 0.99 \rightarrow 0$ is applied, to enable the transformation of the optimization process from exploratory phase to exploitative phase. That is, the process starts from exploration (selecting random actions at each step), and then gradually with the progress of episodes

it becomes less exploratory and more exploitative, i.e. choose the optimal action.

Later, the DQN uses experience replay, i.e. a memory of past transitions (s_t, a_t, R_t, s_{t+1}) . Then, at each episode a random batch from this memory is chosen for optimization.

A more detailed explanation of this algorithm is in Section 3.5.3.

Double DQN (DDQN)

Here, we derive DDQN, an upgraded version of DQN, by distinguishing between behavioral policy (affecting Q-values denoted as Q) and learned policy (affecting Q-values denoted as Q_{target}). This difference is expressed in three RL setups: on-policy RL (demonstrated by SARSA method), off-policy RL (demonstrated by DQN method), and offline RL.

(3.10) assumes that the Q-value is updated via greedy policy only (3.5), i.e. it assumes that the next action a_{t+1} is determined by the maximum Q-values in a given state s_{t+1} for all possible actions $a_i \in A$. This assumption is not necessary what happens in reality. In reality, the actual next action a_{t+1} follows the ϵ -greedy policy (3.11), which may yield either a random or greedy action. Hence, there is discrepancy between the assumption embedded in the Q-value updating rule and the selection of next action a_{t+1} . This divergence between the policy applied (behavioral) and the policy learned (estimation or target policy), is referred to as off-policy RL.

In contrast, SARSA method [123] updates the Q-value given $(s_t, a_t, R_t, s_{t+1}, a_{t+1})$, where unlike Q-learning a_{t+1} is determined by the same (ϵ -greedy) policy. I.e. the update rule follows the same policy as the one that applied:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha (R_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) . \quad (3.12)$$

This means that DQN can use experience replay containing states and actions, which were generated by previous policies that are different than the ones that are applied currently (hence the "off-policy"), while SARSA cannot use it due to previous different behavioral policies (hence the "on-policy"). DQN improves sample efficiency since we do not need to recollect samples whenever a policy is changed. Offline RL separates the behavioral and the learned policies even further. It assumes that the agent does no longer interact with the environment and collect additional transitions using the behavior policy. It uses only stored experience replay generated by some unknown policy. This results with distributional shift, since the learned policy is deviated a lot from the behavioral one. DQN, in contrast, has less shift, since the target policy is induced by the latest behavioral one.

Note that there is a crucial difference between Q-Learning and DQN: a too strong off-policy setting will destroy DQN due to function approximation. Therefore, a better alternative to the regular DQN is proposed: DDQN. DDQN separates the learned and the target policies to be represented by two DNNs instead of one. It learns Q_{target} values according to the learned policy, while it uses Q values according to the applied policy. Then:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(R_t + \gamma \mathbf{Q}_{target}(s_{t+1}, \arg \max_{a_i \in A} \mathbf{Q}(s_{t+1}, a_i)) - Q(s_t, a_t) \right). \quad (3.13)$$

The need for separation is due to TD error: the estimated TD target and the current Q value are interrelated, since they use the same DNN or the same parameters. Hence, updating of current Q-value also updates estimated TD target in the same direction, which makes the optimization unstable. Therefore, DDQN copies $Q \rightarrow Q_{target}$, and keep Q_{target} fixed for a few steps then copy $Q \rightarrow Q_{target}$ again. Alternatively, it uses small learning rate to update Q_{target} from Q at every step.

Additionally, DDQN solves the over-estimation problem that occurs in DQN. Since in updating rule (3.10) we use a greedy policy as only the best estimation of $Q^*(s_t, a_t)$, then the correct action probably has a smaller Q-value. In other words, the agent tends to take the non-optimal action in any given state only because it has the maximum Q-value.

Note that all above updating rules for the Q-functions can also have a different version:

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha (\text{TD error}) \quad (3.14)$$

This version limits the $\alpha \in [0, 1]$ as a trade-off between the current Q-value and its new additional update. While the previous version has no upper limit.

Other related methods

Q-learning algorithm explained above, is based on temporal difference (TD) error, between predicted and actual Q-values. By minimizing this error it updates the estimate of the Q-value. However, TD algorithms are more general, and described as $\text{TD}(\lambda)$ [122]. $\lambda \in [0, 1]$ parameter refers to the trace decay parameter, where $\text{TD}(0)$ is the Q-learning method described above. Conversely, the higher the λ goes, the more it leads to longer lasting traces; that is, a larger proportion of credit from a reward can be given to

more distant states and actions, and at $\lambda = 1$ it reaches Monte Carlo (MC) RL algorithms. More precisely,

$$\text{TD}(\lambda=0) = R_t + \gamma \mathbf{Q}(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \quad (3.15)$$

$$\text{TD}(\lambda=1) = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots + \gamma^{n-1} R_{t+n-1} + \gamma^n \mathbf{Q}(s_{t+n}, a_{t+n}) - Q(s_t, a_t) \quad (3.16)$$

We can see the difference in Fig. 3.2:

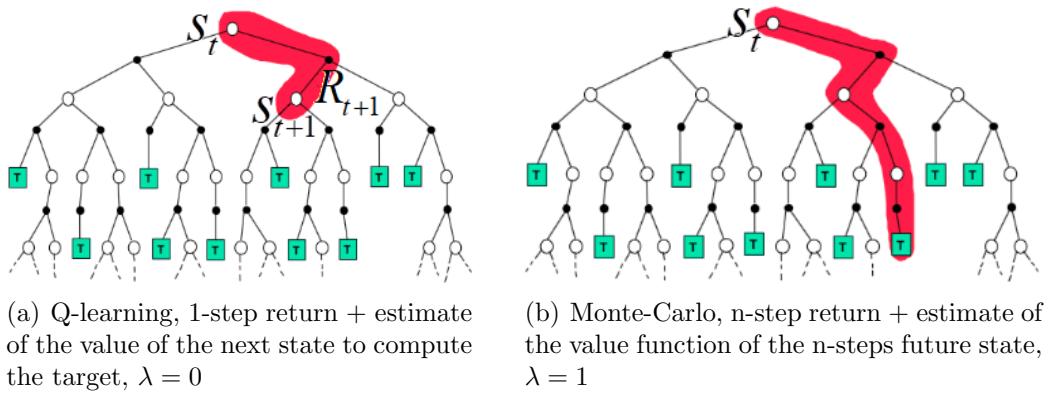


Figure 3.2: State evolution representation of $\text{TD}(\lambda)$, illustrated as a combination Q and MC learning, taken from [161].

Fig. 3.2 illustrates a tree representation of how state and its corresponding reward evolve through time steps. It shows two extreme cases of the continuous $\lambda \in [0, 1]$.

Additionally, Q-learning can be applied either in a discrete state space or in a continuous one [123]. In [123] SARSA(λ) algorithm is used - on-policy algorithm that updates the Q-values based on the actual next action taken and the corresponding reward, see also Section 3.4.2.

Finally, actor-critic(λ) (AC) [122, 125] is a model-free RL algorithm that uses a policy network (actor) to select actions and a value network (critic) to estimate the value function and updates the policy based on the estimated advantage. It has other extensions, such as residual actor-critic(λ), that uses residual networks to improve the accuracy of the value function, and an Asynchronous Advantage Actor-Critic (A3C) [126], that uses multiple agents with separate copies of the AC network to update the policy and value function asynchronously to improve training speed and stability.

3.5 Implementation

This section presents the implementation details, commencing with a discussion of the chosen NN structure. Subsequently, we explore how cooperation among intersection agents was implemented, followed by an examination of how we addressed the requisite traffic constraints. Lastly, we discuss the utilized traffic networks and the DNN models that were used in the DQN, specifically fully-connected and graph types of neural networks.

3.5.1 Neural Network Structure

We first decide upon our control structure. We consider two options: the centralized structure and the decentralized structure. In the centralized case, there is one DQN model for the whole network. Hence the DQN's input is the state of the network and its output are the Q values for all possible actions to be applied on the whole network. After all the Q values are calculated, one action is selected, for the one corresponding to the highest Q value. Additionally, one reward is computed for the whole network. On the other hand, in a decentralized case, there is separated DQN model for each intersection. Hence, each DQN receives the state of its intersection (or optionally of other intersections also) and returns Q values for all possible actions in its intersection. The selected action and the reward are for the intersection.

If it is centralized, then the state should represent the whole network, while the action should represent all possible combinations of actions of every intersection in a network. In this case, the action space may be huge. For example, a 5×5 grid-network, with 25 intersections, each having 4 possible actions such as phases, results with totally 4^{25} #actions for the entire network.

Hence, for the purpose of scalability, we decided to use decentralized structure, similarly to [141], where each intersection learns its best policy independently. Similar to [141], shared parameters are used among all agents.

So far, we chose a decentralized structure, which treats every intersection as isolated, and separated from all the intersections in the network. This is in contrast to the centralized structure, which accounts for all the intersections when learning an optimal policy.

As our case is of decentralized MARL, we implemented local reward, state, and action, for each of the intersections. The state $s \in S = \mathbb{R}^d$ has d features, which could be any subset of the following features: speed, queue, travel-time, density, and the last applied action. While the action $a \in A$, where A is a set of the phases in the intersection.

However, as suggested in [41, 120], for a better cooperation among agents, we also considered implementing a global reward.

Additionally, later in Section 3.6.10, we implemented DDQN via GNN similarly to [157]. However, a neural fitted Q-iteration (NFQI) was used in [157], which is a batch model-free RL method, and for DDQN the authors implemented A2C or dueling network structure of splitting the state-action $Q(s, a)$ -value function into the state value function $V(s)$ and the advantage function $A(s, a)$. We, on the other hand, minimize the L1 norm of the loss described in (3.9), to learn the policy of each agent, and implemented GNN via STGCN.

Note that the first alternative (global reward) was utilized in the non-STGCN case, i.e. where the model is totally separated for each intersection. It is redundant in the STGCN case, since cooperation is applied already, due to neighboring aggregation.

The difference between centralized and decentralized DQN is shown in Fig. 3.3.

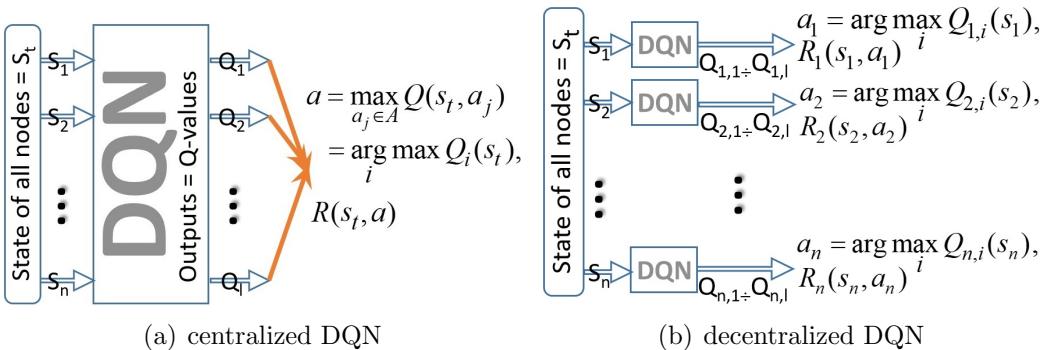


Figure 3.3: Different control structures in DQN (separated versus unified outputs).

Note that $l \neq n$ in Fig. 3.3(a), since $l = |A| = \#$ all possible action combinations in a network and each Q_i represents one Q-value, while in Fig. 3.3(b) each Q_i represents a set of Q-values for each specific intersection. For example, given four possible actions for each intersection, we have $Q_1 = Q_{1,1}, Q_{1,2}, Q_{1,3}, Q_{1,4}$ and $l = 4^n$.

Note also, that when cooperation is involved among agents, in Fig. 3.3(b), the Q-values and the rewards are dependent on additional states of other intersections.

The action space in this chapter is depicted in Fig. 3.4, with green lights for each movement. In this setup, each intersection is assumed to have four

approaches, where the left and the right movements with one lane each, and the straight movement with two lanes. Several sets of phases were considered, but due to large amount of hyper-parameters needed to be tuned, an arbitrary set was selected to be constant throughout all the RL tests. The networks consisting of these intersections are usually grids, and illustrated in Fig. 3.6.

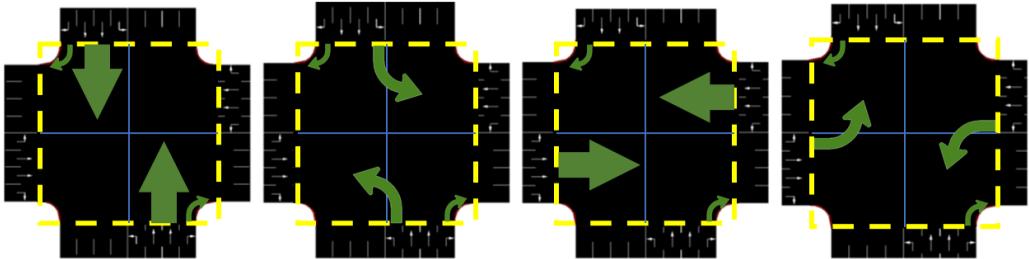


Figure 3.4: Each intersection has four phases of traffic light.

Note that our features are normalized, i.e. relative to their maximum values, in order to be compared among different intersections, especially when applying communication between them via GNN.

We had to decide about the reward $R \in \mathbb{R}$, for each agent (intersection), such that it will represent preference task objective. Hence, we have chosen to represent the preference via weights w , where each weight for its corresponded movement is multiplied with some evaluation measure chosen by the user as hyper-parameter. The evaluation measures could be either the last total delay or the queue length, in the movement. Thus, $w \in [0, 1]$, is a continuous variable, where 0 is for totally un-preferred movements, while 1 is for the totally preferred ones. Consequently, the dynamic objective (3.1) is derived. Note that objective in (3.1) is for minimization. However, since RL objective is to maximize the total reward, i.e. $J(t) = \max R(t)$, a minus sign was added to convert the minimization to maximization:

$$R(t) = - \sum_{i=1}^n w_i(t) d_i(t). \quad (3.17)$$

Among different approaches in RL, such as policy-based, model-based and actor-critic we chose the value-based approach, specifically the Q-learning approach, which maps states to Q-values, and then actions are selected by the maximal Q-values, see Section 3.4.2. We have not chosen AC approach since we produce reward after each step of simulation, hence we do not have sparse reward which requires AC approach.

Usual representation of Q-values is via a 2D table of actions (e.g. rows) and states (e.g. columns) with Q-values for each pair of state and action.

However, if the mapping is via DNN, then it is referred to as deep-Q-learning or DQN. The DNN receives the state in its input, and it outputs Q-values, i.e. accumulative expected reward in a given state s_t , for each of the actions $a_t \in A$, or: $Q(s_t, a_t)$. The output Q-values determine the action, via $a_t = \arg \max_{a_i \in A} Q(s_t, a_i)$.

3.5.2 Constraints

Unlike in general optimization formulation, where constraints are embedded in the problem definition explicitly, RL do not posses this feature. Hence, there is a necessity to address how TSCP constraints should be included in RL.

First, the initial results are executed with unconstrained implementation (constrained only with the definition of phases), and then afterwards, the results are executed with all the constraints needed for a feasible implementation.

Moreover, even in the feasible implementation, we try to induce only the minimal set of constraints, to make the problem as permissive as possible. For example, there is no phase order, and the action is selected from a set of phases. Also, while some works, e.g. [137], use the minimum green time as their step size, we consider it to be too coarse, which results with poorer performance, since we could use smaller step size to gain much more accurate feedback from the simulation environment.

Next, in our RL setting, we use fixed steps in controlling the traffic, and the actions we use are phases. Therefore, the only constraints applied here are those that embedded in the set of phases, i.e. reducing conflicts between movements, and maximizing the flow during each phase. However, to make the control feasible and physically applicable, we should include also the following constraints: (i) minimum green times, to ensure safe clearance of entering vehicles; (ii) maximum green times, to ensure that other movements receive green light also; (iii) penalty for the frequency of phase changes, to reduce wasted inter-green times.

Nevertheless, we could have several control policies dictated by the central control, e.g. a policy where the minimum green time is the step time of the control system, with maximum green time for scenarios in which roads/lanes are closed completely. This option is considered later in Section 3.6.

Many of the reviewed papers in RL literature do not include explicit information or methods of handling these constraints. However, only a few papers did include this information. For example in [132], in case of selecting a new phase, the next iteration will take place only after the minimum green

duration. Otherwise, it will take place after a fixed short pre-defined step. I.e. it applies the minimum green constraint externally to the optimization problem.

On the other hand, one could embed all the constraints in the reward (internally) via an appropriate form as Lagrange multipliers, i.e. by modifying (3.17) as follows:

$$R(t) = - \sum_{i=1}^n w_i(t)d_i(t) + w_c C, \quad (3.18)$$

where w_c is a coefficient that is relative to the preference weights w_i , and C is the form of some constraint.

However, we propose to handle all the constraints mentioned above similarly to [132]. That is, our goal is to induce as less constraints on the RL optimization process as possible, for the following reasons:

1. Even if we include these constraints in the reward (internally), as suggested via Lagrange multipliers, by adding some significant weight w_c - we still would have to enforce these constraints. I.e., in the reward function these constraints considered soft (with some relative importance compared to other weights), while in reality hard constraints will be applied anyway.

For example, if the new reward (3.18) yields in some cases that the minimum green time should not be applied in the next step, the program will override this decision and apply minimum green time anyway.

2. Additionally, in case that constraint parameters have changed (such as the minimum/maximum green time duration), we have to retrain the DQN. Hence, handling the constraints externally is preferred in this case, since it avoids the need for retraining.
3. Moreover, some of the cases we address in our problem include extreme preference weights, i.e. cases with closed roads, lanes, or intersections, where some of the constraints are not imposed.
4. Finally, including constraints internally results with increasing of state size, since additional phase memory for previous phases is required and the duration of a phase is needed to check if minimum green constraint holds. In contrast, externally applied constraints approach is better from this perspective. It eliminates the need to increase the state size, thus reduces the complexity of the problem.

Subsequently, we propose that any constraint that can be separated from the RL optimization process, should be applied externally, e.g. the minimum and the maximum green times constraints for all movements. Minimum green duration is applied after each new phase introduced by the RL, while ensuring all movements receive green times is applied by storing all the phases that were given r.o.w in the last pre-defined horizon, and then inserting the phase that did not receive green time, thus overriding the optimal RL policy. Both of these constraints are applied externally, via simple program, interfering the regular sequence of RL (where each step starts with an action and ends with receiving a reward as the effect of this action). Hence, we can view the program as logic (or symbolic) AI, acting as the higher level, controlling the RL level below, i.e. interrupting it when it fits necessary. This is schematically described in Fig. 3.5.

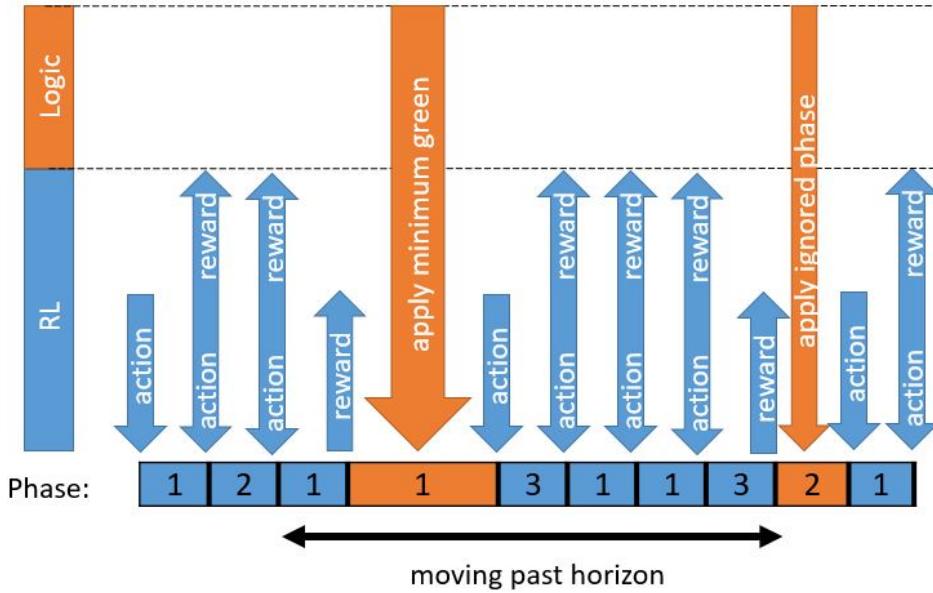


Figure 3.5: The RL system incorporating external logic to enforce constraints.

Fig. 3.5 illustrates logic as the external software, intervening the main RL process. Meaning, using some pre-defined moving horizon or a fixed number of steps, the logic rule determines whether the next step would be RL or overriding RL's next action with another action (phase) to guarantee the appropriate constraint, e.g. minimum green or maximum green. Minimum green is applied by prolonging the current phase if RL determined to change it though it had not provided with the minimum green duration, while maximum green is applied by switching to another phase, that had not been

applied for the last moving horizon number of steps.

The arrows represent interaction with the environment (simulation). The top-down arrows mean applying some action to the simulation, while the bottom-up arrows mean simulation output, in this case a reward, and as seen, it is retrieved only after RL's applying.

3.5.3 The implemented algorithm

To solve our TSCP, several experiments were conducted. We utilize the SUMO software as our simulation tool [162].

For the RL system we implement DDQN, similar to [163], only on different types of states and rewards. In the following, the DDQN Algorithm 1 is presented, then a detailed list of the pre-defined parameters is presented, and finally the parts of the algorithm are explained.

Algorithm 1 DDQN

Input: parameters

```
for episode=1, 2, ... K do
    Reset()
    while done == False do
        action ← Select_Action()
        next_state, reward, done ← Do_Step(action)
        Add_to_Memory(state, action, next_state, reward)
        stateB, actionB, next_stateB, rewardB ← MemoryBatch()
        Optimize_Model(stateB, actionB, next_stateB, rewardB)
        state ← next_state
    end while
end for
```

The parameters in the Algorithm 1 include:

- **state_type**: includes the features of the state, e.g. queue, speed, travel time, last applied action, weights, and more.
- **fixed_weights**: True if throughout the training the weights are fixed.
- **number of episodes**: represents the total number of training episodes that are executed for a given set of hyper-parameters.
- **do_demands_change**: True if demands are changed at each new episode.

- **STGCN parameters:** special parameters of the STGCN model. Parameters such as #spatial-temporal blocks, the type of each block (spatial or temporal), T or #past steps in the spatio-temporal input to STGCN, #layers, activation types (e.g. ReLU or Sigmoid), and more.
- **Optimization parameters:** batch size, gradient clipping value, target update (α, γ in updating rules, see Section 3.4.2), ϵ 's start and ending values (see ϵ -greedy policy in (3.11)), type of optimizer (e.g. Adam, RMSprop, ...).
- **Simulation parameters:** such as range of steps for each episode (starts after #heatup steps, and finishes at some maximum #steps)
- **reward_type:** could be the total delay time at each step or the relative total delay time between consequential steps. Similarly it could be the maximum accumulated waiting time across all vehicles for each intersection, and more.

The overall program for this chapter starts with preparing the simulation, creating the adjacency matrix of the network for GNN model, and initializing the DDQN agent class and the replay memory.

The Algorithm 1 includes the following procedures.

Reset() procedure is responsible for episode initialization. It decides whether a new set of weights should be generated for training, and whether new demands should be introduced in the simulation. Then, the simulation runs for some #heatup steps. At the end we produce our first current state. If $T > 1$ then we run the simulation $T - 1$ more times with random actions at each step, to eventually produce a state of T steps.

Select_Action() returns action for each intersection, following the ϵ -greedy policy (3.11).

Do_Step() applies the actions in the simulation, and returns all the relevant variables: the new state after the actions were applied (**next_state**), the reward at this new state, and *done* boolean value that equals False when the episode is ended.

The values (s_t, a_t, R_t, s_{t+1}) are added to the memory replay buffer. Next, we randomly choose a batch of these values from this memory, and insert them to the optimization function: **Optimize_model**. It checks first if the memory holds enough samples to generate a batch. Then, it calculates the loss, see (3.9). Next, it back-propagates and performs gradient clipping if necessary. Finally, it updates the Q function according to (3.13), and it updates Q_{target} either by copying weights of the Q DQN to Q_{target} DQN

$(Q \rightarrow Q_{target})$ after a fixed #episodes or according to polyak averaging:

$$Q_{target}(s_t, a_t) \leftarrow (1 - \tau)Q_{target}(s_t, a_t) + \tau Q(s_t, a_t). \quad (3.19)$$

Finally, the following state is assigned by **next_state**. Note that if the state is T -size, then it is updated in a cycling manner: step 0 is removed from it, and the new state is added as its final step T .

3.5.4 Networks and models

We first created simple traffic networks on which we applied our RL algorithm. We had two intersections or paired signalized intersections (PSI) network, and a few grid networks: 2×2 , 3×3 , and 5×5 networks, as shown in Fig. 3.6. We perform the analysis gradually, from the simplest network to the most complex one. Throughout the analysis, when a new network was introduced, a hyper-parameter search was conducted.

Note that all the networks shown here and in the results section are truncated for presentation purposes. The external road sections are much longer to ensure a fair comparison between low and high demand entering the networks. This ensures that queues and delays are calculated accurately, avoiding any virtual queues of vehicles attempting to enter the networks. The exact lengths of the links in the networks are 4000 meters for long external (source/destination) roads, and 500 meters for short internal roads.

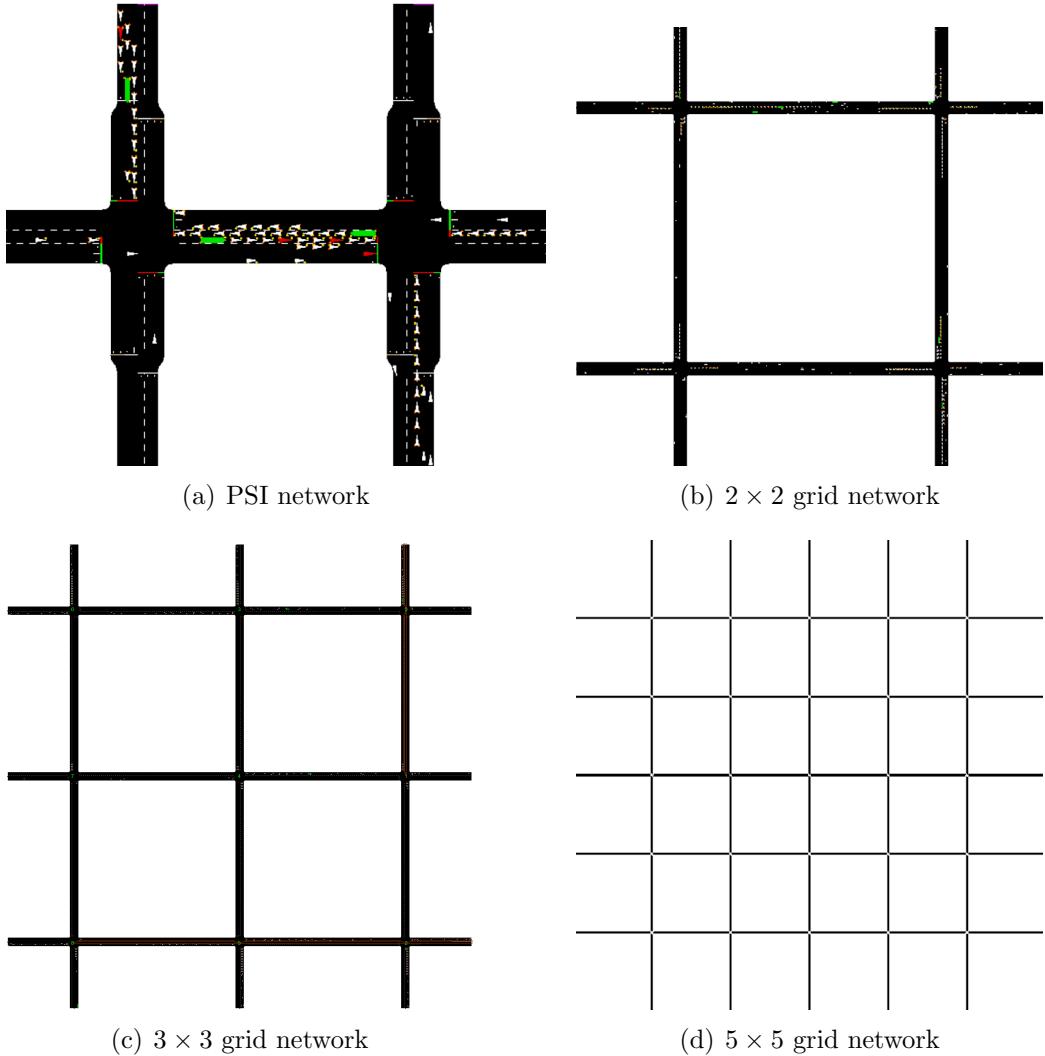


Figure 3.6: Networks used in this chapter.

The DNN models we used are based on the discussed in Section 3.5.1:

1. A simple DNN with FC layers, also referred to as FC NN or FCN. This model is trained solely on every intersection in the network.
2. STGCN model that runs once over all intersections, and then the FCN models above are connected to the STGCN output, running for every intersection.

We analyzed these models as separate instances for each intersection, versus one single model trained over all intersections one after the other.

Both these models are represented by a multi-dimensional state and action tensors: $s_t \in \mathbb{R}^{d \times T \times B \times n}$ where $d = \#$ features, $T = \#$ time steps, $B = \#$ batch size, $n = \#$ intersections; $a_t \in \mathbb{R}^{T \times B \times n}$. Those tensors include additional dimensions over those that were mentioned in Section 3.3: batch B , spatial n and temporal T .

Finally, we use any set of possible features (such as speed, queue length, and travel time), aggregated for each movement in an intersection. That is, d includes all selected features in all movements in a given intersection.

3.6 Results

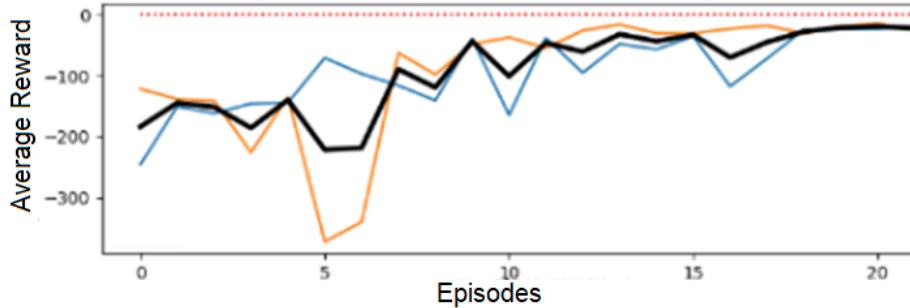
In the following we present the results gathered in this chapter. We start from examining a basic RL setup, with the static objective (3.2) and $w_i = 1 \forall i$. Then, while trying to solve our TSCP (3.1) we present traffic and preference issues encountered during the simulations. Finally, after attaining a working setup that implements preference commands in the network, we investigate different aspects of this setup. Aspects like cooperation among intersections (agents), different preference intensities, generalization, robustness, traffic constraints, and more.

3.6.1 No-preference preliminary results

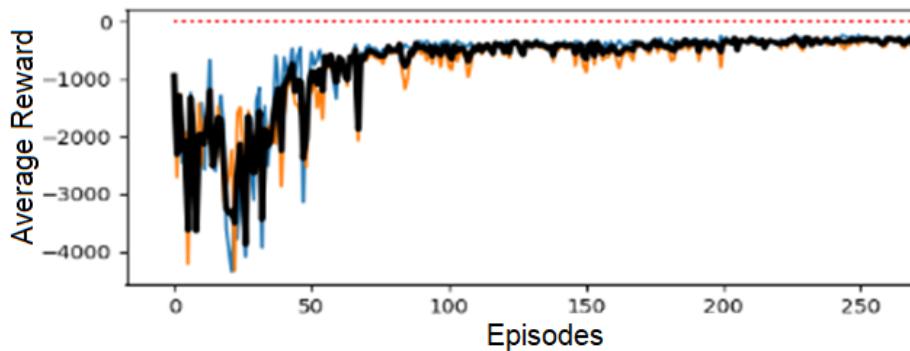
We start from the simplest case: paired intersections (Fig. 3.6(a)) with the static weights objective (3.2) ($w_i = 1, \forall i$), (input, output) $=(x, y)$ and the FCN model. The FCN model assumed here, is one that applies different FC modules for each intersection in the network.

At first we tested aggregated features for the links (road sections) entering an intersection, but obviously it did not work, since there is a lack of information to determine the appropriate action. Meaning the features were over-aggregated, which resulted with losing vital information for determining the next action/phase. It is because a phase is a function of the different movements in the road. Therefore, full information should include state for each movement, or more accurately for each lane. Therefore, all the following results were with the state consisting of features for each lane in every intersection.

We tested low and high demand, see Fig. 3.7. The tested demand was homogeneous in all movements, i.e. similar demand profile for all possible origin-destination (OD) combinations in the network. The high demand was simply a multiplication of 10 with comparison to the low demand, for all movements.



(a) Low demand



(b) High demand

Figure 3.7: Different demands in paired intersections network, fixed weights.

The vertical axis denotes the average training reward at each episode t , and it is defined as follow: $\bar{R}_t = \frac{1}{N} \sum_{\tau=0}^{N-1} \sum_{i=0}^n R_{\tau,i}$, where N =total #steps in a given episode, n =total #intersections. This average reward represents the overall network reward, which is the mean of the rewards at all intersections. The reward for each intersection is the total vehicle delay at that intersection, calculated as the sum of the cumulative waiting times of all vehicles at that intersection up to the current time step.

The orange and blue curves are each of the two intersections, and the black curves are their average.

We see that for low demand the average reward reached the optimum in 20 episodes, while for much higher demand in the same network, the average reached its optimum, after saw profile behavior during almost 90 episodes. Note also that the optimum is different.

Afterwards, we compared the RL optimization to other policies (how action is selected for the next step in simulation): random and fixed, as shown in Fig. 3.8. We see that the fixed policy, i.e. selecting the next phase in a fixed order from the phases set as in fixed signal planning, performs the

worst. It is also shown that the random policy, i.e. where the next phase selected randomly, performs pretty constantly in average. While the optimal policy converges to a significantly better average reward.

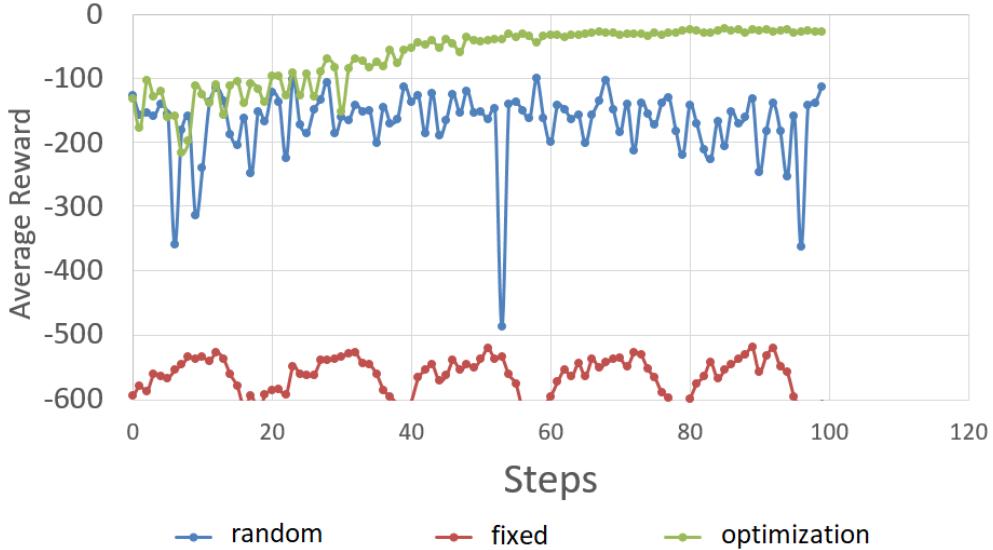


Figure 3.8: Comparison between three different policies.

3.6.2 Issues observed during training

In this stage, we analyzed the 2×2 network (Fig. 3.6(b)). We tested preference in the reward, as defined in (3.1), while the input to the DQN, i.e. the state, remains unchanged, i.e. $(\text{input}, \text{output}) = (x, w \cdot y)$ where x is the state tensor including the preference weights and the $w \cdot y$ is the output y or the reward obtained from applying an action: total weighted delay in an intersection.

We trained the DQN specifically on random preference weights within bounded range: $w_i \in [0, 1], \forall i$. However, it yielded no significant change from static weights, i.e. no visual preference according to w_i was observed in the simulation. We assume, it is because merely random weights, in average, perform similarly to static weights.

Hence, to examine this hypothesis, we decided to test extreme and discrete values of w_i instead of the previous random values in the range $[0, 1]$. More precisely, we tested only on two values for simplicity: $w_i = 0.99$ and $w_i = 0.001$. We chose randomly a small set of movements to apply extreme preference ($w_i = 0.99$), while all the rest of the movements in the network were highly un-preferred ($w_i = 0.001$), i.e. X1000 of relative difference between preferred and un-preferred movements to have stronger effect.

During testing the hypothesis above, we noticed some issues with the ability to learn optimal control policy. These issues can occur in any feedback-control approach:

1. **Issue 1:** When the queue of some lane a is close to its full capacity, vehicles coming from a nearby lane b cannot enter it, thus preventing the vehicles behind to proceed forward in lane b . This creates unnecessary gridlock, and a wasted green signal for lane b , see example in Fig. 3.9(a).
2. **Issue 2:** When the downstream movements are in congestion, no incoming vehicle from the upstream can enter it. This results also with wasted green light for the upstream.

These issues negatively affect the learning of an optimal policy via RL, since the proper actions do not affect their corresponding movements. Hence, our suggestion is to handle the circumstances that result with these issues and thus avoid them.

There are several ways to handle the discussed issues. One approach is by noticing how low-resolution state space limits obtaining the necessary knowledge for an agent to decide a correct action. In issue 1, it lacks the distribution of vehicles over the road section, while in issue 2 it lacks the communication between upstream and downstream movements. Hence, we can extend the state to include more information, e.g. include density or other features, or include downstream features. Another approach to solve these issues, is to encourage cooperation among intersections (agents). As discussed in Section 3.3, it could be done either by introducing global reward or by exchanging neighboring information among intersections (agents). Both of these approaches are tested in the following sections, see Section 3.6.5 and Section 3.6.10.

Fig. 3.9(a) shows an example for issue 1, where vehicles try to enter to the left lane to turn left in the intersection, are thus block the vehicles behind in proceeding straight in their lane to go straight in the intersection. Fig. 3.9(b) shows an example for issue 2, where the congestion in the east downstream road causing a wasted green light in the upstream road in the west, since the vehicles there cannot enter the downstream. We can see how this situation causes a total gridlock in the intersection, which also propagates to further intersections. It starts by the west upstream vehicles inability to exit the intersection, thus creating a congestion in this west road. Then, since RL or any optimal control method has the objective to reduce the total delay in upstream, it gives green to reduce the accumulated congestion in the upstream (we assume the downstream is under control of another

intersection). However, this green has no effect, which causes the controller to keep providing green time for the west-to-east movement. This in its turn, provides red light to the opposite movements, i.e. south and north, which subsequently drive these roads also into congestion.

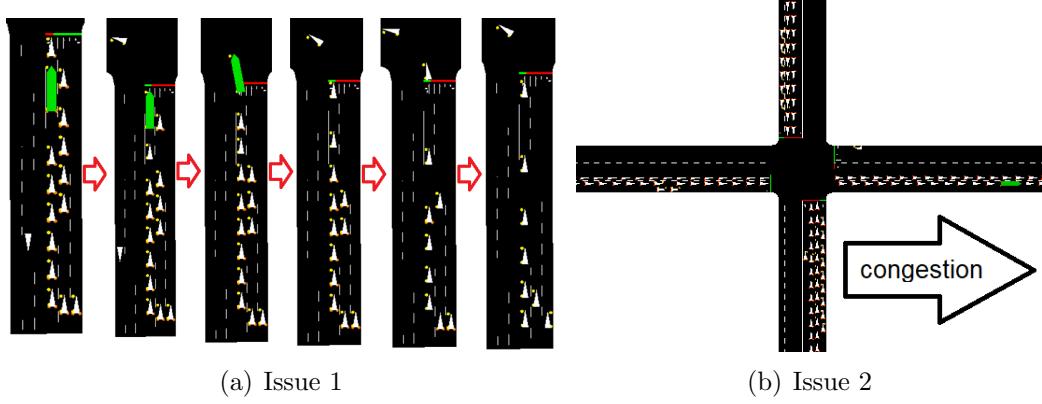


Figure 3.9: Different issues preventing a proper optimal policy learning.

[120, 140] also emphasize the difficulty of RL in highly congested conditions, due to similar issues as above. It is basically since the reward stops being consistent and it is barely affected by the selected action. Thus, yielding the difficulty of learning. Hence, it is recommended sometimes to apply different reward functions depending on the network situation (low, medium or high congestion), [138].

Finally, there was an issue with the preference setup described in the beginning of this section. That is, $(\text{input}, \text{output}) = (x, w \cdot y)$ where x, y are the state and the reward. It worked properly only for fixed weights during each training episode, but it did not work for changing weights in an episode. We assumed it was because $w \cdot y$ output is like non-stationary reward, or a dynamic objective, see a further discussion in Section 3.7.

3.6.3 Deciding upon most appropriate (input, output) for preference command

The current issue we had to solve is the dynamic objective in the reward. Till now we tried to reach this goal by including weights separately as any feature - in the state. Alternatively, we considered an option to embed the weights in the state.

Therefore, we introduce artificial state features. Instead of using the state as it is, we multiply its features with the preference weights. Thus, adapting

the model to regard the state as if it had more or less of the feature than it really does. For example, to see as if a queue is much longer if we would like it to be preferred.

Consequently, the first preference setup to be examined was (input, output) = $(w \cdot x, y)$. Visually (in the simulation) it was allegedly seemed that this setup is succeeding in implementing preference. However, it turned out to be wrong, since the demands in the network were stochastically distributed. Meaning, it just happened to be, that the preferred movements did receive preference, i.e. resulted with less demand. However, it was a wrong interpretation of the results. It is because the network was behaving similarly either with or without preference weights, i.e. the preferred movements had low demand in the first place.

Subsequently, for a better preference examination, an even distribution of demands was applied in the transportation network. This way, employing preference weights would be noticeable (since it will break the demand symmetry in the network). In other words, homogeneous demands were implemented, i.e. demands that are identical in all directions. Additionally, the demands over all left and right movements in the network were reduced significantly, to mitigate the issues of type 1, see Fig. 3.9(a).

In these homogeneous conditions, we tested on different combinations of (input, output) including multiplying these weights: $(w \cdot x, y)$; $(w \cdot x, w \cdot y)$; and $(x, w \cdot y)$.

Eventually, the best setup was $(w \cdot x, w \cdot y)$, which is logical, since it makes the dependency of the state with action consistent, see Fig. 3.10. As seen, consistency preserved when either both multiplied by w or not.

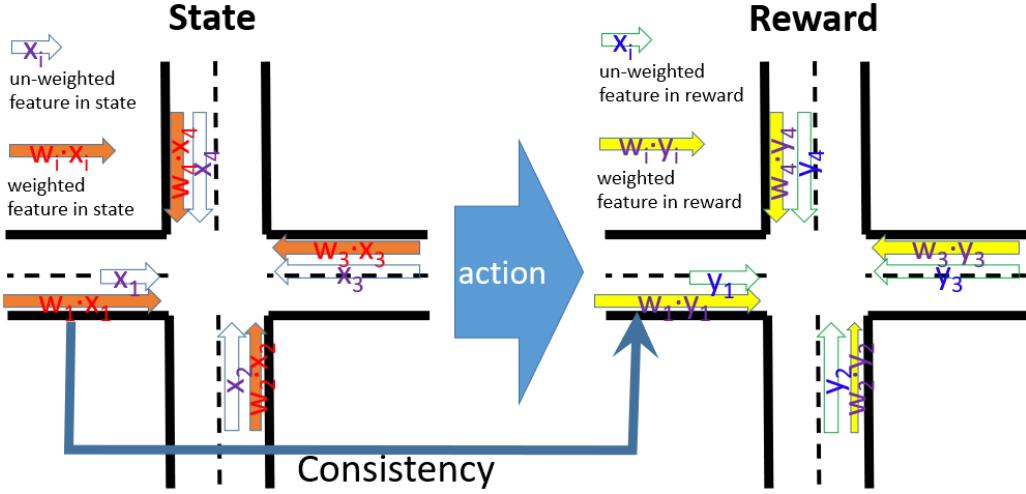


Figure 3.10: Different state and reward combinations, each with un-weighted or weighted features (legend is on the top-left of each intersection).

3.6.4 Issue with preference deployment

In this stage, we trained the RL on high preference on one movement at each intersection in the network. Eventually, there was a new issue while implementing preferences in the network: conflicts between preference movements, see for example Fig. 3.11. The figure demonstrates a preferred route indicated by blue arrows. The green numbers in the middle of each intersection represents the current applied phase, by its ID. The curved white drawing in the middle of the image is for removing the most part of the long vertical road sections. As seen, "Preference 4" road section is mostly applying green signal for the south-to-north direction. Meanwhile, "Preference 1" road section is mostly applying green signal for the west-to-east direction. These two road sections cause a congestion in the red marked road section, since "Preference 1" constantly adds new vehicles to the marked road, while "Preference 4" does not permit the marked road enough green time to dissolve the accumulated vehicles in it.

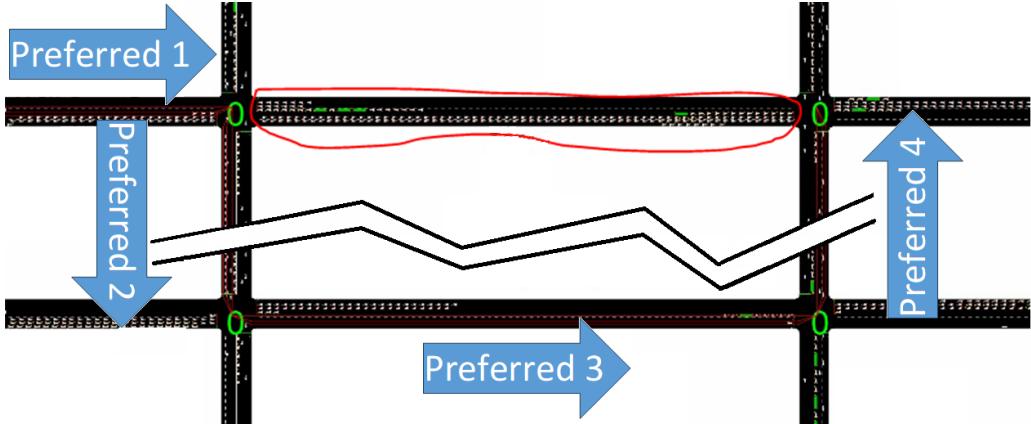


Figure 3.11: Preferred movement 1 is in grid lock due to preferred movement 4. The preferred trajectory comprises of four preferred movements.

This example demonstrates failure in preference planning.²

Note that the training so far was only in extreme and special conditions: preferred $w_i = 0.99$ set on the straight movement at each intersection, while all left and right movements were $w_i = 0.001$. Hence, good results were obtained only on scenarios fitting these conditions. Poor performances occurred in testing also left or/and right preferred movements, or on intersection without any preference. It means it does not generalize on out-of-distribution cases³, i.e. it is not extrapolating but only interpolating, as expected in supervised learning in deep learning.

Similar bad performance occurred in another out-of-distribution case. In this case the trained RL was tested on random preference weights over randomly chosen 0 up to 2 movements in every intersection in the network. It is different from the training setup, since then only one movement was chosen randomly in every intersection.

3.6.5 Hybrid reward

We also analyzed the hybrid reward mentioned in the literature review, i.e. a reward of convex combination: $\beta \cdot \text{reward}_{\text{global}} + (1 - \beta) \cdot \text{reward}_{\text{local}}$. We tested it on the FCN model, with different values for β , as shown in Fig. 3.12. It

²Such issue can be avoided by higher-level commands, in which the agent designs the preference weights autonomously, see Section 6.3.

³While training on specific dataset in supervised learning, the test performs well only on the same data distribution as the one was in the training phase. As the tested distribution diverges from the trained distribution the performance is poorer. This scenario is referred to as "out-of-distribution".

can be seen that the best results are obtained for $\beta = 0.5$. The results show that it is beneficial to include a global reward for a better consideration of neighboring traffic, but not too much, such that it will lose its unique and consistent state-to-action mapping.

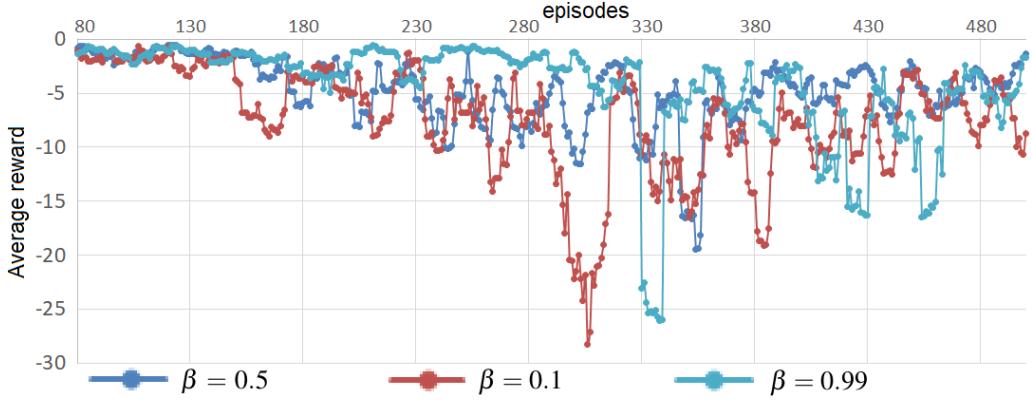
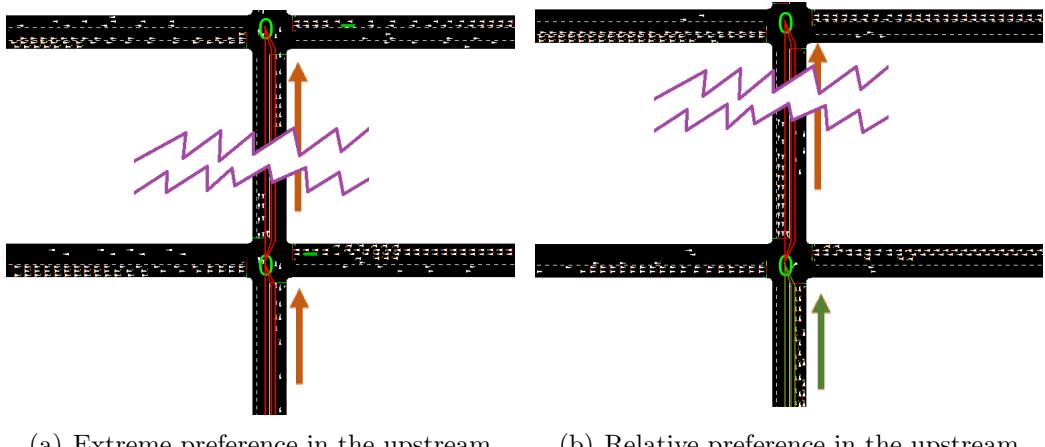


Figure 3.12: Comparing different hybrid rewards, in training the FCN model.

3.6.6 Relative versus extreme preference

In this stage, we trained and tested on non-extreme conditions, i.e. relative preference weights. Fig. 3.13 shows an example of comparison between extreme preference (marked with orange arrow), with $w_i = 0.990$ for preferred movements and $w_i = 0.001$ for all rest movements, and relative preference (marked with green arrow), which is smaller than the extreme preference that has been so far. I.e., $w_i = 0.010$ for preferred movements and $w_i = 0.001$ for all rest movements.

Fig. 3.13(a) shows almost free flow in the south-to-north for the bottom intersection, while accumulated traffic in west and east movements. Also, upper intersection has some small queue in its south-to-north movement. However, in Fig. 3.13(b) due to lower preference in south-to-north movement in the bottom intersection, there is accumulation of traffic in this movement, while the movements in the west and east are less congested compared to the extreme preference scenario. While in the upper intersection there is no queue at all, since a much smaller flow comes from the bottom intersection.



(a) Extreme preference in the upstream (b) Relative preference in the upstream

Figure 3.13: Comparison between relative and extreme preferences.

3.6.7 Generalization from 2×2 network to 5×5

Next, we tested the large 5×5 network, see Fig. 3.6(d). The previous FCN model was learned for an individual intersection in the 2×2 network. Therefore, we can examine how this model generalizes, within the same conditions trained on the 2×2 network, to the 5×5 network, without being trained on it. For similar scenarios, we compared this model with the same FCN model, but after being trained on the 5×5 network. We can see the comparison in Fig. 3.14.

The different scenarios, enumerated from 0, are different preference weight distributions in a 2×2 network, simulated under the same conditions, i.e. the same demand distribution in the 2×2 network. The comparison with 5×5 network was conducted in its upper right corner, within a 2×2 sub-network of the 5×5 network.

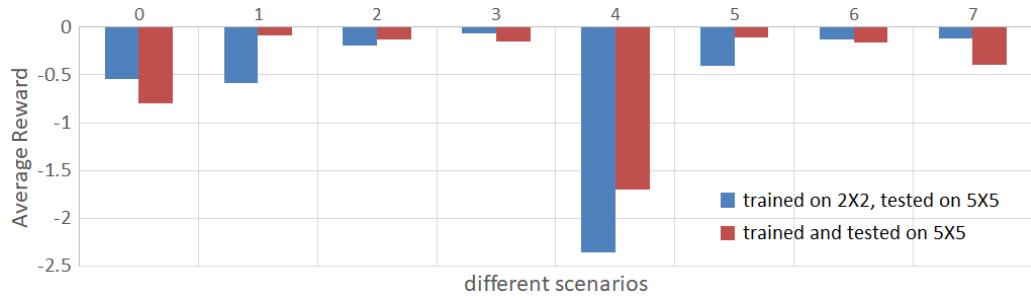


Figure 3.14: Comparison of DQNs trained on 2×2 and 5×5 networks, in different scenarios.

Similar idea was demonstrated in [128] via *transfer planning*, where smaller solutions were used to combine into a larger solution, see Section 3.3. We equivalently examined learning a single intersection model from 2×2 grid network, then applying it to the 5×5 grid network.

3.6.8 Fixed versus changing preference weights

Here, we compared fixed weights versus changing weights, during every training episode, with and without including noise in the state/input of the system, to examine robustness. In other words, we examined how a fixed-preference trained DQN performs in comparison to a changing-preference trained DQN, in a different situation than the one they were trained upon. This test situation is considered as noise in the system's input/state.

Obviously the fixed-preference DQN should perform better than changing-preference DQN, in the fixed-preference scenario that the fixed-preference DQN was trained upon. However, in cases of different scenarios, it is the changing-preference DQN that should prevail (perform better), since it has learned to be adaptive to different preference weight sets.

For scalability reason, we have chosen a single model to be learned, and to be trained on every intersection. While testing different hyper-parameters, we always encountered unexpected results while comparing different fixed-preference DQNs among themselves. Meaning, such comparison should show best performance when tested on the same fixed preference weights. Each DQN should have been the best at its own preference set. However it was not the case.

Consequently, a few things were changed to solve this problem:

1. we reduced the number of left/right turns to reduce Issue 2, see Section 3.6.2.
2. we did not store (s_t, a_t, R_t, s_{t+1}) tuples in the replay buffer, that did not exhibit the maximal saturation flow during green time.
3. Since we trained the RL on robust conditions, i.e. over changing demands, thus the test set should not perform well on some specific demand in the feasible range, but rather over an average of several tested demands in the range, since the learned model is conservative. Meaning, it is good for the range, and not for specific demand in it. Hence, we tested each training DQN model over several sets and averaged their outcomes. Also we kept the customary ratio between training and testing sets in ML: e.g. (train,test)=(80%,20%).

4. Fixed identical initialization for all DNNs (same seed), to make fair comparison.
5. Created new state for every intersection, $Network_state$, which is the concatenation of its own state (previous state definition) with the states of all intersections in the network. The advantage here is that it takes into account all available information, to select the best action. However, the drawback is the huge state dimension, which is slowing the learning significantly.
6. Changed $(state, reward) = (queue, delay)$ to be $(state, reward) = (queue, queue)$, since reward being delay is correlated with state only in specific cases, e.g. in homogeneous demands. Hence, $(state, reward) = (queue, delay)$ performed well when the DQN was trained on fixed set of demands. However, in changing demands (heterogeneity) it may did not have consistency. Therefore, in changing demands case the reward and the state should hold the same information, e.g. queue.

The last change is related to a much broader issue: the selecting of reward and state is mostly heuristic in RL, since designers assume that the RL can learn anything. However, in general it is not true, because one has to make sure RL assumptions hold, e.g. consistency in the relations: $R = R(s, a)$ and $s_{t+1} = f(s_t, a_t)$.

These changes made better performance in 2×2 network, but still had bad results in 3×3 and 5×5 networks. Therefore, from this moment we turn totally to study the 3×3 network case. Although, few changes were remained: 1, 3, 4, and 6.

The preference weight scenarios that were chosen as test set for different models, are illustrated in Fig. 3.15.

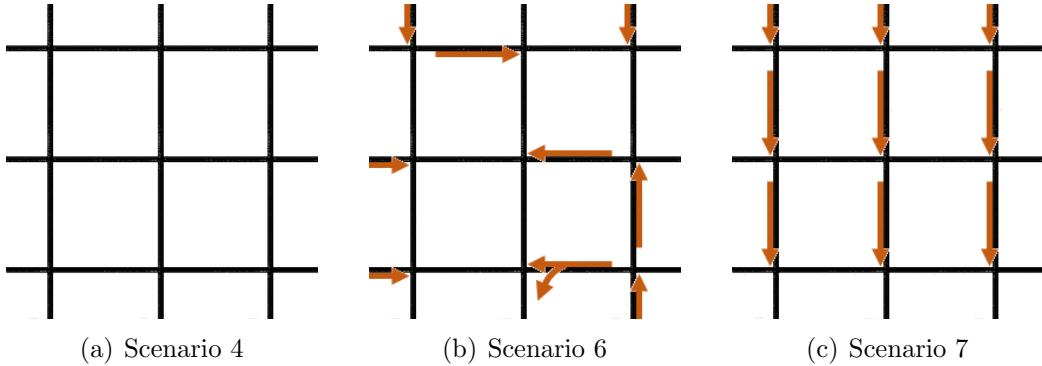


Figure 3.15: Some of the preference weight scenarios for testing, in 3×3 network.

In total there are seven 3×3 test scenarios, ranged in 4...10. Scenario 4 is without any preferences. Scenario 7 is with identical preferences for all intersections. Scenario 6 is a random setup of preference weights. Scenarios 5, 9, and 10 that are not shown in Fig. 3.15 are also random setups of preference weights. Scenario 8 is similar to scenario 7, only with horizontal preferred movements. Finally, scenario 10 is similar to scenario 9, only with relative preference values, i.e. $w_i = 0.5$, instead of the regular extreme values ($w_i = 0.99$).

We used a single FCN model to learn for every intersection in the network. This DQN setup resulted with poor performance. It was because the single FCN model was trained in scenarios where fixed-preferences were applied on different movement configurations within each intersection. That is, it learned best only on exactly the same movements preferred in all intersections, e.g. the west-to-east movements in all intersections. Hence, we switched back to each DQN model learning its own specific intersection, see Fig. 3.16, which is also correct from a traffic engineering point of view: the network has spatial distribution, i.e. every intersection behaves differently depending on its location in the network, e.g. central intersection is different from peripheral one. Fig. 3.16 compares the previous configuration (a) to the new one (b). The same color in (a) represents same model, while different colors in (b) represent separate models for each intersection.

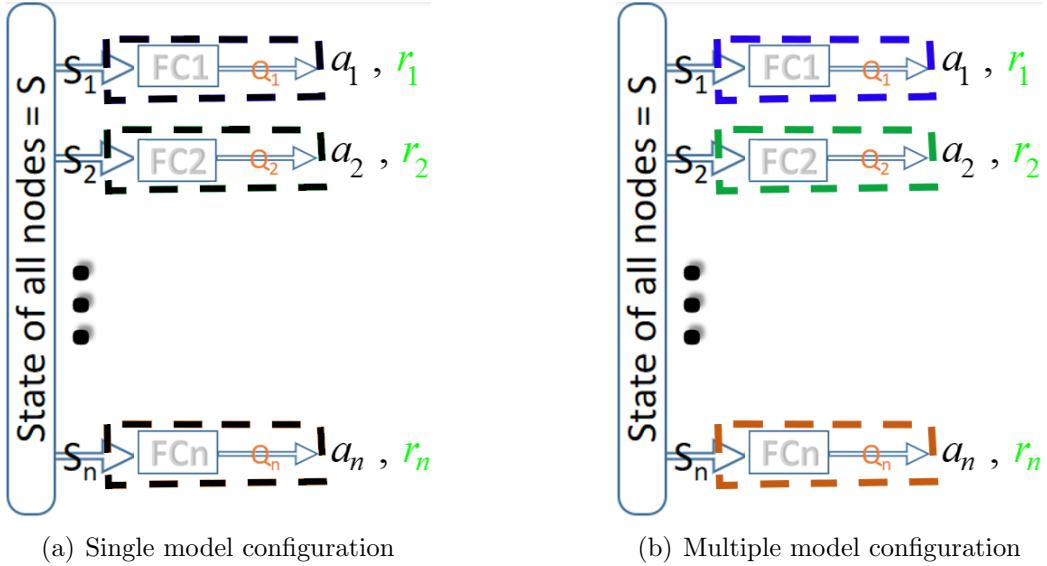


Figure 3.16: Different model configurations of DQN (r =reward).

Next, the multiple model configuration was examined, see Table 3.2. The columns and the rows represent different preference-weights scenarios, i.e. different sets of preference over the given transportation network. The columns represent the DQN models that were trained over each of these scenarios, while the rows represent the tested scenarios. It means that each model that was trained on some of these scenarios should be the best while tested on the same scenario. Hence, we marked in underline and tilted font the diagonal in each table to represent that the best value is expected to be in this cell. We marked in underline and bold font the actual best value obtained, or several cells if they had similar best values. The values in the tables are the average total reward value in all intersections of the network, over a specific test episode of each of the DQN models.

The multiple-model was run for two sets of episodes: 200 and 600. At 200 training episodes the results showed sub-optimality . Hence, we tested further, for 600 episodes. In this case we got better values, and much more optimal. These values are shown in Table 3.2.

As shown, the first 200 episodes yield non-optimal solutions, since the diagonal is mostly contain not the best values. While the 600 episodes yield mostly optimal values in the diagonal. Moreover, the values in Table 3.2(b) are better than the corresponding values in Table 3.2(a).

Additionally, Table 3.2(a) is not similar to Table 3.2(b) since all the rest of the values corresponding to Table 3.2(b) are not optimal, such as models 9, 10, and changing models. Hence, they were removed from the table.

set	4	5	6	7	8	set	4	5	6	7	8	9	10	change1	change2	delta (%)
4	-0.694	-1.188	-1.217	-0.995	-0.808	4	-0.5039	-0.60469	-0.69916	-0.61451	-0.4962	-0.71607	-0.88558	-0.75672	-0.64977	-14.1337
5	-0.022	-0.04	-0.042	-0.036	-0.024	5	-0.01766	-0.01076	-0.01874	-0.0197	-0.01261	-0.0129	-0.02744	-0.01857	-0.01803	-2.88616
6	-0.222	-0.384	-0.35	-0.365	-0.238	6	-0.1267	-0.12633	-0.10011	-0.17464	-0.12415	-0.1384	-0.28738	-0.14846	-0.15078	1.56049
7	-0.24	-0.279	-0.283	-0.3	-0.255	7	-0.18474	-0.238	-0.20813	-0.07915	-0.16618	-0.25187	-0.15478	-0.24451	-0.22135	-9.47108
8	-0.048	-0.244	-0.317	-0.188	-0.05	8	-0.07828	-0.11114	-0.30518	-0.20446	-0.02995	-0.0785	-0.21547	-0.11805	-0.08511	-27.8992
						9	-0.21857	-0.26452	-0.28041	-0.31977	-0.1428	-0.1723	-0.29349	-0.31646	-0.28012	-11.4832
						10	-0.01459	-0.01205	-0.01458	-0.01488	-0.00671	-0.00997	-0.00638	-0.01364	-0.01402	2.79749

(a) After 200 episodes training

(b) After 600 episodes training

Table 3.2: Fixed and changing preference weights, via multiple model configuration, see Fig. 3.16(b).

Table 3.2(b) has two columns for changing preference weight training, where the first (change1) is with $\alpha = 0.00002$ (target update coefficient), which is also in all the fixed preference training models in previous columns (4 to 10). The second column (change2) is with $\alpha = 0.002$. Delta column presents the decrease in percentage of change2 compared to change1. As seen, in average the higher α parameter has better results, since 0.00002 might be too slow learner.

Nevertheless, even for the change1 column, with the same α value as for the fixed models in columns 4...10, we see that for each set, the changing average reward is in the range of best and worst of the fixed models, i.e it is worse than the best model for a given set but it is better than one or more of the rewards of other models on this given set. Also, note that we have only the best values for each set, but not the worst, so the changing performance might be even better.

Eventually, we see robustness when training on changing demands, compared to training on fixed demands.

Next, we changed the state configuration again. Our original definition was consisting of upstream data, i.e. only incoming movements in an intersection. But we gave each intersection more information about the whole network, denoted as *Full_state*, where unlike our usual definition of state - it consists both of upstream data and downstream data.

We tested both fixed-preferences and changing-preferences training, as seen in Table 3.3. Each of those types included three trained models. Then, for each of these models we tested with the same hyper-parameters: with downstream features in the state and without them. The values as usual are the average total reward over all intersections in the network. Only note, that for the sake of clear visualization, the values presented are the absolute

Type	fixed-preferences					
Model	4		6		10	
downstream	with	without	with	without	with	without
test 4	0.212	0.297	0.217	0.31	0.226	0.302
test 5	1.12	1.361	1.185	1.363	1.051	1.525
test 6	1.953	2.431	1.851	2.422	1.866	2.516
test 7	0.499	0.646	0.524	0.627	0.507	0.642
test 8	2.781	3.561	2.719	3.534	2.832	3.72
test 9	0.23	0.345	0.245	0.34	0.237	0.342
test 10	0.269	0.39	0.268	0.407	0.269	0.354
average	1.009	1.29	1.004	1.286	0.998	1.343

Type	changing-preferences					
Model	change1		change2		change3	
downstream	with	without	with	without	with	without
test 4	0.239	0.241	0.269	0.276	0.299	0.298
test 5	1.063	1.031	1.183	1.335	1.423	1.34
test 6	1.939	1.935	2.15	2.495	2.407	2.569
test 7	0.486	0.486	0.55	0.588	0.585	0.643
test 8	2.924	2.989	3.297	3.728	3.602	3.79
test 9	0.244	0.246	0.267	0.279	0.343	0.347
test 10	0.274	0.277	0.348	0.318	0.403	0.431
average	1.024	1.029	1.152	1.288	1.294	1.345

Table 3.3: Comparison of different models in different scenarios, for states with and without downstream features.

and multiplication by 100 of the original values.

We see that optimality of the fixed-preferences are consistent for all scenarios (4,6,10), for states with downstream features, and they are also better than the changing-preferences in the appropriate scenarios.

We also see, that except a minority of cases, most of the values for states with downstream perform better than those without downstream features, and definitely better while averaging over the tests.

3.6.9 Green constraints

So far, we have applied only phases set constraint, i.e. eliminating conflicts during a green light in an intersection, and maximal flow during this phase. However, we did not include minimum and maximum green duration constraints over phases, which is necessary since we do not apply cycles but rather fixed time steps.

Table 3.4 presents a comparison of the so far phase constraints results with the case of including those two green-duration constraints, and it is formatted similarly as Table 3.2. This inclusion of green duration constraints is illustrated in Fig. 3.5.

We can see in Table 3.4(a), that on the one hand including constraints actually improves the reward, which is logical since it eliminates all sorts of issues we encounter. However, on the other hand, applying externally constraints negatively affect the RL optimization, since it inserts unplanned intermediate phases that produce new states. We can see that we do not learn optimal rewards for fixed preference cases. It is shown by comparing Table 3.4(a) to Table 3.4(b), where on the one hand the values in the diagonal are more optimal, while on the other hand are worse (smaller average rewards).

Also, we examined changing preferences in Table 3.4(a), and seen that due to the damaged optimality, the results are not always robust, i.e. not in the range of best to worst values for each tested scenario (sometimes it is worse).

Finally, in different results examined but not presented, we observed that queue length feature in the state as also the reward has performed better than density feature. This issue should be further investigated in a future research.

3.6.10 STGCN

In this stage, STGCN model was introduced. The STGCN model is a distributed model, i.e. it is a multi agent configuration, but unlike the previous

set	4	5	6	7	8	9	10	changing
4	-0.00212	-0.0023	-0.00213	-0.00251	-0.00177	-0.00236	-0.00226	-0.00217
5	-0.0112	-0.01016	-0.01089	-0.01065	-0.01071	-0.01234	-0.01051	-0.01185
6	-0.01953	-0.0188	-0.01739	-0.01926	-0.01724	-0.01852	-0.01866	-0.01851
7	-0.00499	-0.00471	-0.00501	-0.00509	-0.00502	-0.00495	-0.00507	-0.00524
8	-0.02781	-0.02806	-0.02306	-0.02641	-0.02247	-0.02766	-0.02832	-0.02719
9	-0.0023	-0.00249	-0.00234	-0.00237	-0.00221	-0.00239	-0.00237	-0.00245
10	-0.00269	-0.00286	-0.00262	-0.00282	-0.00233	-0.00274	-0.00269	-0.00289

(a) With green constraints

set	4	5	6	7	9	10
4	-0.00235	-0.00303	-0.00242	-0.00341	-0.00317	-0.0029
5	-0.01153	-0.007	-0.01534	-0.01262	-0.0113	-0.01309
6	-0.02376	-0.02464	-0.02282	-0.02662	-0.0249	-0.0248
7	-0.00601	-0.00538	-0.00691	-0.00605	-0.00628	-0.0062
8	-0.03004	-0.03571	-0.02831	-0.03684	-0.0385	-0.03524
9	-0.00265	-0.00298	-0.00293	-0.00322	-0.00301	-0.00315
10	-0.00319	-0.00391	-0.00358	-0.00431	-0.00345	-0.00332

(b) Without green constraints

Table 3.4: Comparing different models in different scenarios, with and without applying minimum and maximum green duration constraints for every phase.

setup of FCN models representing a single intersection - this configuration enables communication between agents. It is since STGCN is a graphical NN, which utilizes convolution over neighbors of every agent, i.e. shared NN weights for neighboring relations in the graph NN.

The STGCN was compared to FCN with different hyper-parameter configurations, on the 2×2 , 3×3 and 5×5 networks, each with different fixed weight scenarios.

First, we test the STGCN model for the 3×3 network. The results are shown in Table 3.5. The table contains several hyper-parameters: `fixed_weights`, `is_single_model`, `set`, `sconv`, `Model` and T . `sconv` is the spatial convolution block type in the STGCN, which is one of its parameters. T is the number of past steps included as an additional dimension in the state tensor. `Model` represents the type of the DQN model: STGCN or FCN. The `set` holds the fixed preference weight scenario that the model was trained upon (if indeed). The test rows are similar to those in Table 3.3. Similarly to Table 3.3, the values of Table 3.5 are absolute reward values and multiplied by 100, for visualization reasons.

Name	Model1	Model2	Model3	Model4	Model5	Model6	Model7	Model8
<code>fixed_weights</code>	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE
<code>is_single_model</code>	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE
<code>Model</code>	STGCN	STGCN	FCN	STGCN	STGCN	STGCN	STGCN	STGCN
T	1	1	1	1	1	4	1	1
<code>sconv</code>	4	4	4	4	4	4	5	5
<code>set</code>	-	-	-	4	8	-	-	-
test 4	0.241	0.27	0.298	0.245	0.295	0.302	0.265	0.257
test 5	1.031	1.339	1.34	0.59	1.214	1.029	1.019	1.199
test 6	1.935	2.275	2.569	1.845	2.338	2.121	2.011	2.307
test 7	0.486	0.536	0.643	0.234	0.483	0.562	0.481	0.53
test 8	2.989	3.629	3.79	3.603	3.904	3.915	3.409	3.983
test 9	0.246	0.295	0.347	0.251	0.295	0.284	0.262	0.275
test 10	0.277	0.285	0.431	0.307	0.35	0.336	0.308	0.357

Table 3.5: Comparing different models (mainly STGCN) with different parameters and scenarios in 3×3 network.

From Table 3.5, we see that STGCN outperforms when its FC layers are all of the same model (`is_single_model=TRUE`) compared to when each intersection has different FC layer (`is_single_model=FALSE`). This is shown in the comparison of `Model1` column versus `Model2` column, for every row (scenario). It fits the fact that STGCN itself is convolutional, i.e. shared parameters are learned for any node and its neighbors, in the graph network. Hence, since STGCN does not distinguish the different intersections in a network, similarly it should be further processed via shared FC layers. See illustration of this setup in Fig. 3.17.

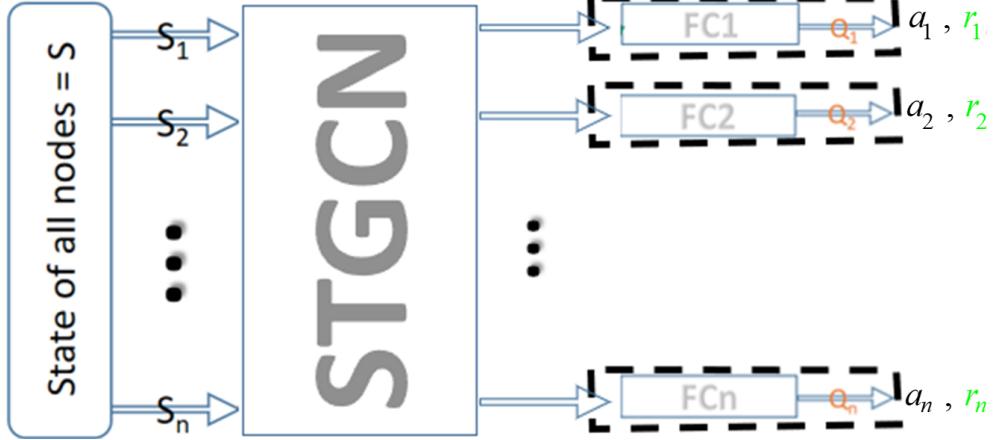


Figure 3.17: STGCN setup: a single and shared FCN model is attached to STGCN for every intersection (r =reward). Similar to Fig. 3.16(a).

Additionally, it is shown that for $T = 4$ (Model6 column), the model performs similarly to the model with $T = 1$ (Model2 column). And when using GAT as the spatial convolution module in STGCN (sconv=5), which utilizes attention, the model behaves similarly to the model with regular GCN spatial convolution module (sconv=4). This is shown by comparing between Model1 versus Model7 and between Model2 and Model8. In average GCN performs slightly better than GAT. It is shown also that STGCN (Model2 column) is better than FCN (Model3), for changing preferences training. STGCN with single FCN models (Model1) performs even better than FCN (Model3). We do not compare to single model FCN, since as was explained and illustrated in Fig. 3.16, single model FCNs do not behave optimally. In conclusion, we do comparison between single and multiple FCN models in STGCN, but we do not compare for FCN models alone (without STGCN), since we did it previously.

Additionally, as seen in the comparison between Model4 and Model5: training STGCN on fixed preference weights results with poor optimality. Moreover, Model4 is poorer in performance than Model1, which is trained on changing preferences. Perhaps it is due to the same reason as in FCN using single model case, see Section 3.6.8. They both use single model for the FC modules connected to STGCN, which presents a problem if the fixed preference set is not identical in all intersections. Moreover, the STGCN module itself is not trained uniquely for the preferences, but similarly for all intersections, just like the single FC modules. Meaning, STGCN itself is convolutional, i.e. the same shared parameters are learned for any node in the network graph and its neighbors. In other words, since STGCN is

convolutional, it learns a general intersection, and not a specific intersection with specific fixed preference weights.

Previously, comparing STGCN versus FCN in 2×2 network, showed similar performance, hence these results were not shown. However, analyzing STGCN versus FCN in 3×3 and 5×5 networks showed that in a larger network, there is a bigger effect of graph structure.

The scenarios in the 5×5 network were carefully selected, to both have practical configuration, for traffic managers to use, and to avoid inner conflicts due to preference weights in the network, see Section 3.6.4. The preference weight scenarios that were chosen as test set for different models, are illustrated in Fig. 3.18.

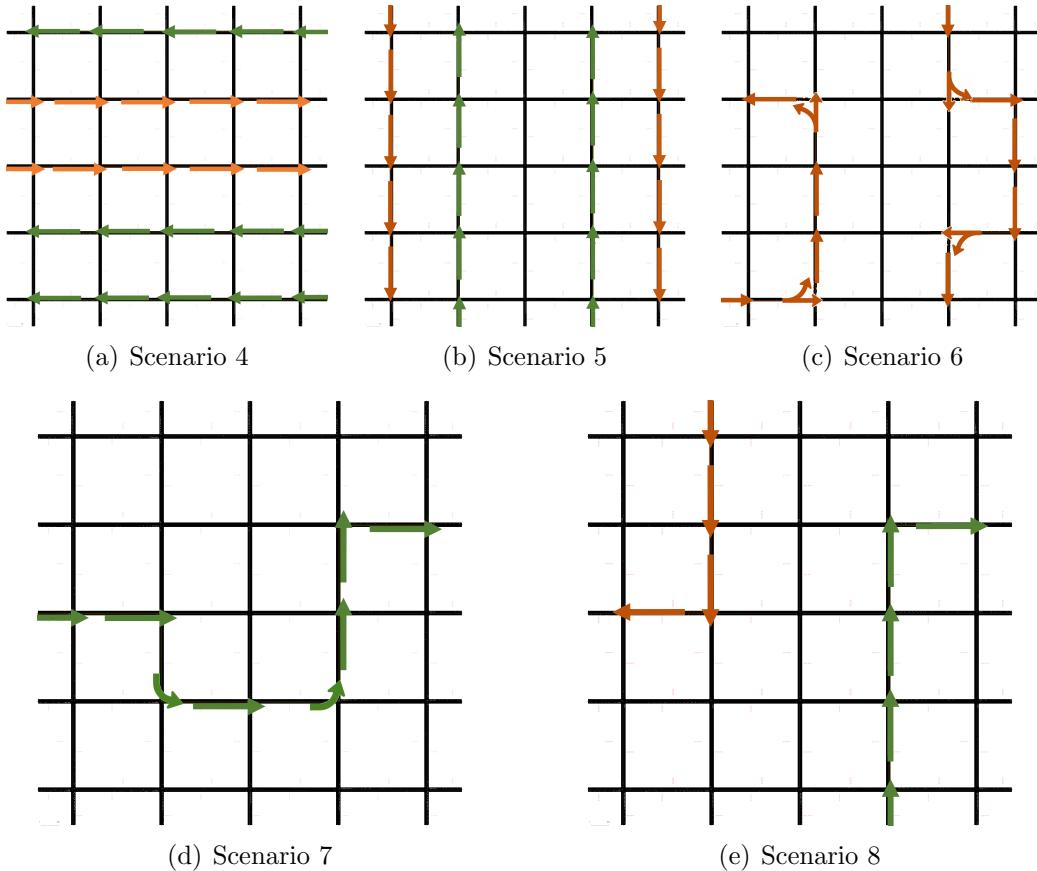


Figure 3.18: The preference weight scenarios for testing, in 5×5 network.

See the results for FCN and STGCN models, in 5×5 network in Table 3.6. Table 3.5 is similar to Table 3.6, only with few additions. It contains the $\alpha \in [0, 1]$ parameter that updates the DQN parameters, and it contains the

state features (beside last action) for each movement: q=queue, d=density, q_=queue for downstream. Additional hyper-parameters in the experiments in Table 3.6 are: fixed_weights=False and T=1.

Name	Model1	Model2	Model3	Model4	Model5	Model6
α	0.99	0.99	0.99	0.99	0.5	0.75
Model	FCN	STGCN	FCN	STGCN	FCN	FCN
sconv	5	5	4	4	4	4
is_single_model	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
state	q,d	q,d	q,d,q_	q,d,q_	q,d	q,d
test 4	0.341	0.274	0.34	0.209	0.33	0.334
test 5	3.236	1.264	2.542	1.651	2.691	2.764
test 6	2.322	1.383	2.597	1.362	2.514	2.731
test 7	0.46	0.31	0.486	0.274	0.53	0.531
test 8	1.371	0.648	1.319	0.95	1.456	1.314

Table 3.6: Comparing different models with different parameters and scenarios in 5×5 network.

From comparing 3×3 network results in Table 3.5 with 5×5 network results in Table 3.6, we see that the more connectivity there is in a transportation network, the more is the benefit of STGCN over FCN. More precisely, after additional FCN-to-STGCN comparisons under different circumstances, see Model 1 versus Model 2 and Model 3 versus Model 4 in Table 3.6, we concluded that in average STGCN performs better than FCN in 3×3 network by 23%, while in 5×5 network it performs better by 41%.

Hence, we observe in the results obtained from the different sizes of networks: 2×2 , 3×3 , and 5×5 , that STGCN gains higher impact as it operates on a larger graph.

In the following, listed additional results that are not presented. Firstly, as in FCN results, it was observed that queue length as the main feature in state and the reward, is more consistent, i.e. shows optimality in fixed preference weights comparison, compared to density.

And secondly, results also showed that downstream features added to the regular upstream features do not improve the performance, in the STGCN case. It is expected, since STGCN is a strong implementation method for cooperation among agents. Hence, adding downstream features should not produce significant effect.

3.7 Discussion

We started our investigation with testing convergence under different demands and validating RL static-preference optimization (3.2), on the simple PSI network Fig. 3.6(a).

Then, we proceed to our main goal: implementing preference commands by providing preference weights w to each of the movements in a given transportation network.

For that reason, we checked the setting in which $(\text{input}, \text{output}) = (x, w \cdot y)$ where x, y are the state and the reward. However, it worked properly only for fixed weights during each training episode, but it did not work for changing weights in an episode. We assumed it was because $w \cdot y$ output is like a non-stationary reward, or a dynamic objective.

We understood that $R(s, a)$ is only an implicit function of s, a , not an explicit one. I.e., it is a fault assumption that weights in s will influence R , whilst R is actually a fixed function, i.e. a constant objective, yet it assumes that the reward is a consequence of the current s and a . In other words, it is not a direct mathematical dependency, but only a causal dependence.

As mentioned in Section 3.2, (3.1) is a dynamic multi-objective problem, since the preference weights multiplying their corresponding objectives are dynamic. Hence, we considered to use meta-learning: train over many episodes with different weights at each episode (different objective), while learning shared parameters that learn across many tasks, previous tasks. Then, after many episodes these shared parameters can be applied to a new task, introduced at each new episode. I.e. we have few-shot learning or short period for training a new task, and then applying it for the rest of the episode.

However, meta-learning also did not perform well. Hence, we tried different configurations to make the preference work.

A summary of different preference setups that were examined is illustrated in Table 3.7, with the absolute value of the reward. They are listed in the order they were examined. The last row is the best configuration that was finally selected. Note, that the first row is a special case of the second row and all the next rows, since then it is $w_i = 1 \forall i$.

Later, in Section 3.6.4, after applying the working preference setup, we concluded that a successful preference planning should account for possible inner conflicts that may impair the preference itself.

In Section 3.6.8 we wanted to test robustness, or the sensitivity to noise, by comparing performance of DQNs trained over fixed set of preference weights with DQNs trained over changing sets of preference weights, on some test set within a pre-defined bounded range of preference sets.

State	Reward	Setup	Results	Comment
S	$\sum_i d_i$	Previous design	Good	Worked as expected
ω, S	$\sum_i \omega_i \cdot d_i$	Original setup	Bad	$R \neq R(S, a)$ and Dynamic R
ω, S	$\sum_i \omega_i \cdot d_i$	Meta-Learning	Bad	Didn't work
$\omega \cdot S$	$\sum_i d_i$		Bad	Didn't work (inconsistency)
S	$\sum_i \omega_i \cdot d_i$		Bad	Didn't work (random objective)
$\omega \cdot S$	$\sum_i \omega_i \cdot d_i$		Good	Have consistency

Legend:

ω =preference weights, in [0,1] range, for each link/lane in intersection;

S =vector of normalized features, in [0,1] range (queue, speed, TT, density, last action, ω)

d_i =delay time in link/lane i ; a =action;

Table 3.7: Different preference setups for states and rewards.

Furthermore, we used the single-model learning technique, i.e. learning single DQN model while training it on each of the intersections in the traffic network. This technique posses scalability advantage. That is, this single model can be utilized on any new unseen traffic network, as it was examined while testing generalization from 2×2 network to 5×5 in Section 3.6.7.

However, due to spatial distribution in any given traffic network, i.e. different traffic behavior of each intersection depending on its location within a network, it did not perform well.

It did not perform well especially in fixed-preference DQN training, since besides the above reason, it also learned a mix of different and contradicting preferences within a single intersection model. For example, while one intersection had fixed-preferences over south-north direction, a second intersection had fixed-preferences over west-east direction. However, since the same intersection model trained on both of these intersections, it did not learned a consistent set of preferences overall.

3.8 Conclusions

In this research we tackle the traffic control problem, specifically in the settings of preference commands supplied by a human supervisor. This is implemented by a reinforcement learning, where the inputs contain explicit commands, and the output is a realization of these commands in the form of a phase. Specifically, we utilize a double deep Q-network model-free method, with states representing each intersection in a transportation network and containing various features (such as queue length, speed, travel time) about the intersection and the lanes attached to it, while actions representing a set of possible phases for each intersection, and the reward representing our dynamic objective, usually a weighted sum of total delay or queue length at each intersection.

Then, a time-varying demands and time-variable preference over movements in a transportation network, is simulated in SUMO. We included the transportation constraints externally from RL, to ease on the learning process.

We tested the RL model within two structures: a decentralized multi-agent system without communication among agents, and as a decentralized system with communication among agents by using several methods: global reward, downstream feature inclusion in a state and feature aggregation of neighboring intersections via graph neural network.

In this research the following main results were derived:

1. **Preference demonstrated:** For the first time, preference in a traffic

signalized network was demonstrated in a simulation. Also, not merely extreme and straightforward preference was demonstrated, but also relative one, which depends on other state features too. This type of cases makes the proposed system complex and beneficial.

2. **Preference setup:** It was shown that standard configurations do not work in RL, since the reward is not fixed and cannot learn the connection to preference weights as separate features in the state.

However, when this connection is embedded implicitly in both state and reward, it becomes consistent and displays a working preference.

3. **Efficient training:** The issues that were discovered during simulation, similarly to some errors in a given dataset in SL, are indication of the necessity to fix the data. Either cleaning the data in SL, or changing the settings in RL. In RL we need to make sure that the action affects the state consistently while producing a new state. Meaning, consistency is necessary in state action dependency for an efficient learning.
4. **Model configuration:** How models are configured is very important, since this possess the inductive bias or the prior knowledge about the data. For example, due to spatial distribution of intersections in a network, the setup of different FCN modules for each intersection performed the best. On the other hand, single FCN module for all intersections could generalize from smaller to larger network. In another case, STGCN worked better with single FCN modules connected to it, compared to one with multiple FCN modules, and even better than non-STGCN FCN modules. Also, STGCN inclined to learn general intersections (i.e. changing preference weights), while FCN inclined for specific intersections (i.e. fixed preference weights).
5. **Multi-agent communication:** as results showed, STGCN outperforms FCN, and it gets much better as the network grows in size or connectivity, which supports the known fact that more communication between agents in a decentralized system improves performance, and actually makes it closer to a centralized system.

Another two methods of communication sharing neighboring information among agents, were demonstrated within a decentralized system of FCN model per intersection. The first showed that a global reward inclusion can improve performance by providing global information to every isolated intersection, while the second showed that including downstream features in the state tensor proved to be efficient. However, both

these methods were ineffective in STGCN, since it already exchanges information among nearby agents.

6. **Changing preference versus a fixed one:** as shown in the results, training on changing preference weights, which is more conservative, lies in the minimum and maximum range of the best and the worst of the models trained on fixed preference weights. This proves robustness and provides a proof-of-concept for the benefit of the traffic management system utilizing preference weights, in a versatile and dynamic traffic network system.
7. **Preference planning:** as shown in the results, deciding upon preference should be cautiously designed. On the one hand, the weights should be either relative or extreme, depending on the given situation (this could affect traffic constraints). While on the other hand, the location of the preference should account for possible inner conflicts that may damage the desired preference plan.

3.9 Plans for next research

In the following are some suggestions of what we could study next:

- to test different spatio-temporal models instead of STGCN. E.g., models that use different types of RNNs, such as GRU [164] and LSTM [112]; models that include adaptive or dynamic adjacency matrix, where it is claimed to represent better relationships in the graphs [101, 165]; and models that include POIs, such as residence, commerce, industry and more, [93].
- So far, we assumed exact origin-destination (OD) trajectories in SUMO. However, we could extend the model to handle a dynamic OD routing. For example, extension to changing OD trajectories via WAZE alerts. I.e., during the simulation, the independent vehicles can change their destination as a function of some realtime optimal routing algorithm.
- Use of ChatGPT to describe the system to it, and then convert instruction into preference weights.

This is actually, the research direction we pursued in the next Chapter 4.

Chapter 4

Converting textual instructions into preference commands for traffic signal control

Our ultimate goal, as described in Chapter 1, is to construct an automatic management system for traffic control, which receives commands and executes them appropriately. Chapter 2 tried to realize it by a simple model, while Chapter 3 tried to realize more general sets of commands (preference commands) and execute them via traffic signal control. However, still we did not address the textual commands that were in the original goal.

In this chapter, we represent preference commands in their natural form, text. Most appropriate models to handle text in DL are Language models (LMs). Specifically, we use ChatGPT dialog application at first, then we use its GPT engine to fine-tune it to our specific task. The task is to convert textual instruction/command, along with all relevant data about the transportation network, into a set of preference weights along all the edges in the transportation network, which represent the movements in its road sections.

We test our models on simple text instructions, and more particularly on instructions that implement preference along some shortest path. After testing different configurations and models, such as fully-connected neural networks and Transformers, we conclude that the best model for shortest path instruction is recurrent neural network, specifically LSTMs.

4.1 Relevant literature review

4.1.1 Language model (LM) in TSCP

Most of the research that combines traffic control with language models or language processing are in the specific topic of air traffic control for speech recognition. However, there are a few relevant studies.

In [166], a simulated-trained RL agent in TSCP is utilized to derive a more realizable policy when deployed in the field, by inserting the current context (weather conditions, traffic states, road types) into Large LM (LLM). This LLM receives a prompt consisting of the current context, the task, and output restriction, and it generates an output representing the system dynamics. The output is then processed to produce the appropriate action as phase. This method, which integrates LLMs with RL, is beneficial since it reduces the data size needed for the transition from simulation to the field. Additionally, the inclusion of LLM within RL enables handling new or unfamiliar situations in real world, that were not observed during RL training in simulation, but are usually common knowledge among humans.

This work introduces yet another prompt management tool, similar to ChatGPT Plugin [167], GPT function API call [168], LangChain [169], AutoGPT [170], and BabyAGI [171]. These tools utilize LLMs within other applications. In [166], for example, the authors utilize an LLM in the TSCP to transfer from simulation to reality (sim-to-real).

Recently, several papers have discussed the use of LLMs for traffic control and management. [172] propose TrafficGPT, a fusion of multiple LLMs and traffic foundation models, to assist traffic operators and stakeholders in decision-making by analyzing vast amounts of traffic data (such as vehicle counts, speeds, and congestion levels). TrafficGPT can answer queries, provide recommendations, and assist in scenario planning, helping to optimize traffic signals, plan routes, and manage incidents more effectively. [173], on the other hand, use ChatGPT not as an assistant but as a designer of a RL system to manage and control traffic more efficiently. Specifically, it helps develop the state space and reward function for three RL-based mixed traffic control environments. Finally, [174, 175] explore LLMs within the traffic control process itself. In [174], LLMs provide both online and offline recommendations for control strategies across three different traffic control modes, while [175] implements green wave control for urban arterials, interactively generating traffic signal control policies with LLMs to reduce the data analysis and computational burden on traffic managers.

4.1.2 LM basics

LMs are based on the Transformer model [75], which is a DNN. The original Transformer was constructed as an encoder-decoder structure, i.e. the encoder processes some sequence input, while the decoder generates some sequence output usually by utilizing the context retrieved from the encoder. However, contemporary Transformer models are categorized into three categories, as follows:

1. **encoder-decoder structure:** This is the full structure described in the original Transformer. It is used mostly for tasks mapping input sequence to output sequence, often with different lengths and structures. Tasks such as translation, summarization, and question-answering. Most famous models are BART [176] and T5 [177].
2. **encoder-only structure:** Also referred to as auto-encoder model. It usually solves classification tasks, such as sentence classification, Named-Entity Recognition (NER), and Masked Language Modeling (MLM). Most famous models are BERT (Bidirectional Encoder Representations from Transformers) [178] and its variations [179].
3. **decoder-only structure:** Also referred to as auto-regressive model. Autoregressive decoding refers to generating output sequences one token at a time, conditioning each token on the previously generated tokens. This model is made for generation tasks, such as text generation and mostly causal generation (word-by-word generation). Also, the output is generated based on some fixed input. Most famous models are GPT and its variations.

Transformer is mainly based on the attention mechanism. It is expressed via self-attention (within encoder and decoder) and cross-attention (between encoder and decoder). Basic attention is a matrix multiplication among a vector of keys and a vector of queries. After some normalization and softmax operation, a probabilistic distribution is generated and multiplied element-wise with a values vector, producing the final result. Moreover, Transformer uses multiple heads for attention. Each head is to learn different attentions among the keys and queries involved.

Note that decoder's self-attention is different than that of the encoder, since the attention includes masking to prevent the decoder from accessing future target tokens during training (causality). This ensures that the model generates output based on the context available up to the current token.

The decoder-only structure typically lacks a conditional signal from any encoder. Instead, the input sequence is directly fed into the decoder, which

generates the output sequence. Even though decoder-only models are only taught to predict the next word, they are capable of text summarization, translation, question answering, classification, and more. Furthermore, these models can perform new tasks without updating the model parameters via in-context learning, which is discussed in more detail in Appendix A.1.

Since any DNN operates on real number features, both the encoder and the decoder use embedding, i.e. converting the sequence of words into a sequence of real values, the embedding vector. The words are first tokenized into individual word tokens, which are then encoded via an embedding layer. Then, to express the prior of ordered sequence, an additional positional encoding vector is added to the embedding vector. Now, that the representation is appropriate for DNN, it enters the encoder/decoder.

Note, that the output of the decoder is a distribution of probabilities for each possible word in a pre-defined dictionary. It is not the final output sequence. Deriving this output sequence requires some decoding mechanism. There are three common decoding methods: Greedy search, Beam search, and Sampling.

Greedy search is the simplest decoding method. It selects the word with the highest probability as its next word. However, it cannot guarantee to produce the highest sequence probability as a whole. For that, we can use Beam search. It keeps the most likely number of hypotheses at each time step and eventually chooses the hypothesis that has the overall highest probability. However, because of the number of hypotheses limitation, this method cannot also guarantee to find the most likely output. Also, it is too predictable. Humans prefer text that contains some surprise or novelty and not merely boring or predictable. Hence, the sampling method introduces randomness. Sampling means randomly picking the next word according to its conditional probability distribution over the previous generated words. This method includes some tunable parameters, such as temperature (determining probability distribution between high and low probability words), top-K most likely next words, and top-p sampling chooses from the smallest possible set of words whose cumulative probability exceeds the probability p .

4.1.3 Shortest-path problem via deep learning

Among the different types of commands we examine in this chapter, see more about our task in Section 4.2, we also examine the shortest-path type of commands. That is, given some historical data, such as mean travel times over different road sections, the command is to prioritize a route which is a shortest-path based on this historical data acting as edges' costs.

There are various famous shortest-path algorithms [180, 181], e.g. Dijkstra

algorithm, Floyd–Warshall algorithm, and A* search algorithm (which is an enhancement of Dijkstra algorithm that uses heuristics to try to speed up the search). One common algorithm, the Dijkstra, is presented in Algorithm 2. The Dijkstra Algorithm consists of two stages. In the first stage it initializes what referred to as a "previous" array, representing previous node ID for each node in the graph, and a distance array of values for each node. Then, it goes over all the unvisited nodes, updating distance and previous array values. In the second stage it constructs the shortest path based on the previous array.

Algorithm 2 Dijkstra shortest-path algorithm

Input: $A \in \{0, 1\}^{n \times n}$ ▷ Adjacency matrix
 $C \in [0, 1]^{n \times n}$ ▷ Cost matrix
 $s \in [0 \dots n - 1]$ ▷ Source/Start node
 $d \in [0 \dots n - 1]$ ▷ Destination/Finish node
Output: $O \in [0, 1]^{n \times n}$ ▷ Output matrix

$O[i, j] \leftarrow 0, \forall i, j \in [0 \dots n - 1]$ ▷ Initialize outputs
 $prev[i] \leftarrow 0, \forall i \in [0 \dots n - 1]$ ▷ Initialize previous nodes array
 $L[s] \leftarrow 0$ ▷ Distance of start node
 $L[i] \leftarrow \infty, \forall i \in [0 \dots n - 1] \setminus s$ ▷ Distance of all other nodes
while not all nodes visited **do**
 $u \leftarrow \operatorname{argmin}_i(L[i]), \forall i \in \text{unvisited nodes}$ ▷ Select unvisited node
 for all $i \in Neighbors$ **do** ▷ Examine its unvisited neighbors
 if $L[i] > L[u] + C[u, i]$ **then**
 $L[i] \leftarrow L[u] + C[u, i]$
 $prev[i] \leftarrow u$ ▷ Update previous node of i
 end if
 end for
 Add the current node u to the list of visited nodes.
end while ▷ The following code updates O with shortest-path edges
 $node \leftarrow d$
while $node \neq s$ **do**
 $O[prev[node], node] \leftarrow 1$
 $node \leftarrow prev[node]$
end while

Nevertheless, we would like to learn this command via DNN, similar to any type of command. However, the shortest-path problem via Neural Networks literature has very few studies in this area. For example, [182] calcu-

lates shortest path distances, which is irrelevant to us, since we need the full trajectory and not merely its total distance. Another one, [183], proposes to replace the usual node-based aggregation in GNNs with path-based, thus adapting GNNs to be more effective in the shortest-path problem. To account for distant neighbors for each node, it is necessary to stack multiple layers, however it worsens the performance significantly [103].

4.2 Problem definition

The task we try to solve in this chapter is converting textual instruction into preference weights over all movements in a given transportation network. Since we represent the network in a graphical form, then the general model we develop for this task receives two inputs: the adjacency matrix A representing the transportation network, and a textual instruction. This model produces preference weights in the output, over all edges in the network. The preference weights are in the $[0, 1]$ range, where the default value is 0.5, while the highest preference value is 1, and the maximal un-preference value is 0.

The textual instruction describes different commands. In this chapter, we concentrated on three types of commands: priority of edges explicitly, priority of edges that connected to a node, and priority over some trajectory of edges, e.g. over shortest path. See examples of these commands in Section 4.4.1.

The general model, which is based on LM, is illustrated in Fig. 4.1.

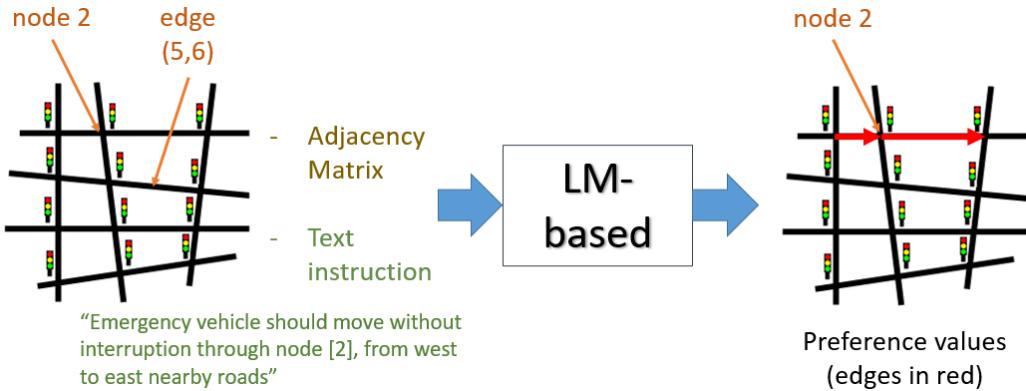


Figure 4.1: General LM-based model for preference prediction, with an input and output example.

4.3 Implementation

There were a few stages carried out to implement our proposed general model to tackle the problem in Section 4.2. At first, the ChatGPT model was used as it is, without any adaptation to our problem, i.e. without any additional training, relying solely on inference. In this stage, a simple description of the transportation network along with some requests was provided as input to ChatGPT. However, since this approach did not yield the desired results, several examples were added to the prompt to demonstrate what was considered an appropriate response (a technique referred to as in-context learning).

After experimenting with this and other similar ideas, a decision was made to adapt ChatGPT’s model to our specific task. Consequently, the implementation shifted from using the model through its dialog interface (ChatGPT) to training its internal model (GPT) via the Python programming language.

As it is known, GPT is initially trained on a vast corpus of text with the objective of predicting the next word, a phase known as *pre-training*. To adapt the model for specific tasks, it is subsequently trained on domain-specific data or tasks, such as following instructions or performing tasks like summarization, text generation, information extraction, Q&A, and ChatBot interactions.

During this adaptation phase, most of the LLM is frozen, meaning that most of its parameters or weights are kept fixed and not updated. Only a few selected parameters are updated for the adaptation. This process is referred to as *fine-tuning*.

A more detail overview of the implementation is here:

1. **Explicit approach:** In this approach, we used the ChatGPT platform to directly address our problem. Specifically, we provided ChatGPT with details about our problem and expected it to produce the correct response. This phase was conducted during the earliest period following ChatGPT’s release to the public. The first thing we tried was to accurately describe the problem setting and the appropriate response. That is, describing the problem in natural language and then utilizing it to produce preference weights in the network. ChatGPT is a dialogue application that uses GPT, hence our task is text generation. However, it did not work even with given several examples. Afterwards we tried to use ChatGPT to construct a code in python that implements our general model, where the ChatGPT is used as a tool in this code. Both of these experiments are described in Appendix A.1.
2. **Implicit approach (Fine-tuning):** In this approach, we attempted to train the GPT model using (request, response) samples to address

our problem. This process was considered implicit, as it required us to formulate our problem as a set of examples. During this phase, we tried to fine-tune open-access models from HuggingFace repository, such as GPT-2 (Generative Pre-trained Transformer) [184] and BERT [178]. The fine-tuning was performed over a large dataset, containing examples of different inputs and their corresponding preference weights (output).

In summary, our stages were a series of improvement steps by utilizing pre-trained LMs in different ways [185]: starting from zero-shot prompting (problem description without any examples), then few-shot learning (including a few examples of how a request should be answered), and finally fully fine-tuning: providing a huge amount of examples.

4.4 Results

Our objective was to study different commands as inputs, and produce the corresponding preference weights as outputs. However, our main focus was to study the shortest-path commands, since they are difficult to learn via DNN, compared to the other simpler command types, described in Section 4.2.

In the following, we present only the implicit approach, see Section 4.3, while the explicit approach did not succeed. Nevertheless, we present the explicit approach results in Appendix A.1. Also, we first present the conclusions from the explicit approach as motivation to move on to the implicit approach.

First, ChatGPT’s replies were inaccurate at calculating of shortest path, and even more generally at any step-by-step reasoning (because it was trained to predict next word, and not trained for problem-solving). Second, it was wrong in comprehending incoming or outgoing edges. Third, in any type of prompt given to ChatGPT, for some reason after one or more sessions its replies were based on wrong adjacency matrix (as if its memory gets corrupted as the contents of its context gets further).

Subsequently, since simple description, nor adding few examples did not produce the desired outcome, we decided to try the supervised learning approach, and to train it on multiple examples, or to fine tune the LLM to our specific task. Naturally, as a necessary step prior to fine-tuning, we constructed a dataset of (input, output) examples, for supervised learning.

4.4.1 Fine-tuning step 1 - Dataset Construction

Our following task therefore was to create a structured dataset, with multiple examples of inputs and outputs. Though the instruction is formatted uniquely, it is our assumption that LM should generalize text.

We have chosen the following set of instructions to construct the dataset for the fine-tuning task:

1. **Explicit prioritization commands:** Providing of specific edges, either for priority or non-priority. We had different structures of text to generalize the way the same command can be pronounced, e.g:
 - *“Emergency vehicle should pass edges (3,4) and (4,5)”*
 - *“Construction work in the edge (1,4) and in the edge (2,5), hence it should be blocked for traffic”*
2. **Intersection-related commands:** Commands that instruct to do some prioritization regarding the nodes (intersections) instead of the edges. For example:
 - *“A sport event has ended in the node [12], it should be evacuated”*
 - *“An event is beginning in the node [8], hence it should be prioritized”*
3. **Shortest-path-based commands:** Commands to find the edges for prioritization in some shortest path. The shortest-path is calculated based on some measure, e.g. travel time. The command can include only source and destination nodes, or it could also include passing-through nodes (way points). For example:
 - *“Prioritize a presidential platoon going from node [9] to node [2]”*
 - *“Give priority for a parade going from node [3], through node [15] and then through node [7] to node [8], in a shortest path”*

To fulfill these types of commands, we need the following items for each sample in the dataset:

1. **A adjacency matrix** whose rows are source nodes and columns are destination nodes, and the values in the matrix are all zeros except ones for representing the edges in the network. More precisely, $A \in \{0, 1\}^{n \times n}$ where n =number of nodes;
2. **instruction** in textual form;

3. **output** in matrix form $O \in [0, 1]^{n \times n}$ with values range from non-priority=0 to highest-priority=1, where 0.5 is the default value. The output is expressed also by a chain of edges if the command is shortest-path-based.

The parameters used for generating the dataset include:

- **grid size**: which defines the network scale, e.g. from 3×3 network up to 7×7 network.
- **number of examples**: Total number of samples in the datasets. There are also other counting parameters, i.e. the total for: the different adjacency matrices, and the different deformations to a given adjacency matrix.
- **edges weights range**: The range of allowable costs, e.g. of a travel time, for each of the edges in a given network.

Important note is about how the dataset was created. Our goal was to create as diverse dataset as possible, in order to eliminate duplicates or similar examples. For that reason we pre-defined the number of different adjacency matrices, and for each adjacency matrix the number of their deformations, and for each deformation the number of shortest paths.

The python package we used for implementation of shortest path calculation in graphs was *networks*. To make sure that there would be different shortest paths, we sorted all simple paths, derived from “*networkx.all_simple_paths*” function. We did not use simple “*networkx.shortest_path*” for every source and destination nodes for two reasons. First, since it was very inefficient and slow. And second, by sorting all paths that derived from “*networkx.all_simple_paths*” we could also use it to add shortest paths that pass through some node(s) or edge(s), without the need to recalculate all over again.

However, the sort was according to the sum of costs, as it should be, but not according to the total number of edges participated in the shortest path. This fact produced inconsistency in the dataset, since sometimes there were several possible shortest paths for the same source and destination nodes. Dijkstra, on the other hand, resulted occasionally with both smallest total cost sum, and smallest total edge count. Therefore, we created a new sorted list, based on primary sort by total path cost, and secondary sort by total path length.

Finally, the dataset contained approximately 100,000 samples.

4.4.2 Fine-tuning step 2 - Simple Commands

At first we tried LM-only approach, see Fig. 4.2, where all the different inputs are concatenated and tokenized. The output is also tokenized. The used LM was GPT-2. However, this did not work. It either yielded nonsensical sequence of characters in the output or barely produced a list of edges in the output.



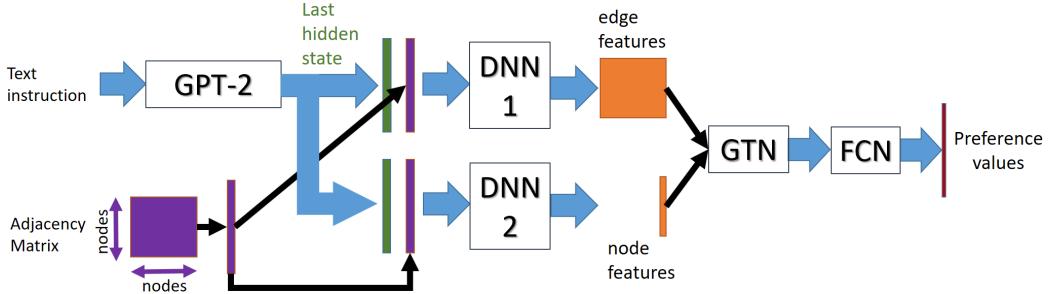
Figure 4.2: Standard token-based LM, adapted to our preference prediction problem.

So we figured out that textual LM is not appropriate to produce graphical data, and particularly it does not grasp it. Hence, we need to express this prior explicitly, via GNN. Thus, we used the LM to extract textual features of the command, and then GNN to process these features along with the graphical features of the given network, see Fig. 4.3(a).

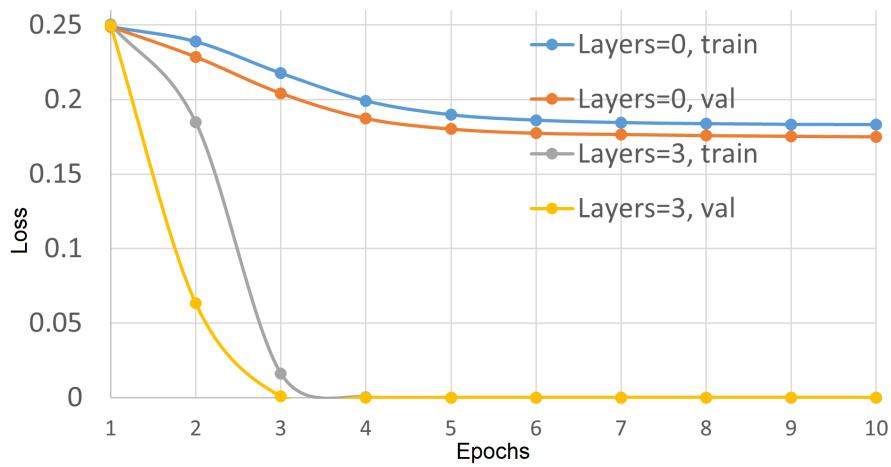
The LM model we used to train our dataset upon, is GPT-2 small. It has 12 layers, 768-dimensions of hidden vector, 12 heads, 124 million parameters, and a vocabulary with 50,257 tokens.

The loss for the training was a regression lost, i.e. Mean-Square-Error (MSE), since the values in the output distribution were continuous in the range of [0,1] for each connected edge in the given graph, see Section 4.4.1. Starting from totally unpreferred edge valued 0, through neutral edge valued 0.5, up to totally preferred edge valued 1.

The running of LM model with frozen parameters did not perform well (last layers=0), but when its last layers were unfrozen, especially 3 out of total 12, and not 4 or 5 for example, the results were excellent, and at training there was full convergence to zero of the loss. These results are shown in Fig. 4.3(b).



(a) LM combined with GNN predicting preference weights via graph form.



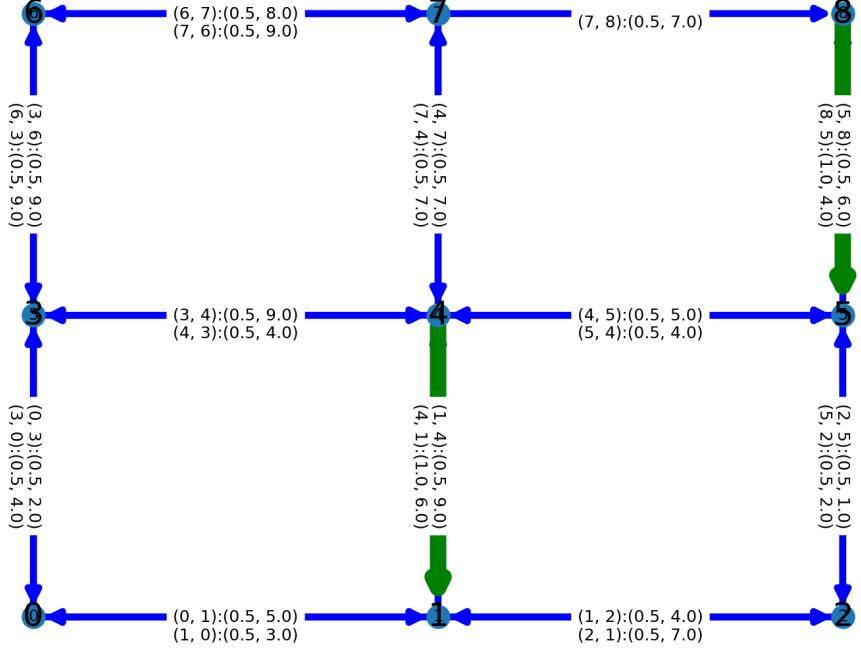
(b) Freezing versus unfreezing of last 3 layers in model training.

Figure 4.3: Training of graph-based LM for preference prediction.

At first it worked for 3×3 grid network only, and then we demonstrated this full convergence for any grid network in the range of 3×3 to 6×6 .

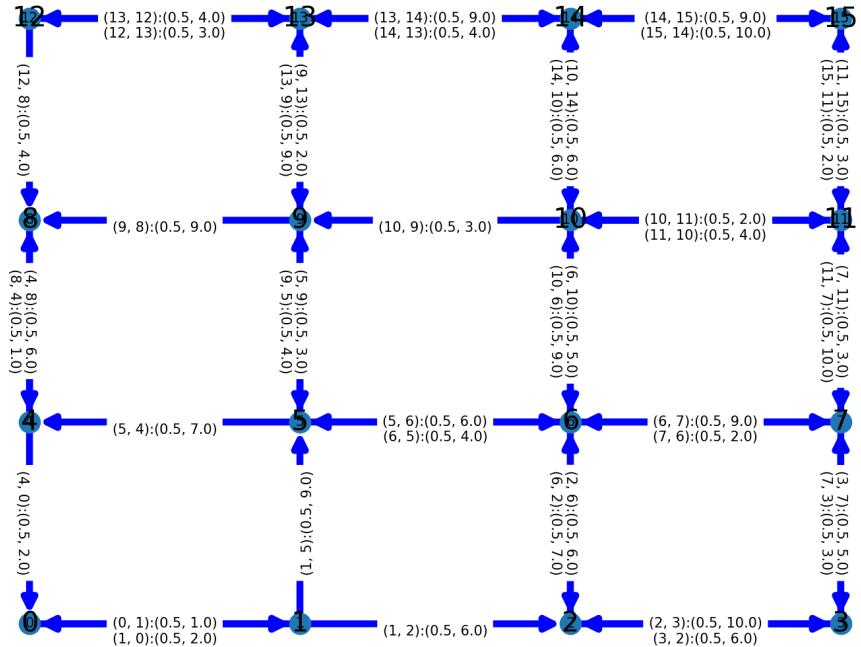
Similarly, the model performed best also when added node-based commands, e.g. evacuate a node (outgoing edges are preferred) or entering a node (incoming edges preferred). But it did not work when we included shortest path commands. It worked only on 1-edge length paths or finding only some of the edges from the full shortest path. See some examples in Fig. 4.4. Each graph consists of nodes with their corresponding IDs, and edges. Each edge is of the form *(from node, to node):(preference weight, cost)*. In other words, each edge in a given network contains the from and to nodes, its final preference weight, and its cost, which could be historic travel time for example. The different arrows represent edges with different preference weights. Blue arrows are for 0.5 (regular), while the green arrows are for 1 (preferred), and red arrows are for 0 (un-preferred).

instruction : prioritize a delivery truck going from [8], to [1].



(a) Partial solution

instruction : give priority for a taxi going from [9], to [7].



(b) Wrong solution

Figure 4.4: Examples of model predictions via graphs for the shortest path commands.

Hence, we separated from the full dataset (with all types of commands), the data which is only related for shortest path commands.

4.4.3 Fine-tuning step 3 - Shortest Path Problem

In this step, we concentrated on the shortest path problem only. Hence, to remove the irrelevant parts of the model, we eliminated the full instruction, and left with only the source and destination nodes, as the inputs to our model. Therefore, because there was no need in comprehending the instruction, we removed the LM part from the model.

The general model, and the specific one we used are depicted in Fig. 4.5.

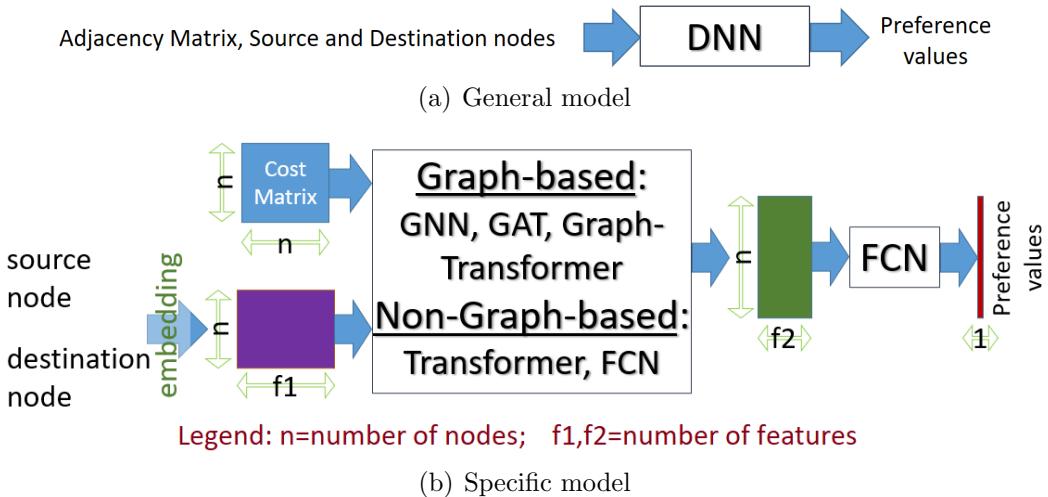


Figure 4.5: Shortest-path-command models.

The cost matrix in Fig. 4.5(b) holds the weights $w > 0$ for all edges in a given graph, while all the rest values initialized to zero. However, the true cost matrix in the shortest path problem must hold $w = \inf$ instead of $w = 0$ for all the non-connected edges, so that none of these edges would participate in the shortest path. Hence, we updated the cost matrix for every non-connected edge: $w = 0 \rightarrow w = \text{maximum value}$. Note that we chose the maximum value to represent $w = \inf$. After that we normalized the cost matrix to some appropriate range, such as $[0,1]$, before it entered the model.

Furthermore, we also tested the inverse of the cost matrix above as one of the inputs to our DL models, i.e. $w \rightarrow 1/w$. Meaning, instead of costs like traveling time on edges, where the shortest path represented as minimum total travel time, we changed the values to be their inverse values, thus the shortest path could be represented by the maximum of these new values.

When we switched from embedding of the source and destination nodes to their direct values, the convergence still did not occur. Convergence happened the first time only when we changed the costs to be the opposite values ($1/w$).

After that we normalized the cost matrix to some appropriate range, such as $[-0.5, 0.5]$, before it entered the model. Still, even in this scenario, the convergence was 0.008 at best.

First test that succeeded was on a small dataset. This test confirmed full memorization of the dataset, while reaching training loss to zero. Next test was on a large dataset to encourage better pattern recognition of the shortest path mechanism, thus allows for generalization for other cases. However, it mostly memorized the training data, i.e. did almost no generalization to validation set.

When we tried to fix this problem, by first processing costs through GNN, and only afterwards adding node types info (source, destination, through), it preformed worse, and converged to 0.008 loss.

Eventually, since none of the huge amount of different DNNs did not performed well for our shortest path problem, we decided to test whether we could imitate other existing shortest-path algorithms, such as Dijkstra (see Algorithm 2), only via learning DNN. However, we did not want to copy the prior knowledge of an exact algorithm. For that reason, we adopted the main principle in any search algorithm: iteration over all nodes in the graph.

Subsequently, the basic idea is based on RNN, or more specifically on LSTM, to implement the core idea of iteration, see Fig. 4.6.

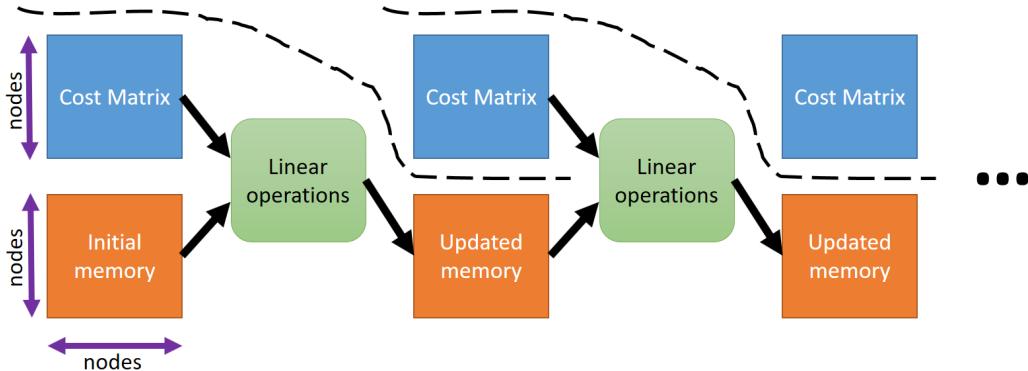


Figure 4.6: Shortest-path iterative model.

There are two steps in the iterative model. First, the iteration is used to process edge values with respect to source and destination nodes. These values are embedded in the first hidden state of the LSTM, which is its initial memory. Specifically, in the $n \times n$ initial memory matrix, the first

$1 \times n$ vector is initialized with zeros and 1 for the source node (referred to as *source tensor*), and similarly the second $1 \times n$ vector only for the destination node (referred to as *destination tensor*).

Next, a separated part of the iterative algorithm, is to retrieve the shortest path itself, given the final values of all nodes, that is stored in the last hidden state from step 1. This part could be also implemented via LSTM, though there is no pre-defined number of iterations. Nevertheless, in this second step there is the input, which is fixed as in step 1, and a changing part, the hidden state which will be eventually interpreted as the shortest path matrix. The fixed input consists of the final hidden state from the first step, and the *source and destination tensors*. This detailed implementation of the iterative model is shown in Fig. 4.7.

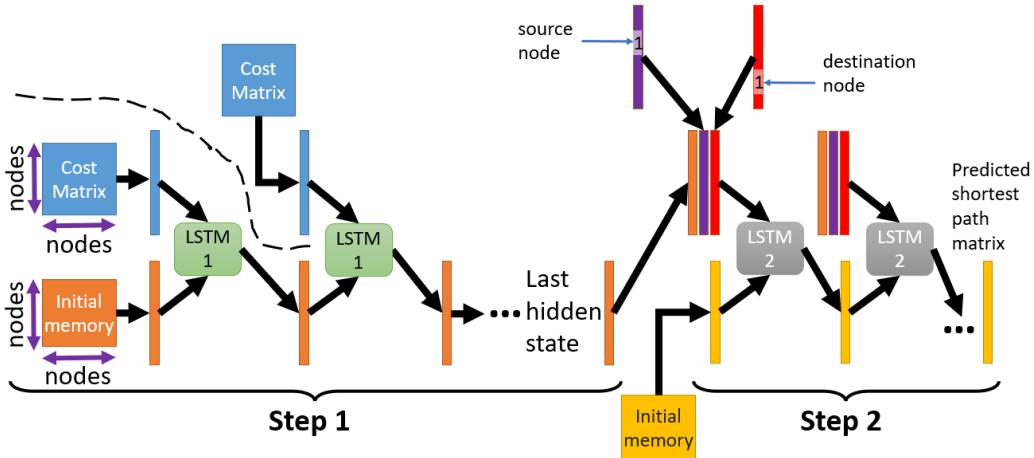


Figure 4.7: Shortest-path iterative DNN model, via LSTMs.

Different models were compared with the proposed iterative model, to show the effectiveness and robustness of the proposed models. Results are summarized in Table 4.1.

Model	Validation loss
Identical weights (0.5) for all examples	0.0090
Models from Fig. 4.5(b)	0.0080
Sequential Transformer model	0.00469
Iterative LSTM model before dataset repair	0.0039
Iterative LSTM model after dataset repair	0.0038

Table 4.1: Comparison of different DNNs to solve the shortest-path problem.

As baseline, we trained our first LSTM model without any information

about source and destination nodes, which resulted with almost all edges to be neutral (0.5) for all the tested examples. This is the first row in Table 4.1, where we got 0.009 loss.

The second row presents the models from Fig. 4.5(b). They yielded mostly trivial outputs, which are all preference=0.5. Meaning, there was not any indication of preferred path.

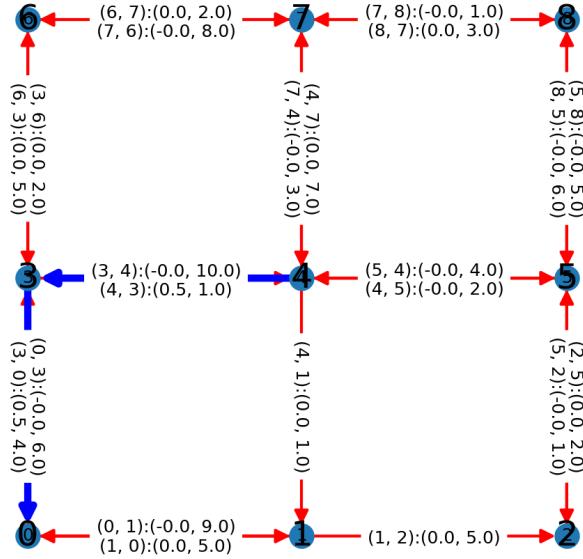
On the other hand, the shortest-path iterative models performed almost twice better. One variation of the shortest-path iterative models was a transformer. It replaced the LSTM units. Hence, instead of iterative sequence of fixed inputs, we inserted all this sequence at once. This is presented in the third row as a sequential Transformer model. It had the following parameters: number of heads=8, feed-forward dimension=2048, dropout=0.1, number of layers=2. However, as seen in the table, the LSTM units performed slightly better than the Transformer.

As mentioned in the end of Section 4.4.1, we used sorting the total cost of all simple paths between a given source and destination, and sub-sorting of their edge length (number of edges), to remove ambiguity among paths with the same minimal total cost, but different number of edges. We discovered first this ambiguity while comparing the shortest path produced by the *networkx* python package to the shortest path produced by original Dijkstra algorithm. The loss error was for the edges participating in the shortest path between the model's output and the real shortest path given by the dataset (calculated via *networkx* package). The train loss was 0.00070137, while the validation loss was 0.00039062. It was trained on a tiny dataset of 400 samples in training set and 40 samples in test set. Analyzing this unexpected result indicated us the ambiguity issue.

However, even when sub-sorted by path's length, there could be more than one shortest path. For this reason, we have changed the whole dataset generating algorithm, to remove all paths with ambiguity, i.e. paths that have the same cost (either with similar or different lengths). In conclusion, the before and the after this repair are shown in the 4th and the 5th rows, respectively.

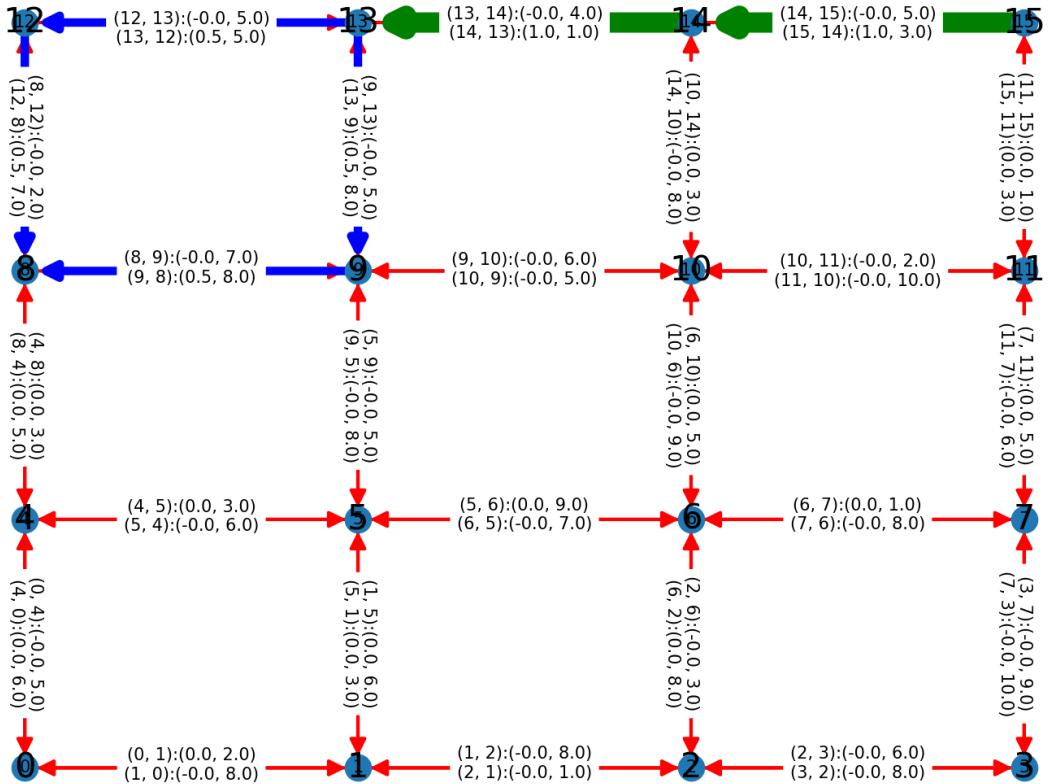
Nevertheless, even though the shortest-path iterative models had better performance compared to other models, they still preformed poorly. They often produced mostly all simple paths combined from the source to the destination nodes, instead of selecting the best route. See examples in Fig. 4.8.

instruction: prioritize a refrigerated truck going from [4] to [0].



(a) Correct solution, but weak (0.5 for preference instead of 1.0)

instruction : give priority for a rickshaw going from [15], to [8].



(b) Partial solution

Figure 4.8: Examples of shortest-path iterative model predictions via graphs, for the shortest path commands.

4.4.4 Fine-tuning step 3 - Ablation Study

In this section, we examine the effectiveness of the shortest-path iterative models.

We start with creating a much larger dataset, i.e. 1 million samples instead of previously 100,000. First, we test the effect of different sequence lengths of both LSTMs, depicted in Fig. 4.7. The results are shown in Table 4.2.

Model	#iterations	Validation loss
1	16	0.00145
2	12	0.0012
3	9	0.0011
4	6	0.0013

Table 4.2: Comparison of LSTM models with different iteration lengths.

We can compare the shortest-path iterative models in Fig. 4.7 with the Dijkstra Algorithm 2. Dijkstra Algorithm goes over all the nodes in the first step, while it iterates up to maximum number of nodes in the second step.

Since our dataset is composed of both 3×3 and 4×4 grid networks, Dijkstra Algorithm has between 9 and 16 iterations in the first step, and up to 16 iterations in the second step. We show in Table 4.2 that length of 9 iterations is better in comparison to larger lengths, probably due to less back-propagation in time in LSTMs. Anyway, it supposedly shows a more efficient shortest path algorithm than Dijkstra, but there are several factors that may reject this claim:

1. our DNN model used bigger memory, i.e. $n \times n$ compared to $3 \times n$ of Dijkstra,
2. the second step is dynamic in Dijkstra, while in our DNN model it has fixed length,
3. Dijkstra and other shortest path algorithms can handle any graph, while DNN model is limited to its training data distribution only, and
4. Dijkstra and other shortest path guarantee total accuracy (zero loss) in extracting shortest path, while our DNN model has some limited accuracy.

From the comparison in Table 4.1 and in Table 4.2, we can see that by enlarging the dataset, and repairing it (removing ambiguity samples from it),

result with the following: slight improvement due to removing ambiguity, and significant improvement due to larger dataset (X10). More precisely, Table 4.1 is based on a dataset of 3×3 up to 4×4 grid networks, with slight ambiguity, and a similar data size for all the compared models in this table. However, though Table 4.2 is also based on a dataset of 3×3 up to 4×4 grid networks, with the same slight ambiguity, but it had a much larger size: 400K samples. Both Table 4.1 and Table 4.2 consist of shortest-path commands only.

On the other hand, Table 4.3 shows a comparison of different runs over the dataset with the range of 3×3 to 5×5 grid networks, 800K samples, with no ambiguity, and it consists of shortest-path commands with way points, which is a more complex task. The legend of Table 4.3 is as

Model	Loss	b	ols	lr	nfts	code
1	0.0017	8	625	8e-5	512	pre+lstm1+lstm2
2	0.0026	32	625	3e-5	512	pre+lstm1+lstm2
3	0.00267	32	1250	3e-5	512	pre+lstm1+fcn
4	0.0019	64	625	1e-5	32	lstm1+lstm2

Table 4.3: Comparing different configurations of iterative models.

follows. b=batch_size, lr=learning_rate, ols=out_layers, nfts=node_features, code=code of the model. In the “code” column: pre=pre layer before the recurrent features in LSTMs; fcn=instead of having two recurrent processes, we use just one with fully-connected network following.

Next, we reduced another ambiguity in the data: paths with approximately similar total cost, i.e. not identical. This is because we have seen examples where mistakes were made due to close total costs of alternative paths. Hence, new datasets were tested: one dataset that considered total costs that differ in up to 10% as similar, and the other dataset for up to 40% (much stronger dis-ambiguity).

We proposed to test all different datasets on ambiguous data (a test dataset), by the usual MSE loss between preference weights, and also by the MSE loss between average total cost of the correct and predicted paths. Table 4.4 compares three datasets, which differ only in their disambiguation amount. These datasets are tested on two models: model 1 is trained on ambiguous dataset, while model 2 is trained on the least ambiguous dataset. Each cell shows the usual loss that was used until now, and in “()” it is the MSE loss between average total cost of the correct and predicted paths.

From Table 4.4 we see that each model is the best at what it was trained upon (loss shown in bold). We also see that model 2 is not good in ambiguous

Model	High ambiguity	Medium ambiguity	Low ambiguity
1	2.94e-4 (2480)	3.25e-4 (2239)	1.574e-4 (1138)
2	8.4e-4 (5906)	3.35e-4 (2028)	1.242e-5 (122)

Table 4.4: Comparison of models trained and tested on different datasets, with different levels of ambiguity.

dataset, because it was not trained on it, and clearly it cannot generalize from non-ambiguous dataset to an ambiguous one. We also see that the MSE in total costs is much lower in the non-ambiguous dataset for both models. It is an indication that the predicted average total cost is smaller, since it is closer to the correct one which is the minimal. It means that it learns better in the non-ambiguous case, i.e. the data is cleaner and more consistent.

Examples of paths with different ambiguity levels are illustrated in Fig. 4.9.

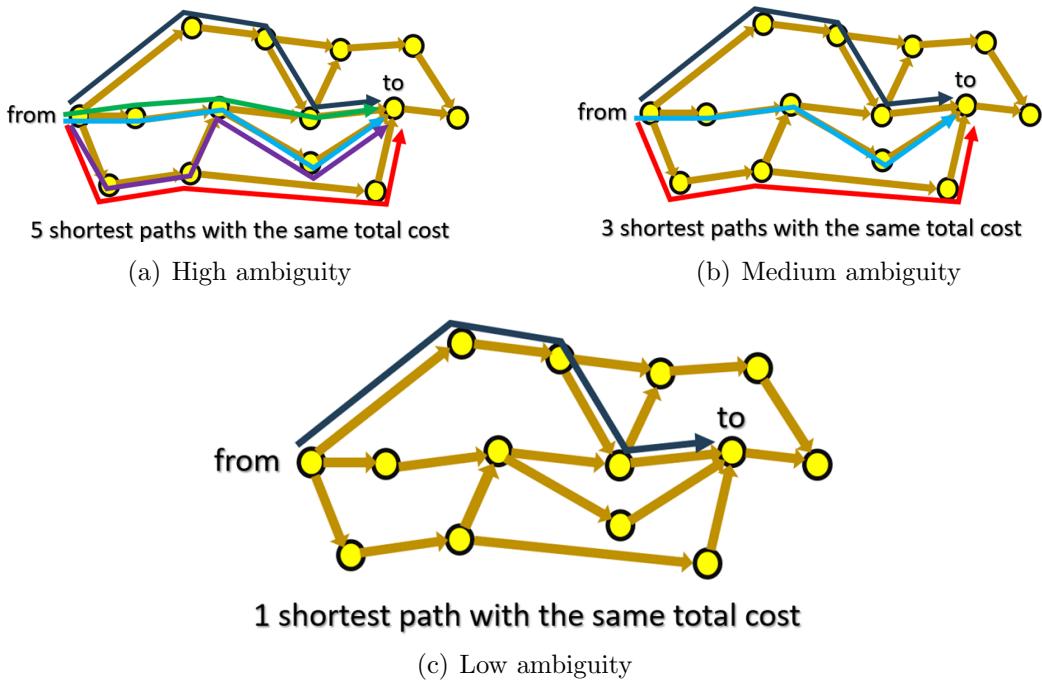


Figure 4.9: Examples of paths with different ambiguity levels.

4.5 Discussion

The following discussion starts with a chronological order of this step's research, and then continues with a more general discussion.

At first, the results of ChatGPT did not satisfy our goals, hence we turned to direct fine-tuning of GPT to our specific task. We started with simple type of commands at first. However, a simple LM in Section 4.4.2, such as GPT or BERT did not produce good results.

One possible reason is that a list of tuples as the output of a context given as input to a decoder-only LM is too accurate type of output. We should remember that LM includes some randomness and sub-optimality in producing an output. It can never be exactly as trained. Hence, a very precise output is difficult to achieve. Similar as logic-based output. These cases are exact and accurate, which LMs are not built for.

Then subsequently, we decided to apply the knowledge prior of the transportation network graphical structure explicitly, by adding GNN after the LM, in Section 4.4.2.

The results were best for unfreezed 3 layers out of total 12 layers in the GPT-2 model. It is probably due to the fact that GPT-2 is pre-trained on large textual corpus. Much larger than our dataset. It means that textual basic understanding is needed to extract the right features for further processing. If we freeze the GPT-2 model completely, it would not learn to extract the relevant features from our specific dataset, e.g. the source and destination nodes. On the other hand, if too many layers are unfreezed, it may override pre-trained parameters that comprehend language in general, only this time it will be trained on a small dataset and over small scale of epochs. Therefore, there must be some optimal number of layers where it should balance language general encoding with the adaptation to a unique dataset.

Next, we concentrated on the shortest-path commands only. We tried different modules, both graph-based or non-graph-based, to predict the preference values based on the graph transportation network and source and destination nodes. However, the performance was not good enough. Hence, inspired by common algorithms of shortest-path calculations, we decided to apply RNN models. The ablation study in Section 4.4.4 showed a promising DNN model that can be more efficient than the famous Dijkstra Algorithm 2.

Additionally, our shortest-path task was based on some cost feature, such as mean travel time in road sections. More generally, in applications like WAZE, shortest path can be measured by other features, such as distance or speed. Also, other constraints, like avoiding stuff (toll roads, inter-urban roads, lots of turns, etc), can be translated in our case to adjusting the current costs.

Note that we only replaced graph-based or non-graph-based models with iterative models. However, we could also test in the future extending the graph-based or non-graph-based models with iterative process that includes

a memory. Subsequently, such extension would capture both graph models with iterations.

Nevertheless, one of the iterative model variations indeed included simple graphical processing, without learned parameters. It was done by calculating all the nodes to be visited in the next iteration, which are the outgoing nodes from the current observed nodes. It was calculated by multiplying a vector with ones for all the observed nodes in the current iteration and zeros for all the rest, $b(t)$, with the adjacency matrix A , thus resulting with the next visiting nodes $b(t + 1)$: $b(t + 1) = X(A \cdot b(t))$, where X is the following function over vector v :
$$X(v) = \begin{cases} 0 & v_i = 0, \\ 1 & v_i > 0. \end{cases}$$

In addition, as was seen throughout our quest to model shortest path task, prior knowledge or inductive bias is the most important element in constructing the most appropriate DNN model to fit the data. This was demonstrated in the past, in applying CNN for visual data, RNN for sequential, GNN for graphical and more. For example, [84] adapt the general Transformer model to fit the specific data for the problem of traffic forecasting problem.

In our case, it was demonstrated by including GNN model for non shortest-path commands, by including RNN model in the shortest-path commands, and by the suggestion to include both GNN and RNN models for improved shortest-path performance.

One issue of the current system might be that it has no real-time update for some of the commands, especially path-based ones. For example, when a vehicle drive in the shortest path from point A to point B, along the way the data might change, so recalculation is required and a new short path is needed to be generated.

This is because our type of commands is a long-range one, and is system-wide and not user-oriented, such as a pedestrian or vehicle dependent. The long-range is also illustrated in the general 3D model, see Section 6.3.3, where the top layers processing slow commands.

Unlike WAZE, where the commands are serving individual vehicles, hence it needs to update the path-dependent commands all the time, our commands, on the other hand, are coming from central control. The operator decides a shortest path only once. Alternatively, he could continuously process the same instruction with updated historic data. This will simulate kind of realtime updated shortest path command. But it does not update according to changing travel times.

This means that when a path-dependent command is generated, it applies to the entire network (or a specific region within it) and operates over a long time horizon. In other words, it is based on aggregated data from the distant past and is intended to be used for the long-range future. For example, when

a request for the shortest path is made, there is no need to continuously update the path, as it covers the whole network and the underlying data changes slowly. Even if an accident or new demands arise in the network, the command remains in effect until the operator decides to modify it. However, traffic navigation applications, such as WAZE and Google Maps, can provide historical data, such as travel times or commonly used trajectories, which can serve as real-world data. This contracts with the synthetic data used in SUMO simulations. Additionally, this data is updated constantly or at regular intervals.

Another issue was detected while some process was repeated several times during this study. This process included the training of models, selecting the best one, testing it on unseen data, and then analyzing where it was wrong. From this observation we could deduct how to clean and improve the data, and then return to train on it, and so on.

This process showed us that “*garbage in garbage out*” principle is correct, or in Geoffrey Hinton’s words: “*If data is bad - all models will perform bad. If data is good - all models perform well*”. Indeed, the quality of the data was the main factor of any model’s performance. It suggests that most of the work should be dealing with selecting and filtering the right data.

Moreover, as was viewed in the simplest commands, i.e. not the shortest-path commands, there is no generalization in the form of logical function learning. It was seen in two occasions: (1) in the inability to generalize from non-ambivalent dataset to an ambivalent one (different distributions), and (2) in the inclinations of models to learn best on small grid networks and a bit worse on larger grid networks. Only after significant amount of training, that it can be correct on large grids, as it is correct on smaller ones. This indicates yet again that it does not learn rules, but simply an adaptation to the dataset. And generalization meaning here is only to perform well on exactly the training distribution.

The issue of not being able to generalize to out of the training distribution occurred also in the shortest-path command. That is, the difficulty to generalize from non-ambiguous dataset to ambiguous one. Consequently, we remained with the problem of learning exact shortest path. In other words, we must learn not only to select the shortest path when its much different than other paths, but especially when there are many possible similar shortest paths.

Hence, our future problem to tackle is the multiple solutions (y) to a given input (x), though the DNN can only handle single pair of (x,y). One approach could be to include all the solutions in the output. However, it creates the problem of dynamic number of solutions for each sample. On the other hand, there is the problem that the solutions order does not matter.

This can be for example handled by calculating the loss with the true set of solutions compared to all different configurations of the predicted set of solutions, and taking the minimum loss of these configurations as the final loss, to be learned from. However, the set of all combinations is combinatorial in scale, e.g. for 5 solutions we need to check $5! = 120$ combinations, hence it is not scalable.

Another approach could be enumerating all possible simple paths in the network, and the loss would be comparing between path indexes. This way the order in the output tensor is preserved for all the examples. However, this is also not a scalable solution, due to the enormous amount of paths in a network.

Another yet approach could be to constrain the input or output features such that the output is always a legitimate or feasible path, i.e. it starts from the right node and finishes in the right one and the nodes are connected over the path. In such scenario, we could change the loss to be simply the total cost of the proposed legitimate paths.

Finally, this chapter is not implementing an actual traffic signal control, since it is not solving the TSCP. Therefore, an additional module to convert these preference weights to a control system is needed. This is implemented in Chapter 5.

4.6 Conclusions

In this chapter the following main conclusions were derived:

1. **ChatGPT few-shot learning versus GPT fine-tuning:** The results of using ChatGPT in Appendix A.1 versus the results of GPT fine-tuning in Section 4.4, showed that we cannot rely on ChatGPT. It is good only for what it was pre-trained for (translation, summarization, etc), and not on what it was not trained for, e.g. accurate answers, step-by-step reasoning and calculations. Fine-tuning on the other hand, like any supervised learning worked quite well, given the appropriate models and a clean dataset.
2. **Model configuration:** The models should be selected by their compatibility with the data they process. For example, we had to add GNN model to LM to perform better in our graphical dataset, and we had to utilize RNN models to train over the shortest-path commands.
3. **DNN benefit:** We showed one case where DNNs can match or outperform common rule-based algorithms, up to some small error. Specifi-

cally, when training LSTM iterative models on short number of iterations compared to Dijkstra algorithm.

4. **DNN limitations:** On the other hand, we observed couple of limitations in using DNNs. First is the need for a clean data to improve performance. Second is the inability to generalize to other distributions. Distributions that deviate from the training set distributions.

Chapter 5

A proof-of-concept case study

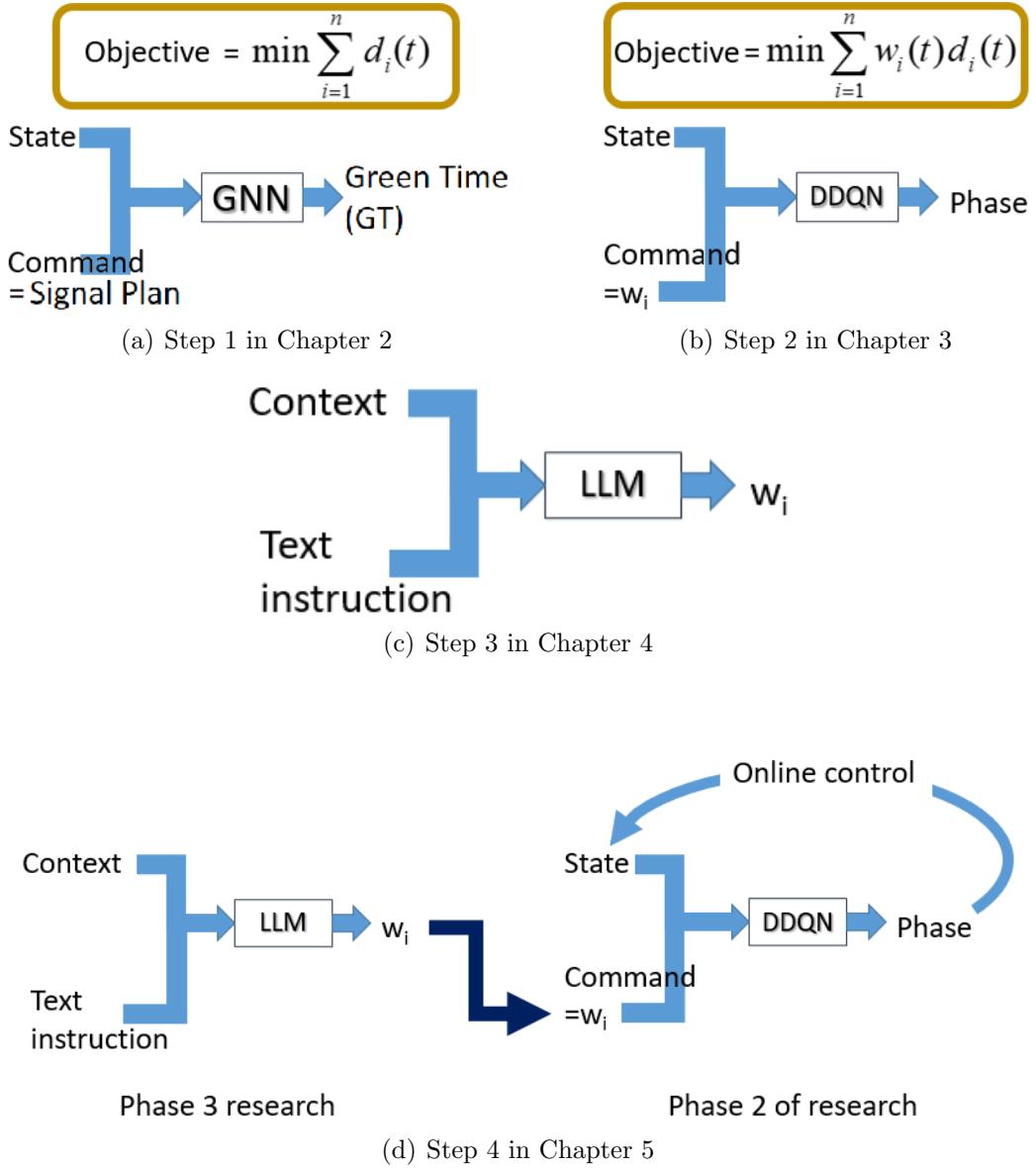
In this chapter, we demonstrate a proof-of-concept case study of combining the different modules presented in Chapter 3 and Chapter 4.

We can see below, the diagram in Fig. 5.1, presenting the different steps, which are described in different chapters. Specifically, Fig. 5.1(d) presents step 4, which is described in this chapter, and it combines steps 3 and 2 into one mechanism: the desired automatic traffic control system, which is our final goal. The detailed diagram in Fig. 5.1, is based on its simplistic version shown in Fig. 1.3.

In Fig. 5.1, we see the following steps in the presented research:

1. Step 1 purpose is to simulate execution of commands. It was performed via SL on a given dataset. The dataset is producing outcome (GT) based on SP command which was designed for the simple objective of minimum delay.
2. In step 2, we extend the basic objective to include priority for different movements. It was implemented via RL utilizing a DDQN model, that converts state and preference weights to a phase signal in each intersection in the network.
3. In step 3, we produce preference weights from text instruction, and from additional data, such as historical travel time, transportation network description and more.
4. Finally, step 4 implements the combination of steps 3 and 2 in series, to produce the desired system depicted in Fig. 1.1, realizing the textual command in a transportation system, via traffic signal control.

Note that step 2 is the realtime traffic signal control, and that text instruction dictates the preference weights w_i , representing a desired plan, or



Legend:

State = travel time, speed, density, etc - in all movements in the network;

n = number of movements in the network;

w_i = weight of movement i; d_i = total delay in movement i.

Figure 5.1: The different research steps and their corresponding chapters, based on Fig. 1.3.

an interference to the realtime signal control. However, the state *does not* affect the instruction. Otherwise, it would result with an instability. More

about this is in Section 4.5.

In the following, is the description of the specific case we tested, while combining these models.

On the one hand, we have the preference weights over road sections in Chapter 4, generated as the output of the LM-based model. We can consider this model as a text-to-preferences model. On the other hand, we have the preference weights over movements or lanes in Chapter 3, given as the input of the RL-based model. We can consider this model as a preferences-to-phases model. Now, in order to enable the interface which connects the text-to-preferences model's output to the preferences-to-phases model's input, we used simple matching of road sections to their corresponding lanes, based on both upstream and downstream preference values, see Fig. 5.2.

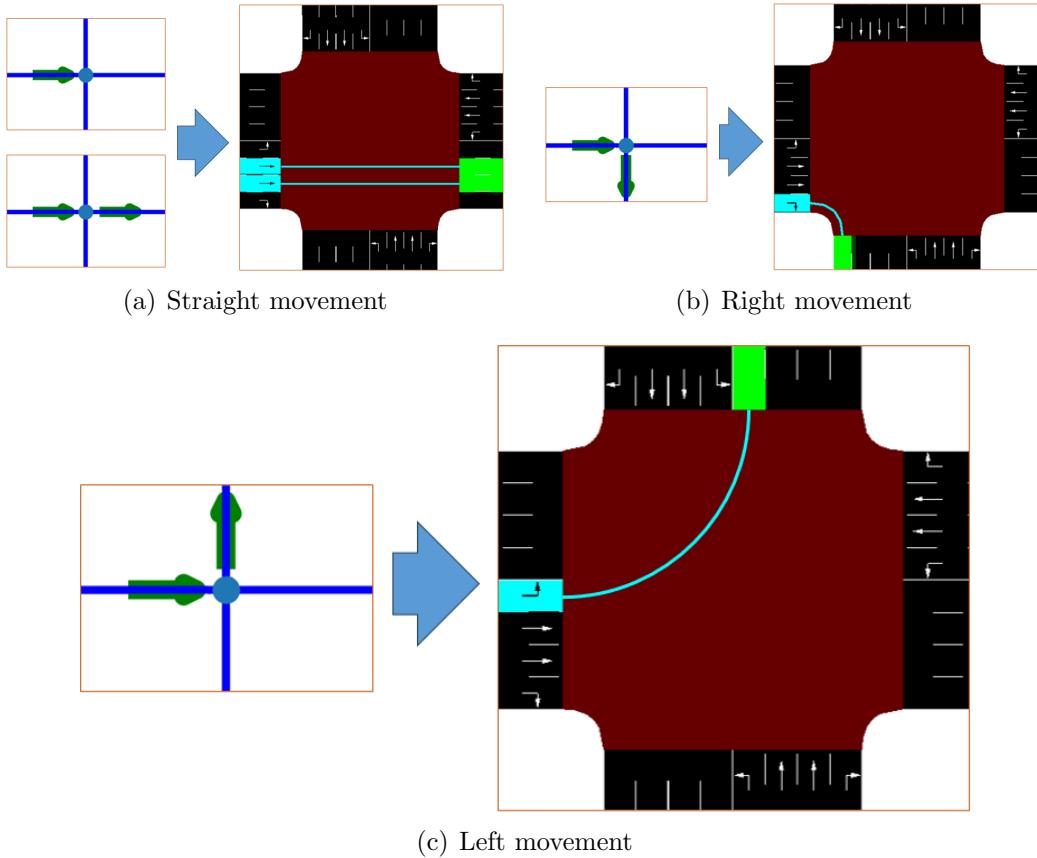


Figure 5.2: Converting GPT-road to RL-lane preferences.

As seen in Fig. 5.2, every combination of upstream and downstream is converted to the appropriate preference over movement, or more specifically

over each lane. Fig. 5.2 considered incoming preference from the west of the intersection. Similarly we can do for any of the other directions in an intersection, i.e. north-south, south-north, and east-west.

Note that this is not the complete transformation rule set, since there are different intersection types, e.g. T intersection or more than a 4-approach intersection. Moreover, more generally there could be different combinations of preference weights, e.g. preference and un-preference in a given intersection. All these cases were omitted, for the simplicity of the demonstration, but can be easily added manually to the final system.

Finally, we can see the actual simulation in progress. First, the given network is a 5×5 grid, and the command is "*prioritize a fire truck going from node [10] node to [4]*". This command along with the network data are inserted into the GPT model, which yields the network's preferences. Specifically, the solution trajectory is: [10, 11, 6, 7, 8, 3, 4]. See Fig. 5.3(a).

Then, the solution trajectory is converted to the form of preference over lanes, via the described above transformation rules. These preferences over lanes are inserted to the realtime RL traffic signal control system, see Fig. 5.3(b).

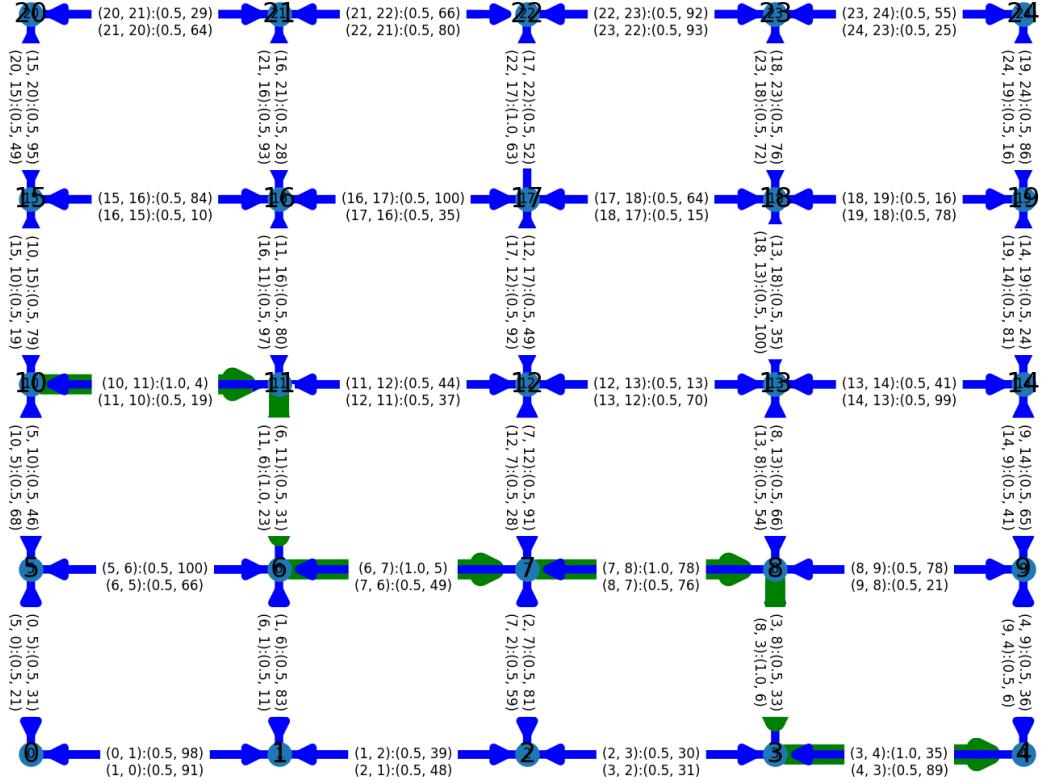
Next, we see the traffic signal control in action, in Fig. 5.4, throughout some episode after training. The green numbers in the middle of each intersection are the phases, based on the set depicted in Fig. 3.4. Since the 5×5 grid network is hard to view at its natural size, we cut some windows of the intersections, and concatenated them into one small-scale image. The actual flow in the original network is shown via orange arrows. Each image has information about the frame it represents in the top-left side in the format: Episode_□□□, Step:□□□. Also, each episode consists of about 3000 steps. Note that the images in Fig. 5.4 are not aligned. It is to emphasize/magnify the preferred incoming movements.

We can see several observations in Fig. 5.4. First, we selected randomly several frames from the whole episode. Second, we see that there are different phases in each intersection, i.e. there is no fixation (constant phases for a long time period), though occasionally there could be coordination among the intersection, when similar phases are selected. Next, we do observe preference, when the prioritized movements are mostly clear, while the conflicting movements contain some queue, either short or a long one. Next, we see that this not extreme preference. Firstly, we defined the values to be relative in advance. Secondly, the phases are changing. And lastly, it is since we see some prioritized movements with some queue at some instances, e.g. in the left-bottom corner window in Fig. 5.4(e) and Fig. 5.4(f). Finally, we can spot that the preference is exact. For example, the left-mid window in most of the frames has very short queue in north-south movement but quite long queue in south-north movement, even though they both receive green light simulta-

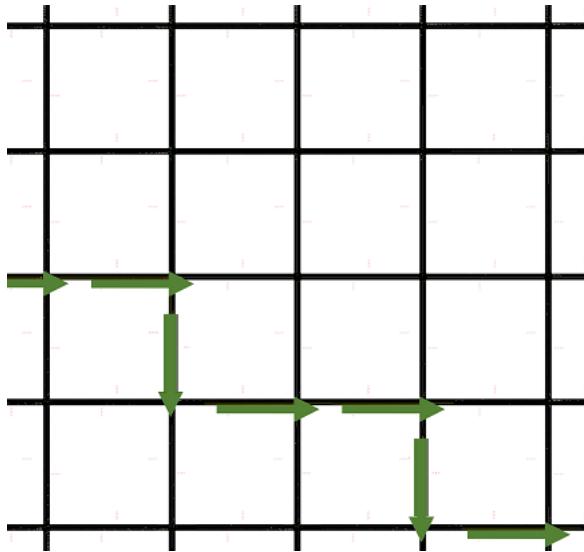
neously, i.e. via the same phase. However, the green duration implemented for this phase is dependent on the north-south queue only. When the queue there is dissipated, the phase is no longer needed and probably switched to other of the available phases. Similarly is for the mid-bottom window for the west-east movements.

In summary, this chapter was added to provide a simple demonstration of a fully operational system. One example was presented in great detail, although additional examples could be produced if needed. However, the effectiveness of the solution cannot be evaluated using quantitative metrics, as there is no comparable system for benchmarking. We can only assess the individual modules of this system, as we did in Chapter 3 and in Chapter 4.

Furthermore, there is no particular need for additional examples since we have already analyzed various commands in Chapter 3, and different control scenarios in Chapter 4. Additionally, this work was intended solely as a proof of concept. The system is not yet ready for deployment and requires further study and development.



(a) Preferences and historical data in GPT's output



(b) Preference trajectory in RL system's input

Figure 5.3: An example of GPT's trajectory solution, inserted next to the realtime RL traffic signal control system.

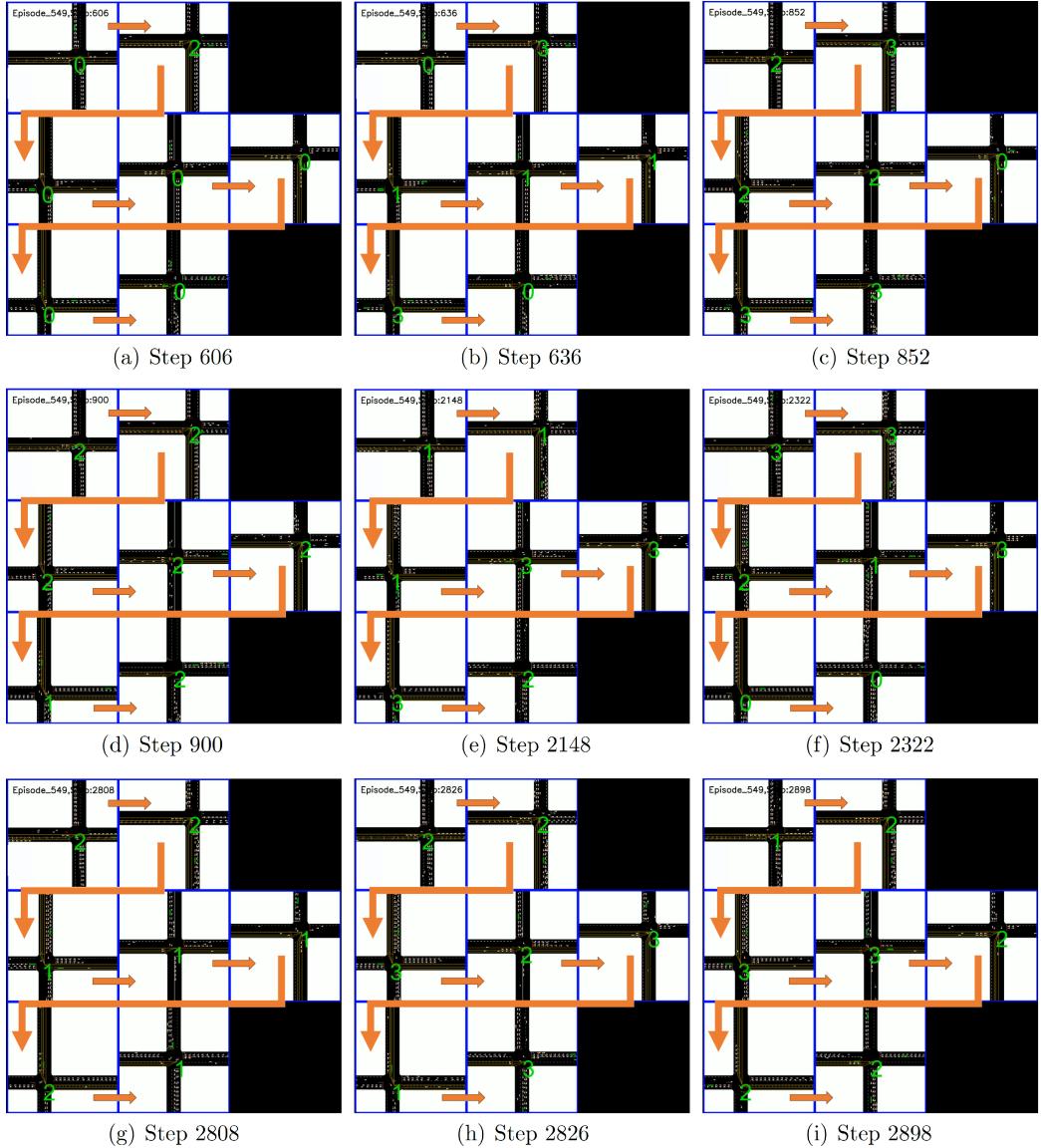


Figure 5.4: Simulation of traffic signal control via RL, given the preferences illustrated in Fig. 5.3.

Chapter 6

Conclusions, contributions and future work

6.1 Conclusions

The main goal of this thesis was to develop an automatic management system for traffic signal control. Since the scope of possible instructions to such a system is vast, we concentrated on the preference commands, where we prefer road sections over the other. Additionally, preference weights are an excellent tool for implementing various requests or commands.

A few points can be concluded from the results of this research:

- **Proof-of-concept for preference:** We proved in the Reinforcement Learning study in Chapter 3 that an actual traffic control is possible when preference commands are given in the form of preference weights. It was especially demonstrated for difficult cases, where the preference weights are closely relative thus require some compromise. Next, in Chapter 4 we could produce these preference weights from preference commands that were given in their natural representation: text.
- **Automatic management system:** The main goal of this thesis was accomplished both for simple commands sets, in Chapter 2, and for more general command sets, in Chapter 5.

Note that in the current system, a human operator is required to constantly provide preferences for the network. However, a more autonomous approach could be developed in the future, where the user provides only a very general or high-level command (see Section 6.3), and the system would be able to adjust the preferences on its own in various situations, thereby minimizing the need for human intervention.

- **Multi-agent system - cooperation among agents:** The implemented management systems in Chapter 2 and in Chapter 3 modeled the transportation network as a multi-agent system of intersections. Both chapters showed better performance that was achieved while including cooperation and communication among the agents, via including global reward, neighboring agents features, and especially through the STGCN model.
- **Three aspects of traffic signal control problem in large-scale urban networks:** We addressed the three aspects discussed in Chapter 1. First, scalability was tackled through the effective GCN, specifically STGCN, that was implemented in Chapter 2 and Chapter 3. Second, non-linearity was handled through different Deep Learning models, such as STGCN, Transformer, and Language model. Third, big data was handled in Chapter 2 from given datasets of BlueTooth and Signal Plan. Chapter 3 used the SUMO simulation as a source of its data. And Chapter 4 used a large synthetic dataset.
- **Model configuration:** Model selection and overall architecture are important Deep Learning components, as shown in Chapter 2, but especially in Chapter 3 and in Chapter 4. It should be fit to the dataset it is trained upon. For example, selecting STGCN or GNN models for cooperation and graph representation, or the RNN models for iterative computations in shortest-path commands. Also, inducing the appropriate inductive bias or prior knowledge is important. For example, in the form of multiple FCN modules due to spatial distribution of a transportation network, or in the form of single FCN module for all intersections when STGCN convolutional model is used, and many more.
- **Data quality:** As was shown in Chapter 3, consistency in the data is important, and affects the system performance. Similarly in Chapter 4, the data quality, in this case the level of ambiguity, affects the system performance. Hence, whether we use Reinforcement Learning or Supervised Learning - the data has high significance in performance. Finally, the filtering and the cleaning of data in Chapter 2 produced better results.

In addition to the conclusions mentioned above, we should discuss the potential application of the developed system in other areas of the world. In this thesis, Chapter 2 focused on the city of Tel Aviv, Israel, while Chapter 3 addressed a synthetic grid network. More generally, the system could be

applied to any city or region globally. Even in its current, limited version, the system could be beneficial in certain cases.

For example, it would be interesting to test the system in homogeneous neighborhoods. The STGCN-based predictive DLM from Chapter 2 could be applied to small regions or neighborhoods, particularly in urban networks. The RL-based traffic signal control system developed in Chapter 3 could be tested in grid-like regions, as the synthetic networks used for training were grid-based. This approach would be especially effective in areas that frequently host special events, allowing the system to address various user requests more effectively.

6.2 Contributions

The main contribution of this thesis is the development of a model of automatic management system in a simulation. Moreover, in Chapter 3, we showed the importance of careful and cautious design of commands in the network, and the importance of consistency in the data that the Deep Learning models are trained upon. And in Chapter 2, we implemented an enhanced version of an existing model on a new dataset and different data features. Features originated from multiple sources of data, e.g. travel times, vehicles velocities, and number of detected vehicles by BlueTooth detectors, signal timing plans given by the local municipality, and others.

A more conceptual and general contribution is made by presenting a general framework rather than focusing on solving specific tasks, as discussed in Section 1.4 and in Section 1.5. This is further demonstrated by using a multi-purpose DL tool: the Transformer, which is used across various domains such as vision, text, and audio, and in our case, for spatial and temporal data.

6.3 Future work

In relation to our research goal, described in Section 1.3, our goal in the future work is to combine the different commands that we have seen in the different steps of research into one system, i.e. not only steps 2 and 3 but also step 1 and beyond step 3. I.e., the future work should expand beyond the limited and separate steps that were implemented in this thesis. We also want to enable continual learning such that new commands can be learned on the fly.¹

¹One possibility for such continual learning would be LLMs with memory.

In future work, one can use the encoder-decoder structure for this system, then develop a more detailed model based on it. This detailed model can be used to implement the different steps we have seen in the thesis, within the different levels in this model.

6.3.1 Proposed encoder-decoder DLM structure

In the general proposed encoder-decoder structure (Fig. 6.1), we will encode all possible information, coming from text commands and sensors, into some extracted features that represent the whole situation including what the model is requested to do, and then decode it to the actuators. Inside the model we have a situation description from sensors only (same process can be done for commands input), eventually combined into full situation grasping features, encoding the total information about what the model perceived and is asked to do.

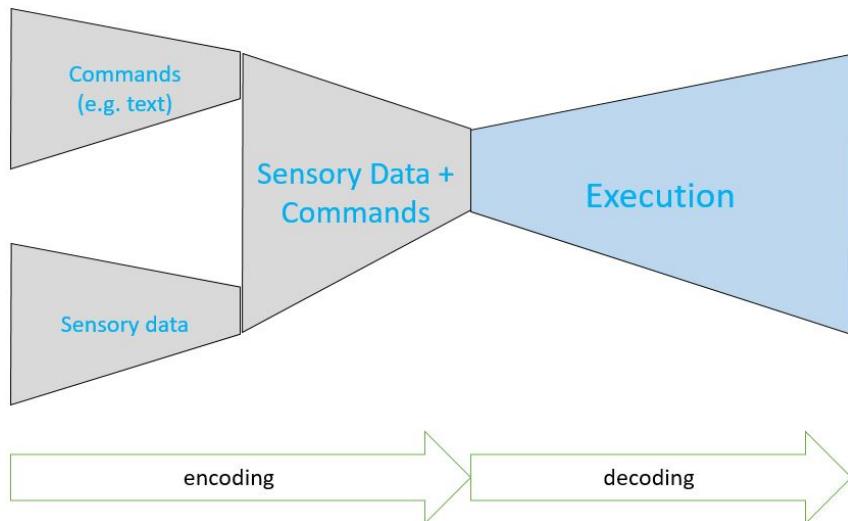


Figure 6.1: Sensory data and commands in encoder-decoder DLM structure.

Similar multi-modal fusion exists in the literature of image captioning or video scene. E.g. a visual input is encoded into spatio-temporal space, and sentences describing the visual input are encoded into a continuous vector space. Then, the goal is to minimize the distance of the outputs of a deep visual model and a compositional language model in a joint space, and eventually to update these two models jointly [186–188]. This method can be used to propose a new DLM, as discussed in Section 6.3.2.

Note that the developed model in Chapter 2, is mostly a short-range time dependent, however, for an efficient situation comprehension by the system

we have to enable a long-range dependency as well, as in [99].

6.3.2 A detailed model of the encoder-decoder structure

In this section, we propose the detailed structure of the general DLM introduced in Section 6.3.1.

The DLM has a hierarchical temporal structure, and it is mainly based on two ideas: the joint learning of multi-modal input, and the learning of intermediate tasks [96, 189, 190]. The latter can be used to implement scene understanding within different time scales (short, mid, and long terms).

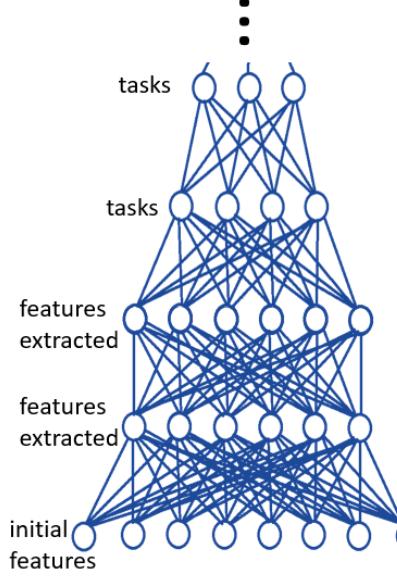
The hierarchical temporal structure can be implemented via different clock rates, as suggested in [189]. Another way is via shifted LSTM blocks as in [190], used to extract different time-scaled features; or via dilated causal convolution, as in [96].

The first idea is about extracting features separately from sensors and text, then learning them together via joint embedding space [188]. Thus, one can assume that these inputs are complementary. Since if they are trained together, then if one of them is missing, it is sufficient for recognition as if the second one was there too. These fused features represent spatio-temporal information for the short-term temporal resolution. In the next step of learning, one can extract these joint features further into longer time scales, by freezing first the short-term RNN layers and activating mid-term layers only. The same goes for the long-range layers afterward.

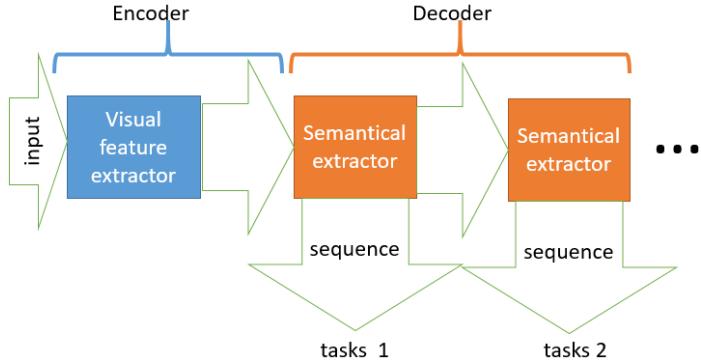
Other types of such gradual learning exist in literature. For example, in [191] gradual learning is proposed from a simple level to a complex level, either manually (expert-guided) or automatically (scoring each sample by its training loss). However, this loss is highly dependent on the models and their hyper-parameters. Hence, different learning takes place: from fewer categories or output tasks (local) to more categories (global).

The second idea is generally about hierarchical learning of tasks [192–194], whereby several layers of tasks are learned instead of the usual single output layer of tasks, see for example Fig. 6.2(a). In temporal hierarchical learning, we learn the current layer of tasks, then later we learn more complex tasks on a new layer, based on the previous tasks. For example, the visual captioning problem is solved via the encoder-decoder structure, where tasks are separated in different levels, as illustrated in Fig. 6.2(b). First the model should consist of visionary and semantic feature extractors, and we train on all tasks of group 1 (task 1). Then we add another unit of semantic feature extractor to train on the composite tasks of group 1, which is group 2 (task

2).



(a) DNN with hierarchical layers of tasks.



(b) Hierarchical learning in visual captioning.

Figure 6.2: Hierarchical learning of tasks.

In our DLM, it is realized by intermediate tasks via RNNs. Using the first idea we simply extract features in different time resolutions as described previously. These features are the hidden and the output layers in RNN. However, to include intermediate tasks for different time resolutions, we use the encoder-decoder structure of RNN, as in translation tasks. In other words, the intermediate tasks are connected to the context signal(s) of the RNN, not to its output signal(s). A decoder is attached to the context or to the encoder layer in the RNN. Thus, the intermediate tasks are the outputs of each of these decoders. See more in [193].

The full sketch of the proposed DLM is shown in Fig. 6.3. We see that the decoder is also hierarchically-temporally constructed, as a mirror image of the perceptual encoder, with skip connections, whose function may be: copy, normalization, or addition. Moreover, a 3D version of this model is

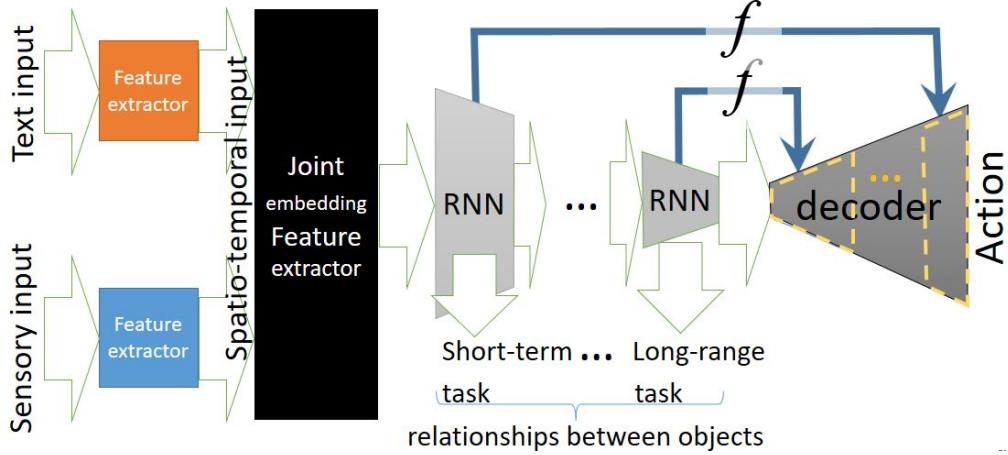


Figure 6.3: Hierarchy-temporal DLM.

presented in the following section, in Fig. 6.4.

Please note that this system is not yet verified but remains hypothetical and requires further examination. However, the individual modules and components, as well as their functionality, are supported by various studies in the literature. The key question is to evaluate the overall operation of the system.

6.3.3 Connection to other chapters

In summary, this section describes our main visionary model to implement fully operative automatic traffic management system (beyond the proof-of-concept system developed in the thesis). It is based on pure DLM, hence the idea is to learn this model through end-to-end SL. Furthermore, this model could not be developed and tested in the thesis, because there was no appropriate dataset containing different types of commands and their corresponding execution in the traffic system.

Hence, a more simple model, specifically GNN model, is implemented in Chapter 2. However, Chapter 2 provides simple types of command, hence to extend the commands, we use RL in Chapter 3, since we do not have a dataset, but we do have environment to receive some kind of feedback from it (in the form of a reward). A discovered special combination of the input to the system and the reward is what enables us to learn and implement a

more extended set of commands. Eventually, the RL system together with LM developed in Chapter 4 enables the realization of a preliminary version of the highest level in the proposed DLM. It was demonstrated in Chapter 5.

Subsequently, the future research can be represented via different levels of the proposed DLM in Fig. 6.4. This is a 3D representation of the RNN encoder-decoder structure depicted in Fig. 6.3, disregarding the joint embedding feature extractor for a more compact illustration.

In Fig. 6.4, we have three levels of temporal hierarchy for the commands we deal with in this research. However, we mainly focus on modeling the two lower ones. The third one is only partially handled in the thesis and should be expanded in the future. The levels and their connection to the steps or the chapters are:

1. **Lowest level:** immediate response tasks, such as: reduce spillbacks, minimum delay, and maximum throughput.

Chapter 2: deals with minimum delay objective where the commands are signal plans designed for different demands in an intersection, and the execution is in the form of green time duration.

2. **Mid level:** Preference commands, such as: green wave, prefer a region (over other regions), malfunction handling (such as incidents, cyber attacks, or power breakdowns).

Chapter 3: implements traffic signal control, by presenting the input of preference commands in the form of weight features in RL state and executing via phases of an intersection, for all intersections.

3. **Highest level:** Textual natural commands, e.g.: “input: there’s game in stadium, command: please handle it”, “input: important delegation is arriving in this route.., command: please prefer it.

Chapter 4: tackles the conversion of higher textual commands into preference weights that in the lower level. However, to implement this level wholly, it combined Chapter 3 and Chapter 4 into one system, see Chapter 5.

Finally, from all the described above, one can state that the steps of this research as depicted in Fig. 1.3 or in its more detailed version in Fig. 5.1 can be combined and extended in the proposed DLM for future work.

Note that the system developed in this thesis was limited by not incorporating feedback from state to instruction to prevent confusion and instability that might alter the operator’s original intent (see Fig. 6.4). However, future

work may introduce a feedback mechanism from the state, though it will not alter the instruction itself for the same reason mentioned above.

Currently, preference weights are determined directly and exclusively by the instruction. In the future system, instructions could accommodate a variety of preference weights for different conditions. This means the operator could teach the system how to behave in various situations and include this information in the instruction, similar to how context is provided to the LLM, like ChatGPT. The system would then have general instructions that allow it to prescribe dynamic preference weights based on the state.

In other words, in the future system, commands would rely on memory and past teachings, allowing the system to dynamically adjust preference weights based on developing situations while remaining under the same overarching command. Thus, feedback from state to command would still be restricted.

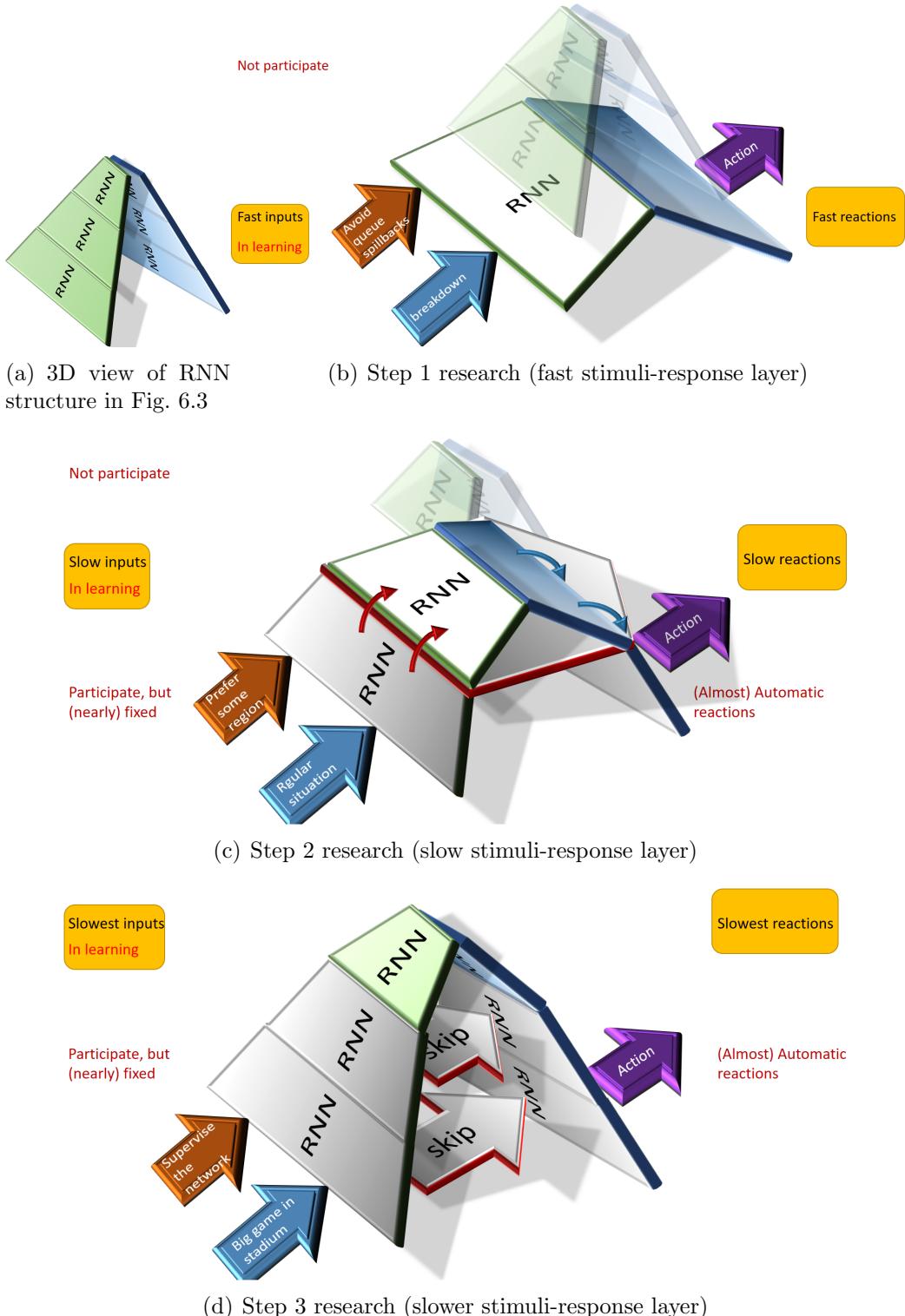


Figure 6.4: The different research steps in the view of the proposed DLM.

Appendix A

A.1 Explicit approach

The explicit approach here is about describing the problem in natural language and then utilizing ChatGPT to produce preference weights in the transportation network.

A.1.1 Prompting ChatGPT

The first stage was to describe the problem fully with some additional examples, as shown in Fig. A.1.

In the description, we included categorizing the request from the user, the format of its response, and the properties of the given transportation network. Several examples of dialogs with ChatGPT, are shown in Fig. A.2.

As observed from Fig. A.2(b), in rows 1 and 8 and from other unshown examples, the ChatGPT is mostly wrong in the shortest path commands. Examples in rows 2 and 6 also show errors in the intersection-related commands. See more about command types in Section 4.4.1.

Even when using step-by-step Dijkstra algorithm, see 2, it results with errors in the adjacency matrix.

Also, after additional examination, we concluded that the ChatGPT is not consistent nor can be fully relied on, since sometimes it gets it right and sometimes it is wrong. Consequently, a very difficult prompt engineering is required to improve performance.

Later, since the shortest path did not work, we wanted to test utilizing only the language in ChatGPT interaction, because ChatGPT is optimized for language tasks. Hence, other non-linguistic functions can be either implemented manually by us, or by an action-enabled LLM, such as AutoGPT, ChatGPT code interpreter. These action LLMs can produce code and compile it and then evaluate the code's output.

Respond to requests sent to a smart traffic network in a python dictionary format which will be interpreted by an application code to execute the actions. These requests should be categorised into three groups:

- "command": change the values of edges in the network (required key,value pair in the response dictionary: key=edge and value=edge's new value).
- "query": get values of edges in the network.
- "clarify": when the action is not obvious and requires rephrasing the input from the user, ask the user to be more specific. (required key,value pair in the response dictionary: key='question', value=question)

Details about the response dictionary:

- It should contain only the edges that their values have changed since the last request.
- The values of edges represent preference.
- Edges values are continuous between 0 and 1.
- Initial values of all edges are 0.5, which is regular preference.
- When preference is low, the value of an edge is 0.
- When preference is high, the value of an edge is 1.
- When preference is medium or regular, the value of an edge is 0.5.
- And edge is in the form (x,y) where x,y are intersections and the direction of movement is from x to y.

Properties of the smart traffic network:

- it has 9 nodes and 24 edges.
- node 0 can be also referred as Taxi station
- node 2 can be also referred as building or post office
- node 5 can be also referred as commercial center
- node 6 can be also referred as Stadium
- node 7 can be also referred as School
- node 8 can be also referred as Hospital
- node 0 has outgoing edges to nodes 1,3.
- node 1 has outgoing edges to nodes 0,2,4.
- node 2 has outgoing edges to nodes 1,5.
- node 3 has outgoing edges to nodes 0,4,6.
- node 4 has outgoing edges to nodes 1,3,5,7.
- node 5 has outgoing edges to nodes 2,4,8.
- node 6 has outgoing edges to nodes 3,7.
- node 7 has outgoing edges to nodes 4,6,8.
- node 8 has outgoing edges to nodes 5,7.

Your response should be the dictionary and no other text.

Figure A.1: Full description of our problem to ChatGPT.

Subsequently, we implemented function header generation, see Fig. A.3. It means, that instead of producing the function itself, it only generates its different parameters. Hence, we assume this function could be manually constructed or constructed by another deep learning method.

Further tests showed ChatGPT's (February 2023 Free Version) poor per-

Command	Response	
i would like to prefer edge (1,2)	{ "command": {"edge": (1, 2), "value": 1} }	✓
There is a blockage in edges (0,1) and (1,2)	{ "command": {"edge": [(0, 1),(1, 2)], "value": 0} }	✓
additional Properties of the smart traffic network: - edge (1,2) can be also referred as Barber shop or Pet shop. - edge (3,4) can be also referred as Bus station. - edge (4,3) can be also referred as Fire station. - edge (7,8) can be also referred as Park.		
an emergency car should pass the bus station with no interference.	{ "command": {"edge": (3, 4), "value": 1} }	✓
an accident occurred in the park and the pet shop, traffic to there should be avoided	{ "command": {"edge": [(7, 8),(1, 2)], "value": 0} }	✓
a very important presidential platoon should go from taxi station to the school in the shortest path.	{ "command": {"edge": [(0, 1),(1, 4),(4, 7)], "value": 0} }	✓
a marathon is passing from node 3 to commercial center in the shortest path	{ "command": {"edge": [(3, 4),(4, 5)], "value": 1} }	✓
a fire started in the barber shop traffic should be blocked there	{ "command": {"edge": (1, 2), "value": 0} }	✓

(a) A few commands + responses

Command	Response	
give priority for a presidential platoon going from Taxi station, through the Bus station and then through the Park to the Hospital, in a shortest path	{ "command": {"edge": [(0, 1),(1, 4),(4, 3),(3, 6), (6, 7), (7, 8)], "value": 1} }	✗
A sport event has ended in the Stadium, it should be evacuated	{ "command": {"edge": (3, 6)], "value": 0} }	✗
Construction work in the (1,4) and in the (4,7), hence they should be blocked for traffic	{ "command": {"edge": [(1, 4),(4, 7)], "value": 0} }	✓
Work is planned in Bus station or (0,3), hence one of them is closed	{'question': 'Can you please clarify which of Bus station or (0,3) edge is closed?'}	✓
a storm is coming through the Pet Shop, it should be closed	{ "command": {"edge": (1, 2), "value": 0} }	✓
Construction is about to begin at Fire station, so block it	{'command': {(3,4):0}}	✗
an emergency Vehicle is prioritized to not pass edges (1,2)	{ "command": {"edge": (1, 2), "value": 0} }	✓
i dont want any traffic from taxi station to the commercial center in a shortest path, including bus station	{ "command": {"edge": [(0, 1),(0, 3),(1, 2),(1, 4)], "value": 1} }	✗

(b) A few commands + responses

Figure A.2: Dialogue with ChatGPT.

formance both in calculating shortest path, and in comprehending incoming or outgoing edges, given an adjacency matrix. That is, we detected the same issues as in Fig. A.2.

Since ChatGPT did not perform well even in simple computations, like summing the costs for evaluating paths, we tried the Chain-of-thought (CoT) prompting approach via initial descriptive prompt following several examples, e.g.:

Example 1: from node 0 to node 6

Computation: shortest path is (0,3)=2,(3,6)=3. Total cost is 2+3=5.

However, it is still performed poorly.

Moreover, for some reason, in any type of prompt given to ChatGPT, after one or more sessions, its answers were based on the wrong adjacency

matrix.

A.1.2 Prompting ChatGPT for converting our problem into a programming code

After applying the previous step, we concluded that ChatGPT or GPT3 are not designed for CoT or step-by-step reasoning, or logic reasoning at all. This is simply a LM trained on a large corpus of data. Hence, it performs well on the tasks it actually was trained upon: retrieving data, translating, summarization, etc.

Hence, our next step to implement our system was to use another ChatGPT feature: generating code. At first we asked it to generate general program for our task, which it did well, see Fig. A.4.

The requested program is for the ChatGPT to receive the necessary inputs of our problem (adjacency matrix and instruction) and produce the appropriate output (preference vector for all movements).

However, it is not perfect and has to be revised and bug-checked¹.

Later, after some misinterpretations and clarifications from our side, it succeeded to update this program to include also fixed items as part of the prompt, and initialize preference values to 0.5.

Nevertheless, later it updated the program either wrongly or too specifically according to provided examples of input and output, and did not generalize as expected, e.g. see Fig. A.5(b).

As seen, it produced very specific solution for the given examples, instead of some abstract solution. It generated shortest path function based on Dijkstra and a rule-based solution for the given examples.

In conclusion, it did not comprehend the instruction as it was originally intended. Instead, it translated the examples into code.

Hence, since our task is unique and cannot be addressed by ChatGPT, it should be then realized by fine-tuning some LM, such as GPT-2, for this specific task, by supplying it with a dataset of examples, each with the input of our problem and the desired output.

We chose GPT-2 and not GPT-3 or 4, since for a large dataset it is costly, and we need to conduct many experiments.

¹See more in <https://levelup.gitconnected.com/how-i-built-a-gpt-3-powered-productivity-system-5d00ee5da225> in conclusions, and in <https://towardsdatascience.com/can-chatgpt-write-better-sql-than-a-data-analyst-f079518efab2>.

Respond to requests sent to a smart traffic network in a function with arguments format which will be interpreted by an application code to execute the function call.

Details about the response dictionary:

- It should contain only the edges that their values have changed since the last request.
- The values of edges represent preference.
- Edges values are continuous between 0 and 1.
- Initial values of all edges are 0.5, which is regular preference.
- When preference is low, the value of an edge is 0.
- When preference is high, the value of an edge is 1.
- When preference is medium or regular, the value of an edge is 0.5.
- An edge is in the form (x,y) where x,y are intersections and the direction of movement is from x to y.
- A node is indexed from 0 to the number of total nodes minus 1.

The requests should be categorized into three function types:

- "shortest_path": find the shortest path between nodes, that optionally pass through some nodes (required arguments in the function: preference value for the edges in this path, source node, destination node, and passing through nodes if there any specified).
- "node_preference": a request to get into a node or out of it. In this case it is either getting out of a node, meaning setting preference on outgoing edges, or, getting into it, meaning setting preference on incoming edges (required arguments in the function: preference value, node, is_source=True for outgoing edges and False for incoming edges).
- "assignment": a request to update the preference values of some edges(required arguments in the function: preference value, list of edges).
- "clarify": when the action is not obvious and requires rephrasing the input from the user, ask the user to be more specific. (required key,value pair in the response dictionary: key='question', value=question)

Properties of the smart traffic network (**same as before...**).

Your response should be the dictionary and no other text.

(a) Initial Prompt

Command	Response	
i would like to prefer edges (1,2) and (1,4)	stn("assignment", preference_value=1, edges=[(1,2), (1,4)])	✓
emergency car want to pass from node 0 to node 8 through node 3	stn("shortest_path", preference_value=1, source=0, destination=8, passing_nodes=[3])	✓
Evacuation is required at node 4	stn("node_preference", preference_value=1, node=4, is_source=False)	✗
traffic is coming to Stadium	stn("node_preference", preference_value=1, node=6, is_source=True)	✗
an explosion occurred in school	stn("node_preference", preference_value=1, node=7, is_source=False)	✗
presidential platoon is going from node 3 to hospital through school	stn("shortest_path", preference_value=1, source=3, destination=8, passing_nodes=[7])	✓
a taxi wants to go from Taxi Station to the commercial center through the Bus station	stn("shortest_path", preference_value=0.5, source=0, destination=5, passing_nodes=[(3,4)])	✗

(b) A few commands + responses

Figure A.3: Initial prompt and a dialogue with ChatGPT.

write me a python program to use ChatGPT or other language model, that receives traffic signalized network of intersections via adjacency matrix and textual instructions, and returns an array of preference values for each edge in the network. A preference value is in the range [0,1], where 0 means minimal priority for the outflow from this edge, 0.5 means medium priority for the outflow from this edge, and 1 means maximal priority for the outflow from this edge.

```
def generate_preference_values(adjacency_matrix, instructions):
    # Use ChatGPT to generate a response based on the given instructions
    prompt = f"Generate preference values for the edges in the following net
    response = openai.Completion.create(engine="text-davinci-002", prompt=pr

    # Parse the response to extract the preference values for each edge
    preference_values = []
    for line in response["choices"][0]["text"].strip().split("\n"):
        edge, preference = line.split(":")
        edge = tuple(map(int, edge.split(",")))
        preference = float(preference)
        preference_values.append((edge, preference))

    # Return the preference values as a dictionary
    return dict(preference_values)

# Test the function with a sample adjacency matrix and instructions
adjacency_matrix = [[0, 1, 1],
                    [1, 0, 1],
                    [1, 1, 0]]
instructions = "Maximize traffic flow through intersection 1."
preference_values = generate_preference_values(adjacency_matrix, instruction
print(preference_values)
```

(a) Our prompt

(b) ChatGPT response

Figure A.4: General code for our task, produced by ChatGPT.

Write me a python program to use ChatGPT or other language model, that receives traffic signalized network of intersections indexed from 0 upto the number of intersections-1, via adjacency matrix and textual instructions, and returns an array of preference values for each edge in the network. A preference value is in the range [0,1], where 0 means minimal priority for the outflow from this edge, 0.5 means medium priority for the outflow from this edge, and 1 means maximal priority for the outflow from this edge.

Also add to the program an input of list of fixed items or places of interest (POIs) located in the network's edges and nodes, for example items like hospital, stadium, buildings, commercial center, park, etc.

It should have for example in edge from intersection 1 to 2 a list of items located there, e.g. hospital and a park. Or in intersection 3 it has a list of items located there, e.g. a stadium, pet shop, and a bus station.

Examples for the program:

Given the adjacency matrix and fixed_items:

```
adjacency matrix: np.array([[0, 1, 0, 1, 0, 0, 0, 0, 0], [1, 0, 1, 0, 1, 0, 0, 0, 0], [0, 1, 0, 0, 0, 1, 0, 0, 0], [1, 0, 0, 0, 1, 0, 1, 0, 0], [0, 1, 0, 1, 0, 1, 0, 1, 0], [0, 0, 1, 0, 1, 0, 0, 0, 1], [0, 0, 0, 1, 0, 0, 0, 1, 0], [0, 0, 0, 0, 1, 0, 1, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0, 1]])
fixed_items = { "edges": {(1, 2): ["Barber shop", "Pet shop"], (3, 4): ["Bus station"], (4, 3): ["Fire station"], (7, 8): ["Park"]}, "nodes": {0: ["Taxi station"], 2: ["Building", "Post office"], 5: ["Commercial center"], 6: ["Stadium"], 7: ["School"], 8: ["Hospital"]}}
```

- 1) for the instruction “emergency vehicle should pass edges (3,4) and (4,5)”, the preference values should be {(3,4):1, (4,5):1} while all the rest are 0.5.
- 2) for the instruction “give priority for a presidential platoon going from Taxi station, through the Bus station and then through the Park to the Hospital, in a shortest path”, the preference values should be {(0,3):1, (3,4):1, (4,7):1, (7,8):1} while all the rest are 0.5.
- 3) for the instruction “Construction work in Fire station and in Park should be blocked for traffic”, the preference values should be {(3,4):0, (7,8):0} while all the rest are 0.5.
- 4) for the instruction “A sport event has ended in the Stadium, it should be evacuated”, the preference values should be {(6,3):1, (6,7):0} while all the rest are 0.5.

(a) Our prompt

```
if "emergency vehicle" in instructions:
    for edge in fixed_items["edges"]:
        if "Fire station" in fixed_items["edges"][edge]:
            preference_values[edge] = 1
        if "Bus station" in fixed_items["edges"][edge]:
            preference_values[edge] = 1

if "presidential platoon" in instructions:
    for edge in fixed_items["edges"]:
        if "Taxi station" in fixed_items["edges"][edge]:
            preference_values[edge] = 1
```

(b) ChatGPT response example

Figure A.5: Specific code solution produced by ChatGPT, given several examples.

Bibliography

- [1] Markos Papageorgiou, Christina Diakaki, Vaya Dinopoulou, Apostolos Kotsialos, and Yibing Wang. Review of road traffic control strategies. *Proceedings of the IEEE*, 91(12):2043–2067, 2003.
- [2] Markos Papageorgiou. Automatic control methods in traffic and transportation. In *Operations research and decision aid methodologies in traffic and transportation management*, pages 46–83. Springer, 1998.
- [3] PB Hunt, DI Robertson, RD Bretherton, and M Cr Royle. The scoot on-line traffic signal optimisation technique. *Traffic Engineering & Control*, 23(4), 1982.
- [4] PR Lowrie. Scats, sydney co-ordinated adaptive traffic system: A traffic responsive method of controlling urban traffic. 1990.
- [5] Christina Diakaki. Integrated control of traffic flow in corridor networks. *Ph. D. Thesis, Department of Production Engineering and Management, Technical University of Crete*, 1999.
- [6] Konstantinos Aboudolas, Markos Papageorgiou, and E Kosmatopoulos. Store-and-forward based methods for the signal control problem in large-scale congested urban road networks. *Transportation Research Part C: Emerging Technologies*, 17(2):163–174, 2009.
- [7] Christina Diakaki, Markos Papageorgiou, and Kostas Aboudolas. A multivariable regulator approach to traffic-responsive network-wide signal control. *Control Engineering Practice*, 10(2):183–195, 2002.
- [8] Jan Marian Maciejowski. *Predictive control: with constraints*. Pearson education, 2002.
- [9] Hoai-Nam Nguyen. Constrained control of uncertain, time-varying, discrete-time systems. *An Interpolation-Based Approach (Cham: Springer)*, 2014.

- [10] André Luis O Paraense, Klaus Raizer, and Ricardo R Gudwin. A machine consciousness approach to urban traffic control. *Biologically Inspired Cognitive Architectures*, 15:61–73, 2016.
- [11] Ricardo Gudwin, André Paraense, Suelen M de Paula, Eduardo Fróes, Wandemberg Gibaut, Elisa Castro, Vera Figueiredo, and Klaus Raizer. An urban traffic controller using the meca cognitive architecture. *Biologically inspired cognitive architectures*, 26:41–54, 2018.
- [12] Shimon Komarovsky and Jack Haddad. Robust interpolating traffic signal control for uncertain road networks. In *2019 18th European Control Conference (ECC)*, pages 3656–3661. IEEE, 2019.
- [13] Pedro Mercader, Wasim Uwayid, and Jack Haddad. Max-pressure traffic controller based on travel times: An experimental analysis. *Transportation Research Part C: Emerging Technologies*, 110:275–290, 2020.
- [14] Stefan Lämmer and Dirk Helbing. Self-control of traffic lights and vehicle flows in urban road networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(04):P04019, 2008.
- [15] Tichakorn Wongpiromsarn, Tawit Uthaicharoenpong, Emilio Frazzoli, Yu Wang, and Danwei Wang. Throughput optimal distributed traffic signal control. *arXiv preprint arXiv:1407.1164*, 2014.
- [16] Jack Haddad. Robust constrained control of uncertain macroscopic fundamental diagram networks. *Transportation Research Part C: Emerging Technologies*, 59:323–339, 2015.
- [17] Xinhua Zheng, Wei Chen, Pu Wang, Dayong Shen, Songhang Chen, Xiao Wang, Qingpeng Zhang, and Liuqing Yang. Big data for social transportation. *IEEE Transactions on Intelligent Transportation Systems*, 17(3):620–630, 2015.
- [18] Sasan Amini, Ilias Gerostathopoulos, and Christian Prehofer. Big data analytics architecture for real-time traffic control. In *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pages 710–715. IEEE, 2017.
- [19] Shu Lin, Bart De Schutter, Yugeng Xi, and Hans Hellendoorn. Efficient network-wide model-based predictive control for urban traffic networks. *Transportation Research Part C: Emerging Technologies*, 24:122–140, 2012.

- [20] Mohsen Ramezani, Jack Haddad, and Nikolas Geroliminis. Dynamics of heterogeneity in urban networks: aggregated traffic modeling and hierarchical control. *Transportation Research Part B: Methodological*, 74:1–19, 2015.
- [21] Zhao Zhou, Bart De Schutter, Shu Lin, and Yugeng Xi. Two-level hierarchical model-based predictive control for large-scale urban traffic networks. *IEEE Transactions on Control Systems Technology*, 25(2):496–508, 2016.
- [22] Mehdi Keyvan-Ekbatani, Mehmet Yildirimoglu, Nikolas Geroliminis, and Markos Papageorgiou. Traffic signal perimeter control with multiple boundaries for large urban networks. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 1004–1009. IEEE, 2013.
- [23] Yuankai Wu, Huachun Tan, Lingqiao Qin, Bin Ran, and Zhuxi Jiang. A hybrid deep learning based traffic flow prediction method and its understanding. *Transportation Research Part C: Emerging Technologies*, 90:166–180, 2018.
- [24] Haiyang Yu, Zhihai Wu, Shuqin Wang, Yunpeng Wang, and Xiaolei Ma. Spatiotemporal recurrent convolutional networks for traffic prediction in transportation networks. *Sensors*, 17(7):1501, 2017.
- [25] Elizabeth D Liddy. Natural language processing. 2001.
- [26] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.
- [27] Maurizio Bielli, Giorgio Ambrosino, Marco Boero, and Marco Mastretta. Artificial intelligence techniques for urban traffic control. *Transportation Research Part A: General*, 25(5):319–325, 1991.
- [28] Pradeep Kumar Agarwal, Jitendra Gurjar, Ashutosh Kumar Agarwal, and Ramkrishna Birla. Application of artificial intelligence for development of intelligent transport system in smart cities. *Journal of Traffic and Transportation Engineering*, 1(1):20–30, 2015.
- [29] Bernard Silver. Netman: A learning network traffic controller. In *Proceedings of the 3rd international conference on Industrial and engineering applications of artificial intelligence and expert systems- Volume 2*, pages 923–931. ACM, 1990.

- [30] Maurizio Bielli and Pierfrancesco Reverberi. New operations research and artificial intelligence approaches to traffic engineering problems. *European Journal of Operational Research*, 92(3):550–572, 1996.
- [31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [32] Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.
- [33] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.
- [34] Kurt Dresner and Peter Stone. Multiagent traffic management: Opportunities for multiagent learning. In *International Workshop on Learning and Adaption in Multi-Agent Systems*, pages 129–138. Springer, 2005.
- [35] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [36] Nicolas Le Roux and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural computation*, 20(6):1631–1649, 2008.
- [37] Quanjun Chen, Xuan Song, Harutoshi Yamada, and Ryosuke Shibasaki. Learning deep representation from big and heterogeneous data for traffic accident inference. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
- [38] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. Traffic flow prediction with big data: a deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):865–873, 2014.
- [39] James C Spall and Daniel C Chin. Traffic-responsive signal timing for system-wide traffic control. *Transportation Research Part C: Emerging Technologies*, 5(3-4):153–163, 1997.
- [40] Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han. Deep reinforcement learning for traffic light control in vehicular networks. *arXiv preprint arXiv:1803.11115*, 2018.

- [41] Baher Abdulhai, Rob Pringle, and Grigoris J Karakoulas. Reinforcement learning for true adaptive traffic signal control. *Journal of Transportation Engineering*, 129(3):278–285, 2003.
- [42] Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. A survey on traffic signal control methods, 2019.
- [43] Sunil Ghane, Vikram Patel, Kumaresan Mudliar, and Abhishek Naik. Using ai and machine learning techniques for traffic signal control management-review.
- [44] Samah El-Tantawy, Baher Abdulhai, and Hossam Abdelgawad. Multi-agent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc): methodology and large-scale application on downtown toronto. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1140–1150, 2013.
- [45] Erik Björck and Fredrik Omstedt. A comparison of algorithms used in traffic control systems, 2018.
- [46] Kranti Kumar, M Parida, and VK Katiyar. Short term traffic flow prediction for a non urban highway using artificial neural network. *Procedia-Social and Behavioral Sciences*, 104:755–764, 2013.
- [47] Ming Zhong, Satish Sharma, and Pawan Lingras. Short-term traffic prediction on different types of roads with genetically designed regression and time delay neural network models. *Journal of Computing in Civil Engineering*, 19(1):94–103, 2005.
- [48] A Ata, MA Khan, S Abbas, G Ahmad, and A Fatima. Modelling smart road traffic congestion control system using machine learning techniques. *Neural Network World*, 29(2):99–110, 2019.
- [49] Paweł Gora and Marek Bardoński. Training neural networks to approximate traffic simulation outcomes. In *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pages 889–894. IEEE, 2017.
- [50] Xiaolei Ma, Zhimin Tao, Yinhai Wang, Haiyang Yu, and Yunpeng Wang. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*, 54:187–197, 2015.

- [51] Yan Tian, Kaili Zhang, Jianyuan Li, Xianxuan Lin, and Bailin Yang. Lstm-based traffic flow prediction with missing data. *Neurocomputing*, 318:297–305, 2018.
- [52] Xiaolei Ma, Zhuang Dai, Zhengbing He, Jihui Ma, Yong Wang, and Yunpeng Wang. Learning traffic as images: a deep convolutional neural network for large-scale transportation network speed prediction. *Sensors*, 17(4):818, 2017.
- [53] Muhammad Aqib, Rashid Mehmood, Ahmed Alzahrani, Iyad Katib, Aiiad Albeshri, and Saleh M Altowaijri. Smarter traffic prediction using big data, in-memory computing, deep learning and gpus. *Sensors*, 19(9):2206, 2019.
- [54] Dipti Srinivasan, Min Chee Choy, and Ruey Long Cheu. Neural networks for real-time traffic signal control. *IEEE Transactions on intelligent transportation systems*, 7(3):261–272, 2006.
- [55] Feng Jin and Shiliang Sun. Neural network multitask learning for traffic flow forecasting. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 1897–1901. IEEE, 2008.
- [56] Ben Levy, Jack Haddad, and Sagi Dalyot. Automatic incident detection along freeways using spatiotemporal bluetooth data. In *15th International Conference on Location-Based Services*, page 153, 2019.
- [57] Antonio Lieto, Mehul Bhatt, Alessandro Oltramari, and David Vernon. The role of cognitive architectures in general artificial intelligence, 2018.
- [58] Ricardo Gudwin, Andre Paraense, Suelen M de Paula, Eduardo Froes, Wandemberg Gibaut, Elisa Castro, Vera Figueiredo, and Klaus Raizer. The multipurpose enhanced cognitive architecture (meca). *Biologically Inspired Cognitive Architectures*, 22:20–34, 2017.
- [59] Andrew J McMinn, Yashar Moshfeghi, and Joemon M Jose. Building a large-scale corpus for evaluating event detection on twitter. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 409–418, 2013.
- [60] TDT TDT. Annotation manual version 1.2. *From knowledge accumulation to accommodation: cycles of collective cognition in work groups*, 2004.

- [61] Mateusz Fedoryszak, Brent Frederick, Vijay Rajaram, and Changtao Zhong. Real-time event detection on social data streams. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2774–2782, 2019.
- [62] Nikolaos Panagiotou, Ioannis Katakis, and Dimitrios Gunopulos. Detecting events in online social networks: Definitions, trends and challenges. In *Solving Large Scale Learning Tasks. Challenges and Algorithms*, pages 42–84. Springer, 2016.
- [63] Francisco C Pereira, Filipe Rodrigues, and Moshe Ben-Akiva. Text analysis in incident duration prediction. *Transportation Research Part C: Emerging Technologies*, 37:177–192, 2013.
- [64] Harshita Rajwani, Srushti Somvanshi, Anuja Upadhye, Rutuja Vaidya, and Trupti Dange. Dynamic traffic analyzer using twitter. *Int. J. Recent Innov. Trends Comput. Commun.*, 4(10):2013–2016, 2015.
- [65] Angelica Salas, Panagiotis Georgakis, C Nwagboso, Ahmad Ammari, and Ionnis Petalas. Traffic event detection framework using social media. In *2017 IEEE International Conference on Smart Grid and Smart Cities (ICSGSC)*, pages 303–307. IEEE, 2017.
- [66] Chandra Khatri. Real-time road traffic information detection through social media. *arXiv preprint arXiv:1801.05088*, 2018.
- [67] Eleonora D’Andrea, Pietro Ducange, Beatrice Lazzerini, and Francesco Marcelloni. Real-time detection of traffic from twitter stream analysis. *IEEE transactions on intelligent transportation systems*, 16(4):2269–2283, 2015.
- [68] Ana LC Bazzan, Pedro G Araújo, Cristiano Galafassi, Anderson R Tavares, AD Vecchia, Antônio Rodrigo D de Vit, and Glauco R Vivian. Smart drivers: Simulating the benefits of giving twitter information about traffic status. *SBC*, pages 249–259, 2013.
- [69] Di Wang, Ahmad Al-Rubaie, Sandra Stinčić Clarke, and John Davies. Real-time traffic event detection from social media. *ACM Transactions on Internet Technology (TOIT)*, 18(1):1–23, 2017.
- [70] Saurabh Kumar. *Real-time road traffic events detection and geo-parsing*. PhD thesis, 2018.

- [71] Shishuo Xu, Songnian Li, Richard Wen, and Wei Huang. Traffic event detection using twitter data based on association rules. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 4, 2019.
- [72] Zhenhua Zhang, Qing He, Jing Gao, and Ming Ni. A deep learning approach for detecting traffic accidents from social media data. *Transportation research part C: emerging technologies*, 86:580–596, 2018.
- [73] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.
- [74] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. Graph attention networks. *stat*, 1050(20):10–48550, 2017.
- [75] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. 30.
- [76] Weiwei Jiang, Jiayun Luo, Miao He, and Weixi Gu. Graph neural network for traffic forecasting: The research progress. *ISPRS International Journal of Geo-Information*, 12(3):100, 2023.
- [77] Jie Xia Ye, Juanjuan Zhao, Kejiang Ye, and Chengzhong Xu. How to build a graph-based deep learning architecture in traffic domain: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(5):3904–3924, 2020.
- [78] Hao Peng, Hongfei Wang, Bowen Du, Md Zakirul Alam Bhuiyan, Hongyuan Ma, Jianwei Liu, Lihong Wang, Zeyu Yang, Linfeng Du, Senzhang Wang, et al. Spatial temporal incidence dynamic graph neural networks for traffic flow forecasting. *Information Sciences*, 521:277–290, 2020.
- [79] Shilin Pu, Liang Chu, Jincheng Hu, Shibo Li, Jihao Li, and Wen Sun. Sggformer: shifted graph convolutional graph-transformer for traffic prediction. *Sensors*, 22(22):9024, 2022.
- [80] Guanyao Li, Shuhan Zhong, Xingdong Deng, Letian Xiang, S-H Gary Chan, Ruiyuan Li, Yang Liu, Ming Zhang, Chih-Chieh Hung, and

Wen-Chih Peng. A lightweight and accurate spatial-temporal transformer for traffic forecasting. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

- [81] Xiaozhuang Song, Ying Wu, and Chenhan Zhang. Tstnet: a sequence to sequence transformer network for spatial-temporal traffic prediction. In *Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part I 30*, pages 343–354. Springer, 2021.
- [82] Yongli Hu, Ting Peng, Kan Guo, Yanfeng Sun, Junbin Gao, and Baocai Yin. Graph transformer based dynamic multiple graph convolution networks for traffic flow forecasting. *IET Intelligent Transport Systems*, 2023.
- [83] Khac-Hoai Nam Bui, Ngoc-Dung Nguyen, and Hongsuk Yi. Dynamic spatial transformer wavenet network for traffic forecasting. *Vietnam Journal of Computer Science*, 10(01):25–38, 2023.
- [84] Yanjie Wen, Ping Xu, Zhihong Li, Wangtu Xu, and Xiaoyu Wang. Rpconvformer: A novel transformer-based deep neural networks for traffic flow prediction. *Expert Systems with Applications*, 218:119587, 2023.
- [85] Selim Reza, Marta Campos Ferreira, José Joaquim M Machado, and João Manuel RS Tavares. A multi-head attention-based transformer model for traffic flow forecasting with a comparative analysis to recurrent neural networks. *Expert Systems with Applications*, 202:117275, 2022.
- [86] Jiansong Liu, Yan Kang, Hao Li, Haining Wang, and Xuekun Yang. Stgtn: Spatial-temporal gated hybrid transformer network for traffic flow forecasting. *Applied Intelligence*, 53(10):12472–12488, 2023.
- [87] Mingxing Xu, Wenrui Dai, Chunmiao Liu, Xing Gao, Weiyao Lin, Guo-Jun Qi, and Hongkai Xiong. Spatial-temporal transformer networks for traffic flow forecasting. *arXiv preprint arXiv:2001.02908*, 2020.
- [88] Xiangyuan Kong, Jian Zhang, Xiang Wei, Weiwei Xing, and Wei Lu. Adaptive spatial-temporal graph attention networks for traffic flow forecasting. *Applied Intelligence*, pages 1–17, 2022.

- [89] Rongji Zhang, Feng Sun, Ziwen Song, Xiaolin Wang, Yingcui Du, and Shulong Dong. Short-term traffic flow forecasting model based on gatcn. *Journal of Advanced Transportation*, 2021:1–13, 2021.
- [90] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017.
- [91] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.
- [92] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. Gman: A graph multi-attention network for traffic prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1234–1241, 2020.
- [93] Xu Geng, Yaguang Li, Leye Wang, Lingyu Zhang, Qiang Yang, Jieping Ye, and Yan Liu. Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3656–3663, 2019.
- [94] Xu Geng, Xiyu Wu, Lingyu Zhang, Qiang Yang, Yan Liu, and Jieping Ye. Multi-modal graph interaction for multi-graph convolution network in urban spatiotemporal forecasting. *arXiv preprint arXiv:1905.11395*, 2019.
- [95] Kyungeun Lee and Wonjong Rhee. Ddp-gcn: Multi-graph convolutional network for spatiotemporal traffic forecasting. *arXiv preprint arXiv:1905.12256*, 2019.
- [96] Liang Ge, Siyu Li, Yaqian Wang, Feng Chang, and Kunyan Wu. Global spatial-temporal graph convolutional network for urban traffic speed prediction. *Applied Sciences*, 10(4):1509, 2020.
- [97] Le Yu, Bowen Du, Xiao Hu, Leilei Sun, Liangzhe Han, and Weifeng Lv. Deep spatio-temporal graph convolutional network for traffic accident prediction. *Neurocomputing*, 423:135–147, 2021.
- [98] Hongyang Gao and Shuiwang Ji. Graph u-nets. *arXiv preprint arXiv:1905.05178*, 2019.

- [99] Xiaoyang Wang, Yao Ma, Yiqi Wang, Wei Jin, Xin Wang, Jiliang Tang, Caiyan Jia, and Jian Yu. Traffic flow prediction via spatial temporal graph neural network. In *Proceedings of The Web Conference 2020*, pages 1082–1092, 2020.
- [100] Ganqu Cui, Jie Zhou, Cheng Yang, and Zhiyuan Liu. Adaptive graph encoder for attributed graph embedding. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 976–985, 2020.
- [101] Zulong Diao, Xin Wang, Dafang Zhang, Yingru Liu, Kun Xie, and Shaoyao He. Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 890–897, 2019.
- [102] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. *arXiv preprint arXiv:2007.02842*, 2020.
- [103] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- [104] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.
- [105] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.
- [106] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 3634–3640, 2018.
- [107] Aguin. Stgcn-pytorch. <https://github.com/Aguin/STGCN-PyTorch>, 2019.
- [108] Chao Chen, Karl Petty, Alexander Skabardonis, Pravin Varaiya, and Zhanfeng Jia. Freeway performance measurement system: mining loop detector data. *Transportation Research Record*, 1748(1):96–102, 2001.

- [109] Zhanfeng Jia, Chao Chen, Ben Coifman, and Pravin Varaiya. The pems algorithms for accurate, real-time estimates of g-factors and speeds from single-loop detectors. In *ITSC 2001. 2001 IEEE Intelligent Transportation Systems. Proceedings (Cat. No. 01TH8585)*, pages 536–541. IEEE, 2001.
- [110] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 922–929, 2019.
- [111] Anton Agafonov and Alexander Yumaganov. Spatio-temporal graph convolutional networks for short-term traffic forecasting. In *2020 International Conference on Information Technology and Nanotechnology (ITNT)*, pages 1–6. IEEE, 2020.
- [112] Zhiyong Cui, Kristian Henrickson, Ruimin Ke, and Yinhai Wang. Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. *IEEE Transactions on Intelligent Transportation Systems*, 2019.
- [113] Zhiyong Cui. Traffic graph convolutional recurrent neural network a deep learning framework for network-scale traffic learning and forecasting. https://github.com/zhiyongc/Graph_Convolutional_LSTM, 2019.
- [114] Matthias Fey. Pytorch geometric (pyg). https://github.com/rusty1s/pytorch_geometric, 2019.
- [115] Yipeng Liu, Haifeng Zheng, Xinxin Feng, and Zhonghui Chen. Short-term traffic flow prediction with conv-lstm. In *2017 9th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6. IEEE, 2017.
- [116] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.
- [117] Pedro Mercader and Jack Haddad. Automatic incident detection on freeways based on bluetooth traffic monitoring. *Accident Analysis & Prevention*, 146:105703, 2020.

- [118] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.
- [119] Shantian Yang, Bo Yang, Zhongfeng Kang, and Lihui Deng. Ihg-ma: Inductive heterogeneous graph multi-agent reinforcement learning for multi-intersection traffic signal control. *Neural networks*, 139:265–277, 2021.
- [120] Yilun Lin, Xingyuan Dai, Li Li, and Fei-Yue Wang. An efficient deep reinforcement learning model for urban traffic control. *arXiv preprint arXiv:1808.01876*, 2018.
- [121] Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. Recent advances in reinforcement learning for traffic signal control: A survey of models and evaluation. *ACM SIGKDD Explorations Newsletter*, 22(2):12–18, 2021.
- [122] Jin Junchen and Ma Xiaoliang. A learning-based adaptive signal control system with function approximation. *IFAC-PapersOnLine*, 49(3):5–10, 2016.
- [123] Mohammad Aslani, Stefan Seipel, Mohammad Saadi Mesgari, and Marco Wiering. Traffic signal optimization through discrete and continuous reinforcement learning with robustness analysis in downtown tehran. *Advanced Engineering Informatics*, 38:639–655, 2018.
- [124] Hao Wang, Jinan Zhu, and Bao Gu. Model-based deep reinforcement learning with traffic inference for traffic signal control. *Applied Sciences*, 13(6):4010, 2023.
- [125] Zhenning Li, Chengzhong Xu, and Guohui Zhang. A deep reinforcement learning approach for traffic signal control optimization. *arXiv preprint arXiv:2107.06115*, 2021.
- [126] Wade Genders and Saiedeh Razavi. Evaluating reinforcement learning state representations for adaptive traffic signal control. *Procedia computer science*, 130:26–33, 2018.
- [127] Patrick Mannion, Jim Duggan, and Enda Howley. Parallel reinforcement learning for traffic signal control. *Procedia Computer Science*, 52:956–961, 2015.

- [128] Elise Van der Pol and Frans A Oliehoek. Coordinated deep reinforcement learners for traffic light control. *Proceedings of Learning, Inference and Control of Multi-Agent Systems (at NIPS 2016)*, 2016.
- [129] Zheng Zeng. Graphlight: Graph-based reinforcement learning for traffic signal control. In *2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS)*, pages 645–650. IEEE, 2021.
- [130] Hao Huang, Zhiqun Hu, Zhaoming Lu, and Xiangming Wen. Network-scale traffic signal control via multiagent reinforcement learning with deep spatiotemporal attentive network. *IEEE transactions on cybernetics*, 2021.
- [131] Junchen Jin and Xiaoliang Ma. Adaptive group-based signal control by reinforcement learning. *Transportation research procedia*, 10:207–216, 2015.
- [132] Saad Touhbi, Mohamed Ait Babram, Tri Nguyen-Huu, Nicolas Marilleau, Moulay L Hbid, Christophe Cambier, and Serge Stinckwich. Adaptive traffic signal control: Exploring reward definition for reinforcement learning. *Procedia Computer Science*, 109:513–520, 2017.
- [133] Wade Genders and Saiedeh Razavi. Using a deep reinforcement learning agent for traffic signal control. *arXiv preprint arXiv:1611.01142*, 2016.
- [134] Li Li, Yisheng Lv, and Fei-Yue Wang. Traffic signal timing via deep reinforcement learning. *IEEE/CAA Journal of Automatica Sinica*, 3(3):247–254, 2016.
- [135] Guanjie Zheng, Xinshi Zang, Nan Xu, Hua Wei, Zhengyao Yu, Vikash Gayah, Kai Xu, and Zhenhui Li. Diagnosing reinforcement learning for traffic signal control. *arXiv preprint arXiv:1905.04716*, 2019.
- [136] Jaun Gu, Minhyuck Lee, Chulmin Jun, Yohee Han, Youngchan Kim, and Junwon Kim. Traffic signal optimization for multiple intersections based on reinforcement learning. *Applied Sciences*, 11(22):10688, 2021.
- [137] Kaidi Yang, Isabelle Tan, and Monica Menendez. A reinforcement learning based traffic signal control algorithm in a connected vehicle environment. In *17th Swiss Transport Research Conference (STRC 2017)*. STRC, 2017.

- [138] HM Abdul Aziz, Feng Zhu, and Satish V Ukkusuri. Learning-based traffic signal control algorithms with neighborhood information sharing: An application for sustainable mobility. *Journal of Intelligent Transportation Systems*, 22(1):40–52, 2018.
- [139] Bunyodbek Ibrokhimov, Young-Joo Kim, and Sanggil Kang. Biased pressure: cyclic reinforcement learning model for intelligent traffic signal control. *Sensors*, 22(7):2818, 2022.
- [140] Jorge A Laval and Hao Zhou. Large-scale traffic signal control using machine learning: some traffic flow considerations. *arXiv preprint arXiv:1908.02673*, 2019.
- [141] Chacha Chen, Hua Wei, Nan Xu, Guanjie Zheng, Ming Yang, Yuanhao Xiong, Kai Xu, and Zhenhui Li. Toward a thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3414–3421, 2020.
- [142] Chunming Liu, Xin Xu, and Dewen Hu. Multiobjective reinforcement learning: A comprehensive overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(3):385–398, 2014.
- [143] Itamar Arel, Cong Liu, T Urbanik, and AG Kohls. Reinforcement learning-based multi-agent system for network traffic signal control. *IET Intelligent Transport Systems*, 4(2):128–135, 2010.
- [144] Bram Bakker, Shimon Whiteson, Leon Kester, and Frans CA Groen. Traffic light control by multiagent reinforcement learning systems. In *Interactive Collaborative Information Systems*, pages 475–510. Springer, 2010.
- [145] RR Negenborn and H Hellendoorn. Intelligence in transportation infrastructures via model-based predictive control. In *Intelligent Infrastructures*, pages 3–24. Springer, 2010.
- [146] Anastasios Kouvelas, Dimitris Triantafyllos, and Nikolas Geroliminis. Two-layer hierarchical control for large-scale urban traffic networks. In *2018 European Control Conference (ECC)*, pages 1295–1300. IEEE, 2018.
- [147] Junchen Jin and Xiaoliang Ma. Hierarchical multi-agent control of traffic lights based on collective learning. *Engineering applications of artificial intelligence*, 68:236–248, 2018.

- [148] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.
- [149] Nan Wu, Yuan Xie, and Cong Hao. Ironman: Gnn-assisted design space exploration in high-level synthesis via reinforcement learning. In *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, pages 39–44, 2021.
- [150] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada*, volume 30, 2018.
- [151] Qingbiao Li, Fernando Gama, Alejandro Ribeiro, and Amanda Prorok. Graph neural networks for decentralized multi-robot path planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11785–11792. IEEE, 2020.
- [152] Hang Zhu, Varun Gupta, Satyajeet Singh Ahuja, Yuandong Tian, Ying Zhang, and Xin Jin. Network planning with deep reinforcement learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 258–271, 2021.
- [153] Paul Almasan, José Suárez-Varela, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Deep reinforcement learning meets graph neural networks: exploring a routing optimization use case. *Computer Communications*, 196:184–194, 2022.
- [154] Yun Peng, Byron Choi, and Jianliang Xu. Graph learning for combinatorial optimization: a survey of state-of-the-art. *Data Science and Engineering*, 6(2):119–141, 2021.
- [155] Daniele Gammelli, Kaidi Yang, James Harrison, Filipe Rodrigues, Francisco C Pereira, and Marco Pavone. Graph neural network reinforcement learning for autonomous mobility-on-demand systems. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 2996–3003. IEEE, 2021.
- [156] Eli Meirom, Haggai Maron, Shie Mannor, and Gal Chechik. Controlling graph dynamics with reinforcement learning and graph neural

- networks. In *International Conference on Machine Learning*, pages 7565–7577. PMLR, 2021.
- [157] Tomoki Nishi, Keisuke Otaki, Keiichiro Hayakawa, and Takayoshi Yoshimura. Traffic signal control based on reinforcement learning with graph convolutional neural nets. In *2018 21st International conference on intelligent transportation systems (ITSC)*, pages 877–883. IEEE, 2018.
 - [158] François-Xavier Devailly, Denis Larocque, and Laurent Charlin. Ig-rl: Inductive graph reinforcement learning for massive-scale traffic signal control. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):7496–7507, 2021.
 - [159] Shimon Komarovsky and Jack Haddad. Spatio-temporal graph convolutional neural network for traffic signal control in large-scale urban networks (under review). 2023.
 - [160] Zhen Zhao. Variants of bellman equation on reinforcement learning problems. In *2nd International Conference on Artificial Intelligence, Automation, and High-Performance Computing (AIAHPC 2022)*, volume 12348, pages 467–478. SPIE, 2022.
 - [161] Chayoung Kim. Deep reinforcement learning by balancing offline monte carlo and online temporal difference use based on environment experiences. *Symmetry*, 12(10):1685, 2020.
 - [162] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of sumo-simulation of urban mobility. *International journal on advances in systems and measurements*, 5(3&4), 2012.
 - [163] Alvaro Cabrejas-Egea, Raymond Zhang, and Neil Walton. Reinforcement learning for traffic signal control: Comparison with commercial systems. *Transportation research procedia*, 58:638–645, 2021.
 - [164] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 2019.
 - [165] Rui Zhang, Yunxing Zhang, and Xuelong Li. Graph convolutional auto-encoder with bi-decoder and adaptive-sharing adjacency. *arXiv preprint arXiv:2003.04508*, 2020.

- [166] Longchao Da, Minchiuan Gao, Hao Mei, and Hua Wei. Llm powered sim-to-real transfer for traffic signal control. *arXiv preprint arXiv:2308.14284*, 2023.
- [167] Openai. 2023a. openai. chatgpt plugins.
- [168] Openai. 2023b. openai. function calling and other apiupdates.
- [169] langchain. 2023. langchain. introduction — langchain.
- [170] Auto-gpt: An autonomous gpt-4 experiment.
- [171] Babyagi. 2023. translations: — babyagi.
- [172] Siyao Zhang, Daocheng Fu, Wenzhe Liang, Zhao Zhang, Bin Yu, Pinlong Cai, and Baozhen Yao. Trafficgpt: Viewing, processing and interacting with traffic foundation models. *Transport Policy*, 150:95–105, 2024.
- [173] Michael Villarreal, Bibek Poudel, and Weizi Li. Can chatgpt enable its? the case of mixed traffic control via reinforcement learning. In *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, pages 3749–3755. IEEE, 2023.
- [174] Yiqing Tang, Xingyuan Dai, Chen Zhao, Qi Cheng, and Yisheng Lv. Large language model-driven urban traffic signal control. In *2024 Australian & New Zealand Control Conference (ANZCC)*, pages 67–71. IEEE, 2024.
- [175] Yiqing Tang, Xingyuan Dai, and Yisheng Lv. Large language model-assisted arterial traffic signal control. *IEEE Journal of Radio Frequency Identification*, 2024.
- [176] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.
- [177] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

- [178] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [179] Acheampong Francisca Adoma, Nunoo-Mensah Henry, and Wenyu Chen. Comparative analyses of bert, roberta, distilbert, and xlnet for text-based emotion recognition. In *2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pages 117–121. IEEE, 2020.
- [180] Boris V Cherkassky, Andrew V Goldberg, and Tomasz Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical programming*, 73(2):129–174, 1996.
- [181] Amgad Madkour, Walid G Aref, Faizan Ur Rehman, Mohamed Abdur Rahman, and Saleh Basalamah. A survey of shortest-path algorithms. *arXiv preprint arXiv:1705.02044*, 2017.
- [182] Fatemeh Salehi Rizi, Joerg Schloetterer, and Michael Granitzer. Shortest path distance approximation using deep learning techniques. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 1007–1014. IEEE, 2018.
- [183] Yiding Yang, Xinchao Wang, Mingli Song, Junsong Yuan, and Dacheng Tao. Spagan: Shortest path graph attention network. *arXiv preprint arXiv:2101.03464*, 2021.
- [184] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. 2018.
- [185] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.
- [186] Jianfeng Dong, Xirong Li, and Cees GM Snoek. Predicting visual features from text for image and video caption retrieval. *IEEE Transactions on Multimedia*, 20(12):3377–3388, 2018.
- [187] Ran Xu, Caiming Xiong, Wei Chen, and Jason J Corso. Jointly modeling deep video and compositional text to bridge vision and language in a unified framework. In *AAAI*, volume 5, page 6. Citeseer, 2015.
- [188] Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. Grounded compositional semantics for finding and

- describing images with sentences. *Transactions of the Association for Computational Linguistics*, 2:207–218, 2014.
- [189] Kyuyeon Hwang and Wonyong Sung. Character-level language modeling with hierarchical recurrent neural networks. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5720–5724. IEEE, 2017.
 - [190] Xiaolei Diao, Xiaoqiang Li, and Chen Huang. Multi-term attention networks for skeleton-based action recognition. *Applied Sciences*, 10(15):5326, 2020.
 - [191] Hao Cheng, Dongze Lian, Bowen Deng, Shenghua Gao, Tao Tan, and Yanlin Geng. Local to global learning: Gradually adding classes for training deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4748–4756, 2019.
 - [192] Ricardo Cerri, Rodrigo C Barros, and André CPLF De Carvalho. Hierarchical multi-label classification using local neural networks. *Journal of Computer and System Sciences*, 80(1):39–56, 2014.
 - [193] Duy-Kien Nguyen and Takayuki Okatani. Multi-task learning of hierarchical vision-language representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10492–10501, 2019.
 - [194] Jiuxiang Gu, Handong Zhao, Zhe Lin, Sheng Li, Jianfei Cai, and Mingyang Ling. Scene graph generation with external knowledge and image reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1969–1978, 2019.

תקציר

מחקר זה נועד לפתח מערכת ניהול תנועה אוטומטית, שתוכל למש פקודות של מפעילי תנועה באמצעות הפעלת רמזורים. מערכת ניהול התנועה צריכה לטפל בשלושה נושאים עיקריים בראשות עירוניות בקנה מידה גדול: (ז) סקלబיליות של הרשת, (ז'י) אי-ליניאריות בדינמיקת התנועה ויחסים מורכבים בנתונים, ו-(ז''י) זמינות של ביג דטה הטרוגני.

כדי להשיג את מטרת המחקר, פותחה גישת למידה עמוקה. המודלים השונים שפותחו מבוססים על מבנה הגרפים של רשת התחבורה.

המחקר נערך בשלושה שלבים. מטרתו של שלב 1 היא להציג לדמותו ביצוע פקודות. זה מתבצע על מערך נתונים נתון, שיש לו תוצאות שונות: משך זמן יрок (GT) המבוסס על פקודות תוכנית רמזור (SP) שתוכננה למטרה של עיכוב מינימלי.

שלב 2, המטרה הבסיסית הורחבה כך שתכלול עדיפות לתנועות שונות. בשלב 3 הפקנו משקלים העדפות מהירות טקסט, ולפעמים מנתונים נוספים, כגון זמן נסעה היסטוריים, אם ברצונו להשתמש בהם כדי לתת עדיפות לנטייה הקצר ביותר מצומת מקור כלשהו לצומת יעד.

המערכת הסופית צריכה לישם את המטרה הרצiosa של מחקר זה. היא עשוה זאת על-ידי שרשור שלבים 3 ו-2 ברכף. כמובן, בהינתן פקודות טקסט כלשהי הנכנסת בשלב 3, הפלט הוא משקלים העדפות בכל צומת בגרף המציג את רשת התחבורה. לאחר מכן, פلت זה הופך להיות הקלט למערכת של שלב 2, אשר בתורו מייצר את אות הבקרה המתאים בכל הצמתים בראשת התחבורה.

שלב 1 מאמן באמצעות למידה מפוקחת, על ידי מודל רשת עצבית של גרפ מרחב-זמן (STGCN), תוך שימוש בראשת העירונית רחבות ההיקף של תל אביב.

הוא מורכב מתכונות כגון מהירות, המופוקות ממערך חיישני Bluetooth הממוקמים בצמתים של הרשת, ו- SPs המציגים במקרה שלנו גם את הפקודות של מפעילי התנועה במרכז בקרת התנועה.

שלב 1 היה הניסיון הראשון שלנו לישם את המטרה העיקרית שלנו של יישום פקודות בראש תחבורה. עם זאת, הפקודות היו פשוטניות מדי. לפיכך, הוצעה משפה כללית יותר של פקודות לשלב 2: פקודות העדפה.

בשל מספר סיבות, כגון היעדר מערך נתונים למטרה זו, אנו משתמשים בגישה למידה מחזוקים (RL). באופן ספציפי, אנו משתמשים בראשת Q عمוקה (DQN) ומישמים אותה בסימולציה תחבורה SUMO. המודל הסופי הוא מערכת משוב סגורה, בניגוד למערכת המשוב הפתוח בשלב 1. במטרה זו, אנו משתמשים

בתכונות דומות עברו הקלט של DQN, המכונה מצב ב- RL, בעוד שהפלט הוא תוחלת התגמול הכלול בעבר כל אחד מהצמתים. לאחר מכן אנו בוחרים פעולה, מתוך קבוצה של פעולות מוגדרת מראש, בהתבסס על הפלטים מה- DQN, שיוחלו ברשות ההתחורה.

מכיוון שצעד זה היה מוצלח למדי, החלטנו להסיר את כוונון המשקלים הידני לכל קצה ברשות ההתחורה, וליצג את הפוקודה בצורה טبيعית וקלה יותר: על ידי טקסט. שלב 3 יועד למטרה זו. בשלב זה השתמשנו הן במשחק הדיאלוג ChatGPT ובעיקר במודל GPT לצורך כוונון עדין למשימה הספציפית שלנו.

שלב 1 הראה את היתרון של מודל STGCN שלנו על פני מודלים גרפיים פוטנציאליים אחרים של במידה עמוקה:

הביצועים הטובים יותר של נתוני צעדי זמן מחרוז סינכרוניים מסוננים בהשוואה לנוטוני הצעדים הקבועים האסינכרוניים המקוריים שמוסכו למשך זמן של 5 דקות, הביצועים הטובים יותר של תכונות GT במשימת חיזוי GT בהשוואה להכללת תכונות BT, עקב מדידות שונות, והביצועים הטובים יותר של הוספה תכונות רבות ככל האפשר למשימת חיזוי המהירות.

שלב 2 הראה שההעדפה עובדת בסימולציה כאשר משקלי העדפות מוכפלים הן עם המצב והן עם התגמול, היתרון של תקשורת בין סוכנים באמצעות שיטות (STGCN, תגמול גלובלי, תכונות במورد התנועה), החוסן (רגישות לרעש) של דגמים שאומנו על משקלות העדפות משתנות בהשוואה לדגמים שאומנו על משקלות העדפה קבועות, והחשיבות של תכנון נכון של העדפה, בין אם באמצעות בחירת ערכי העדפה קבועים או יחסיים בין תנויות, ובין אם על ידי בחירת התנויות הנכונות כדי למנוע קונפליקטים פנימיים בין תנויות מועדות.

שלב 3 הראה כי כוונון עדין של מודלי שפה עבור המשימה הספציפית שלנו עדיף על הסטמוכות על שימוש במערכת הדו-שיך ChatGPT עם קבוצה קטנה של דוגמאות. הוא גם הראה הכללה גורעה על תנאים שונים מהנתונים שהמודלים אומנו עליהם, אם כי קל להציג אותם עבור בני אדם.
לבסוף, התוצאות מהשלבים לעיל הראו מספר נקודות:

(i) הוכחת היתכנות להעדפות בסימולציה של רשות ההתחורה עירונית, במיוחד עבור מקרים מורכבים עם העדפות יחסיות. (ii) שלב נוסף לשילוב שלבים 3-2 הדגים מערכת ניהול אוטומטית הפעלת במלואה. כמו כן, גרסה פשוטה יותר של מערכת צו נבדקה בשלב 1. (iii) לאור כל השלבים, עקרון התקשרות בין סוכנים מבוסנה רב-סוכנים הוכח כמועיל בהשוואה להיעדרו. (iv) הראיינו במהלך הפרקים שהמערכת שלנו מטפלת בשלושה נושאים חיוניים ברשותות עירונית בקנה מידה

גדול, שהם: סקלאלביות, אי-ליניאריות וביג דאטה. (v) השלבים השונים הראו שיש התאמה בין סוג הנתונים לבין הארכיטקטורה העצבית העמוקה לעיבודם. לדוגמה, נתונים גרפיים באמצעות רשתות עצביות גרפיות, עיבוד איטרטיבי באמצעות רשתות עצביות נישנות, נתונים טקסטואליים באמצעות מודלים של שפה ועוד. (vi) חשיבות ניקוי הנתונים בלמידה عمוקה ועקבויות נתונים, במיוחד בשלב 2.

מחבר/ת חיבור זה מצהיר/ה כי המחבר, כולל איסוף הנתונים, עיבודם והציגם, התייחסות והשוואה למחקרים קודמים וכו', נעשה כלו בצורה ישירה, כמצופה מחקר מדעי המבוצע לפי אמות המידה האתניות של העולם האקדמי. כמו כן, הדיווח על המחבר ותוציאותיו בחיבור זה נעשה בצורה ישירה ומלאה, לפי אותן אמות מידה.

המחקר נעשה בהנחיית פרופסור חבר ג'אק חדאד בפקולטה הנדסה
ازרחות וסביבה.

בינה מלאכותית בברית רמזורים עבר רשתות עירוניות גדולות

חיבור על מחקר

לשם مليוי חלקו של הדרישות לקבלת התואר דוקטור לפילוסופיה

שמעון קומרובסקי

הוגש לסנט הטכניון - מכון טכנולוגי לישראל

טבת ה'תשפ"ד חיפה דצמבר 2023

בינה מלאכותית בברית רמזורים עבר רשתות עירוניות גדולות

שמעון קומרובסקי