KAZAKH-BRITISH TECHNICAL UNIVERSITY

UDK 004.89, 519.876.2

TOLEBI GULNUR ABDYKADYRKYZY

Adaptive Control of Traffic Flows

6D070500 Mathematical and Computer Modeling

Dissertation submitted in fulfillment of the requirements for the degree Doctor of Philosophy (PhD)

Supervisors:
Doctor of Sciences in physics and mathematics,
Prof. N.S. Dairbekov
Satbayev University

PhD, Prof. H-K. Lam
King's College London, UK

Republic of Kazakhstan
Almaty, 2021

Contents

Normative References

This thesis uses references to the following standards:

- ”Instructions for the preparation of a dissertation and author’s abstract” Ministry of education and science of the Republic of Kazakhstan, 377-3 zh.
- GOST 7.32-2001. Report on research work. Structure and design rules.
- GOST 7.1-2003. Bibliographic record. Bibliographic description. General requirements and compilation rules.
- GOST 7.32-2017. System of standards of information, librarianship and publishing. Research report. Structure and design rules.

Symbols and abbreviations

A_i - set of actions
AI - Artificial Intelligence
ANN - Artificial Neural Network
ARIMA - Autoregressive integrated moving average
b_j^k - bias
c_j^k - results of node j at the layer k after convolution
CI - Computational Intelligence
CNN - Convolutional Neural Network
CPU - a central processing unit
DL - Deep Learning
EC - Exolutionary Computations
EW - East-West
FCNN - Fully-connected neural network
FL - Fuzzy Logic
F_i, t - link flow on the link i at time t
GPU - a graphics processing unit
GRU - Gated Recurrent Unit
h_{t-1} - holds the information for the previous t-1 units
h_t^j - current memory content, use the reset gate to store the relevant information from the past
kNN - k-nearest neighbours
LSTM - Long-Short-Term memory
MARL- Multi-agent Reinforcement Learning
MDP - Markov Decision Process
MSE - Mean Squared Error
n - number of estimations
NLP - Natural Language Processing
NS - North-South
OD - Origin-destination
pooling - layer of CNN that would reduce the number of parameters when the images are too large
r_t^j - reset gate of GRU is to decide how much of the past information to forget
RL - Reinforcement Learning
RNN - Recurrent Neural Network
SI - Swarm Intelligence
SN - South-North
SUMO DLR - Simulation of Urban MObility
SVR- Support Vector Regression
tanh - nonlinear activation function. It is bound to the range (-1, 1)
TSC - Traffic Signal Control

U - weight of h_{t-1}
U_z - own weight of update gate for h_{t-1}
U_r - own weight of reset gate for h_{t-1}
WE - West-East
W - weights of input x_t
W_r - own weights of reset gate for x_t
W_z - own weights of update gate for x_t
w_{ij}^k -weights at the layer k
x_t - data at timestep t
x_i^{k-1} - input data at k-1 layer
x_j^k - output data of j at the layer k after applying activation function
x_j^{k+1} - output data after pooling layer
Y_i - predicted value
$\widehat{Y}_i$ - target value
x_t^j - update gate z for timestep t in the GRU, which decides how much the past state affects the current state
1D - one dimensional
2D - two dimensional
θ- activation function
σ- sigmoid activation function is applied to squash the result between 0 and 1

Chapter 1

Introduction

General characteristics of the work. The given work is devoted for research and development of the link flow estimation and adaptive traffic signal control models to reduce the traffic congestion in the transport network. The study proposed estimation and control models based on Computational Intelligence Techniques.

Relevance of the work. Nowadays the rapid growth of vehicles in urban areas is a serious problem. Traffic congestion entails many negative consequences, such as travel delays, air pollution and health problems. Therefore, there is an urgent need in managing traffic flows as efficiently as possible.

A transport network is a complex non-stationary open environment with multiple heterogeneous stochastic agents such as intersection controllers and road users. Changes in such a network may occur due to random demand fluctuations, supply degradation, or actions of agents. Thus, it makes the task of adaptive Traffic Signal Control (TSC) extremely challenging.

Goal of the research. The goal of the work is researching and developing the effective methods for link flow estimation and traffic signal control by using Computational Intelligence Techniques.

Objectives of the research. To achieve this goal we set several objectives:

- A review of previous researches in traditional traffic signal control and computational intelligence methods for development of effective traffic signal controllers.
- An analysis of computational intelligence techniques including neural network, reinforcement learning, fuzzy logic systems, multi-agent systems for traffic signal management.
- Building an experimental environment on a simulator.
- Designing online model-free Q-learning traffic signal controller for an isolated intersection.
- Development of the Reinforcement Learning intersection controller with deep Q-network.

- Designing link flow estimation models using deep learning techniques.
- Development of the adaptive traffic signal controller based on Artificial Neural Network
- Implementing simulation and computational experiments

Subject and Object of the research. The object of the research is a traffic flow in a transport network. The subject of the study is a model for intelligent traffic flow management.

Research methods. The main research approaches are computational intelligence techniques. The design of control and estimation models is based on simulations of the traffic flow at the microscopic level. The Reinforcement Learning method is used for adaptive control of traffic flows. The tabular method is used for a simple formulation of the problem, and the Deep QNetwork approach for an extended view. In addition, a different approach for traffic management is presented, where the tasks of estimation and control are separated. Deep learning methods are used to build a link flow estimation model. The data is treated as a time series. The models are based on convolutional and recurrent neural networks. Estimation-based control is performed by a model based on a feed-forward fully connected neural network. Simulator SUMO is used for building experimental environments. Numerical experiments are used to evaluate the quality of models and make appropriate corrections if necessary. Synthetic data obtained by generating on the simulator were used for the experiments.

Scientific novelty of the work. Adaptive traffic controller that works without any knowledge of the environment is proposed in the given work. System based on Reinforcement Learning method, where the reward function is one of the main components, is designed. Novel reward function is invented. Singularity of the given reward function is that it consists of equilibrium and queue reduction terms.

In the given work together with lots of advantages of RL there were identified some disadvantages in the scope of TSC problem. For instance, it is basically designed for closed environments rather than for dynamically changing ones. If an RL agent is placed in the dynamically changing environment, every time the demand changes it has to relearn its decision making policy. One of the other disadvantages of RL is that the agents are designed to operate without reference to each other. They are not good at collaborative work since any actions of a single agent changes the environment for the other ones. Hence, it affects the stationary assumption. Moreover, when the agents work in collaboration, they have to share their states and coordinate actions. That implies the exponential rise of the state and action space as the number of agents increments.

A number of deep learning models proposed in this thesis represent a new approach to estimating link flows in transport networks. The link flow is treated as the probability of vehicles being generated in unit time. The value is not the exact proportion of the vehicles arrived, but represents some other properties of traffic flow in

the given link. The models that currently dominate in scientific journals use instant results as the number of halting cars or waiting time at the given moment. However, we consider input data as a timeseries. There are two key issues when time series are used for flow estimation: the time series analysis requires the process to be stationary, however, the traffic flow is dynamically changing; the spatial characteristics of the time series data are not taken into account in the classical methods. However, when sequential data are considered it is important to take into account a more deep representation of features. The given issues have been solved by a hybrid model of RNN and CNN, which was first proposed in this study. The results obtained in this dissertation will significantly advance the field of AI in TSC.

Key statements to be defended. According to the results of the study, the following statements are to be defended:

- the state of the problem and existing solutions is described;
- online model-free Q-learning based adaptive traffic signal controller is proposed (Q-table, Deep-QNetwork);
- novel reward formulas are presented;
- possibility to predict the link flow in the near future based on pregained data, if given data goes back and is sampled every period of time is proved;
- novel link flow estimation models based on deep learning are presented;
- solving the problem of TSC using the MAS approach for a non-stationary environment is proposed;
- estimation of optimal signal plan for a given traffic network if we know the demand is implemented;
- numerical experiments results and discussions are provided.

Theoretical and practical value of the work. The algorithms and models developed during the research can be used for developing new methods for traffic signal control. They also can be integrated in any kind of projects related to the Intelligent Transportation System. The obtained research results can be used for a further theoretical investigation of the given topic, in the traffic control, traffic flow prediction and for the optimization of the existing traffic signal control systems.

The performed studies allow us to expand and deepen knowledge in the field of adaptive control of traffic flows and consider the problem as a task of Artificial Intelligence.

The degree of validity and reliability. The reliability of the obtained results was proved on the basis of algorithms and computational experiments with the use of simulator, publications in journals, including journals cited in SCOPUS, 2 certificates of copyright, as well as numerous discussions at seminars held in King's

College London (London, UK), Kazakh-British Technical University, Satbayev University, School of Mathematical and Computer Modeling.

The relation of topic to the plans of scientific research programs. The thesis is carried out as part of research work "Software development for 3D modeling, online monitoring and prediction of the air contamination level in urban and industrial areas" no.BRO05236316 performed at the Satbayev University MES RK according to financial program, designed for 2018-2020 years.

Publications. 9 works are published, including: 3 in journals recommended by committee, 1 in the journal indexed by SCOPUS, 4 in the proceedings of international conferences (3 indexed by SCOPUS):

1. Link Flow Estimation on an Isolated Intersection Based on Deep Learning Models. Gulnur Tolebi, Nurlan S. Dairbekov , Daniyar Kurmankhojayev. International Review of Automatic Control (I.RE.A.CO.), Vol. 13, N. 1. ISSN 1974-6059, pp. 19-26, January 2020.

2. Deep learning models for link flow estimation. Tolebi G., Dairbekov N. Traditional April international mathematical conference in honor of Day of science workers of the Republic of Kazakhstan,dedicated to the 1150th anniversary of Abu Nasir al-Farabi and the 75th anniversary of the Institute of mathematics and mathematical modeling, pp 141-142, Almaty, 2020.

3. Road Traffic Demand Estimation and Traffic Signal Control. Kurmankhojayev D., Tolebi G., Dairbekov N. The 5th International Conference on Engineering; MIS 2019. June 6–8, 2019, Astana, Kazakhstan. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3330431.3330433. (SCOPUS)

4. Link flow estimation using neural network. Tolebi G., Kurmankhojayev D. Herald of the KBTU, Vol. 16, Issue 2, pp. 170-174, June 2019.

5. Analysis of existing traffic detectors for taking measurements. Tolebi G., Kurmankhojayev D., Herald of the KBTU, Vol. 16, Issue 1, pp. 73-79, March 2019.

6. Adaptive control of traffic flows. Dairbekov N.S., Kurmankhojayev D., Tolebi G. III International innovation Forum "Digital Kazakhstan: sustainable development of urban planning systems in the XXI century, pp. 120-124, April 2019.

7. Reinforcement Learning Intersection Controller. Tolebi G., Dairbekov N.S., Kurmankhojayev D., Mussabayev R. Proceedings of 14th International Conference on Electronics and Computer and Computation (ICECCO), pp. 65-69, Electronic ISBN: 978-1-7281-0132-3, DOI:10.1109/ICECCO.2018.8634692, Kaskelen, 2018. (SCOPUS)

8. Analysis of the traffic flow modeling systems. Kurmankhojayev D., Tolebi G., Herald of the KBTU, ISSN 1998-6688, Vol. 15, Issue 4, pp.86-91, December 2018

9. Online model-free adaptive traffic signal controller for an isolated intersection. Kurmankhojayev D., Suleymenov N., Tolebi G. XIII International Asian School-seminar "Problems of complex systems' optimization" in the scope of International multi-conference IEEE SIBIRCON 2017, pp. 109-112, Novosibirsk, Russia, 2017. (SCOPUS)

Acknowledgments. The author is grateful to the scientific supervisor Doctor of Sciences in physics and mathematics, Professor N.S. Dairbekov, external supervisor PhD, Professor H-K.Lam, and also colleagues in the project team for invaluable assistance in completing the dissertation work.

Structure and scope of the thesis. The thesis is presented on 92 pages of typewritten text. It consists of normative references, list of symbols and abbreviations, introduction, five main chapters, conclusion and references. The dissertation includes 19 tables, 26 figures. List of references consists of 91 titles.

First chapter presents an introduction that describes the problems of the dissertation work and its content.

In the second chapter general concepts of traffic signal control are described. Background information about basic notations, general architectures for traffic optimization, simulation types are given. In addition, detailed review of the existing traffic signal control methods are shown.

Third chapter presents an overview of Reinforcement Learning methods. Elements and basic types of the given methods are described in the current section.

In the fourth chapter traffic signal control method based on Reinforcement Learning is proposed. Two approaches are implemented in the scope of the TSC problem. Numerical experiments, simulations and results are shown in the given chapter.

Fifth chapter proposed link flow estimation module based on deep learning models. An overview of the method and experimental results with appropriate tables and charts with their discussion are given. In addition, comparative analysis of the proposed models are presented.

In the sixth chapter adaptive traffic signal controller using link flow estimations, that is designed on the basis of ANN is presented. Related works and proposed methods are described in the chapter. Experimental results and discussion are given.

Conclusion provides the main outcomes of the current work.

Chapter 2

Traffic Signal Control (TSC) Background

The world is experiencing population growth and urbanization, which leads to an increment in the number of vehicles in transport networks. This fact points to the need to create efficient transport systems. Congestion that takes a long time strongly affects the social, economic, environmental and psychological sphere of life of the population. Reconstruction of road systems or modification of infrastructure is one of the methods to raise highway capacity. However, this is very expensive, and sometimes not a real approach. Therefore, the optimization of the existing control system through artificial intelligence is a necessary topic for research. Intelligent traffic con- trol systems ensure efficient operation of traffic lights in real time. Such systems have the capability to adjust to changes in the situation on the roads. With optimal management, the following goals can be achieved: minimizing the number of waiting cars, reducing delays at intersections,the safety of all users of the transport network,as well as balanced traffic at intersections. The effective management of vehicles entails a reduction in emissions into the around, which has a positive impact on the environment and public health.

Various scientific studies [1],[2],[3] are conducted in the field of implementation and use of computational intelligence techniques for solving the task of intelligent traffic signal control. Due to their ability to learn, these methods have the force to figure out concrete issues. The purpose of this work is to apply computational intelligence methods to the problem of traffic signal control.

Traffic signals or traffic lights, traffic signal controllers are signaling devices for controlling conflicting directions in the road space. Traffic lights are installed at intersections, pedestrian crossings, where they give the right to move in one direction for a certain length of time. The first traffic lights appeared in the early 20th century [4]. With the change in road transport infrastructure, traffic lights were also modified. There are three main generations of traffic signals: fixed, actuated and adaptive. The fixed control has a predetermined sequence of traffic light signals duration. It is completely fixed and unchangeable. The second type of traffic lights has a specific

plan of action. If a traffic jam is detected on the road, the traffic light changes the signal plan to a predefined one. That is various kinds of signal plans are provided in advance for distinct situations. An inductive loop detector [5] is used to detect traffic jams. This method is not fully adaptive, but at the same time adapts to the situation in real time. The adaptive traffic signal controller is trained and optimized gradually, adapts to the environment, is able to detect changes on the roads and make a decision in real time. Information about road conditions is obtained from detectors installed on road sections or from outdoor surveillance cameras, and data can also be obtained through GSM operators. After receiving the data, the corresponding duration time of each traffic light signal is calculated. More detailed explanations about the traffic light controller are provided in the following section.

This chapter provides a review of related materials and the basic terminologies used in traffic signal controlling methods. Also types and comparison of existing traffic light control methods, general architectures that can be applied for traffic optimization and types of simulation are included.

2.1 Literature review

A signalised intersection is designed for the efficient and safe passage of conflicting directions of traffic flow in time. The main unit in the traffic signal control system is a *signal group*, which defines a set of lights that matches to certain traffic in the intersection: green, red, amber.

- *Green signal time* - period of time while the movement of vehicles in the given lane of intersection is allowed.
- *Red signal time* - period of time in which movement is not allowed.
- *Amber* - requires to slow down and prepare for the traffic light to switch to the red signal.

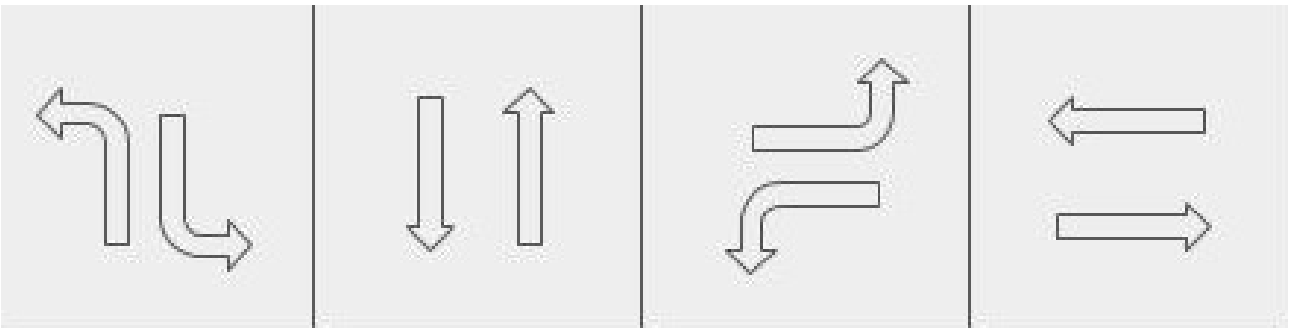

Figure 2.1: Sample of 4 phases

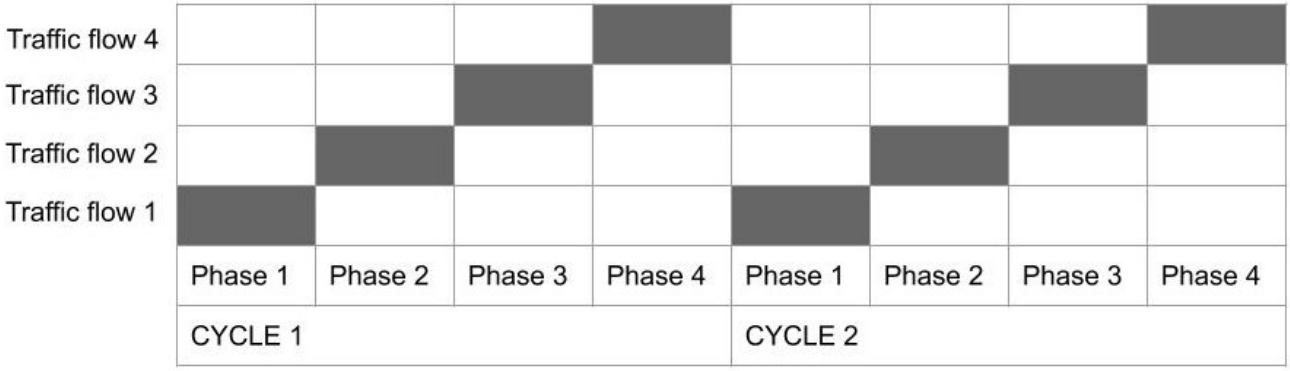

Figure 2.2: Sample of phase-cycle-traffic flow relationship.

- *A phase* - is a part of cycle length that corresponds to a combination of traffic movements that have the order of passage for the green signal time (Figure 2.1). These movements follow simultaneously without conflicting with other traffic flows. The sum of all phases gives a cycle length (Figure 2.2).
- *Link* is an adjacent lane group where traffic forms a single queue.
- *Cycle length (cycle time)* - the time during which each phase has been enabled once.
- *Split* is the duration of each phase in a cycle. It represents a distribution of the cycle time to the individual phases. Represents the green time allocated to each phase.
- *Offset* is a time shift among the beginning of a green signal for neighboring intersections to ensure the devoid movement of vehicles without encountering all red signals.
- *Delay* is a total averagehalting time per vehicle in each lane in the transport network.
- *Safety requirements* - Due to presence of several conflicting traffic flows at each intersection, some safety requirements must be realized. First, each signal plan must not proceed at the same time with the conflicting traffic flows. Second, there must be a certain short period of time when all the traffic flows stop before switching signal plans. This will let vehicles that entered the intersection exit it without disturbing conflicting traffic flow. In addition, there should be implemented some requirements that ensure the safety of pedestrians [6]. As it was mentioned above they fall outside of the given work.

Traffic signal controllers based on architecture consider general three strategies:

- Fixed-time control
- Actuated control

- Adaptive or intellectual control

2.1.1 Fixed-time control

Fixed-time or so called pre-timed control method is an algorithm of offline optimization, where the duration of phase orders is fixed and does not adapt according to changes in traffic flow. The given approach takes the traffic demand as stable. Signal time and cycle length are selected offline and based on data obtained previously. In comparison with other methods fixed-time control reduces the algorithm implementation cost due to the absence of any detectors that should consider traffic jams and other situations that occur at the current time.

The control parameters in the fixed-time controller are mostly calculated based on Highway Capacity Manual (HCM) [7] or Webster formula [8]. Webster method is described below. The best cycle length is calculated as following:

$$C = \frac{1.5L + 5}{1 - \sum_{i=1}^{n} y_{ci}} \tag{2.1}$$

where

- C is proper cycle length in seconds;
- L is a total unusable time at each cycle in seconds;
- y_{ci} is critical lane volume each phase/saturation flow
- n is a number of phases in one cycle.

The best green time that is depending on v/s relation is calculated as following:

$$g_i = \frac{(\frac{v}{s})_{ci}}{\sum_{i=1}^{n} (\frac{v}{s})_{ci}} xG \tag{2.2}$$

where

- g_i is optimal green time duration in phase i;
- v is flow rate;
- s is saturation flow rate;
- $G = C - L$ is the total green time duration in one cycle in seconds.

The HCM method is based on the similar approach as the Webster method. To allocate green time among different signal phases, which is to equalize the degree of

saturation (v/c ratio) of the critical lane group of each signal phase [7]. The saturation degree is calculated as the following:

$$X_{ci} = \frac{v_{ci}}{c_{ci}} = \frac{v_{ci}C}{s_{ci}g_{ci}} \tag{2.3}$$

where

- C is cycle length in seconds;
- X_{ci} is critical degree of saturation for phase i;
- $c_{ci} = s_{ci}\frac{g_{ci}}{C}$ is critical lane group capacity for phase i;
- g_{ci} is green time in phase i (s)

X_c, the critical $\frac{v}{c}$ ratio for the whole intersection is defined as

$$X_c = \sum_{i=1}^{n} (\frac{v}{s})_{ci} (\frac{C}{C-L}) \tag{2.4}$$

The cycle length is calculating as following:

$$C = \frac{LX_c}{X_c - \sum_{i=1} n(\frac{v}{s})_{ci}} \tag{2.5}$$

Timing plans of fixed-time control methods are defined by calculating the mean amount of traffic. In everyday life, traffic flow is possible to vary very rapidly according to day of the week, time and the accidents that sometimes occur. It is obvious that given methods cannot consider traffic flow effectively. Therefore, in real life the pre-timed controllers should be used in network locations where traffic flows at a low level.

The settings of pre-timed control necessary to be periodically calibrated to display intermediate or long traffic flow model variations.

In neighboring intersections on traffic networks, entering vehicles to a glassroot intersection are usually impacted by the policies of management of the upstream intersections. In the transport network vehicles also move in groups. Therefore, it is preferable to relate the pre-timed traffic signals of neighbouring intersections such that a group of vehicles may pass several intersections without stopping. For this purpose, offset is also used as a parameter that will affect the management of neighboring intersections as well as other parameters.

2.1.2 Actuated control

In comparison with fixed-timed signal control, actuated traffic signal control systems are able to respond to traffic flow fluctuations. Actuated traffic control requires

some expenses for actuated traffic controllers and vehicle detectors which must be located next to the intersection. The actuated timing plan takes into account the demand fluctuations sending a signal about the presence or absence of vehicles approaching or leaving the intersection. When a signal is received, the controller makes a decision whether to extend the green phase or terminate it [9].

Two types of actuated control can be distinguished: Semi-Actuated form and Fully Actuated form.

- In semi-actuated control vehicle detectors are placed only on insignificant streets. The movements on main streets are called non-callable phases. Therefore, they are activated for the entire split time every cycle, without paying attention to the traffic demand.
  The design of controllers is completed in such a way that a permitted period is defined. During that period the controller allows the callable phases to be served upon a detection request. The callable phases are usually calculated to maintain a minimum green time, after which the green time can only be increased for 10 requests and up to the maximum limit. In some cases, it is possible that minor streets do not require all their allocated green time within a cycle. As a result green time remains unused. Such unused green periods are automatically added to the non-callable phases (main streets).Thus, semi-actuated traffic control is best for traffic roads where local minor streets intersect with arterials.

- In fully-actuated control detectors are basically placed on the traffic approaches. If there is no detected vehicle demand phases can be skipped. Similar to semi-actuated control, callable phases run their splits with a green interval which varies between minimum and maximum values depending on the traffic demand. After reaching the minimum time for a green light phase switching logic is run, and eventually the phase is completed in accordance with a particular criterion. Therefore, fully-actuated control is suitable for less predictable and high volume intersections on all approaches.

2.1.3 Adaptive control

Adaptive traffic signal control systems are basically complicated. Some well known adaptive traffic signal control systems are considered in the given section.

inSync - InSync was developed in 2008 by Rhythm Engineering. In addition to software, the company also offers its own equipment. This makes InSync available in 3 versions:

- InSync - use their own video cameras (up to 4 IP cameras per every intersection)

- InSync: Tesla - existing recording devices are used - magnetometers, radars, video cameras
- InSync: Fusion - combines devices from Rhythm Engineering and existing city devices

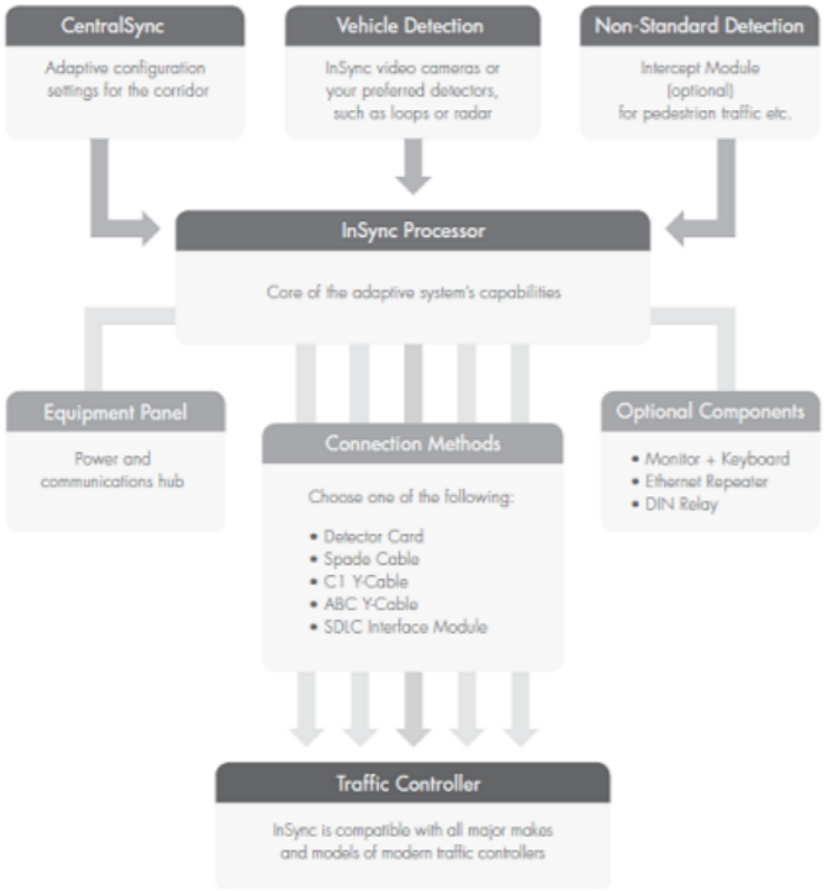

Figure 2.3: Architecture of inSync [10]

Figure 2.3 shows the architecture of the InSync system. As it is shown on the diagram the central module of the system is the processor. This sets it apart from other systems. InSync uses processors at each intersection. Therefore, it does not depend on the main server. It can be monitored from a laptop and mobile devices. The three main elements of a standard InSync system are: an IP camera, an InSync processor and a method for transferring data from detectors.

The system controls three parameters at the local level: clock, phase and cycle. Globally, the system manages signals along the corridor to progress transport through the main streets at the required speed, and also minimize stops and congestion along the corridor. In case of camera failure, InSync uses 4 weeks data.

InSync: Tesla works with all kinds of detectors: stop line detectors, microwave detectors, magnetometers, induction loops, dashboard cameras and radars.

Split, Cycle and Offset Optimization Technique (SCOOT) [11] is a centralized adaptive real-time traffic management system. The system continuously responds to road conditions by making small changes to the signal plans. SCOOT was advanced at the Transport Research Laboratory (TRL) in contribution with the UK transport systems production. The SCOOT has been used in more than 200 cities and megacities in 14 countries. countries around the world, including London, Beijing, and Toronto.

According to TRAFFIC usage reports, SCOOT reduces vehicle latency by an average of 12-20 %.

Current versions of SCOOT have the following features:

- traffic management
- determination of the amount of exhaust gas emissions
- prioritization of public transport
- creating corridors
- detection of an accident

The system consists of two components: the core of the system (Kernel Software) and the software interface (ITC software), designed to interact the core of the system with devices on the road. The system core contains a real-time traffic flow simulation tool and three optimizers (one for each traffic light control parameter). The system's programming interface is specific to the device manufacturer. The second part of the system also contains a graphical user interface for operator interaction with the system. All data obtained as a result of the system operation is stored in the ASTRID database. The following data is recorded in ASTRID:

- traffic flow per hour as modeled by the system
- number of cars per hour as received from sensors
- the ratio of the total delay time to the number of machines per unit of time

Other parameters are calculated.

Sydney Coordinated Adaptive Traffic System (SCATS) Sydney Coordinated Adaptive Traffic System (SCATS) [12] is one of the earliest adaptive control systems. It was developed in Australia in the 70s. Excrement is used and introduced at 37,000 intersections in 27 countries, including: Australia, Brazil, China, India, Iran, New Zealand, Pakistan, Saudi Arabia, Singapore, USA, etc. It has similarity with SCOOT. The main difference between SCATS and SCOOT is that the first one has no traffic model or traffic signal control plan optimizer. SCATS selects optimal phase duration and the offset among several predefined plans [13] depending on current traffic flow conditions. SCATS controls three main parameters for controlling signals:

1. Cycle: the total duration of all signals

2. Regulation Section: percentage of the regulation cycle distributed for every phase

3. Offset: difference of the green signal time beginning at traffic lights

The management of traffic flows is released on 2 levels tactical and strategic. Strategic control is released by regional computers. Which define the optimal cycles, offsets and sections. Tactical control is released by local controllers. It provides the ability to switch off the green light in case of weak traffic demand or to avoid switching it on at all, if there is no vehicle on the road. However the changes of tactical control signals are made by regional computers. Local SCATS controllers can work in different modes: Hurry Call, Masterlink, Flexilink, Isolated, etc. Any of these modes can be set manually or automatically by regional computers and local controllers. For instance, in Flexilink mode control phases are defined by local controllers according to signal plan. SCATS was developed in modular configuration in order to be able to satisfy the needs of small, medium and large cities. In the simplest form the system is able to easily control up to 250 intersections. The enlargement of the system is implemented by setting extra regional computers. All of the systems have central computers for common data control and making backup of the system. Induction loops are located on the stop lines. Also radars and video recorders can be used. The data of geolocation of public transport is used. In Australian large cities data from thousands of buses is gathered to analyze and give priority for public transport.

Dynamic Programmed Intersection Control (DYPIC) Robertson and Bretherton [14] have created the method for optimal management called DYPIC. The DYPIC founded on dynamic programming techniques for isolated intersections. Authors for illustrating their method used a basic intersection. It has two conflicting directions. Therefore the decisions of signal management consists of two actions: prolong or stop the green signal. The given research proposed that the system has certain information about receiving vehicles in the next few minutes. In fact it is impossible in real con- ditions. Consequently, the given method is commonly used for theoretical research and for evaluation of other traffic signal management techniques.

In the method the whole action space is split into N parts. Each of them longs for 5 seconds. When each interval is finished the control system selects to prolong or stop the green signal for the given direction. There were restrictions in the duration of the green phase, no maximum or minimum period. The researchers defined this type of control as a problem of dynamic programming. In particular, decision making points meet the concept of dynamic programming. The condition on every stage was described by the signal state and queues at the approaches. Since it was considered that certain information about approaching traffic flow exists for the action space, the queue length in every direction could be calculated. The goal of optimization was to define the appropriate control strategy containing actions sequence. On

the base of queue lengths, initial signal plans and future vehicle coming data that minimizes the total delay.

2.2 Traffic Signal Control existing methods

This segment provides foundation data with respect to a number of the methods which have been realized within the adaptive traffic signal control optimization.

2.2.1 Multi-agent system

A multi-agent system comprises of a subjective number of intelligent agents which are interacting in one environment. Moreover, agent of such a system must have the following characteristics [15]:

- Autonomy: The agents must be able to act on their claim agreement, making educated choices based on the information accessible to them.
- Decentralized: There is no one centralized agent that controls and makes decisions for all other agents.
- Local view:Do not possess a global scan of the whole surroundings. Ac- tion selection must be made based on locally accessible information. In spite of the fact that locally accessible information may have been sent to the agent from other distant agents.

The multi-agent frameworks are especially well suited to tackling issues which cannot easily be illuminated by a single agent.[16] proposed the system to solve traffic control problems in the transport network.

Fuzzy Logic (FL), Swarm Intelligence (SI), Reinforcement Learning (RL), Deep Learning (DL) are the widely used CI techniques that consider TSC problems as MAS [17, 18].

RL is a machine learning method that tries to figure out optimal actions by interacting with an environment [19]. RL is based on states, actions, and rewards. An RL agent chooses an action for a current state according to its decision policy; the environment then returns a reward for the action; and based on the sequence of such rewards the agent finally updates its decision policy.

RL algorithms may respond to dynamic changes of both current and longer term traffic which is incorporated in the state value or state-action value functions. Yau et al. [20] surveys many publications for TSC RL-based. It thoroughly analyzes the parameter sets engaged by different methods and groups them as follows: transport network parameters, RL-context parameters, performance measure metrics, and the complexities of the used algorithms. It also highlights the following few approaches:

- MARL [21, 22]
- Max-plus RL [23]
- Model based [24]
- Actor-Critic RL [25]
- Multistep backups RL [26]
- RL with function approximation [27]

2.2.2 Artificial Neural Network

Artificial neural networks are widely used in traffic signal management in the transport network. Consider some of the work in this area.

The study [28] presents a model based on ANN for the traffic signal control task on isolated intersections, where the mutual influence of the modes of operation of traffic lights on adjacent sections of the network is not considered. In this work, the total delay of vehicles at the intersection is minimized. For simulation purposes, the delay flux, determined by the phase of the traffic signals, is considered as a quadratic function of the green light duration in this phase.

Araghi, et al. [29] provide a comparison of classical ANN and fuzzy controllers as traffic signal control systems. The authors suggest using a neural network with one hidden layer, input vector containing the number of halting cars in each traffic signal controller, and at the output they receive the duration of each phase. ANN is trained using a genetic algorithm.

Castro, et al. [30] consider biologically inspired neural networks (BiNN) for intersection control. In such methods, emphasis is placed on the study of dynamics, in contrast to classical ANNs, which mainly deal with learning procedures. BiNN is investigated on a complex intersection model. The BiNN structure is as follows: input neurons describe a queue of vehicles in each lane. Output neurons correspond to phases on stripes. All output neurons are associated with inhibitory neurons, which suppress the activity of other output neurons. The duration of the phases is limited by the equation describing the concept of ”immanent plasticity” of the neuron.

In the article of Spall, et al. [31] have presented a system based on ANN which takes the information about traffic flow at the given time and gives the duration of signals. The system is model-free, since it does not need any traffic flow model. A feed-forward NN used as a controller. Architecture of the network is as follows:the input layer consists of 42 neurons, and there are two hidden layers with 12 and 10 neurons respectively. Inputs included: the queue at each cycle termination for 21 traffic queues of the simulation; 11 per-cycle vehicle arrivals in the system; simulation start time; and the nine outputs from the previous control solution. The output layer contains 9 nodes for each signal split.

The article [32] considers a deep convolutional artificial neural network for adaptive traffic management. For ANN training, reinforcement learning is used. In the terminology of the underpinning learning paradigm ANN is called an agent. The input signal to the ANN is formed from the state space proposed by the authors - discrete state coding traffic (DTSE). The following neural network architecture is proposed. Two neural networks are used with an identical structure, but a different set of input signals. In the first, a binary vector describing the presence / absence of a car on the road sections is fed to the input. In the second network a vector of real numbers describing vehicle speed on the road sections is given to the input. The outputs of neural networks deployed into a vector are glued to each other and with the current state of the phases and fed to the input fully connected ANN. The output from the ANN is an indicator vector showing the action that the agent must perform, namely, contains the number phase to be included.

In papers [28]-[30], phase lengths are obtained at the outputs of neural networks, and in [32] the solution is to decide which of the phases to switch.

Choi, et al. [33] present a new hybrid, synergistic approach of a multiagent system for real-time traffic signal control of a large-scale traffic network. The transport network management task was divided into several subproblems and each considered by an agent with fuzzy neural decision making capability. The proposed system reduced total waiting time of vehicles by 50% and average delay by 40%.

2.2.3 Fuzzy Logic Systems

Researchers have started to apply the Fuzzy Logic Systems in traffic control problems since 1970. [34], [35] are some papers that presented the FLS controller for an isolated intersection.

Favilla, et al. [36] applied FLS to traffic signal management on an isolated intersection with two-way roads. The input that served to the fuzzy rules consists of the number of vehicles that had already transmitted the intersection and the number of the halting vehicles in the red signal. The result of the system is the duration of the prolong of the green phase.

Authors [37] presenting another FLS TSC. The considered intersection has two-way highways. The given controller has two actions: continue the green signal or terminate it. In the articles [38], [39] a two-stage fuzzy online controller was developed.

[40] proposed a controller with genetic algorithms. Queue lengths, cars and motorcycles considered as an input. Output is the duration of green signal extension.

2.3 Simulation types

The main function of simulators is to implement experiments of traffic flow modeling and optimization. It can be observed how some changes in traffic flow such as

difference in signal location, widening the streets, prohibition or permission of turns, one-way traffic organization would affect the amount of air pollution. There are a lot of different traffic flow simulators for estimation of transport policy, strategies and projects.

Figure 2.4 schematically shows a simulation model of road-traffic flows.

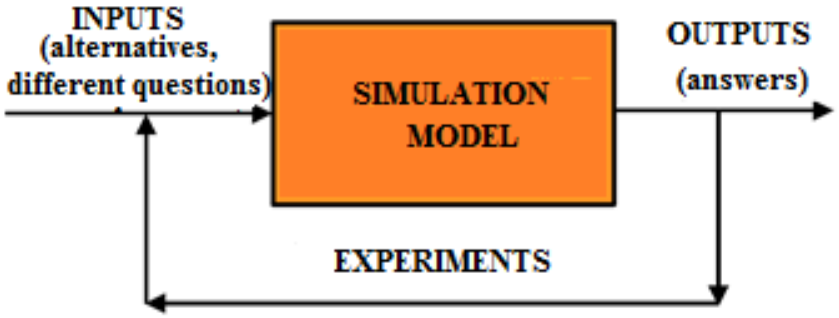

Figure 2.4: Simulation model of traffic flow

However, any simulation model contains several models such as demand model, model of traffic flow and transport network model. The construction features and principles of these models are shown below. Traffic simulation is modeling of traffic systems in order to plan the transport systems. These simulations give opportunity to research traffic flows in a convenient and safe environment.

Traffic simulation is the modelling of vehicle traffic systems for the purpose of planning transportation systems. These simulations offer a safe and convenient environment to investigate possible modifications to transportation systems.
Traffic simulation as a whole can be largely divided into tree approaches:

- microscopic
- macroscopic
- mesoscopic (a hybrid of the previous two)

At the **macroscopic level**, traffic movement is considered as a consecutive flow, which is defined by its speed and density. It means that the participants of traffic movement are considered as an entire flow, not separately. All values are averaged.

At the **microscopic level**, the speed and behavior of each traffic movement participant is modeled separately. The attention is paid to simulation of drivers and pedestrians behavior.

Some important modules in microscopic simulation:

- Transport network - Creation of a transport network is the process of road system design for use in simulation. Usually it consists of defining some traffic features such as speed restrictions, number of lanes, starting and destination locations, connections between lanes at intersections. Some simulators need

these specifications to be entered manually, while others are able to directly import traffic networks from other sources or geographic information systems.

- Route generation -Route generation includes route definition for each vehicle that will participate in the simulation. The settings of route generation differ according to the simulator type, as each of them accepts data at different levels and formats. Usually the route of each vehicle is set clearly. A set of road sections to be crossed, where a route is dynamically generated when the vehicle enters the simulation. Using any of these methods it would be unreal to create a set of routes for modeling manually. For this reason the majority of simulators provide tools that automatically create vehicle routes based on specified parameters.
  Using the origin / destination (OD) approach usually includes the number of sub-networks and amount of vehicles that start their way in the same area and finish in the other. These numbers can be used for journey estimation in a certain time period where routes between two points are calculated using the route algorithm.
  The coefficients of turning (Turning ratio) When the routes are determined with the use of probability coefficient of turn for each option there must be a mentioned turn point of the network. Those probabilities must be equal to 1 for each turn point, since each vehicle must move in one of possible directions. Vehicle implementation metrics that determine how many vehicles are included in the simulation and where they come from along with ”drain edges” must also be specified for the network. When the vehicle route reaches the ”edge of failure” the route ends and the vehicle is removed from the simulation. Then the route creation program defines possible vehicle routes from origin till it reaches the edge.

- Output - The output possibilities of the simulation play an important role in determining the benefit of the environment. This is due to the fact that it would be impossible to draw any conclusions without the ability to measure the relevant indicators. The basic data which is included in almost every simulator is generated on the base of microscopic simulations, valuable information about vehicles on the network, such as speed and travel time. Some simulators give general information about the road part inside the system often including such parameters as average speed and amount of the vehicles and their density. Some advanced simulators are able to make calculations, which include the level of noise and air pollution. It is very important for those researchers who are interested in developing systems minimizing pollution. Some other simulators have features of providing output information in the form of graphs so that many people could easily obtain and compare the results of simulation.

- Car following models - Car tracking model is a form of driver behavior widely used in microscopic simulators of traffic movement. The attempt to duplicate

the behavior of neighbor vehicles drivers. The use of car following models allows observing the behavior of drivers in the traffic flow interaction. The mesoscopic level is conditional, since mesoscopic models can be considered less detailed analogs of microscopic ones. For example, if at the microscopic level we can consider several types of transport, distinguish different brands of vehicles, then at the mesoscopic level they will be represented by one class of vehicles. The choice of simulation level strictly depends on the objectives of the project.

The mesoscopic level is conditional, since mesoscopic models can be considered less detailed analogs of microscopic ones. For example, if at the microscopic level we can consider several types of transport, distinguish different brands of vehicles, then at the mesoscopic level they will be represented by one class of vehicles. The choice of simulation level strictly depends on the objectives of the project.

In the given work Simulator DLR SUMO [41] was chosen as an environment for experiments.

SUMO is a simulator for microscopic modeling of traffic movement. Each vehicle is defined by its own ID, departure time and the movement route in the network. Simulation is discrete in time with 1 second default increment. It is spatially continuous. The position of each vehicle is determined by the lane where the vehicle is located and the distance from the beginning of that lane. The speed of each vehicle is calculated by using the model of following cars. SUMO uses the car models developed by Stefan Kraus [8]. The change of the lane is released by using the model developed while implementing SUMO.

Benefits of the SUMO:

- Full tool for traffic flow modeling
- demand generation
- multimodal modeling
- emission modeling
- simulation modeling
- interactive interaction
- different input sources: destination matrices, traffic counts, etc.
- high-performance modeling
- “remote control” interface (TraCI) for adapting online modeling
- V2X - vehicle and vehicle to infrastructure

- evaluation of the developed traffic light programs
- “Traffic Assignment” on a microscopic basis
- evaluation of traffic surveillance systems
- modeling of road traffic in large cities
- Car-After and Lane-Change API
- Personal simulation of intermodal traffic
- Emission and noise modeling.

Chapter 3

Overview of the methodologies

In this section basics of Artificial Neural Network (ANN) and Reinforcement Learning (RL),formulation of the given problem within its framework are presented.

3.1 Reinforcement Learning (RL)

Reinforcement learning (RL) [19] studies how to teach an agent to interact efficiently with the *environment* . An *agent* is the learner which is able to sense the state of its environment and take actions that affect the state. On each consecutive interaction the environment responds by sending a numerical *reward signal* to the agent which indicates how good the action was. The agent must have a *goal* relating to the state of the environment. The purpose of reinforcement learning is to make the agent learn the action selection rule which ensures the maximum possible total reward the agent receives over the long run so as to achieve the goal.

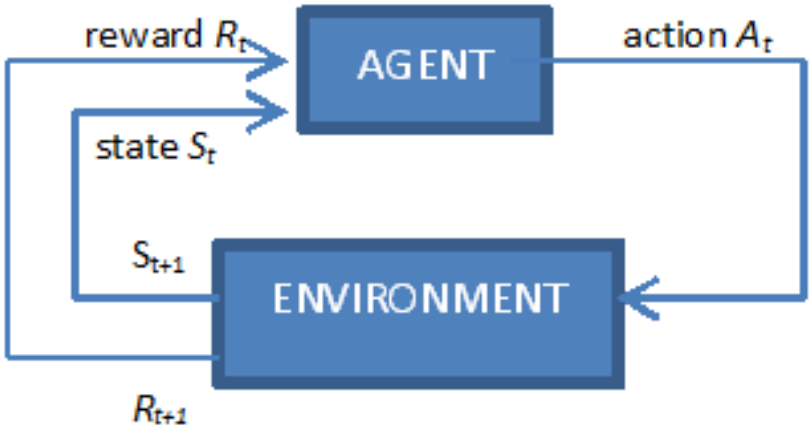

Figure 3.1: Reinforcement Learning

The agent-environment interaction is continual and occurs over time t as follows

(see Figure 3.1):

$$S_t \underset{A_t}{\longrightarrow} (R_{t+1}, S_{t+1}), \tag{3.1}$$

where $S_t \in \mathscr{S}$ and $S_{t+1} \in \mathscr{S}$ are the states of the environment at the previous and current times respectively; $A_t \in \mathscr{A}$ is the action taken, and $R_{t+1} \in \mathscr{R}$ is the reward value obtained. If spaces $\mathscr{S}$, $\mathscr{A}$ and $\mathscr{R}$ are all finite, there is a discrete probability distribution for the random variables S_t and R_t dependent only on the preceding state and action:

$$p(s', r|s, a) \doteq Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{3.2}$$

These conditional probabilities constitute the *model* of an environment and completely characterize its dynamics. The algorithms that rely on the knowledge of the model are called *model-based*; those which do not assume it are called *model-free*.

The idea of a total cumulative reward is formalized by means of the *discounted return* at time t:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots, \tag{3.3}$$

where $0 \leq \gamma \leq 1$ is the *discount factor*. The agent's action selection rule is called the *policy* and is also a probability distribution:

$$\pi(a|s) \doteq \Pr\{A_t = a \mid S_t = s\} \tag{3.4}$$

Finding the optimal policy that maximizes the expected return is the objective for learning.

While rewards show what is good in an immediate sense, the *value functions* determine the long-term desirability of the environmental states taking into consideration the states that are likely to follow under the given policy, and the rewards available in those states [19]. The *state-value function* $v_\pi(s)$ for policy π evaluated at state $s \in \mathscr{S}$ is the return value which the agent can expect to obtain starting from that state and following policy π thereafter:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] && (3.5) \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] && (3.6) \end{aligned}$$

Similarly, the *action-value function* $q_\pi(s, a)$ for policy π evaluated at state $s \in \mathscr{S}$

and action $a \in \mathscr{A}$ is defined as:

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{3.7}$$

Notice that the formula (3.7) gives a way to express $q_\pi(s,a)$ in terms of $v_\pi(s)$. It can also be done the other way around:

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a) \tag{3.8}$$

Intuitively, the value $v_\pi(s)$ measures how good it is for the agent to be in state s, whereas the value $q_\pi(s,a)$ indicates how good it is to perform action a in state s.

If we define a partial ordering on the set of all policies as

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s) \; \forall s \in \mathscr{S}, \tag{3.9}$$

then there is at least one policy that is better than or equal to all other policies — the *optimal policy*. Let us denote by π_* all the optimal policies. They share the same *optimal state-value function* v_*

$$v_*(s) \doteq \max_\pi v_\pi(s), \; \forall s \in \mathscr{S} \tag{3.10}$$

as well as the *optimal action-value function* q_*:

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a), \; \forall s \in \mathscr{S}, \forall a \in \mathscr{A} \tag{3.11}$$

The optimal value functions, v_* and q_*, are related to one another by the *Bellman optimality equation*:

$$v_*(s) = \max_{a \in \mathscr{A}} q_*(s,a), \tag{3.12}$$

which comes from the fact that if we know q_* to be optimal, then the optimal value of a state is the expected return for the best action from that state.

The deterministic *greedy* action selection policy with respect to the optimal action-value function is always one of the optimal policies:

$$\pi_*(a|s) = \begin{cases} 1 & a = \text{argmax}_a \, q_*(s,a) \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

The reinforcement learning methods aim at finding plausible estimates of the

optimal value functions. The agent can adopt the greedy policy with respect to these estimates and it will be nearly optimal. However, if the agent follows a completely greedy policy all the time with respect to the estimates at each time step, it will exploit the current knowledge about the states and actions it has observed, but may fail to explore new, and possibly better, action choices. This is called the *exploration-exploitation dilemma*. *On-policy* methods estimate the value functions of a policy while simultaneously using it for controlling the agent's behavior [19, Sec 5.7]. In *off-policy* methods there are two different policies. The policy used to generate the agent's behavior is called the *behavior policy b*, and it may in fact be unrelated to the policy that is evaluated and improved, called the *target policy* π. An advantage of this separation is that the target policy may be deterministic (e.g. greedy), while the behavior policy controls the agent so that it continues to explore new action selections.

3.1.1 Q-learning and SARSA

There are numerous basic approaches to solve the reinforcement learning task. If the environment's dynamics is known, the methods of *dynamic programming (DP)* [19, Chap 4] are employed which directly solve for v_* by the method of successive approximations for the system of linear equations defined by (3.6). If the task is episodic [19, Sec 3.3], the *Monte Carlo* methods are frequently used which involve estimating v_* by averaging sample returns obtained at the end of each successive episode [19, Chap 5]. However, neither of these approaches is suitable to our task per se, since the model of the environment is not known beforehand, and the task is not episodic.

The *temporal-difference learning (TD)* is a combination of both the dynamic programming and Monte Carlo methods. Like DP, TD methods update estimates of the value functions obtained in turn from other learned estimates. However, in contrast to DP, TD assumes no predefined model of the environment whatsoever. Like Monte Carlo, TD methods can learn directly from raw experience. However, in contrast to Monte Carlo, the TD approach updates the estimates in the *online* fashion, meaning that it does not wait until the end of an episode, but rather does this on each time step (i.e. *bootstraps*).

The TD method estimates the true expected value (3.5) by averaging the value of $R_{t+1} + \gamma v_\pi(S_{t+1})$ at each time step. However, the value $v_\pi(S_{t+1})$ is not known, so its current estimate $V(S_{t+1})$ is used instead. Upon transition to a new state S_{t+1} and receiving R_{t+1}, the TD methods make the following update:

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right] \tag{3.14}$$

Another perspective on the above formula is:

$$V(S_t) \leftarrow V(S_t) + \alpha \delta_t \tag{3.15}$$

$$\delta_t \doteq Target - Estimate \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{3.16}$$

Thus, the update shifts the current estimate in the direction of the error between the target at time t and the current estimate of the value function. This error δ_t is called the *TD error*. Notice that the TD error at each time is the error in the estimate made at that time. Because the TD error depends on the next state and the next reward, it is not actually available until one time step later. The update formula (3.14) follows the following general rule that occurs frequently throughout reinforcement learning:

$$NewEstimate \leftarrow OldEstimate + StepSize\,[Target - OldEstimate] \tag{3.17}$$

This general update rule can be treated as averaging of the old estimate and the target with the constant parameter $StepSize = \alpha$ that defines the extent to which the new target affects the average.

The majority of TD methods estimate q_* instead of v_*, since we cannot construct a policy derived from the estimate of v_* without the model of the environment. The estimation of q_* is done using essentially the same TD update (3.14) described above. One of the on-policy TD methods is *SARSA* [19, Sec 6.4], in which the target and behavior policies are the same and are both derived from the current action-value estimate $Q(S_t, a)$. For instance, the ε-*greedy* policy can be adopted which is defined as follows:

$$\pi(a|s) = \begin{cases} \varepsilon & a \text{ is a random action} \\ 1-\varepsilon & a = \operatorname{argmax}_a q_*(s,a) \end{cases} \tag{3.18}$$

The update rule for SARSA is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right], \tag{3.19}$$

where A_{t+1} is selected according to the current policy.

One of the off-policy TD algorithms is *Q-learning* [19, Sec 6.5]. The target policy for Q-learning is the greedy policy, whereas the behavior policy may be any policy derived from Q which ensures exploration, i.e. all the state-action pairs are continually visited and updated. The update rule for Q-learning appears as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right] \tag{3.20}$$

If the state and action spaces are finite, the function Q can be represented as a table.

3.1.2 Q-network

In the Q-network algorithm the estimate of the true action-value function is represented not as a table, but rather as a parametrized continuous function — the neural

network with a shallow architecture. That is,

$$q_*(s,a) \approx \hat{q}(s,a,w), \tag{3.21}$$

where $\hat{q}(s,a,w)$ represents the value of the neuron corresponding to the action a in the output of the neural network evaluated at state s. The neural network is parametrized by the set of weight matrices $w \doteq \{w^{(1)}, w^{(2)}, \ldots, w^{(L-1)}\}$, where L is the total number of layers in the neural network. The matrix $w^{(k)}$ for $k = \overline{1, L-1}$ defines the transition from the k^{th} to the $(k+1)^{th}$ layer.

In order to be able to train the neural network, we need to define the cost function. The algorithm should adjust the weight matrices w so as to reduce the cost function as much as possible. The natural way to define a cost function which is consistent with our goal of approximating the optimal action-value function q_* is:

$$J(w) \doteq \frac{1}{2} \sum_{s \in \mathscr{S}, a \in \mathscr{A}} \left(q_*(s,a) - \hat{q}(s,a,w)\right)^2 \tag{3.22}$$

However, the true values of $q_*(s,a)$ in the sum above are not known, so this cost function is inappropriate. To remedy this state of affairs, the temporal-difference learning comes in.

The SQN algorithm proceeds in the online manner: at each time step t it makes a TD estimate $U_t(S_t, A_t)$ to the target value $q_*(S_t, A_t)$ as follows:

$$U_t(S_t, A_t) \doteq R_{t+1} + \gamma \max_{a' \in \mathscr{A}} \hat{q}(S_{t+1}, a', w_t) \tag{3.23}$$

Afterwards, we define the cost function as

$$J_t(w_t) \doteq \frac{1}{2} \left(U_t(S_t, A_t) - \hat{q}(S_t, A_t, w_t)\right)^2, \tag{3.24}$$

which involves the difference between the new and current estimates of the action-value function, with the latter produced by the neural network with the weights at time t.

Gradient descent is used to update the weights of the neural network by a small amount along the anti gradient of the cost function (3.24). Note that the cost function (3.24) does not involve summation over all the states and actions, but rather it uses only state S_t and action A_t from the current time step. These were provided stochastically by the environment. That is the reason why this is the *stochastic gradient descent*: it uses the cost function involving a training example which is chosen stochastically, and seeks to reduce error just on it.

Let us compute the gradient of the cost function (3.24):

$$\nabla J_t(w_t) = \left(U_t(S_t, A_t) - \hat{q}(S_t, A_t, w_t)\right) \nabla_{w_t} \hat{q}(S_t, A_t, w_t) \tag{3.25}$$

Afterwards, the algorithm updates the weights w_t of the neural network as follows:

$$w_{t+1} \doteq w_t - \alpha \nabla J_t(w_t), \tag{3.26}$$

where α is the learning rate.

Note also that (3.25) is not the true gradient of (3.24), since in computing it we did not take into account the fact that the value $U_t(S_t, A_t)$ is dependent on the network weights w_t. After $U_t(S_t, A_t)$ has been evaluated, it is treated to be constant. This is why (3.26) is called *semi-gradient descent*.

In general, the gradient $\nabla_{w_t} \hat{q}(S_t, A_t, w_t)$ on the right-hand side of (3.25) is not easy to compute for the neural networks having one or more hidden layers, therefore the whole $\nabla J_t(w_t)$ is computed by means of the backpropagation algorithm. The error δ_t is backpropagated at each time step t which is defined as follows:

$$\delta_t \doteq U_t(S_t, A_t) - \hat{q}(S_t, A_t, w_t) \tag{3.27}$$

3.1.3 Average Reward Network (ARN)

Another approach of training the neural network by learning a numerical *preference* for each action a to be selected in state s, which we denote $H(a, s, w_t)$, where w_t is again a set of weight matrices of the neural network. You can think of $H(\cdot, s, w_t)$ as a vector of size $k \doteq |\mathscr{A}|$ produced at the output layer of the neural network when the activation function has not yet been applied to it. The larger the preference of an action, the more favorable it is to be taken; however, the preference has no interpretation in terms of reward. Only the relative preference of one action over another is important; if we add 1000 to all the preferences, there is no effect on the action probabilities, which are determined according to the *softmax* (or Boltzmann) *distribution* as follows:

$$\pi(a, s, w_t) \doteq \frac{e^{H(a,s,w_t)}}{\sum_{b=1}^{k} e^{H(b,s,w_t)}}, \tag{3.28}$$

The parametrized probability distribution $\pi(\cdot, s, w_t)$ constitutes the current stochastic action selection policy.

Next, we need to maintain the average reward $R_t^*(s)$ at time t for each state $s \in \mathscr{S}$ updated by the following recurrent rule:

$$R_t^*(s) \doteq R_{t-1}^*(s) + \alpha \left(R_t(s) - R_{t-1}^*(s) \right) \tag{3.29}$$

$$R_0^*(s) \doteq 0 \tag{3.30}$$

Note that the above update follows the general rule (17), and here we assume that the state space $\mathscr{S}$ is finite.

Upon selecting action A_t in state S_t and receiving reward R_t, the average reward $R_t^*(S_t)$ is updated, and the error $\delta_t \in \mathbb{R}^k$ is computed according to (3.31). The term $R_t^*(S_t)$ serves as a baseline with which the current immediate reward is compared. If the reward is higher than the baseline, then the probability of taking A_t in the future is increased; and if the reward is below baseline, then the probability is decreased. The non-selected actions move in the opposite direction.

$$\begin{aligned} \delta_t(A_t) &\doteq -\left(R_t - R_t^*(S_t)\right)\left(1 - \pi(A_t, S_t, w_t)\right), \\ \delta_t(a) &\doteq \left(R_t - R_t^*(S_t)\right)\pi(a, S_t, w_t) \text{ for all } a \neq A_t \end{aligned} \tag{3.31}$$

Afterwards, the error δ_t is backpropagated through the network using the backpropagation algorithm, and the weight matrices w_t are updated.

We name the algorithm described in this section as *Average Reward Network (ARN)*. ARN is an online, model-free and on-policy method. The error defined in (3.31) is analogous to the update rule (2.10) in [19, Sec 2.8]. The distinguishing feature of ARN is that, unlike SQN, it learns solely by immediate rewards.

3.1.4 A model based solution

The proposed method requires a model of the environment. The model learns the transition function $T(s,a,s'))$ from the pair of current state s and action a to the next state s'. It is a conditional probability, which is derived from the history statistics: $P(s'|(s,a)) = \dfrac{\#(s,a,s')}{\#(s,a)}$

For the Q-value update based on the Bellman equation:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} (T(s,a,s') \max_{a'} Q(s',a')) \tag{3.32}$$

3.2 Artificial Neural Network (ANN)

ANN is a network that consists of many simple processors that interconnect between each other. These units are called *neurons*. Neurons transmit signals to each other by the connections that have different strength values. This network parameter is named as *a weight*. Neurons operate with their own local data and perform calculation in parallel [42].

$$z_j = \sigma_j + \sum_{i=1}^{n} a_{ij} x_i \tag{3.33}$$

where z_j is the output, a_{ij} are the weights, x_i is the input, σ is the bias.

The main advantage of the ANN is the ability to learn without analytical knowledge about the environment. Learning process is the setting of the weights such that

the model is able to give the right solutions. The weights are adjusted based on the data provided by the neural network. Finally, the network can predict the behaviour of the system. The data that is provided for the learning process is called *training data*. The data that is used for evaluation of the model is called *test data*. Overall data is merged as *a dataset*.

Three types of learning in ANN are distinguished: *supervised, unsupervised and semi supervised*. Supervised learning requires input data and desired output or target (label). Since the correct answers are known, the ANN's output is compared with the targets and the error can be calculated. Based on this, the weights are updated, which lead to minimizing errors. This is how the model is trained in supervised learning [43].

In the unsupervised learning process data is not annotated, which means that there are no desired outputs. In this method, the hidden features and characteristics of the given data are studied and divided into groups based on certain characteristics. The network self-organize the input data.

In a semi supervised learning approach some parts of data has a target, some none. Practical applications of NNs most often deal with supervised learning as in this thesis.

The success in usage of ANN depends on provided data. A large dataset is required for the training process. ANN is like a function approximator that constructs dependencies between input and output. The rightness of the result directly depends on data quality. There are some difficulties in ANN: too many parameters to adjust, difficult to train, since slow convergence, local minima and design of the network itself. Architecture definition (number of layers, hidden units) requires expert knowledge in the given scope.

3.2.1 ANN Architecture

The group of neurons that process in one unit time join into one layer [44]. There exist three types of layer: *input, output and hidden.* The first two are compulsory, while the third is optional. Therefore, the architecture of ANN is defined as the number of layers and number of neurons in each layer (See Figure 3.2).

Input is the first layer that accepts the provided data. The size of this layer depends on the feature vector of the input sample. Some ANN configurations add one node for bias.

Output is the last layer that presents the final result. The number of neurons on this layer depends on the problem that is considered. In classification problems, the number of classes in the output set determines the number of neurons. In a regression problem, there is a single node that is predicted.

Hidden is a middle layer between input and output. It adds flexibility to the network. If the input data is linearly separable then there is no need for a hidden layer. If need more complex connections for data approximation hidden layers are added. We need to add several hidden layers to make network more flexible. There

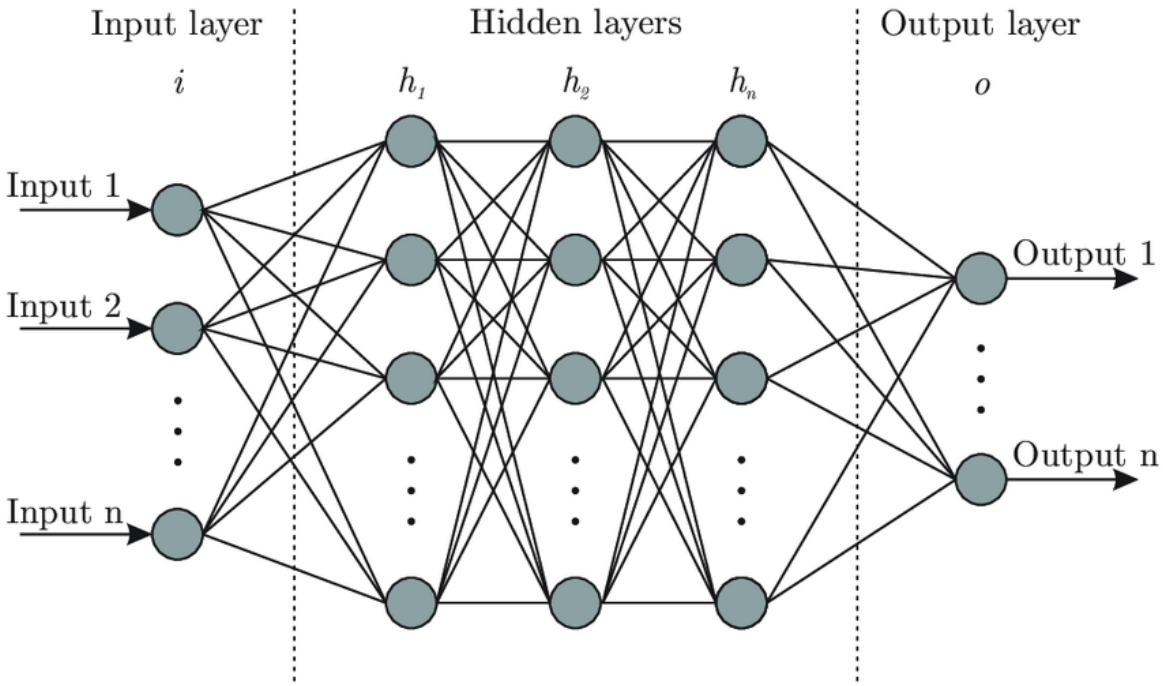

Figure 3.2: ANN architecture [45].

is no strong rule to find how many hidden units are needed to approximate any given function. Usually, the number of neurons is selected by experimental way. If the network is overfitting during the training and testing process, the number of neurons should decrease.

3.2.2 Activation functions

Activation functions conduct the data through the specific function that converts it to needed format.

$$z_j = f(\sigma_j + \sum_{i=1}^{n} a_{ij} x_i) \tag{3.34}$$

where z_j is the output, f is the activation function. ANN uses 2 different activation functions in one time: hidden layer activation function and output layer activation function. These two have different goals. Activation function of hidden neurons adds nonlinearity to the network. Activation function of output neurons depends on target value.

Examples of activation functions [46]:

Linear:

$$f(x) = x \tag{3.35}$$

Hyperbolic:

$$f(x) = tanh(x) \tag{3.36}$$

Sigmoid:

$$f(x) = (1 + \exp^{-x})^{-1} \tag{3.37}$$

Softmax:

$$f(x_i) = \frac{\exp^{x_i}}{\sum_j x_j} \tag{3.38}$$

Activation functions for different tasks:

- Sigmoid : if problem requires binary output;
- Softmax: classification targets;
- Tanh: continuous value output in range -1 and 1;
- Linear: unlimited continuous value output.

3.2.3 Backpropogation Algorithm

ANN is often use *backpropogation algorithm* [47] to training process. The weights are adjusted so that difference between output o and target t value is minimized. Gradient descent is used to minimize the error. The *cost function* C is calculated for each sample by Euclidian distance:

$$C = \frac{1}{2}||o - t||^2 \tag{3.39}$$

where $o = (o_1, o_2, ..., o_n)$, $t = (t_1, t_2, ..., t_n)$.

Weights w are updated by gradient descent algorithm:

$$w(t+1) = w(t) - \eta \frac{\partial E}{\partial w} \tag{3.40}$$

where η is learning rate. It is free parameter, that identifies step size and affect to learning speed.

Weights between hidden and output layers d are updated as following:

$$d_{jk}(t+1) = d_{jk}(t) - \eta \frac{\partial C(t)}{\partial d_{jk}} = \tag{3.41}$$

$$= d_{jk}(t) - \eta \frac{\partial C(t)}{\partial o_{jk}} \frac{\partial o(t)}{\partial d_{jk}} = \tag{3.42}$$

$$= d_{jk}(t) - \eta (o_{jk} - t_k) z_j(t) \tag{3.43}$$

where cost function:

$$C = \frac{1}{2} ||o - t||^2 = \tag{3.44}$$

$$= \frac{1}{2} \sum_{k=1}^{p} (o_k - t_k)^2 = \tag{3.45}$$

$$= \frac{1}{2} \sum_{k=1}^{p} \sum_{j=1}^{h} (d_{jk} z_j - t_k)^2 = \tag{3.46}$$

For update weigths between input and hidden layer a the cost function will be modified:

$$C = \frac{1}{2} ||o - t||^2 = \tag{3.47}$$

$$= \frac{1}{2} \sum_{k=1}^{p} (o_k - t_k)^2 = \tag{3.48}$$

$$= \frac{1}{2} \sum_{k=1}^{p} \sum_{j=1}^{h} (d_{jk} z_j - t_k)^2 = \tag{3.49}$$

$$= \frac{1}{2} \sum_{k=1}^{p} (\sum_{j=1}^{h} (d_{jk} f(\sum_{i=1}^{n} a_{ij} x_i) - t_k)^2 = \tag{3.50}$$

Update rule for weights a:

$$a_{ij}(t+1) = a_{ij}(t) - \eta \frac{\partial C(t)}{\partial a_{ij}} = \tag{3.51}$$

$$= a_{ij}(t) - \eta (\sum_{k=1}^{p} \frac{\partial C(t)}{\partial o_k(t)} \frac{\partial o(t)}{\partial z_j(t)}) \frac{\partial z_j(t)}{\partial a_{ij}} = \tag{3.52}$$

$$= a_{ij}(t) - \eta (\sum_{k=1}^{p} (o_k(t) - t_k) d_{jk}(t)) z_j(t)(1 - z_j(t)) x_i(t). \tag{3.53}$$

In the above formula, sigmoid function is considered as the activation function.

Chapter 4

Traffic Signal Control using Reinforcement Learning (RL)

This chapter presents an online model-free adaptive traffic signal controller for an isolated intersection. The problem is formulated as a Reinforcement Learning (RL) task. We base our solution on the Q-learning algorithm. In contrast with other studies in the field, we use the queue length rather than the average delay as a measure of performance. Also, the number of queuing vehicles in non-conflicting directions is aggregated to represent a state. Then, instead of predefined phase splits or direct switching, the duration of phases is updated by a small amount of time. Finally, we include the queue reduction and equilibrium terms in the equation of an immediate reward. The performance of the proposed method is compared with an optimal symmetric and far-from-optimal asymmetric fixed signal plan. The experimental results show that the proposed method performs almost as good as an optimal one.

4.1 Q-learning-Q-table

In recent years significant efforts are made in order to utilize Computational Intelligence (CI) techniques in the traffic signal control problem. Reinforcement Learning (RL) is one of them.

The main focus in Traffic Signal Control (TSC) systems has moved as following:

- from traffic flow model-based methods to traffic flow model-free methods
- from pure Fuzzy Logic (FL), Neural Network (NN) [29], and Genetic Algorithm (GA) solutions to RL solutions
- from model-based RL [48] to model-free RL [49, 50]
- from single-objective to multiple-objective RL [51, 21]
- from RL to RL with function approximation [52] and global optimization techniques.

Additionally, different agent types [52, 53] as well as agent collaboration techniques [54, 55, 56] had been investigated. The most recent trend is to consider RL in combination with Swarm Intelligence (SI) [57, 58] Therefore, the researchers mostly work on decentralized TSC systems based on collaboratively operating and self-organized multi-type agents, which have no prior knowledge of the environment.

Our aim is to create an easy to deploy system of adaptive traffic signal control that is based on one of the well-known methods and that can be integrated without difficulties into the current local traffic network with the minimal change of the latter. Such a solution is an off-policy, one-step, tabular, model-free Temporal Difference (TD) Q-learning method. Our approach is closely related to the papers by Abdoos et al. [49] and Araghi et al. [50].

As in [49, 50], we formally describe the problem as an infinite horizon Markov Decision Process (MDP) with a non-stationary unknown environment. In other words, we define finite sets of states and actions and use ε-greedy policy to choose actions based on state signals received from the environment. We solve the problem using one of the Reinforcement Learning (RL) algorithms, namely Q-learning [59].

Compared to [49, 50], first, we use the queue length rather than the average delay as a measure of performance. Second, we aggregate the number of queuing vehicles in non-conflicting directions to represent a state. Third, instead of predefined phase splits, we update the duration of phases by a small amount of time. Finally, we include the queue reduction and equilibrium terms in the equation of immediate reward.

4.1.1 Q-learning

According to [19], today the most widely used RL methods are: Q-learing, SARSA, Actor-Critic, and R-learning. SARSA and actor-critic methods are on-policy methods, while Q-learning and R-learning are off-policy methods. Due to their great simplicity, they can interact with the environment on-line with a minimal amount of computation. On-policy and off-policy methods mainly differ in the value function and policy update, which mostly influence the convergence rate of the process. Since all the methods have been proven to converge to an optimal policy, in our context those differences are subtle. Therefore, any of the aforementioned methods may be equally applied to the problem.

We implement an off-policy, one-step, tabular, model-free Temporal Difference (TD) Q-learning method:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t) \right] \tag{4.1}$$

where $Q(s_t,a_t)$ is an action-value of action a_t taken at state s_t, s_{t+1} is the next state, $0 < \alpha \leq 1$ is a constant step-size parameter, r_{t+1} is an immediate reward of action a_t at state s_t, and $0 < \gamma \leq 1$ is a discount factor. See [59, 19] for more details.

4.1.2 The proposed method

The proposed solution for a traffic signal controller needs neither a traffic flow model nor a model of the environment. Additionally, it can be almost painlessly used and deployed on a traffic network of any size.

In this context, a signal controller of an intersection is considered as an agent and the intersection itself as its environment. Given a non-static environment, during each cycle the agent first receives a state signal, then it performs an action, and, finally, it gets a reward for the action. The objective is to minimize the queue length, keeping the balance between the conflicting directions.

The state signal represents the aggregated number of queuing vehicles in non-conflicting directions. The links D10 and D30 are considered to have non-conflicting directions, so do D20 and D40 (Figure 4.1). Therefore, each state has two components: the queue length in NS (North-South and South-North) and the queue length in WE (West-East and East-West) directions. We also categorize both components of a state as L (Low), M (Medium), and H (High), according to the level of congestion in a particular direction. Hence, there is a discrete state space with $3^2 = 9$ states: LL, LM, ML, MM, MH, HM, LH, HL, HH.

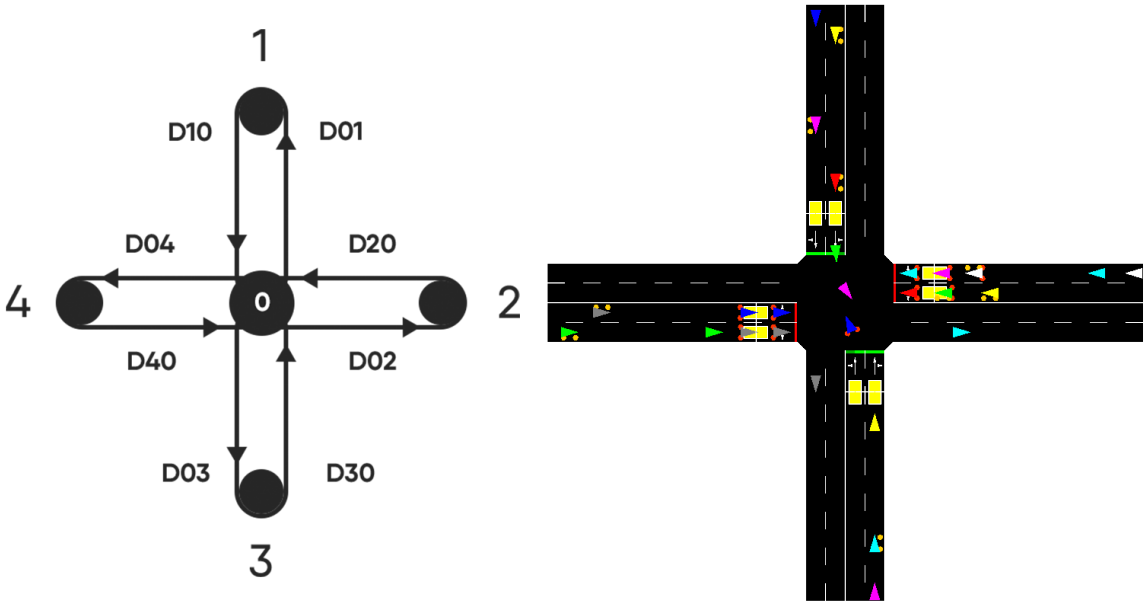

Figure 4.1: Test transport network.

In Table 4.1 n denotes the aggregated number of queuing vehicles in a direction. We uniformly distribute the queue length among the categories. However, an impact of any other distribution on the system has not been investigated, because in future work we consider replacing the discrete state space by a continuous analog.

During each cycle the agent can choose one of the three actions: first, do nothing, i.e $a_0 = (0,0)$; second, extend the NS green phase and accordingly shorten the WE

Table 4.1: Initialization of the categories

Category	Short	Range
Low	L	$0 \leq n \leq 30$
Medium	M	$30 < n \leq 60$
High	H	$60 < n \leq 90$

green phase, i.e., $a_1 = (+dt, -dt)$; third, extend the WE green phase and shorten the NS one, i.e., $a_2 = (-dt, +dt)$. Those actions change the signal plan of the next cycle. Here dt is a constant extension parameter.

After the agent performs an action the environment gives an immediate reward. We represent the immediate reward as a function R(s,s') of the previous and current states:

$$R = w_1 \overbrace{\left(|NS - WE| - |NS' - WE'|\right)}^{\text{equilibrium term}} + w_2 \underbrace{\left((NS + WE) - (NS' + WE')\right)}_{\text{queue reduction term}} \tag{4.2}$$

where $0 \leq w_1 \leq 1$ and $0 \leq w_2 \leq 1$ are control weights, $s = [NS, WE]$ and $s' = [NS', WE']$ are previous and current states.

In 4.2 there are two terms. The first is an equilibrium term which tries to keep the congestion in both directions in balance, and the second is a queue reduction term which serves to reduce the congestion in general.

The ε-greedy policy is taken as the behavioral policy of action selections. This means that we choose an action with the maximal estimated value with probability $1 - \varepsilon$, and choose a random action with probability ε.

4.1.3 Experiments

In order to evaluate the performance of our traffic light controller and test different scenarios we use one of the microscopic traffic simulation software [**?**]. We built an isolated four-way intersection with two-lane links and two-phase signal plan (Figure 4.1). Technically speaking, the intersection has green, red and amber phases. We modify only green and red phases, while the amber remains constant. The cycle length is fixed(Eq. 4.3):

$$t_c = t_g + t_y + t_r \tag{4.3}$$

The test intersection is Right-Hand Traffic (RHT) intersection. From each link a vehicle can go straight, turn right or left. Those moves are considered as non-conflicting moves. A vehicle may turn right from the rightmost lane, and left from

the leftmost one. All vehicles turning left clear the intersection during the amber light.

We set average lengths of links and vehicles to 100 meters and 5 meters respectively. The problem is simplified by excluding the following: U-turns, public transport, pedestrians, traffic rules violation, traffic road accidents, and parking. It is also assumed that the input data are available on request.

For the given intersection a set of random trips with a uniform distribution is generated. The generated demand model may be unrealistic, but it is good enough for the comparison of the performance of different methods.

The agent-environment interaction is an infinite horizon process (which does not naturally break into episodic tasks and continues without limits). Therefore, we let the simulation run for ≈ 28 hours with the following structure of the signal plan:

1. Green (NS) - Red (WE)
2. Amber
3. Green (WE) - Red (NS)
4. Amber

We conduct three main experiments with the same uniform demand model on the given test intersection. Since the generated demand model has a uniform distribution, the optimal signal plan should have more or less equal green phase duration in all directions. Our first experiment uses exactly these equal green phase durations, i.e., it exploits the fixed symmetric signal plan. Our second experiment uses the fixed signal plan which is far from optimal. The signal plan is fixed to be 24 seconds for the NS green light and 42 seconds for the WE green light. Finally, our third experiment starts from the same setting as in the second experiment, but it uses the proposed method to react to the demand changes. For the last experiment, in order to prevent a negative phase duration, we set a maximal and a minimal phase duration. Additionally, we penalize any unfeasible action by some negative reward.

For all three experiments the amber light is set to 12 seconds and the cycle length sums up to 90 seconds. For the proposed solution, we set $\varepsilon = 0.1$, $\gamma = 0.6$, $\alpha = 0.3$, $dt = 3$ seconds, $w_1 = w_2 = 1$. The Q-table is initialized as the zero matrix.

4.1.4 Results and evaluations

The first experiment shows no traffic congestion. The phase split perfectly unloads the intersection in both directions (Figure 4.2, top). The most frequently occurring state of the experiment is LL (Figure 4.3, blue bars).

By contrast, in the second experiment there are huge queues on the corresponding links due to considerably less amount of time allocated to the green light in the NS direction. The Figure 4.2 (middle) shows that the fixed asymmetric signal plan with the specified demand definition on the given intersection does not work properly. The dominating state is HL (Figure 4.3, cyan bars).

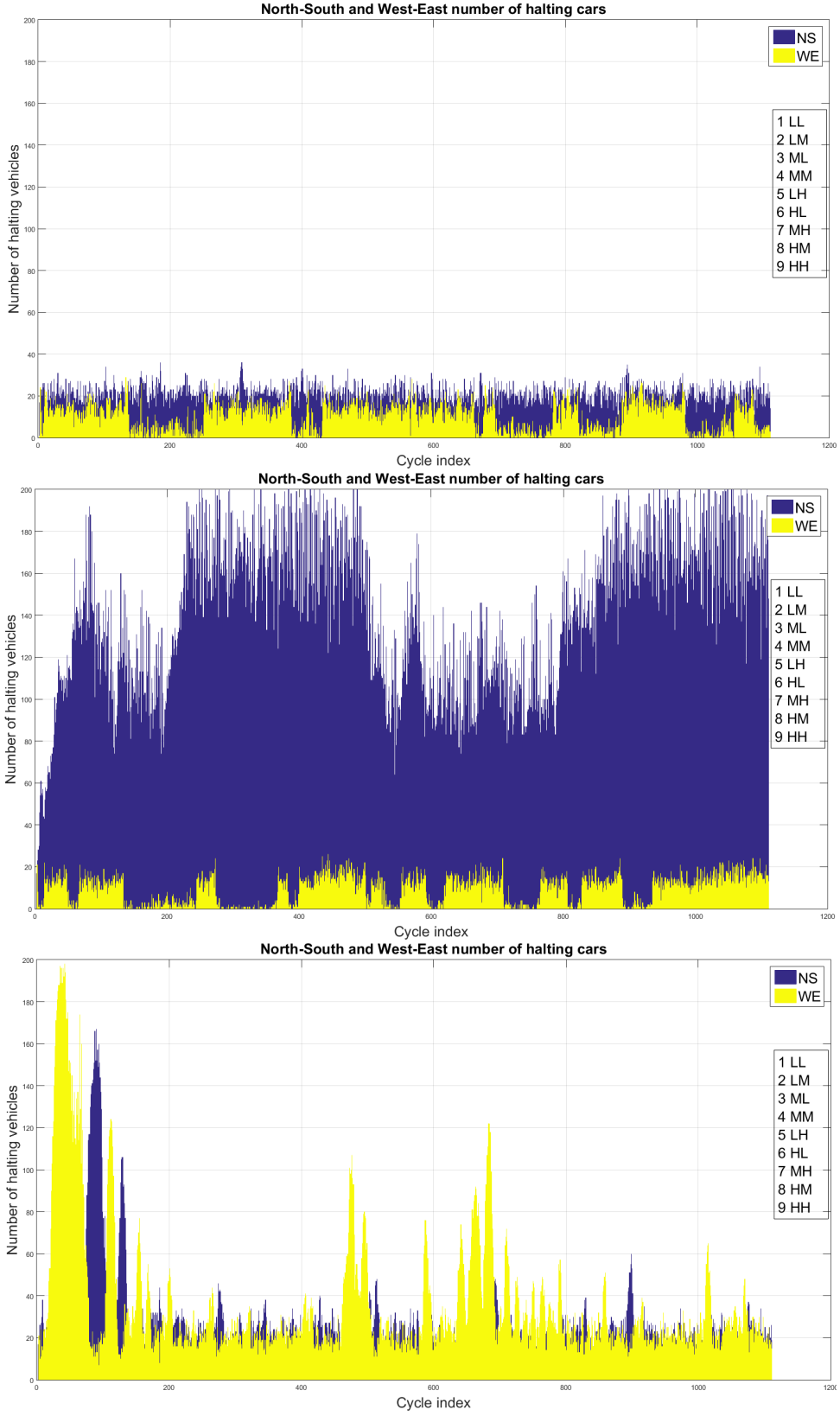

Figure 4.2: Number of halting vehicles in two conflicting directions for the symmetric fixed signal plan(top), fixed asymmetric signal plan(middle) and adaptive controller(bottom).

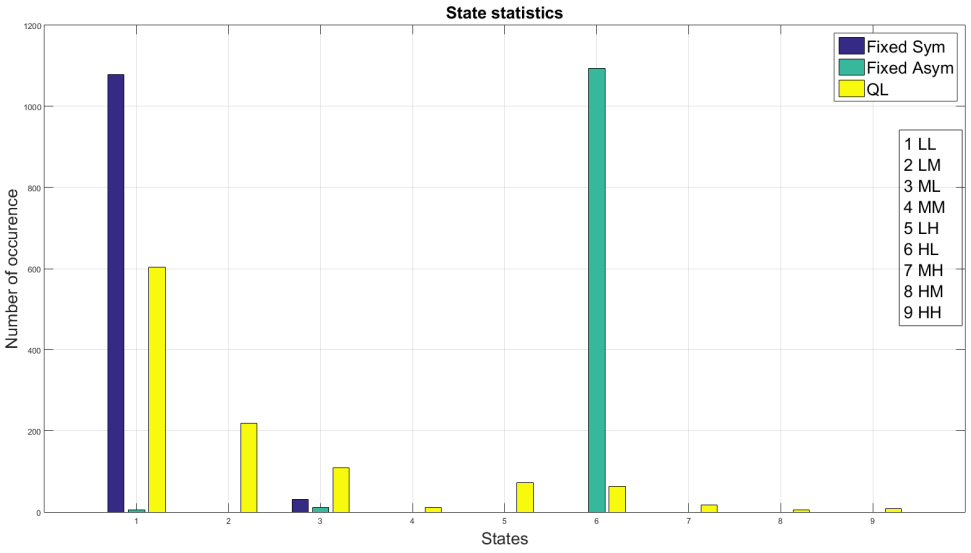

Figure 4.3: Quantitative evaluation of state occurrence for all three experiments.

In the third experiment we investigate our approach. The initial phase split of the adaptive controller has the same setting as in the second experiment. According to Figure 4.2 (bottom), the controller reaches a near-optimal policy within the first $\approx$ 150 cycles.As in the first experiment, LL is the most frequent state (Figure 4.3, yellow bars).

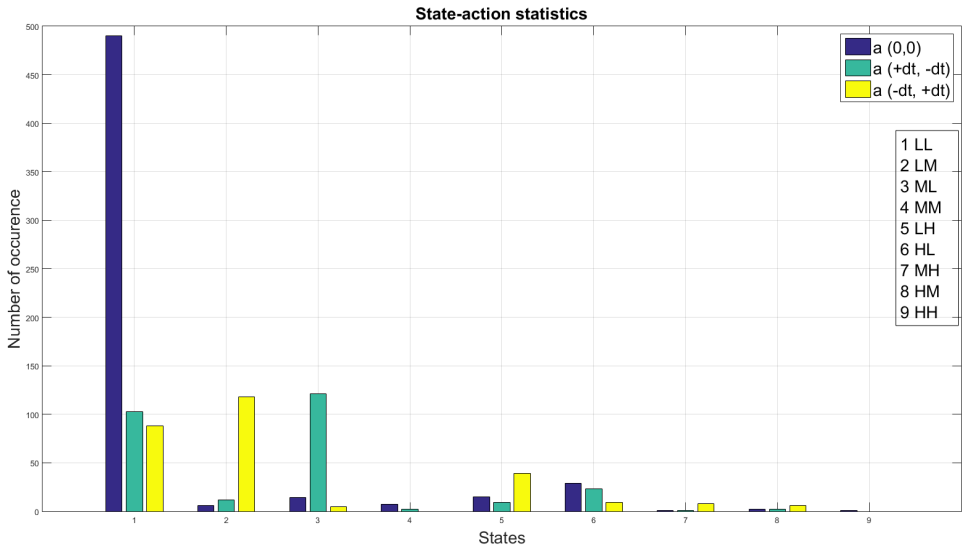

Figure 4.4: Quantitative evaluation of state-action pair occurrence for the adaptive controller.

Additionally, we provide state-action statistics, which show how often the actions have been taken at a particular state (Figure 4.4), and the reward function evolution (Figure 4.5).

As shown in Figure 4.4, at the LL state the adaptive controller almost never changes the signal plan; at the LM state the controller decreases the green light in the NS direction and increases it in the WE direction; at the ML state it proceeds in the opposite way. All other states are not visited frequently enough.

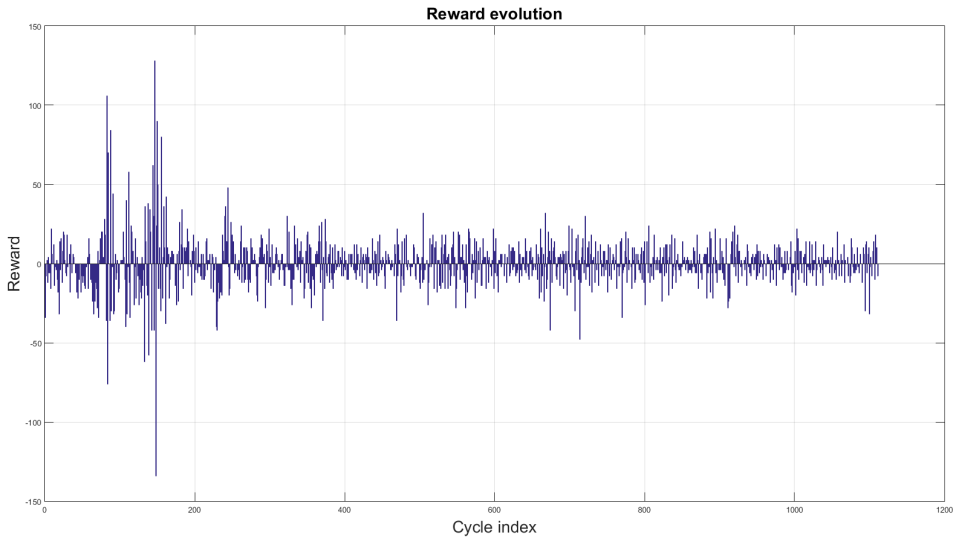

Figure 4.5: Evolution of the adaptive controller's reward function.

The reward function is expected to slowly approach zero with time span. Specifically, when an optimal phase split is found, further changes of phase durations would not improve the subsequent states and the immediate rewards become small. The Figure 4.5 shows that the immediate rewards oscillate around zero and decrease.

In this chapter we propose an adaptive model-free signal controller system based on a multi-objective one-step Q-learning algorithm.

We compared the performance of our solution to the ground truth phase split. In practice, however, improper phase splits are common issues. Therefore, we examined our solution against this non-optimal intersection setting.

The results of the experiments have shown that the adaptive method approached a near-optimal performance.It is possible to replace the discrete state space by a continuous analog and apply this method for a multi-intersection network.

4.1.5 Extension of the state and action spaces

As a continuation of the previous experiments we apply the proposed method with an extended action and state spaces. The action space consists of the following 9 elements:

1. $[-dt, -dt]$
2. $[-dt, 0]$
3. $[-dt, +dt]$
4. $[0, -dt]$
5. $[0, 0]$
6. $[0, +dt]$

Table 4.2: Initialization of the state category

Category	Short	Range
Extra Low	EL	$0 \leq n \leq 16$
Low	L	$17 < n \leq 32$
Medium	M	$33 < n \leq 48$
High	H	$49 < n \leq 64$
Extra High	EH	$65 < n \leq 80$

7. $[+dt, -dt]$
8. $[+dt, 0]$
9. $[+dt, +dt]$

where $dt \in N$. Note that the cycle length is not fixed anymore. Each component of the action is green phase duration in the NS and WE directions.

Each state has the following components:

1. Vehicle approaching the intersection from the north and the south
2. Vehicle crossed the intersection from the north and the south
3. Vehicle approaching the intersection from the west and the east
4. Vehicle crossed the intersection from the west and the east

$State : (D10 + D30)(D01 + D03)(D20 + D40)(D02 + D04)$ (see Figure 4.1)

Each state elements component is categorized as shown in the Table 4.2:
The state space consists of 625 elements.
We used the following continuous reward formula:

$$S_t = (NS, WE) \rightarrow S_{t+1} = (NS', WE'), \tag{4.4}$$

the reward signal is defined as follows:

$$\begin{aligned} R(S_t, S_{t+1}) \doteq \beta \overbrace{\left(|NS - WE| - |NS' - WE'|\right)}^{\text{equilibrium term}} + \\ + (1-\beta) \underbrace{\left(NS + WE - \left(NS' + WE'\right)\right)}_{\text{queue reduction term}}, \end{aligned} \tag{4.5}$$

where $\beta \in [0,1]$ is control weight;

$$NS := NS + \alpha WE_{crossed}; \tag{4.6}$$

$$WE := WE + \alpha NS_{crossed}; \tag{4.7}$$

$$NS' := NS' + \alpha WE'_{crossed}; \tag{4.8}$$

$$WE' := WE' + \alpha NS'_{crossed}; \tag{4.9}$$

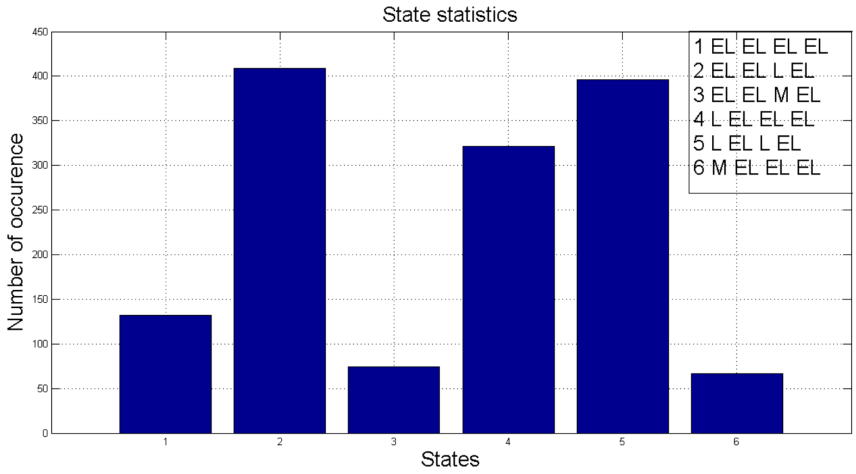

Figure 4.6: Quantitative evaluation of the most state occurrence for experiments

We provided state statistics of the most occured states (See Figure 4.6). From the results we can see that states consist of only EL, L and M components.

From the state-action pair statistics (see Figure 4.7) it can be concluded that actions are selected in an optimal way. The number of halting vehicles in two directions in the most cases converges (see Figure 4.8), and the intersection is in the equilibrium state.

The reward function is expected to slowly approach zero with time span. The Figure. 4.9 shows that the immediate rewards converge to zero.

The results of the experiments have shown that the adaptive method approaches a near-optimal performance when we extend our discrete state space and try to approach to continuous analog. Also by changing the action space we achieved the flexibility of the cycle length. Considering directions separating to enter and cross the intersection. Since, our reward formula is also modified.

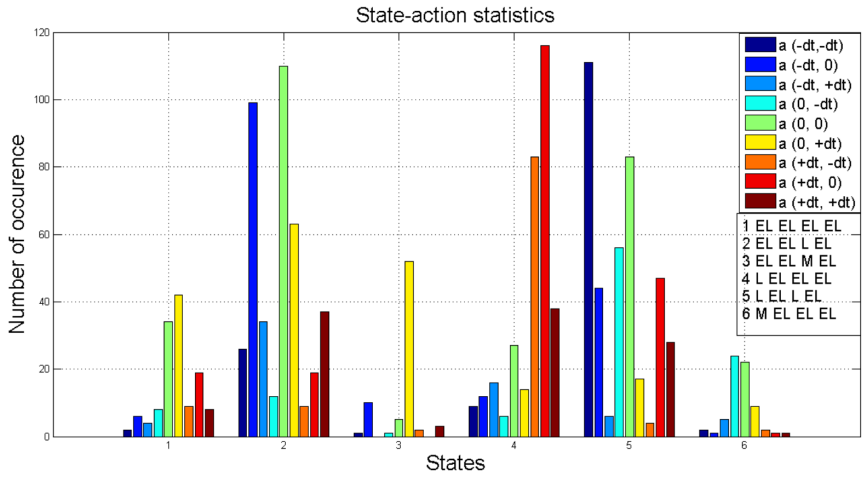

Figure 4.7: Quantitative evaluation of state-action pair occurrence

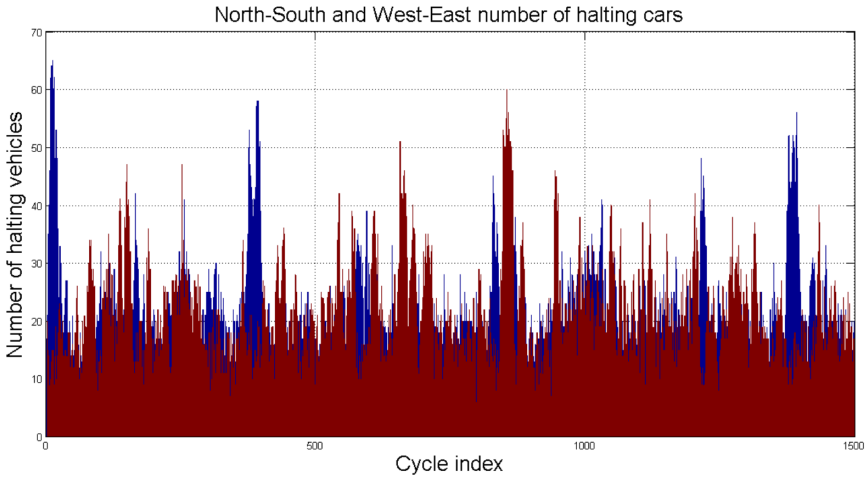

Figure 4.8: Number of halting vehicles in two conflicting directions

4.2 DQN

In the given section, one of the solutions - RL with action-value approximation (Deep Q-Network, DQN) considered to tackle the problem of traffic flow control on an isolated intersection. The approach that presented in the previous section has a serious limitation. It can not easily incorporate data from different sources. For instance, week day, day time, temperature, season, and etc. The action-value function approximation by a neural network overcomes the issue. After that, we decided to change the action space such that instead of adding and subtraction from green phases *dt* we directly set their duration. In other words, new action space consists of a set of phase durations. That does not change the way how the system works, but changes conceptually means a lot. To be more specific, now it implicitly takes into account the previous green phase durations since the information about lengths of the phases directly incorporated in the action itself.

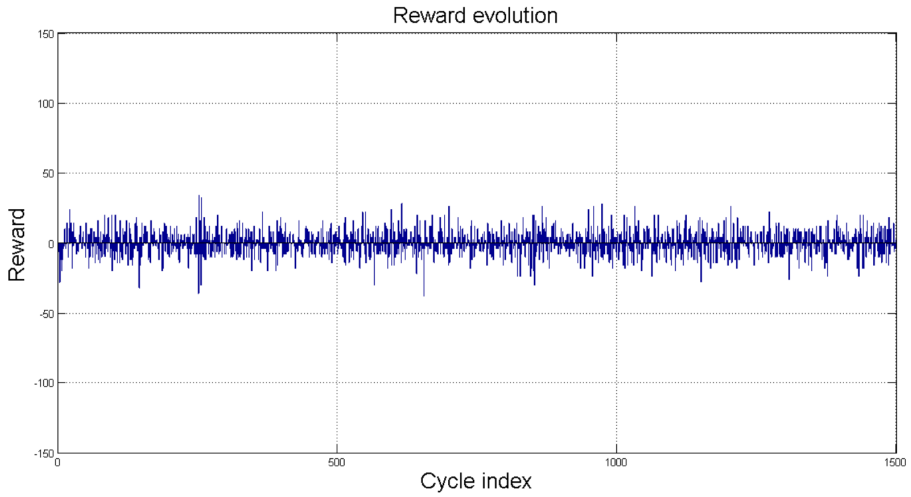

Figure 4.9: Evolution of the reward function.

The limitations of our immediate reward formula were relieved and the updated formula was proposed. Therefore, each of the four directions is considered separately.

4.2.1 Problem Formulation

The aim of the given work is to build an intelligent traffic signal controller for an isolated intersection using RL approach. The model trained using Deep Q-learning (DQN). Traffic signal controllers are considered as an agent, intersection as an environment. The goal of the agent is to reduce the queue length at the intersection and keep the balance between the all directions. Since the environment is non-stationary, traffic controllers should be adaptive and be able to adjust to the dynamically changing traffic road situation. DQN is adopted to enlarge state space and be suitable to be extended to a transport network of any size.

State space

The state space element represents the number of waiting vehicles in each direction and green phase signal for non-conflicting directions. The links D10 and D30 are considered to have non-conflicting directions, so do D20 and D40 (See Figure 2). Consequently, at each time step the state signal is represented by a 6-vector of values as follows:

$$NS \doteq D10 \tag{4.10}$$

$$EW \doteq D20 \tag{4.11}$$

$$SN \doteq D30 \tag{4.12}$$

$$WE \doteq D40 \tag{4.13}$$

$$grPhaseNS \doteq \text{green phase duration for } D10 \text{ and } D30 \tag{4.14}$$

$$grPhaseWE \doteq \text{green phase duration for } D20 \text{ and } D40 \tag{4.15}$$

$$S \doteq (NS, SN, WE, EW, grPhaseNS, grPhaseWE) \tag{4.16}$$

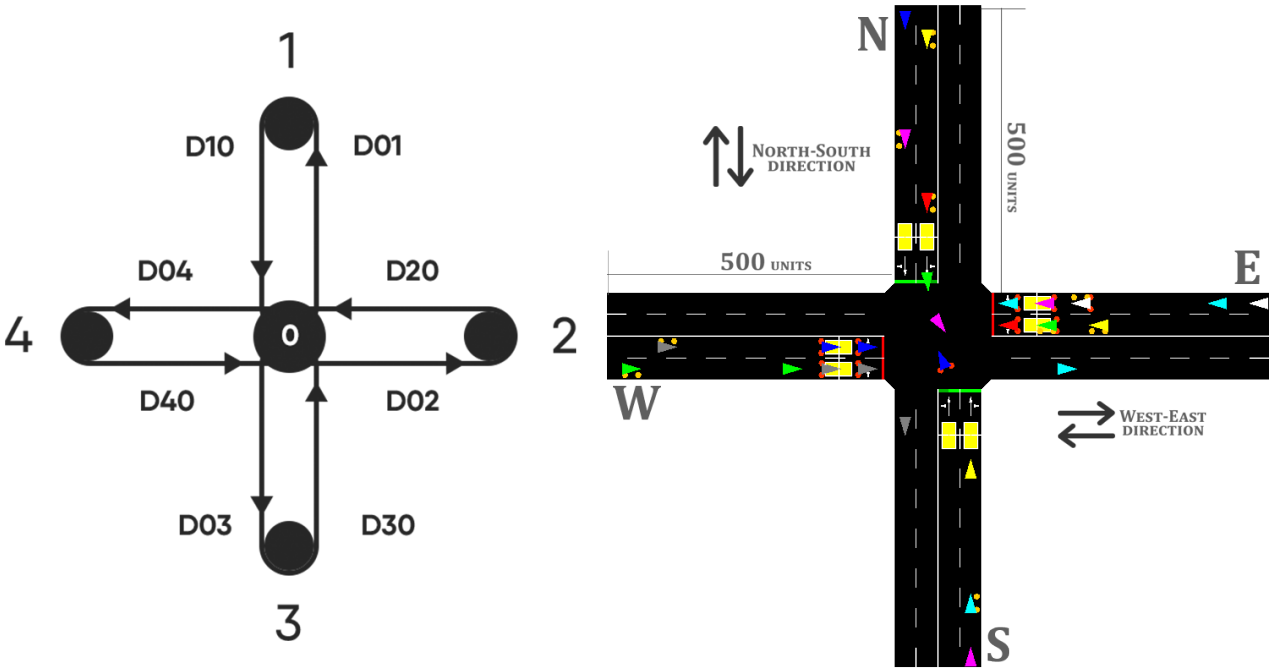

Figure 4.10: The test transport network

Thus, the state space $\mathscr{S}$ in this setting is unbounded:

$$\mathscr{S} \doteq \{(s_0, s_1, s_2, s_3, s_4, s_5) \mid s_0, s_1, s_2, s_3, s_4, s_5 \in Z^+\} \tag{4.17}$$

Besides the number of halting vehicles and current phase duration a variety of other factor influencing decision making might be included into the state vector, such as the current, time, weather, etc. This might be a major improvement to the accuracy of decision making and will be considered in the future work.

Action space

At the beginning of each cycle the agent first receives a state signal, then it performs an action, and, finally, it gets a reward for the action. The agent-environment interaction is an infinite horizon process (which does not naturally break into episodic tasks and continues without limits). The intersection has the green, red and amber phases. The signal plan of each cycle comprises the following components:

1. Green (NS) - Red (WE)
2. Amber
3. Green (WE) - Red (NS)
4. Amber

We modify only the green and red phases, while the amber remains constant.Action element is a precise value of the green phase duration in the non-conflicting directions, therefore, cycle length is non-fixed. The minimum duration is 12 seconds, the maximum is 60. Step size is dt. After each cycle the agent can choose one of the action from the action space:

- $a_0 \doteq (12, 12)$
- $a_1 \doteq (12, 12+dt)$
- ...
- $a_{n-1} \doteq (60-dt, 60)$
- $a_n \doteq (60, 60)$

These actions change the signal plan of the next cycle.

$$\mathscr{A} \doteq \{a_0, a_1, a_2.., a_n\} \tag{4.18}$$

Reward formula

Insofar as the task of the agent is to reduce the number of halting vehicles as much as possible, while keeping the balance between the conflicting directions, the reward formula ought to clearly reflect these goals. For this task in our previous work [60] we used the following continuous reward formula:

$$S_t = (NS, WE) \rightarrow S_{t+1} = (NS', WE'), \tag{4.19}$$

the reward signal is defined as follows:

$$R(S_t, S_{t+1}) \doteq \beta \overbrace{\left(|NS - WE| - |NS' - WE'|\right)}^{\text{equilibrium term}} + \\ +(1-\beta) \underbrace{\left(NS + WE - \left(NS' + WE'\right)\right)}_{\text{queue reduction term}}, \tag{4.20}$$

where $\beta \in [0,1]$ is the trade-off between the queue reduction and equilibrium terms. Throughout all the proposed algorithms we have used $\beta = 0.5$, i.e. the queue reduction and equilibrium goals are equally important for learning.

The reward formula 4.20 captures the following intuition: if the difference between the conflicting directions has decreased, then the first term will be positive; if the total number of standing vehicles has decreased, then the second term will be positive, and the reward value will also be positive in proportion to the magnitude the mentioned values have decreased. In case the values of both goals increase, the overall reward will be negative in the magnitude of the increases. Finally, when there is an increase in the value of one goal and decrease in the other, the two terms of the formula will eat each other and the resultant value will indicate whose impact is stronger — this accounts for the presence of the plus sign in the formula. However, if we consider a case where the total number of halting vehicles in the NS and SN directions is increasing/decreasing, but distribution is high in one direction and low in other, the proposed reward formula does not take this into account. In addition, if the number of halting vehicles is large and during the next time interval it increases/decreases slightly, and if the number of halting vehicles is small the system can not distinguish these cases. Therefore, the queue reduction term will give the same results. For solving described problems, we decided to distinguish four directions and proposed to use two continuous reward formulas. The first one is based one the queue lengths:

$$S_t = (NS, SN, WE, EW) \rightarrow \\ S_{t+1} = (NS', SN', WE', EW'), \tag{4.21}$$

We divided the previously used NS and WE directions into two components NS, SN and WE, EW, respectively.
The reward signal is defined as follows:

$$R(S_t, S_{t+1}) \doteq$$

$$\doteq \beta \overbrace{\big(min(\mu - NS', \mu - SN', \mu - WE', \mu - EW')\big)}^{\text{equilibrium term}} + \\ +(1-\beta) \underbrace{\big(-max(NS', SN', WE', EW')\big)}_{\text{queue reduction term}}, \tag{4.22}$$

where

$$\mu = \frac{(NS + SN + WE + EW)}{4} \tag{4.23}$$

Minimum from the deviation of the mean in all directions gives information about how much the intersection is in balance.The queue reduction term considers only the current state in order to see the worst state from all directions. Now, the value of our reward in the best case will be 0. When all four directions have the same number of waiting vehicles or there are no ones, first term will be zero. The second term is zero when there is no queue.
The second reward formula based on the waiting time. It is common reward representation used in the previous researches:

$$R(S_t, S_{t+1}) \quad \doteq \sum_{i=1}^{N} \frac{w_t^i}{N_t} - \sum_{i=1}^{N} \frac{w_{t+1}^i}{N_{t+1}} \tag{4.24}$$

where w is the total waiting time of vehicles in the intersection in current and previous states. N - number of halting vehicles.

4.2.2 Experiments and results

The environment for the experiments is a traffic network in Figure 1. An isolated intersection built in simulation platform SUMO DLR [41]. The average length of links is 100 meters. We are excluding the U-turns, public transport, pedestrians, traffic rules violations, traffic road accidents, and parking. Travel demand models are given in 4.5. Initial signal plan: the green phase in NS direction is set to 24 seconds, and the green phase in WE direction is set to 42 seconds. Our simulation runs approximately 28 hours. The parameter settings are given in the Table 4.3 and Table 4.4.

For the evaluation of our method rewards with average queue lengths and waiting time were used. Fixed-signal control with predefined phase split is considered as ground truth.

Two groups of experiments conducted using synthetic data (See Table 4.5). In the first part of our experiment uniform distributed demand model (Configuration 1) is used as an environment. The result of the simulations is given in Table 4.7. The DQN model with proposed rewards and fixed-time controller are compared. For the fixed control several signal plans were tested, and the best one was chosen: (24s,

Table 4.3: Parameter settings

Free parameter	**Value**
Action time interval Δt	6 seconds
Learning rate for Q-learning α_1	0.6
Discount factor for reward γ	0.8
ε	0.1

Table 4.4: The NN parameters

Parameter	**Value**
Number of layers	4
Number of neurons	6, 14, 34, 81
Learning rate α_2	0.001
Activation function on hidden layers	ReLu
Activation function on output layer	Linear

24s).

The Reward 1 is oscillated around zero and decreasing (See Figure 4.11). During the experiment it was noticed that the signal plan had stopped changing. However, the intersection was overloaded. A situation occurred when our reward formula incorrectly reflects the state of the environment. The Reward 2 allows us to avoid this problem. The maximum value of this reward may be zero.

Experiments show that the system with the proposed rewards performs near-optimal and is able to stabilize when using uniform distributed demand. In the second group of experiments we used a mixed demand model, where the arrival rate of vehicles is changing during simulation time. The results show that system can not

Table 4.5: Demand models

Config	**NS**	**SN**	**WE**	**EW**
1 (0-100000 s)	0.2	0.2	0.2	0.2
2 (0-100000s):				
(0-20000 s)	0.2	0.2	0.2	0.2
(20001-40000 s)	0.4	0.4	0.2	0.2
(40001-60000 s)	0.2	0.2	0.2	0.2
(60001-80000 s)	0.2	0.2	0.4	0.4
(80001-100000 s)	0.2	0.2	0.2	0.2

Table 4.6: Performance on Configuration 1.(QL-queue length, DT-delay time)

Model	**QL NS**	**QL WE**	**QL SN**	**QL EW**
DQN: Reward 1 (12)	5.64	14.23	5.72	11.53
DQN: Reward 2 (14)	6.21	8.19	6.25	9.03
DQN: Reward 3 (16)	3.35	4.56	3.37	4.71
Fixed-time control	5.37	5.31	5.42	5.24
Model	**DT NS**	**DT WE**	**DT SN**	**DT EW**
DQN: Reward 1 (12)	112.57	369.63	113.68	290.99
DQN: Reward 2 (14)	135.82	195.65	135.25	223.35
DQN: Reward 3 (16)	50.26	76.71	50.58	80.19
Fixed-time control	92.93	92.26	93.50	90.03

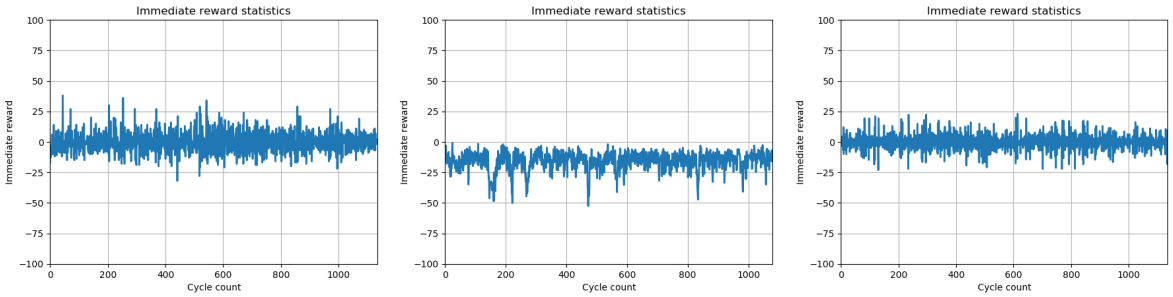

Figure 4.11: Configuration 1. Reward 1, Reward 2, Reward 3

properly arrange to the new situation and reward value has large deviation (See Figure 4.12). The controller is being rebuilt all the time and cannot stabilize. There may be several reasons: lack of NN flexibility, often update (occurs each step), statistics (history) are not taken into account.

Table 4.7: Performance on Configuration 2. (QL-queue length, DT-delay time)

Model	**QL NS**	**QL WE**	**QL SN**	**QL EW**
DQN: Reward 1 (12)	65.86	25.65	66.16	26.33
DQN: Reward 2 (14)	56.21	34.47	56.85	34.48
DQN: Reward 3 (16)	55.52	31.41	54.62	31.16
Fixed-time control	53.03	29.18	53.15	29.15
Model	**DT NS**	**DT WE**	**DT SN**	**DT EW**
DQN: Reward 1 (12)	1843.41	475.52	1858.93	529.27
DQN: Reward 2 (14)	1460.05	888.77	1509.09	2932.12
DQN: Reward 3 (16)	1257.15	717.91	1171.41	714.81
Fixed-time control	991.66	531.02	985.22	525.54

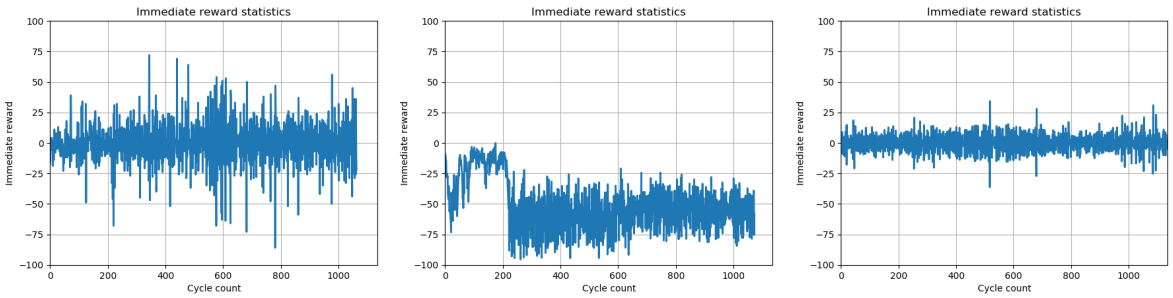

Figure 4.12: Configuration 2. Reward 1, Reward 2, Reward 3

In this chapter, we propose a model-free, online, off-policy reinforcement learning traffic control system. The model is training based on the temporal-difference learning with ANN. The main bias was on the reward function. Based on the shortcomings of the previous work, a new reward formula was proposed. The adaptive controller gave near-optimal performance. The experiments were conducted on a uniform and mixed demand model. The result shows that the system based on the neural network has difficulties in a non-uniform environment. Some reasons were identified. To eliminate the disadvantages, it is planned to further expand the neural network architecture and consider the possibility of describing the state of the environment with more stable information based on statistics.

Chapter 5

Link flow estimation based on deep learning models

This chapter deals with the problem of estimating a certain observed amount of traffic flow in a given link of a transport network for a short-term forecast [61]. The aim is to evaluate (calculate) the changes in the flow at the road section over time. The time interval for future estimation is not fixed and is varied. The link flow is treated as the probability of vehicles being generated in unit time. The value is not the exact proportion of the vehicles arrived, but represents some other properties of traffic flow in the given link. The link flow estimation problem will be formulated as follows:

- Given data going back and sampled every period of period, is it possible to predict the link flow $F_{i,t+m}$ in the near future (at time $t+m$) based on pre gained data? $F_{i,t}$ is the flow rate on the link i at time t.

The presented dataset is a time series data, with a timeseries being a sequence of observations taken sequentially in time [62]. A time series involves a time component, which gives an explicit ordered relationship between observations. There are two key issues when time series are used for flow estimation:

- The time series analysis requires the process to be stationary. However, the traffic flow is dynamically changing;
- The spatial characteristics of the time series data are not taken into account in the classical methods [63]. However, when sequential data are considered it is important to consider a more deep representation of features.

These issues can be solved by a hybrid model of RNN and CNN, which is presented in this study. The problem is formulated as a supervised learning regression task.

The following questions are addressed:

- Is it possible to improve the performance of the baseline model, which is a fully connected neural network?

- How far ahead can the model predict the link flow?

In order to solve the problem, different approaches have been undertaken in recent studies, based on mathematical models of traffic flow [64], [65] and data-driven methods [66], [67]. Early classical methods rely on complex mathematical models that show low efficiency. Nonparametric methods such as k-nearest neighbor (kNN), auto regressive statistics in time series (ARIMA), Support Vector Regression (SVR), and artificial neural network (ANN) have been widely used due to the stochastic and nonlinear nature of traffic flow data [68] -[70]. The k-NN could not overcome linear methods, although it gave good results. The Online Support Vector Regression (Online SVR) proved to be slightly better in comparison with baselines and exceeded them under atypical non-repeating traffic conditions. ANN with shallow architecture could not adapt well to the changes in traffic flow, since such a network is not flexible enough to capture the deep features of traffic flow. Deep learning methods allow highlighting these features at different levels and have the ability to reveal the spatial-temporal properties of data. These types of data-driven methods are widely and successfully used to solve the computer vision problem and NLP problems [71], [72]. Deep learing methods for forecasting the traffic flow have been used in [73]-[75]. A new method based on automatic Stacked Autoencoders has been proposed in [74]. In [75] Stacked Denoising Autoencoders have been suggested to predict the characteristics of the traffic flow and reveal its hidden features. However, these types of neural networks are difficult to train due to the complex interconnections between the layers. The special architecture of ANN that reduces connections and does not suffer from the curse of measurement is CNN. It is very effective when working with spatial structure data. Moreover, the combination of several layers allows detecting global dependencies. The estimation of the link flow by means of a fully-connected NN has been performed in [76]. However, such networks can consider only a fixed amount of previous state data to predict the next one. An RNN can take into account all the previous states in time steps. A combination of recurrent neural networks and convolutional neural networks would allow understanding spatial-temporal sequences well.

5.1 An overview of the proposed models

In this study, three models of deep neural networks have been trained and tested on a simulator, as well as a fully connected feed-forward network (FCNN). FCNN is considered as the baseline. The first model is based on the Recurrent Neural Network (RNN). It is a type of neural network that uses internal memory in order to process sequential data and it takes both the new data and the output from the previous step as input to the current time step. The main difference between RNN and other networks is the presence of memory. Fully-connected and convolutional neural networks cannot remember the previous state. States are not saved between inputs,

and each one is processed independently. The presented dataset is considered as a sequence in time. In order to process data using simple networks, it is needed toprocess the entire sequence at once. However, the RNN has an internal loop that allows it to remember the states and reuse the values calculated in the previous state(iteration). Classical RNNs cannot remember long-term dependencies due to the vanishing gradient problem.

This problem is solved by a type of the RNN-architecture: Long-Short-Term Memory (LSTM) [77] and the Gate Recurrent Unit (GRU) [78]. The GRU can detect short-term and long-term dependencies together.

The presented model uses the GRU type. It has an update and reset gates. An update gate z_t^j decides how much the past state affects the current state and is calculated by the following formula:

$$z_t^j = \sigma(W_z x_t + U_z h_{t-1})^j \tag{5.1}$$

The reset gate is calculated as follows:

$$r_t^j = \sigma(W_r x_t + U_r h_{t-1})^j \tag{5.2}$$

The activation h_t^j of the GRU at time t is a linear interpolation between the previous activation value h_{t-1}^j and the updated activation value $\hat{h}_t^j$:

$$h_t^j = (1 - z_t^j) h_t^j - 1 + z_t^j \hat{h}_t^j \tag{5.3}$$

This procedure is taking a linear sum between the existing state and the newly computed state. The candidate activation $\hat{h}_t^j$ is computed as follows:

$$\hat{h}_t^j = tanh(W x_t + U(r_t \bigodot h_{t-1}))^j \tag{5.4}$$

where r_t is a set of reset gates.

CNN is a type of neural network with specific architecture that considers and treats input data as spatial. It differs from a fully-connected network in structure, where neurons are connected with nearest neurons and all have the same weight, instead of connecting to every neuron in the previous layer. CNN has a very high performance in solving computer vision problems since the architecture of such a network fits well to an internal representation of a two-dimensional image. This allows the model to learn the position and scale invariant structures in the data, which is important when working with images. This feature of CNN has been used to process sequence data and find ordered relationships in the time steps of a time series. CNN is considered as a cheaper alternative model of RNN for time series forecasting problems.

Classical CNN has the following stacked layers:

- convolutional :

$$c_j^k = \sum_i x_i^{k-1} * w_{ij}^k + b_j^k \tag{5.5}$$

- activation

$$x_j^k = \theta(c_i^k) \tag{5.6}$$

- pooling

$$x_j^{k+1} = pooling(x_j^k) \tag{5.7}$$

where x_i^{k-1} is an input, k is an engaged layer.

A 1D CNN is a modified version of CNN where a convolutional hidden layer operates with a 1D sequence. It is very effective when you expect to derive informative features from fixed-length segments of the overall dataset.

The main difference between the classical 2D CNN and 1D is in the dimension of input data and the way how the filter passes through the data.

Recent studies show that for certain applications, including analysis of time sequences of sensor data and analysis of audio signals, 1D CNNs outperform 2D counterparts due to the following reasons:

- lower computational complexity due to dimensionality reduction
- relatively shallow architecture easier to train

The third model (see Figure 5.1) consists of CNN and RNN modules, where the first part serves as data conversion for the input of the second module. Preprocessing data with CNN allows RNN to increase its sensitivity to the data sequence.

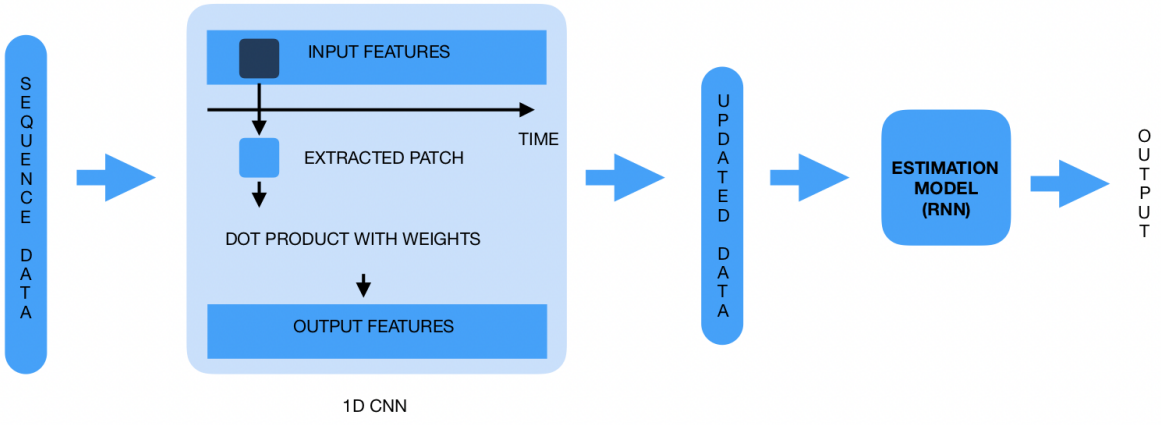

Figure 5.1: Combination of CNN and RNN

The loss function used the Mean Squared Error (MSE):

$$MSE = \frac{1}{n} \sum_n^{i=1} (Y_i - \hat{Y}_i)^2 \tag{5.8}$$

where n - number of estimations, Y_i - target value, $\hat{Y}_i$ - predicted value.

5.2 Experiments and Results

The traffic flow simulation interface SUMO DLR [41] is used as a tool for numeric experiments. The environment for experiments on synthetic data is an isolated intersection with two 2-lane edges (see Figure 5.2). Every edge length is 100 meters. The average vehicle length is 5 meters. The total number of halting vehicles at the intersection is in the range of $0 \leq N \leq 80$.

In order to simplify the dynamics of traffic flows, the left turn (turns) is prohibited, neither public transport, pedestrians, traffic violations, traffic accidents nor parking are considered.

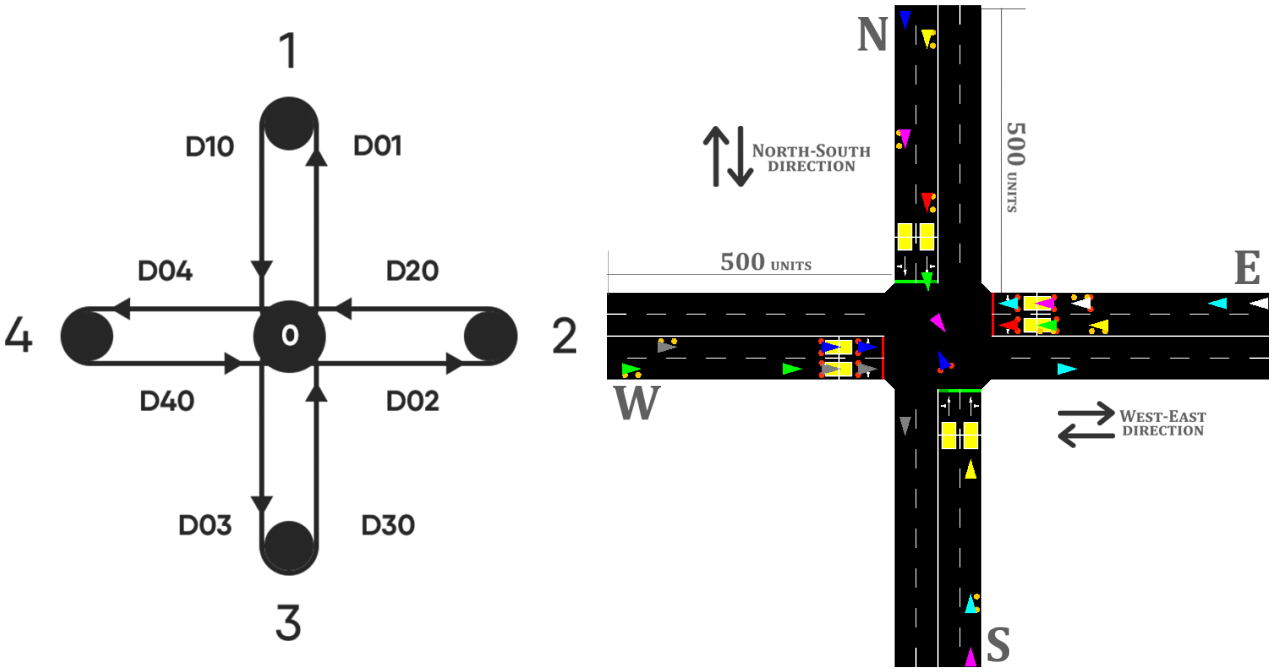

Figure 5.2: Test intersection. (5 nodes, D01, D10 ... D40 - edge id)

5.2.1 Dataset

Since synthetic data obtained by means of the simulator are generated, it is not needed to annotate them, as soon as the parameters of the link flow are known. The task is considered as a supervised learning problem. Several traffic flow settings and signal plans are used for the data collection process.

The test intersection consists of four directions, and each one is considered independently. Vehicles from different directions do not interact with each other since the left turns are excluded from the simulation for simplicity. This allows training the model in order to estimate the flow in a single direction alone. Then it is used four times for each direction separately. The equally steady four level traffic is simulated

(6.1). In this paper, the demand values are the probabilities of generating vehicles every second in a specified direction. The demand higher than 0.4 for each direction is not taken into account due to the physical limitation of the test network. The test intersection with the current settings cannot accommodate more vehicles. Each simulation lasts 50000 seconds.

Table 5.1: Demand models

Config	**NS**	**SN**	**WE**	**EW**
1 (0-50000 s)	0.1	0.1	0.1	0.1
2 (0-50000 s)	0.2	0.2	0.2	0.2
3 (0-50000 s)	0.3	0.3	0.3	0.3
4 (0-50000 s)	0.4	0.4	0.4	0.4

Each demand model has been simulated with 81 predefined signal plans. In total, 4500 simulation hours with the stationary environment have been launched to collect data. In this work the traffic light contains two phases: Green (NS) – the green signal on North-South and South-North directions with the red signal on WestEast and East-West directions; Green (WE) – the green signal on West-East and East-West directions with the red signal on North-South and South-North directions. The signal plan of each cycle comprises the following components:

1. Green (NS) - Red (WE)
2. Amber
3. Green (WE) - Red (NS)
4. Amber

The amber signal is fixed and equal to 3 seconds. The duration of the green phase is controlled in the NorthSouth and West-East directions. More complex signal plans can be conducted in a similar way. The predefined signal plans are as follows:

- $a_0 \doteq (12, 12)$
- $a_1 \doteq (12, 12 + dt)$
- ...
- $a_{n-1} \doteq (60 - dt, 60)$
- $a_n \doteq (60, 60)$

Here $dt = 6$ seconds is assumed, so that there are 81 different signal plans in total.

The time step for data extraction is 6 seconds. The following data are recorded:

- The number of halting cars in the North-South direction
- The number of halting cars in the South-North direction
- The number of halting cars in the West-East direction
- The number of halting cars in the East-West direction
- The average delay time of vehicles in the North-South direction
- The average delay time of vehicles in the South-Northdirection
- The average delay time of vehicles in the West-East direction
- The average delay time of vehicles in the East-West direction
- The green phase duration in North-South and South-North directions
- The green phase duration in West-East and East-West directions
- The current phase id (since the data is obtained after a certain period of time, and not in a single phase)
- The remaining time in seconds until the end of the given phase

Finally, for further work, a dataset has been created with 2700540 rows of data. It is divided into two parts: training (80% of each type of data) and testing data (20%).

In prediction problems, RNN is used because it considers the sequential nature of a process. The presented RNN model takes some data from a recent past (a few last minutes) as input and predicts the link flow that will be in the near future (after a few minutes).

Many-to-one approach of RNN is used, where the model uses multiple steps as input to estimate a single prediction. Since the appearance of vehicles in different directions does not depend on each other, and the test intersection is isolated, the flow prediction for each direction is carried out separately. A model has been trained in order to estimate the flow in a single direction.

The problem is formulated as a supervised learning regression task. The exact statement of the problem is as follows:

- Given data going as far back as lookback timesteps (a timestep is 6 seconds) and sampled every steps timestep, is it possible to predict the link flow in delay timesteps?

The input vector for the estimation model is as follows:

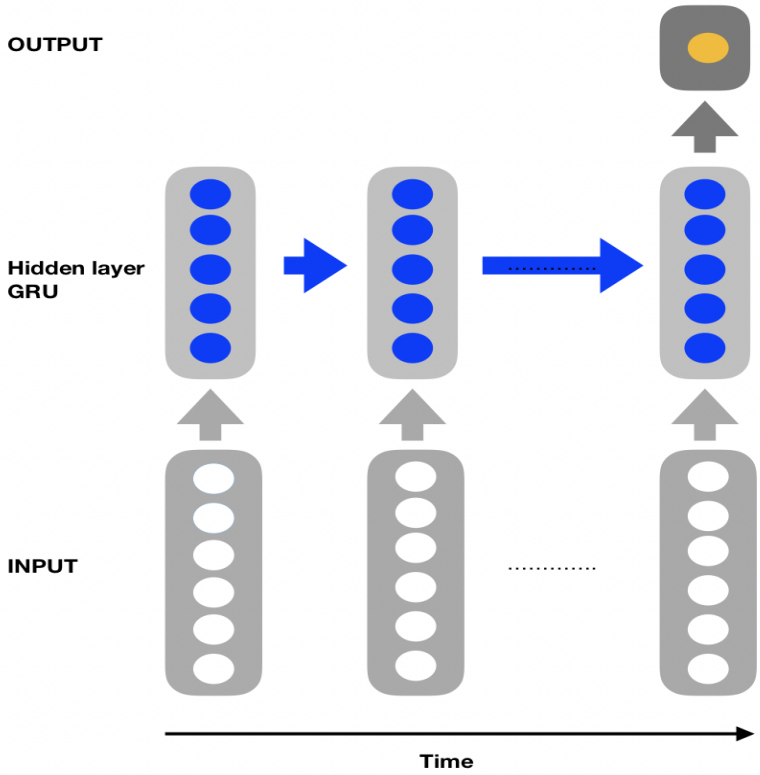

Figure 5.3: GRU link flow estimation model

- the number of halting cars in the specified direction
- the average delay time in the specified direction
- the green phase duration in the North-South and South-North directions
- the green phase duration in the West-East and East-West directions
- the phase ID
- the remaining time in seconds until the end of the given phase.

The presented GRU based model is depicted in Figure 5.3. Each input vector (feature vector) has six values. At time unit one input block is generated which contains a certain number of feature vectors. The size of the input block depends on how far back it is desired to analyze the data. The sequence proceeds to the GRU layer, finally predicts the link flow in the given direction, and gives one value as output.

The following parameter values are used:

- lookback = 100 i.e. our observations of the last 10 minutes
- step = 10 i.e. observations will be sampled at one data point per minute.

- delay = 10 i.e. targets will be 1 minute in the future

Before training the model, the following two steps are needed:

- Preprocessing the data to a format that the neural network can accept. The mean normalization has been used for preprocessing.
- Using a Python generator that takes our current dataset and releases data packages from the previous states along with a target link flow in the future.

The optimal values of free parameters have been selected experimentally.

Table 5.2: GRU model settings

Parameter	**Value**
GRU	5
Optimizer	Adam
Loss	MSE
Dropout	0,2
Recurrent dropout	0,4

The execution time is 945.88 seconds, of which 914.09 seconds are for model training. The results of losses in training and validation of the drop-out regularized GRU model are excellent (See Figure 5.4). There is no overfitting during the 30 epochs. The differences in loss values between training and validation are very small. It can be concluded that both values are about the same and optimal.

5.2.2 Convolutional Neural Network based model

A CNN based model has been trained to process sequence data. The model takes the time series sequence from the past as input and forecasts the link flow. The presented model is a one-dimensional CNN interface with the following structure (see Figure 5.5):

- a 1D Convolution Layer for recognizing local patterns in a sequence,
- a pooling layer to reduce the input lengths,
- a flatten layer for vectorization.
- a dense layer for regression or classification tasks.

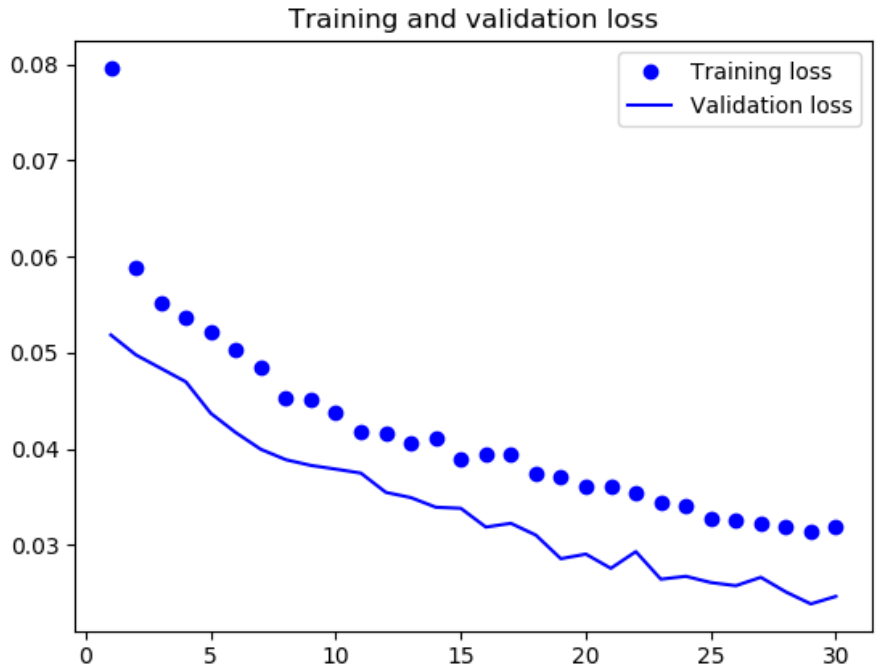

Figure 5.4: The loss values of training and testing the GRU based model (the loss is calculated using the MSE function)

Table 5.3: Model

Parameter	**Value**
1D CONV layer	
Filters	32
Kernel size	3
Pooling size	3
Hidden layer neurons	5
Optimizer	Adam
Loss	MSE

The statement of the problem and input vector are the same as for the RNN-based model in the previous section.

lookback = 100 timesteps (i.e. 10 minutes) is set.

The running time is 350.46 seconds, of which 310.245 seconds are for training. Our training and validation results are slightly lower than in the RNN-based model (See Figure 5.6). However, the training and validation loss are converging very well.

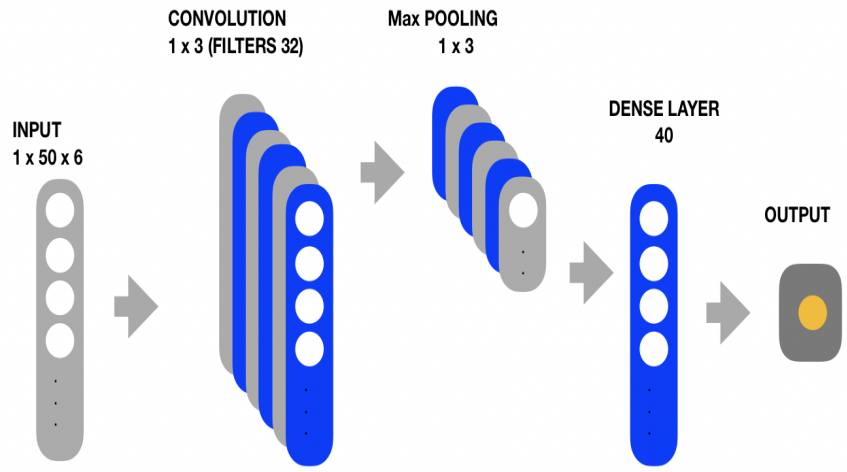

Figure 5.5: CNN estimation model

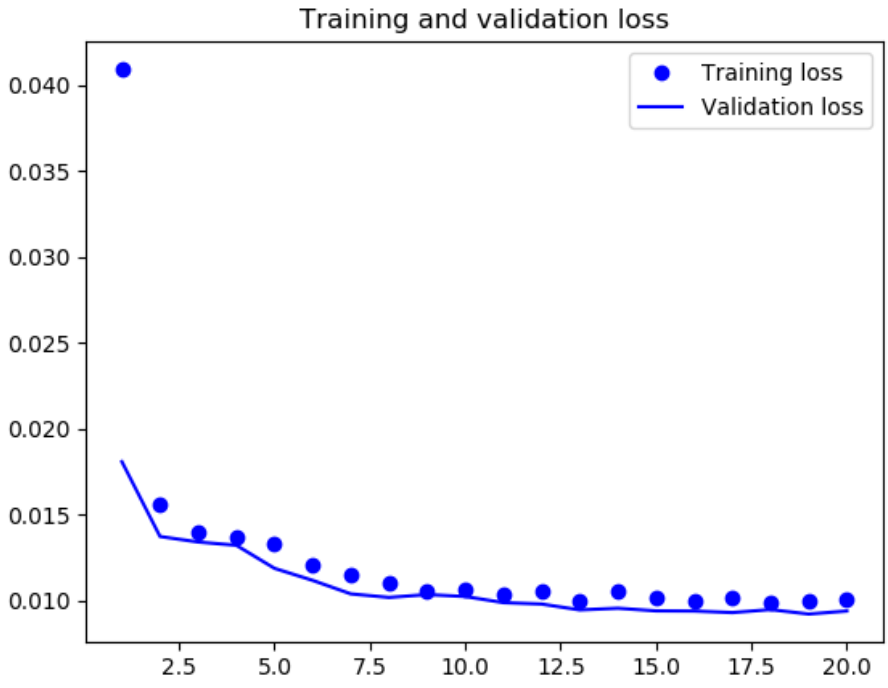

Figure 5.6: The loss values of training and testing the CNN based model (the loss is calculated using the MSE function)

5.2.3 Hybrid model based on CNN and RNN

In the last model, a combination of a one-dimensional CNN and a regularized GRU is used. The preprocessing of data with 1D CNN and the order-sensitivity of RNN give good results. Both the networks are connected in series and settings for the models are the same as in the previous sections.

This technique is not widely used for sequence data in recent studies [79] and has not been previously used for the link flow estimation problem. However, the results of these experiments show that such a combination happens to be very effective and powerful. This approach allows proceeding using much longer sequences, but, in order to make a comparison, the lookback (100) and delay timesteps (10) are not changed.

According to the results of training and validation loss, the given setup is very good, and it is significantly faster than the other proposed models (See Figure 5.7). Both loss values are about the same and eventually converge. Therefore, it can be concluded that the given model has perfect fitting for solving the presented problem. The execution time is 515.90 seconds, of which 501.19 seconds are for the model training.

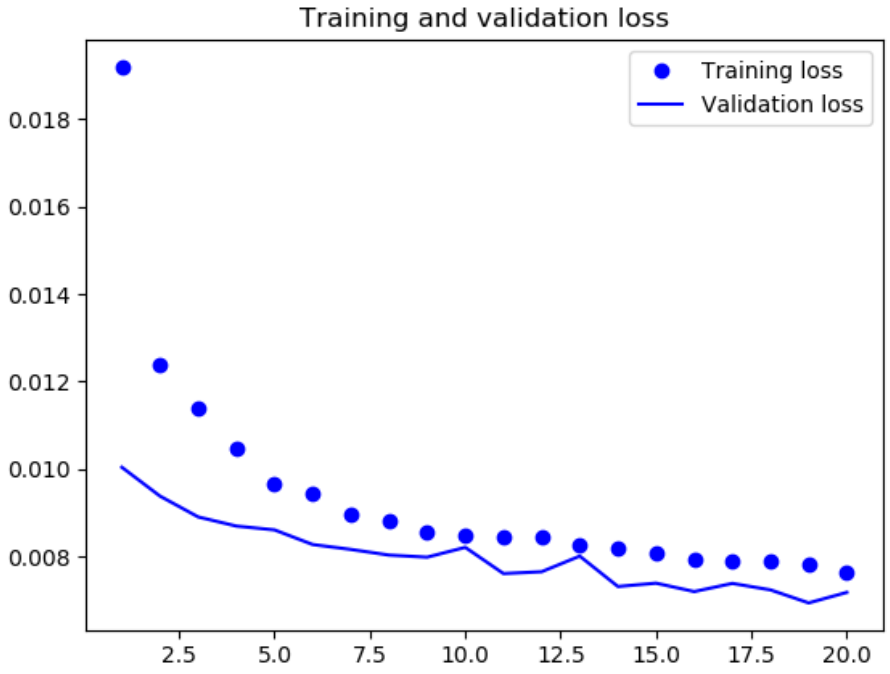

Figure 5.7: Hybrid (CNN and RNN) model link flow estimation training, testing losses.

5.2.4 Fully connected neural network (FCNN) based model

In order to evaluate the effectiveness of the presented models, a fully connected neural network architecture is used as a baseline, where all the neurons in all the layers fully interact with each other. The architecture of the model is chosen by a series of experiments. As a result, a three-layer fully connected NN has been obtained: two hidden layers and an output layer (Table 5.4). Since a fully connected network has no memory, it is needed to provide it with data for the last few timesteps at the same time. Here data for 10 timesteps are provided to the neural network as one feature vector. Therefore, the input vector has length 60.

The results of the experiments show that a simple neural network can solve the problem with an accuracy of 82 percent. The model is well trained, the training and the validation loss values are converging, but the FCNN cannot outperform the previous deep learning models (See Figure 5.8).

Table 5.4: FCSNN Model

Layer	**Number of neurons**
Input	60
First hidden	30
Second hidden	44
Output	4

5.2.5 Comparison of the given methods

Table 5.5: Comparison of the testing loss values of the proposed models

Model	**Min loss**	**Diff**
CNN	0.011	0.505
RNN	0.015	0.007
CNN + RNN	0.007	0.003
FCNN	0.430	0.09

All the models are implemented in Tensorflow [80] and are trained by minimizing the standard error with the Adam optimization method[81]. The batch size is 32 and the learning rate is 0.001. For evaluation, the standard deviation (MSE) is used.

Table 5.5 shows the comparison of different approaches for predicting the link flow rates 10 minutes ahead. The results show that the methods based on CNN networks work better than the fully-connected one, while the hybrid method is the best on taking into account the estimating metrics (See Table 5.5).

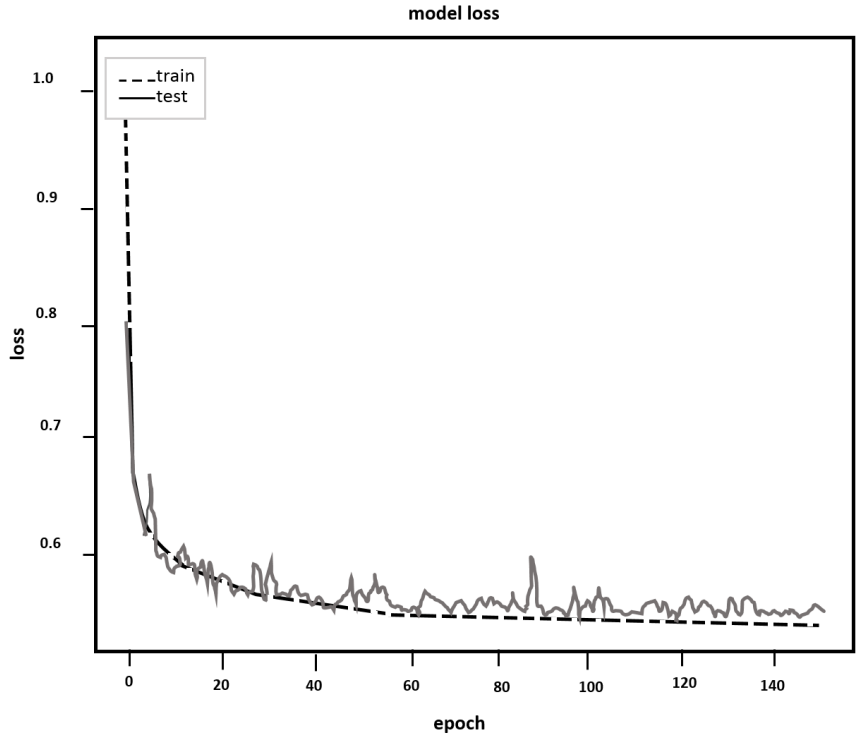

Figure 5.8: FCSNN link flow estimation training, testing loss

The situation in a transport network is changing rapidly and is hard to predict.

Therefore, the lookback is chosen to be pretty short. Several numerical experiments have been conducted to reveal the influence of the length of the prediction time interval.

According to the results, the models can forecast the link flow in the range from 1 to 30 minutes quite successfully. The validation loss values of the models are compared (Table 5.6). The longer prediction time yields the greater prediction error.

Table 5.6: Testing average loss values with different prediction times

Model	**5 min**	**10 min**	**15 min**	**30 min**
CNN	0.0094	0.0099	0.0104	0.0102
RNN	0.02639	0.0342	0.0343	0.0345
CNN + RNN	0.0108	0.0108	0.0109	0.0108

The number of tunable parameters in various models is presented in Table 5.7 (for the FCNN, in parentheses is the amount of neurons in each layer). Surely, the smaller this number is, the bigger the reduction in computations is.

In addition, the presented models have been launched on CPU and GPU in order to compare the training time. The results show that the CPU works twice faster than GPU (Table 5.8). This is due to the sequential computation in the GRU layer. The RNN requires sequential input to calculate the hidden layer weights iteratively. A

model waits for the hidden state value on the previous state to calculate the next value.

Table 5.7: Number of training parameters of models

Model	**Number of parameters**
RNN	186
CNN	939
RNN+CNN	1184
FCNN	10560 (2640)

The GPU cores well-doing computations in parallel, sequential computation cannot utilize fully their computing power. Moreover, while using GPU it is reasonable to increase the size of the batch in order to allocate the whole memory of GPU, while in CPU this will increase the training time of the model. Usually, the batch size is set to be big when data with high noise are used. In this study, it is assumed that data are noise-free and the small size of the batch is preferable. In addition, GPU is advantageous in deep learning when applied to images and texts, where the data are very rich (a many pixels = many variables) and the model has millions of parameters. The trained models are not complex and do not have a large size.

Table 5.8: Running time of models

type	**RNN:Avg 1epoch**	**CNN: Avg 1 epoch**	**CNN+RNN: Avg 1 epoch**
GPU	30.6s	15.5s	25.1s
CPU	14.7s	11.25s	16.7s

Intelligent control methods mostly have been designed for closed environments rather than for dynamically changing ones, as a transport network. If an agent is placed in such an environment then it has to relearn its decision making policy each time the link flow changes.

This chapter investigates the problem of TSC under nonstationary demand, which is approximated by a set of piecewise constant link flows.The objective of this study is to propose a link flow estimation system that can effectively operate in environments where the demand changes independently of the agents' actions and approximate link flow in near feature based on historical data. The TSC problem has been studied with unsteady flow and estimates the changes in the link flow over time.

The novelties of this approach are the following ones:

- The process of link flow estimating does not depend on the traffic control system;

- Novel models used for link flow estimating process, that did not used before in the given problem scope;
- The estimated link flows can be used to feed to the traffic control system.

Traffic data is considered as a time series. Three models based on RNN, CNN, and a hybrid one are presented. Models are trained on synthetic data, which is generated on a simulator. Since the estimation is done in one direction, these methods can be used in any architecture of the transport network. However, it is necessary to do forecasting several times. The results show that the given methods outperform the baseline. The obtained link flow estimations can be used in different traffic control algorithms to minimize the transport network congestion.

Chapter 6

ANN Traffic Signal Control using demand estimates

There are many Artificial Intelligence (AI) solutions to TSC. One of the most widely used AI techniques is Reinforcement Learning (RL) [19], which is an agent based learning. It became popular because the RL agents are able to learn on the fly in a trial-and-error fashion. Additionally, RL can be easily combined with other AI techniques such as, for instance, Deep Neural Networks (DNN) [29, 27]. On the other hand, a classical RL has a few serious disadvantages, which restrict its application in the context of TLC. Firstly, the agents are supposed to operate independently of one another and, secondly, the environment is assumed to be stationary [17, 18].

When the TSC problem is considered as MAS [17, 18], the most widely used techniques are Reinforcement Learning (RL), Deep Learning (DL), Fuzzy Logic (FL), Evolutionary Computations (EC), Swarm Intelligence (SI), and their various combinations.

Despite the many advantages of RL it has a few restricting disadvantages. First of all, it is mostly designed for closed environments rather than for dynamically changing ones. If an RL agent is placed in such an environment then it has to relearn its decision making policy each time the demand changes. For instance, Kurmankhojayev et al. [82] consider the problem of adaptive light control at a single isolated intersection. They report the queue length reduction under the stationary demand conditions, but the approach performs poorly, when the demand changes dynamically, irrespective of the choice of the parameters mentioned by Yau et al. [20].

Next, the RL agents are designed to operate independently of each other [17, 18]. They are bad at collaborative work because the actions of a single agent change the environment for others, which, in turn, violates the stationarity assumption. Additionally, if the agents work in collaboration then they have to share their states and coordinate actions. As a result the state-action space grows exponentially as the number of agents increases. An excellent explanation of this issue provided in [22]. On the other hand, the curse of dimensionality can be partially solved by the function approximation [27, 83, 84, 85]. Furthermore, this also allows the system to

incorporate different kinds of information in the state representation [83].

The objective of this study is to propose an Adaptive Traffic Light Control system that is able to effectively operate in environments where the demand changes independently of the agents' actions. The research is focused on two main questions:

1. Can we solve the problem of TSC using the MAS approach for a non-stationary environment?

2. Can the optimal signal plan be estimated for a given traffic network if we know the demand?

6.1 Related works

According to the surveys [17, 18, 20] many attempts were made to apply RL to the MAS control problem. Initially, the researchers propose to use independent agents. To be more specific, the actions of agents are chosen independently of one another based on their local information only. This leads to suboptimal policies. Later, different kinds of collaboration techniques are employed [22, 86, 51]. These are based on the concepts of Game Theory, i.e. stochastic staged games, where the agents share the state but choose their own actions in response to the opponent's action. This has a few limitations. Firstly, centralized system architectures cannot be used, since even a small number of agents makes the task computationally difficult due to the joint representation of states. Secondly, any agent's action changes the environment at least for its neighboring agents. For this reason, the majority of the considered MARL systems decentralize system architectures, i.e. the agents collaborate only with a neighborhood. As an alternative, swarm intelligence algorithms can be applied for improving the performance of the cooperative MARL [87, 88]. For example, the approach of El-Tantawy et al. [22] combines two modes of operations: independent mode and cooperative mode. The authors report the average delay decrease compared to the actuated and fixed controls. They also note that the cooperative mode outperforms the individual one.

Collaborative MARL systems allow us to slightly improve the control of a traffic network. However, as mentioned, these approaches still assume stationary environments. They will not work properly in the case of dynamic demands because they are forced to relearn their policies each time the demand changes. This happens because the state values or state-action values are forgotten. For instance, assume an agent follows an optimal policy for a given particular demand. If the demand changes and the agent continues to choose actions according to the new outdated policy then it gets punished by the system because the chosen action would not improve the state of the environment. In other words, the agent receives a negative reward. This leads to the policy change. The next time when the environment returns to the initial demand the agent would have a different policy and, hence, it would need to relearn the optimal policy again.

An intuitive step towards TSC under the non-stationary demand conditions is to let the system know when the demand change occurs. This concept requires the piecewise constant demands. In other words, the demand should have a number of intervals where it remains constant. Whenever the change is detected the system activates an appropriate model. However, there are only a few number of publications related to this issue [89, 90, 91, 86].

In [89, 90, 91] the authors design a non-stationary demand by a set of piecewise-constant demand submodels or so-named contexts [91]. The first two papers manually define the number of such demand sub-models, and the last one dynamically creates and assigns the appropriate context model. These models assume that

- the changes occur rarely
- the change of environment is independent of agents' actions
- the dynamic demand can be approximated by a set of piecewise-constant demand models

Additionally, the authors assume that the context cannot be directly obtained, but can be estimated according to the changes of the transition function and the reward function.

Our study is similar to [89, 90, 91] as it considers a finite number of demand sub-models too. However, the principal difference is that it explicitly separates the phase of context detection and the control. The main hypothesis is that if there is a simulation model with a known dynamic demand model (target value), we can train NN to detect changes in demand. As a consequence, we can consider this problem from the perspective of supervised learning. The training of NN is done offline.

After that, when the NN is trained it is ready for exploitation. It accepts the sensor data in real time and, based on it, estimates the demand, which is further fed to the pre-trained control model.

An approximation of the non-stationary demand model by piecewise-constant sub-models allows us to outline relatively recurrent traffic events such as morning, noon, and evening congestion peaks.

This study proposes to split the task into two parts: traffic pattern recognition and decision making. In other words, first, we extensively train a model to recognize different demands or traffic patterns; then use the estimated demand as an input to the multi-agent decision making system.

6.2 The proposed method

We explicitly split the demand estimation and traffic light control tasks. For traffic light control tasks we trained a fully connected Neural Network (NN). The synthetic data generated by the simulation model are used. The problem is addressed

as a supervised learning task. We designed a model to control the intersection based on the demand estimate.

6.2.1 Data collection

First, we collect the data. In order to do that we run the simulation with different parameters. Each simulation lasts for 50 000 seconds (simulation time units). We record the simulation states each 6 seconds. The tracked features are the following:

- number of vehicles in North-South direction
- number of vehicles in South-North direction
- number of vehicles in West-East direction
- number of vehicles in East-West direction
- average delay in in North-South direction
- average delay in South-North direction
- average delay in West-East direction
- average delay in East-West direction
- green phase duration in North-South direction (it is the same for South-North)
- green phase duration in West-East direction (it is the same for East-West)
- current phase index
- remaining time (in seconds) till the end of the phase

We choose between 256 demand models and 81 signal plans. It results in 20 736 simulations.

We do not need to annotate the data for the demand estimation task since we ourselves generated the demand models, hence, their parameters are known to us. The demand estimation model (NN) is trained on this data.

Second, we annotate the data for the traffic light control, i.e. for each demand model we find the best signal plan (among the given 81). In order to do that we compare the averaged queue lengths and delay times of all signal plans, and for each demand model choose the one that has the lowest queue and delay values.

As a consequence, each demand model has its own best (ground truth) signal plan. It results in 256 (X, y) records, where X is a 256×4 matrix and y is the corresponding best signal plan.

We may represent y as a 81 dimensional binary vector, which decodes the signal plan indices.

6.2.2 Demand models

Our demands are based on the second-based probabilities of vehicles being generated. For instance, if it is 0.1 in North-South direction then it means that the probability of a vehicle being generated in this direction each second is 0.1.

Table 6.1: Demand models in terms of simulation step based probability of vehicles being generated in specified directions

NS	**SN**	**WE**	**EW**
0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.2
0.1	0.1	0.2	0.1
...	...	...	...
0.4	0.4	0.4	0.4

6.2.3 Signal plans

We assume that the intersection is a two-phase intersection. We have limited ourselves only to controll the green phase duration in North-South and West-East directions, i.e. we fixed the duration of the amber phase.

Table 6.2: Defined list of signal plans. The numbers represent the durations of green phases in the specified directions in seconds

North-South and South-North	West-East and East-West
12	12
12	18
12	24
...	...
60	60

6.2.4 Traffic light control

Based on a demand estimate, i.e. the four-dimensional vector containing the probabilities of a vehicle being generated, we choose the signal plan by means of another offline-trained fully-connected feed-forward neural network. The output of this model is an index of the optimal signal plan in the list (see Table 6.2). Therefore, the problem is considered as a classification task.

Table 6.3: NN architecture

Layer	Number of neurons
First hidden	50
Second hidden	55
Output	81

The architecture of the model is chosen by the series of experiments starting from the simplest structure with one hidden layer and complicated as needed. For the calculation of the number of hidden neurons the following 'rule of thumb' is used as a starting point:

$$N_{hn} = \frac{(N_i + N_o)}{2} \tag{6.1}$$

where N_{hn} is a number of hidden neurons, N_i and N_o are the number of neurons in the input and output layers respectively. The chosen architecture is shown in Table 6.3. It has two hidden layers.

Experiments launched in the same environment as shown in the previous chapter.

6.3 Results and evaluations

The optimal values of the free parameters were manually selected and shown in Table 6.4.

Table 6.4: Hyperparameter values of NNs

Parameter	**Value**
Leaning rate	0.001
β_1	0.9
β_2	0.99
Epochs	150
Batch size	64
Activation function on hidden layers	ReLu
Activation function on output layer	softmax

This model highly overfits and gives the accuracy close to one (and the loss close to zero, correspondingly). It happens due to the small training set which contains only 256 entries.

By the design of the problem, there is not much we can do about this issue because the input of the previous step is a discrete and finite, i.e. there are only 256

possible combinations that could be fed to the model. On the other hand, that perfectly suits us because the model is able to learn the best possible signal plan (among defined signal plans) given the demand estimate.

This chapter investigates the problem of TSC under non-stationary demand, which is approximated by a set of piecewise constant demand models. The novelties of our approach are the following:

- the process of demand estimating is explicitly separated from decision making system
- the estimated demand is fed to the decision making system

There have only been a few methods related to this problem. Therefore, this work contributes to the general knowledge of the diverse TSC community. Firstly, the stationary demand assumption is relaxed; secondly, a new approach of transport network control is proposed. It is based on separation of the general process to the state detection and control sub-processes.

The bottleneck of this method resides in the process of data generation and annotation - the number of simulations needed for data generation polynomially grows with the increase of the number of the intersections. Since we need to run simulation for all possible combinations of demands, signal plans, this might prevent the current approach from being applied to transport networks with multiple intersections.

Chapter 7

Conclusion

The TSC task is one of the complex parts of the AI sphere. The given thesis considers current issues in TSC sphere such as development of model-free adaptive controllers, that are easily scalable for any network size.

In the given work, a model-free, online, off-policy reinforcement learning traffic control system is proposed. The model is training based on the temporal-difference learning with ANN. The main part of the RL is the reward function. For effective solving the problem a new reward formula was proposed. Several variations of the reward were implemented. Singularity of the given reward functions is that they consist of equilibrium and queue reduction terms. The adaptive controller gave near-optimal performance. The experiments were conducted on a uniform and mixed demand model.

In the given thesis despite many advantages of RL were identified a few restricting disadvantages in the scope of TSC problems. The result shows that the system based on the neural network has difficulties in a non-uniform environment.

Most intelligent control methods have been designed for closed environments rather than for dynamically changing ones, as a transport network. If an agent is placed in such an environment then it has to relearn its decision making policy each time the link flow changes.

In this study a link flow estimation system that can effectively operate in environments where the demand changes independently of the agents' actions and approximate link flow in near features based on historical data is proposed. The TSC problem has been studied with unsteady flow and estimations the changes in the link flow over time.

The novelties of this approach are the following ones: The process of link flow estimation does not depend on the traffic control system; Novel models used for link flow estimation process, that haven't been used before in the given problem scope; The estimated link flows can be used to feed the traffic control system. Traffic data is considered as a time series and three models based on RNN, CNN and a hybrid one are presented. These methods can be used in any architecture of the transport network. According to observed results, the given methods outperform the baseline. The obtained link flow estimations can be used in different traffic control algorithms

to minimize the transport network congestion. This work contributes to the general knowledge of the diverse TSC community. First, the stationary demand assumption is relaxed; second, a new approach of transport network control is proposed which is based on separation of the general process to the state detection and control sub-processes. All of the goals are achieved. The obtained results have a high theoretical and practical significance. The methods of developing and studying models used in this work can be useful for further study of issues on the given area. The given work is a significant contribution to intelligent TSC and can be used for further development in solving problems of adaptive traffic control.

Bibliography

[1] Chin Y., Bolong N., Kiring A.,Yang S., Teo K. Q-learning based traffic optimization in management of signal timing plan. International Journal of Simulation: Systems, Science and Technology, 2011. – Vol. 12, no. 3. – pp. 29–35.

[2] Cai C. Adaptive Traffic Signal Control Using Approximate Dynamic Programming/ Ph.D. thesis: University College London, 2010.

[3] Schouten S. Reinforcement Learning of Traffic Light Controllers under Partial Observability/ Ph.D. Thesis: Faculty of Science University of Amsterdam, The Netherlands, 2007.

[4] Wagner L. 1868-2019: a Brief History of Traffic Lights. https://www.inclusivecitymaker.com/1868-2019-a-brief-history-of-traffic-lights/

[5] Federal Highway Administration Research and Technology. Traffic Detector Handbook: Third Edition. 2006. – Vol.1, Chapter 2. https://www.fhwa.dot.gov/publications/research/operations/its/06108/02.cfm

[6] McKenney D. Distributed and adaptive traffic signal control within in realistic traffic simulation: MSc thesis: Computer Science; Carleton University, Ottawa, 2011 – p. 9.

[7] Reilly W. Highway Capacity Manual 2000. –Transportation Research Board, National Research Council, 2000.

[8] Webster F. V. Traffic Signal Settings: Road Research Technical Paper No. 39. // Department of Scientific and Industrial Research, Road Research Laboratory, UK – 1958, pp. 13-15.

[9] Roess R. P., Prassas E. S., McShane W. R. Traffic Engineering 4th edition. Upper Saddle River, N.J: Prentice Hall , 2011. – pp.527-528.

[10] Chandra R., Georgy C. InSyncAdaptice Traffic Signal Technology: Real-Time Artificial Intelligence Delivering Real-World Results, Lenexa, Kansas, 2012.–p. 13.

[11] Hunt, P. B. A Traffic Responsive Method of Coordinating Signals: Report No.LR1014: Transport and Road Research Laboratory, UK, – 1981.

[12] Sims A. G., Dobinson K. W. The Sydney Coordinated Adaptive Traffic (SCAT) System Philosophy and Benefits // IEEE Transactions on Vehicular Technology. – 1980.– Vol.VT-29, 2. – pp. 130-137.

[13] Abbas M. M. A Real Time Offset Transitioning Algorithm for Coordinating Traffic Signals/ Ph.D. Dissertation. Department of Civil Engineering, Purdue University, West Lafayette, Indiana, USA. – 2001.

[14] Robertson D. I., Bretherton R. D. Optimal Control of an Intersection for Any Known Sequence of Vehicle Arrivals // In Proceedings of the 2nd IFAC-IFIP-IFORS Symposium of Traffic Control and Transportation Systems. – 1974. – pp. 3-17.

[15] McKenney D. Distributed and adaptive traffic signal control within a realistic traffic simulation / MSc thesis. Carleton University, Ottawa, Canada. – 2011.

[16] Balaji P.G., Srinivasan D. Multi-Agent System in Urban Traffic Signal Control // Computational Intelligence Magazine. IEEE – 2010. – VOl. 5(4). – pp. 43-51.

[17] Bazzan, A. L. C. , Klügl,F. A review on agent-based technology for traffic and transportation // Knowledge Engineering Review. – 2014. – Vol. 29. No 3. – pp. 375-403.

[18] Bazzan L. C. Opportunities for multiagent systems and multiagent reinforcement learning in traffic control // Autonomous Agents and Multi-Agent Systems. – 2008. –Vol. 18. No 3. – pp. 342.

[19] Sutton R. S., Barto A. G. Reinforcement Learning: An Introduction. – London: The MIT Press , 2017.

[20] Yau K. A., Qadir J., Khoo H. L., Ling M. H., Komisarczuk P. A Survey on Reinforcement Learning Models and Algorithms for Traffic Signal Control // ACM Comput. Surv. – 2017. – pp. 34:1-34:38.

[21] Prabuchandran K. J., Hemanth Kumar A. N, Bhatnagar S. Decentralized learning for traffic signal control // 2015 7th International Conference on Communication Systems and Networks (COMSNETS). – 2015. – pp. 1-6. – doi=10.1109/COMSNETS.2015.7098712, ISSN=2155-2487.

[22] El-Tantawy S., Abdulhai B., Abdelgawad H. Multiagent 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. – 2013. – Vol. 14. No 3. – pp. 1140-1150.

[23] Medina J. C., Benekohal R. F. Traffic signal control using reinforcement learning and the max-plus algorithm as a coordinating strategy // IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC. – 2012. – pp. 596-601.

[24] Wiering M. Multi-agent reinforcement learning for traffic light control // In Proceedings of the 17th ICML. – 2000. – pp. 1151-1158.

[25] Li C., Wang M, Sun Z., Lin F., Zhang Z. Urban traffic signal learning control using fuzzy actor-critic methods // 5th International Conference on Natural Computation, ICNC 2009. – 2009. – vol.1. – pp. 368-372.

[26] Jin J, Ma X. Adaptive Group-Based Signal Control Using Reinforcement Learning with Eligibility Traces // IEEE Conference on Intelligent Transportation Systems Proceedings ITSC. – 2015. – pp. 2412-2417.

[27] Prashanth L. A., Bhatnagar S. Reinforcement learning with function approximation for traffic signal control // IEEE Transactions on Intelligent Transportation Systems. – 2011. – Vol. 12, N2.– pp. 412-421.

[28] Chedjou J.C., Kyamakya K. Cellular neural networks based on local traffic signals control at a junction/intersection // Proceedings of the 1st IFAC Conference on Embedded Systems 2012. – 2012. – pp. 81—85.

[29] Araghi S., Khosravi A., Johnstone M., Chreighton D. Intelligent Traffic Light Control of Isolated Intersections using Machine Learning Methods // IEEE International Conference on Systems. Man and Cybernetics. – 2013. – pp.3621-3622.

[30] Castro G. B., Martini J. C., Hirakawa A. Biologically-inspired neural network for traffic signal control // Proceedings of 17th International Conference on Intelligent Transportation Systems 2014 (ITSC). – 2014. – pp. 2144—2149.

[31] Spall J. C., Chin D. C. Traffic-responsive signal timing for system-wide traffic Control // Transportation Research Part C: Emerging Technologies. – 1997. – Vol. 5. no. 3–4. – pp. 153–163.

[32] Genders W., Razavi S. Using a deep reinforcement learning agent for traffic signal control. – 2016. – e-print https://arxiv.org/abs/1611.01142.

[33] Choy M. C., Srinivasan D., Cheu R. L., Cooperative, hybrid agent architecture for real-time traffic signal control // IEEE Transactions on Systems. Man, and Cybernetics - Part A: Systems and Humans. – 2003. – Vol. 33. no. 5. – pp. 597–607.

[34]] Wei W., Zhang Y., Mbede J., Zhang Z., Song J, Traffic signal control using fuzzy logic and moga // Proceedings of the IEEE International Conference on Systems. Man and Cybernetics. – 2001. – Vol. 2. – pp. 1335–1340.

[35] Zeng R., Li G., Lin L. Adaptive Traffic Signals Control by Using Fuzzy Logic // International Conference on Innovative Computing. Information and Control. – 2007. – pp. 527–527.

[36] Favilla J., Machion A., Gomide F. Fuzzy traffic control: adaptive strategies // IEEE International Conference on Fuzzy Systems. – 1993.– pp. 506–511.

[37] Zhang L., Li H., Prevedouros P. Signal Control for Oversaturated Intersections Using Fuzzy Logic // Transportation and Development Innovative Best Practices 2008.– American Society of Civil Engineers. – 2008. – pp. 179–184.

[38] Zhang W.- B., Wu B.-Z., Liu W.-J. Anti-Congestion Fuzzy Algorithm for Traffic Control of a Class of Traffic Networks // IEEE International Conference on Granular Computing. – 2007. – pp. 124–124.

[39] Wenchen Y., Lun Z., Zhaocheng H., Lijian Z. Optimized two-stage fuzzy control for urban traffic signals at isolated intersection and Paramics simulation // IEEE Conference on Intelligent Transportation Systems. – 2012. – pp. 391–396.

[40] Chiou Y.- C., Huang, Y.-F. Stepwise genetic fuzzy logic signal control under mixed traffic conditions // Journal of Advanced Transportation. – 2013. – Vol. 47, no. 1. – pp. 43–60.

[41] Krajzewicz D., Erdmann J., Behrisch M., Bieker L., Recent Development and Applications of SUMO - Simulation of Urban MObility // International Journal On Advances in Systems and Measurements. – 2012. – Vol. 5. – pp.128–138.

[42] Warren S., Neural Network and Statistical Jargon. – 1996. ftp://ftp.sas.com/pub/neural/jargon.txt¿

[43] S.Haikyn, Neural Networks and Learning Machines. – Third edition. Ontario: Prentice Hall. – 2009.

[44] Cichocki A., Unbehauen R., Neural Networks for Optimization and Signal Processing.– John Wiley and Sons. – 1993.

[45] Bre F., Gimenez J., Fachinotti V, Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks // Energy and Buildings., 158. – 2018. – pp. 1429-1441.

[46] Warren S., Neural Networks and Statistical Models // Proceedings of the 19th Annual SAS Users Group International Conference. – 1994. – pp. 1-13. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.209.672&rep=rep1 &type=pdf

[47] Rumelhart D.E., Hinton G. E., Williams R.J., Learning Internal Representations by Error Propagation. In: David E. Rumelhart, James L. McClelland (eds.): Paral lel Distributed Processing. Explorations in the Microstructure of Cognition I. MIT Press. – 1986.

[48] Wiering M. Multi-agent reinforcement learning for traffic light control // In Proceedings of the 17th ICML. – 2000. – pp. 1151-1158.

[49] Abdoos M., Mozayani N., Bazzan A. L. C. Traffic light control in non-stationary environments based on multi agent Q-learning // 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC). – 2011. – pp. 1580-1585.

[50] Araghi S.,Khosravi A., Johnstone M., Creighton D. Q-learning method for controlling traffic signal phase time in a single intersection // 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013). – 2013. – pp. 1261-1265.

[51] Prabuchandran K. J., Hemanth Kumar A. N., Bhatnagar S. Multi-agent reinforcement learning for traffic signal control // 17th International IEEE Conference on Intelligent Transportation Systems (ITSC). – 2014. – pp. 2529-2534.

[52] Arel I., Liu C., Urbanik T., Kohls A. G. Reinforcement learning-based multi-agent system for network traffic signal control // IET Intelligent Transport Systems.– 2010.– pp. 128-135.

[53] Yaping W., Zheng Zh. A method of Reinforcement Learning Based Automatic Traffic Signal Control // In: Third International Conference on Measuring Technology and Mechatronics Automation. – 2011. – pp.119-122.

[54] Li C. G., Yan X. l., Lin F. Y., Zhang H.I. Multi-intersections traffic signal intelligent control using collaborative q-learning algorithm // 2011 Seventh International Conference on Natural Computation. – 2011. – pp. 185-188.

[55] Gaikwad V. V., Kadarkar S. S., Kasbekar G. S. Intelligent Traffic Signal Duration Adaptation Using Q-Learning with an Evolving State Space // 2016 IEEE 84th Vehicular Technology Conference (VTC-Fall).– 2016. – pp. 1-6. doi=10.1109/VTCFall.2016.7881050

[56] RosyadiA. R., Wirayuda T. A. B. . Al-Faraby S. Intelligent traffic light control using collaborative Q-Learning algorithms // 2016 4th International Conference on Information and Communication Technology (ICoICT). – 2016. – pp. 1-6.

[57] El Hatri C., Boumhidi J. Q-learning based intelligent multi-objective particle swarm optimization of light control for traffic urban congestion management // 2016 4th IEEE International Colloquium on Information Science and Technology (CiSt).– 2016. – pp. 794-799.

[58] Tahifa M., Boumhidi J.,Yahyaouy A. Swarm reinforcement learning for traffic signal control based on cooperative multi-agent framework //2015 Intelligent Systems and Computer Vision (ISCV). – 2015. – pp. 1-6.

[59] Watkins C. J. Learning from Delayed Rewards. – 1989. – Thesis (Ph. D.) .– King's College, Cambridge.

[60] Kurmankhojayev D., Tolebi G., Suleymenov N. Online model-free adaptive traffic signal controller for an isolated intersection // IEEE International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON). – 2017. – pp. 109-112.

[61] Tolebi G., Dairbekov N.S., Kurmankhojayev D. Link flow estimation on an isolated intersection based on deep learning models // International Review of Automatic Control (I.RE.A.CO.). – 2020. – Vol. 13. N.1.– pp. 19-26

[62] Box George E. P., Jenkins M., Reinsel Gregory C., Ljung G.M. Time Series Analysis: Forecasting and Control. – Wiley Series in Probability and Statistics, 2015.

[63] Brownlee J. Comparing Classical and Machine Learning Algorithms for Time Series Forecasting. // Lecture notes. Deep Learning for time series .Machine Learning Mastery Pty. Ltd. – Australia. – 2018.

[64] Song J. Mathematical modelling and simulations of traffic flow // Ph.D. dissertation. Dept. Math., University of Michigan, Ann Arbor, MI . – 2019.

[65] Keerthika C., Greeshma N., Vyshnavi P., Reddy K. K., Indhira K., Chandrasekaran V. M. Mathematical model for traffic flow // International Journal of Engineering and Technology. – 2018. – Vol. 7 (No 4.10, Issue 10). – pp. 940-941.

[66] Nubert J., Truong N.G., Lim A., Tanujaya H.I., Lim L., Vu A.V., Traffic Density Estimation using a Convolutional Neural Network // Machine Learning Project. National University of Singapore. – 2018.

[67] Du S., Li T., Gong X., Horng S. A hybrid method for traffic flow forecasting using multimodal deep learning. –2018. – e-print https://arxiv.org/abs/1803.02099. 01.12.20.

[68] Hasnat A., Rahman F. I. Traffic flow prediction performance comparison between ARIMA and Monte Carlo simulation // Transport and Logistics the International Journal. – 2019. – Vol. 19 (No 46). – pp. 12-21.

[69] Mousavizadeh Kashi S. O., Akbarzadeh M., A framework for short-term traffic flow forecasting using the combination of wavelet transformation and artificial

neural networks // Journal of Intelligent Transportation Systems. – 2019.– Vol. 23 (no. 2). – pp. 60-71.

[70] Luo X, Li D., Zhang S. Traffic flow prediction during the holidays based on DFT and SVR //Journal of Sensors. – 2019. – Vol. 2019 (Special Issue: Sensing and Data-Driven Control for Smart Building and Smart City Systems). – Article ID 6461450.– 10 pages. doi: https://doi.org/10.1155/2019/6461450

[71] Chai J., Li A., Deep Learning in Natural Language Processing: A-state-of-the-art Survey // 2019 International Conference on Machine Learning and Cybernetics (ICMLC). – 2019.– pp. 1-6. Kobe, Japan.

[72] Al-Safar A. M., Tao H., Talab M. A., Review of deep convolutional neural networks in image classification // 2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET). – 2017. – pp. 26-31.– Jakarta, Indonesia.

[73] Polson N. G., Sokolov V. Deep learning for short-term traffic flow prediction //Transportation Research Part C Emerging Technologies. – 2017. – Vol. 79. – pp. 1-17.

[74] Zheng Z., Yang Y. , Liu J., Dai H-N., Zhang Y., Deep and Embedded Learning Approach for Traffic Flow Prediction in Urban Informatics // IEEE Transactions on Intelligent Transportation Systems. – 2019. – Vol. 20 (Issue 10). – pp. 3927-3939.

[75] Li Y., Yu R., Shahabi C. et al. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecating. – 2017. – pp. 1–12.// e-print https://arxiv.org/abs/1707.01926. 01.12.20.

[76] Kurmankhojayev D., Tolebi G., Dairbekov N.S. Road Traffic Demand estimation and Traffic Signal control // International Conference on Engineering and MIS. – 2019. – Article No.:2. – ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3330431.3330433

[77] Hochreiter S., Schmidhuber J. Long-short term memory // Neural Computation. – 1997. – Vol. 9(8). – pp. 1735-1780.

[78] Cho K., Bart V. M., Gulcehre C., Bahdanau D., Bougares F., Holger H., Bengio Y. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. – 2014. – e-print https://arxiv.org/abs/1406.1078. 01.12.20

[79] Chollet F. Deep Learning with Python. Version 6. – Manning Publications, 2017.– pp. 212-213.

[80] Abadi M., et al., TensorFlow Large-Scale Machine Learning on Heterogeneous Distributed Systems // Google Research. – 2015. – e-print https://arxiv.org/abs/1603.04467. 01.12.20.

[81] Kingma P., Ba J. Adam: A method for stochastic optimization // The 3rd International Conference for Learning Representations. – 2015. – pp. 1-15. / e-print https://arxiv.org/pdf/1412.6980. 01.12.20.

[82] Kurmankhojayev D., Suleymenov N., Tolebi G. Online model-free adaptive traffic signal controller or an isolated intersection // 2017 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON). – 2017. – pp. 109-112.

[83] Wei H., Yao H., Zheng G., Li Z. IntelliLight: A reinforcement learning approach for intelligent traffic light control // Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. – 2018. – pp. 2496-2505.

[84] Genders W., Razavi S. Evaluating reinforcement learning state representations for adaptive traffic signal control // Procedia Computer Science. – 2018.– pp. 26-33.

[85] Liang X., Du X., Wang G., Han Z. Deep Reinforcement Learning for Traffic Light Control in Vehicular Network // IEEE transactions on vehicular technology (eprint). – 2018. – https://arxiv.org/pdf/1803.11115.pdf. 20.11.20.

[86] Abdoos M., Mozayani N., Bazzan Ana.L.C. Traffic Light Control in Non-stationary Environments based on Multi Agent Q-learning //14th International IEEE Conference on Intelligent Transportation Systems. – 2011. – pp.1580-1585.

[87] Tahifa M., Boumhidi J., Yahyaouy A. Swarm reinforcement learning for traffic signal control based on cooperative multi-agent framework // 2015 Intelligent Systems and Computer Vision (ISCV). – 2015. – pp. 1-6. doi=10.1109/ISACV.2015.7105536.

[88] El Hatri C. , Boumhidi J. Q-learning based intelligent multi-objective particle swarm optimization of light control for traffic urban congestion management // 4th IEEE International Colloquium on Information Science and Technology (CiSt). – 2016. – pp. 794-799. doi=10.1109/CIST.2016.7804996.

[89] Choi S. P. M., Yeung D., Zhang N. L. An environment model for nonstationary reinforcement learning // Advances in Neural Information Processing Systems. – 2000.– pp. 307-313.

[90] Doya K., Samejima K., Katagiri K., Kawato M. Multiple model-based reinforcement learning // Neural computation.– 2002. – Vol. 14, N6.– pp. 1347-1369.

[91] Da Silva B. C., Basso E. W., Bazzan A. L. C., Engel P. M. Dealing with non-stationary environments using context detection // ACM International Conference Proceeding Series. – 2002. – Vol. 148. – pp. 217-224.