Simulation of road traffic flow in Hangzhou

Bachelor's Thesis

by

Anders Holmberg and Louise Brandt

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

LUND UNIVERSITY

2022

Abstract

Is it possible to increase traffic flows and decrease stand stills in rush hour traffic by tweaking existing traffic light signals? A vehicle's engine running on idle is not useful in physical terms and emits unnecessary exhausts. By investigating and observing signal phase times on site in Hangzhou, China, an effort was made to find out if it was possible to get better flow and thus lower emissions by changing the traffic signal programs. The Simulation of Urban MObility suite developed by the German Aerospace center, DLR, was used to simulate traffic. Three scenarios were simulated. Scenario one tried to find better and smoother flow by reducing the duration of the traffic signals' phase times. Scenario two allowed both extended and reduced signal phases. Scenario three was based on already existing signal phase times. The findings will show that it is possible to obtain increased throughput of vehicles if some green signal phases and thus cycle times are shortened. This translates as shorter time spent in congested traffic, which results in lower emissions per vehicle.

Acknowledgments

Swedish International Development Cooperation Agency (Sida)

Sida granted us a US$ 3,000 scholarship each for our Minor Field Study. It made the research feasible. This was a great help for both of us.

Lund University, Lund, Sweden (LU)

Professor Charlotta Jonsson got us in crucial contact with her Chinese counterpart Professor Hongye SU. The institution of Automatic Control provided us with gifts to all helpful friends at Yuquan Campus, Zhejiang University, Hangzhou, China. Our supervisors, Associate Professor Christian Nyberg and Professor Karl-Erik Årzen have dedicated us their time and shown a lot of patience. We are very grateful for the efforts showed by these individuals.

Zhejiang University, Hangzhou, China (ZJU)

Professor Hongye SU, Institute of Cyber-Systems and Control, ZJU, made the official and very important invitation for studies at ZJU. This was crucial in order to obtain Chinese entry visas for our 10 weeks stay.

Postgraduate Hanshan SHAO, was appointed by Professor SU as our contact person at ZJU. Hanshan skillfully took care of practical matters like documents for visas and general advice for our stay on campus. He also presented the CSC department and parts of Yuquan campus to us upon arrival. Hanshan is seen standing between the authors of this report in Figure 0.1.

'George' Xiaohang XU, Office of Admission Affairs, International College, ZJU, helped us with lodging during our stay on Yuquan campus. George has been a contact person for the China specialization option for civil engineering students at LU. Three mandatory courses were earlier held at ZJU. George is experienced and very helpful. He offered to assist with extending ours visas, should we want to travel abroad during Chinese New Year break.

In general, we thank the good and striving people at Yuquan Campus, ZJU, who were kind enough to host us for 10 weeks around the turn of the years 2018 and 2019.

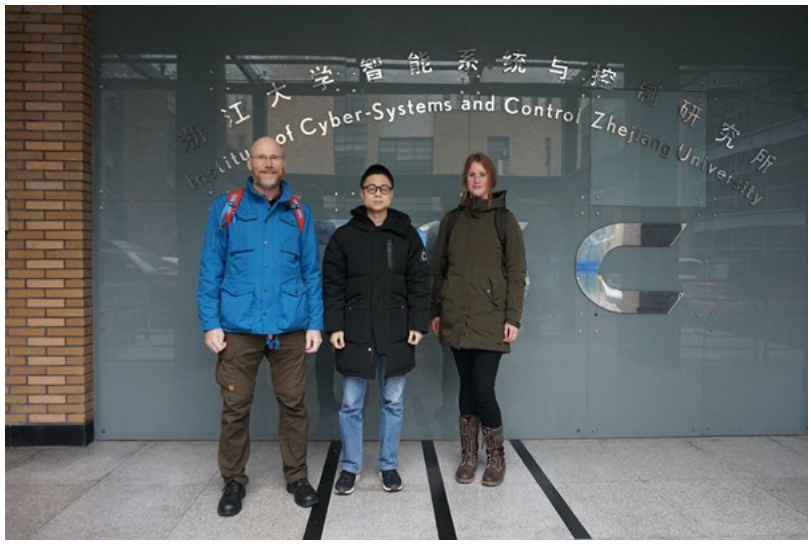


Figure 0.1. The authors flanking Hanshan Shao (汉山).

Anders Holmberg (安德斯)

Louise Brandt (路易斯)

Figure 0.2. The Logotype of the City of Hangzhou, Zhejiang, China.

The Hangzhou logotype shown in Figure 0.2 was widely seen on public transportation and across the city in 2018 and 2019.

Contents

List of figures

List of tables

Acronyms and terminology

100_33	simulation scenario with 100 to 33 % of sampled phase times
50_150	simulation scenario with phase times in range 50-150 %
cycle time	time period for all phases to occur once in a TLI
edge	network edge representing a piece of road
gui	graphical user interface
IRL	simulation scenario representing real (100 %) phase times
MATLAB	high performance technical computing application
Netedit	gui network editor bundled with SUMO
phase time	time period for one and the same signal (color)
Python	high level open source programming language
S1, S3, …	name of signal-controlled intersection 1, 3, …
SUMO	Simulation of Urban Mobility
sumo	terminal application with or without gui
TLI	traffic light intersection
TLS	traffic light signal
TRVMB	Trafikverkets metodbeskrivning för beräkning av kapacitet och framkomlighetseffekter i vägtrafikanläggningar (Swedish regulation)
XML	Extensible Markup Language
ZJU	Zhejiang University, Hangzhou, Zhejiang, China
Ø1, Ø2, Ø3	pedestrian crossing 1, 2 and 3

1 Introduction

Large cities across the world suffer from rush hour traffic, meaning streets fully saturated where cars are mostly standing still in a bumper-to-bumper position. In a vehicle without engine auto-stop[1] drivers do not tend to kill the engine while at a full stop, even if it lasts several minutes, resulting in a wasteful idle time for the car's engine. This reasoning is based on fossil fuel driven vehicles.

Unnecessary combustion outlets, whether fully combusted or not, worsens the negative impact that cars have on human health and on our climate. The question is if it is possible to create smoother traffic flow, thus reducing idle running time, by using a city's already existing fixed-time traffic signal infrastructure?

1.1 Background

This project is an initiative of two students who on different occasions took part in a course about queuing systems.

Much research has been done about traffic congestion since cities are growing all over the world. There exist clear frameworks on how to best build road networks and how regulating traffic should be done. In Sweden road constructors use TRVMB [1]. One of these traffic signal regulations is the "green wave", which simply gives green light to traffic on a certain route utilizing interconnected traffic signals that change according to the approach of vehicles. Thus, making that route a priority at its consecutive intersections.

There exist such things as "intelligent" traffic signal systems, like the Adaptive Traffic Control System (ATCS) [2], which is a traffic management strategy in which traffic signal timing changes, or adapts, based on actual traffic demand. In the USA systems like the ATCS have been deployed on less than 1% of the country's existing traffic signals.

It is fair to say that most or a great deal of existing traffic signals usually function according to local information.

It is not known to the authors if it is possible to, either via time settings or via a remote device, impose control on the light phases of ordinary traffic signals. Such deployment is beyond the scope of this work, but it is most likely very accomplishable. Manually setting traffic signals to operate

[1] Automatically stops the engine from idling when the vehicle is stationary.

differently around the clock, would however be at a relatively low or no cost at all - assuming the traffic signals in an intersection have timing devices for different scenarios.

By applying queuing theory on a rush hour traffic scenario, we want to find out if there exist realistic light phase settings that will create a nearly constant flow of traffic in a system of signal regulated intersections. For this we assume that traffic signals can be set to act according to a certain phase pattern either by a manually preset time setting or via remote control. Should we find this to be true, then the next question is if such a queue system is scalable to operate on a complete city.

Is it possible to control the traffic flow on a large number of streets merely by the use of traffic signals that operate with respect to local data only? Well, in one way this is what actually happens on an even larger scale inside the world wide web, where routers and servers act as traffic signals. These routers and servers by default know nothing more than which is its closest neighboring server or router. Still the web functions well.

1.2 Purpose

The overall purpose is to find a simple way to reduce unnecessary emissions by using infrastructure already in place, namely the traffic signals. By creating a model that corresponds to a geographical part of Hangzhou's road network, the aim is to simulate different scenarios for that specific area. As previously mentioned there already exist some traffic signal regulators that are supposed to reduce traffic congestions. The aim for this thesis is to see whether it is possible to reduce the idle waiting time for the chosen area by simulating different traffic light cycle times and compare the result to the actual traffic signal time phases that are being used. If the theory proves to be successful, this can help reduce the idle time for thousands of fossil driven vehicles and reduce the emissions from incomplete combustions, besides the obvious reduced time spent in urban traffic congestions. In a global perspective it is known that fossil fueled vehicles have a negative impact on the climate and that they also form serious health hazards. Therefore, the final goal and what fuels the authors is to reduce this negative impact.

In this project the open-source traffic simulation program SUMO [3] is used to build scenarios according to the traffic observations made at one of the city's areas which regularly has traffic congestion. To produce a basis for simulation, one must export geographical data from OpenStreetMap [4] and run a few premade Python scripts on it. In short, the processing is done by applying some logical rules. Depending on how detailed the maps are, the critical data needs to be corrected manually. In this case this required a visit

to the geographical location to collect data on signal phase times, speed limits, amount and types of vehicles and other local parameters that are not shown on OpenStreetMap. In this aspect it was crucial to detect which routes are used by what type of vehicles. The simulation itself is microscopic which means each vehicle and its dynamics are modeled individually. The simulation itself can be visualized on a very detailed graphical user interface.

1.3 Goal

The goal is to see if the traffic flow for an area in Hangzhou can be optimized by changing the traffic signal programs. The results will be obtained by comparing calculated vehicles for the different scenarios to the currently existing traffic signal programs.

1.4 Problem

There may exist many ways to try to increase traffic flows. Here we deemed it reasonable to look at the existing traffic signal programs and try to optimize it. Hence, we created an algorithm to calculate two traffic signal programs and compare their results with the existing program to answer the following questions.

- Is it possible to increase traffic flow through the system?
- Is it possible to reduce idling time for traffic in this area?
- Will the algorithm come up with a better signal-regulation solution for optimizing the flow of traffic than the current traffic signal program?

1.5 Motivation of thesis

After attending a course on queueing systems, we saw a resemblance with cars queuing at traffic light regulated road networks. This gave us an idea to try to alter traffic light phases in order to get optimal flow.

Much research has previously been done on traffic congestion and usage of traffic lights. See Section 1.7. *Previous work on similar subjects*, for more details. This thesis is based on a case study and usage of SUMO simulations to test how these calculated scenarios would work in a real-life environment. This differs from the previous thesis that has been reviewed.

1.6 Demarcation

The theoretical model is restricted to the six largest intersections, shown in Figure 1.1 of the area because they are interconnected as well as having entries and exits to the system. In the SUMO model all intersections are implemented, i.e., 10 working traffic light intersections (TLI) as well as the 3 separate pedestrian crossings and all other non-controlled intersections. Thus 6 intersections are modelled with 14 internal directions and 9 roads with bidirectional traffic going into and out of the system. This is a rough estimation, because there are at least double the number of entries and exits to and from this area.

Note that there may exist more lanes along the streets than is the case with their entrances. This is ignored in the theoretical model. Restrictions for the simulation model were made to omit smaller streets within residential areas.

Figure 1.1 The six TLIs in the reduced model.

1.7 Previous work on similar subjects

One of the most fundamental papers regarding traffic signals was written by F.V Webster back in 1957 [5]. It brought up the topic of how to determine the delay at traffic signals as well as how to determine the green time. The article is based on calculations, observations, and simulations for three intersections in London. It clearly describes how to perform calculations to reduce the delay and how to adjust the calculations based on the situation that may occur before the crossing. The calculations are based on actual flow, saturated flow and time cycles for the traffic signal system. With the collected data they came up with a formula nowadays being referred to as Webster's minimum delay optimal cycle length formula. The paper and the approaches used in this paper is still very up to date.

The Swedish traffic authority, Trafikverket [1] published a document in 2013 that is supposed to be used as a guide for traffic planning in Sweden. It is based on earlier published research and is performed to describe and help solve capacity and accessibility issues in the road network. This document goes into detail how to calculate the minimum green time, the cycle time of the traffic signal, how to calculate the minimum crossing time for pedestrians and so forth. Webster's formula is also referred to here when calculating the cycle time. It is of interest to this case study as a means of comparison when viewing the simulation results.

There have been case studies carried out in both Florida, United States and in Beijing, China that have studied similar questions as us. Their main questions being similar to ours but their approaches being different. Simulation study of mixed traffic in China- a practice in Beijing [6] focused on how to use simulation software, Vissim, to create a realistic traffic model and then use optimized traffic signals as well as changing the original lane turning directions to increase the traffic flow in the model to reduce the travel time.

The case study in Florida, Modeling Signalized Intersection Using Queuing Theory [7], focused on two approaches. One using queue theory to try to increase the throughput of the system by increasing the green time and by that also decreasing the waiting time for the traffic. The other being the effect of increasing the number of lanes. This study had more of a theoretical approach without the use of simulations.

When doing research for what previous work has been done on traffic signal systems, we noticed that most of what we came across were for series connected road networks. The location that this case study is carried out in is more complex with a parallel road network. The choice of simulation tool is also different compared to the more well-known and older TRANSYT and PTV Vissim that the two case studies above have used. PTV Vissim is a microscopic simulation software, while TRANSYT is a macroscopic simulation software, where the traffic flow is the basic entity.

2 Technical background and methodology

After a course in Queuing Systems, an idea arose of resolving traffic congestions by changing the time phases of traffic signals within such an area. This idea came from looking at the phenomenon as a deterministic queueing network, where an intersection is a server node and time with green light is related to the service time.

Reading earlier studies on traffic congestion revealed that it was a research area at the Department of Automatic Control at Lund Institute of Technology. One person at the department offered himself to become supervisor of this thesis, but since the work would interfere with his PhD thesis work, he withdrew his offer. He did however present some valuable angles of approach and suggested the SUMO simulation software to be a good choice of tool.

2.1 Thesis fundamentals

The authors have identified the following buildings blocks as fundamental in order to compile this thesis.

2.1.1 Sampling of real TLS phase times

The collection of traffic signal phase times on location in Hangzhou, China, is further described in section 2.3 *Field work (data gathering)*. The compilation and analysis of the sampled data is described in Section 2.4 *Data analysis.*

2.1.2 Calculate optimal traffic flow

Section 2.5 *Optimizing TLS phase times*, explains the approach how to find a balance between resolving a traffic jam through evacuation of vehicles on streets and still having a comparatively steady flow among regulated streets.

2.1.3 Create a simulation model

Section 2.6 *Simulation with SUMO*, briefly explains how to build a network model and how to simulate road traffic by using SUMO simulation software.

2.1.4 Simulation of road traffic flow

To control the simulations, detect values and compile data Python code has been used, which is briefly explained in Section 2.7 *TraCI.*

2.1.5 Analysis, results, discussion and conclusion

The simulations revealed a few things worth mentioning, which are covered in the Sections 3 *Analysis*, followed by the compiled results in Section 4 *Results*. An elaboration based on the results is made in Section 0 *Discussion*. Finally, the answers to the original questions from Section 1.4. *Problem* is presented and further elaborated in Section 1 *Conclusion.*

2.2 Preparations

During the fall semester of 2012 the authors were students at Zhejiang University in Hangzhou, China. It came naturally to try to revisit the city and apply queueing theory there. A 10-week stay in Hangzhou during the time period December of 2018 - February of 2019 was planned.

Professor Charlotta Johnsson is once more acknowledged for assisting in getting contact with and finding the important reference person needed in order to register as students at ZJU. Back in 2012 Professor Johnsson was collaborating with ZJU on location in Hangzhou and participated on an information meeting at the beginning of the autumn semester. Professor Johnsson helped us by contacting her Chinese colleague, Professor Hongye Su, who appointed one of his postgraduates, Hanshan Shao, to be our contact person. Mr Shao helped with preparing lodging inside ZJU:s Yuquan campus. Professor Hongye issued an official university invitation in order to obtain student visas. Luckily some vacant rooms on Yuquan Campus were found for us, due to other foreign students leaving campus at the end of each year.

Before leaving for China and to get an idea of necessary equipment and work approach, three consecutive days (including a hotel stay) were spent around Södervärn in Malmö.

A study in China requires time and money, to say the least. Therefore, two applications were made for a minor field study scholarship (MFS). They were both granted, which provided a welcomed financial help. An MFS scholarship imposes further restrictions such as a mandatory 3-day course at Sida[2] in Härnösand. The preparation courses were attended in November

[2] Swedish International Development Cooperation Agency.

2018, each on two different occasions. After meeting at Shanghai airport on November 15, the authors continued by train to Hangzhou.

2.3 Field work (data gathering)

Before departure to China a shared Google Document folder was prepared. Therefore, a working Internet access was a key to this field work. But the solution including Google had to be abandoned in favor of Microsoft OneNote, Microsoft OneDrive and Microsoft Bing for online search. Google worked very poorly inside China.

2.3.1 Start up

"George" Xiaohang XU[3] at the International College of ZJU gave the advice to buy student SIM-cards for an unlimited data traffic plan within the campus limits. This way the mobile phones can provide Internet access by sharing Wi-Fi to a laptop. The mobile phones were thus crucial instruments and without them the task would most likely be much harder.

2.3.2 Course of action

During the first couple of weeks, the data gathering consisted of sketches, general notes of findings and notes from measuring the time length of various traffic signals. The mobile phone's built-in stopwatch was used. Figure 2.1 and Figure 2.2 show examples of two of almost 50 sampling perspectives.

[3] Xiaohang XU was the contact person when Lund University held courses on ZJU Yuquan Campus.

Figure 2.1 View from a recording of the top western intersection S1.

It was often rainy and cold which made it a bit awkward to be accurate. Consequently, all intersections were later video recorded, still making use of our mobile phones. This way the data collection became more accurate and could be reviewed repeatedly. For the sake of the simulations a detailed map was needed. Therefore, parallel with data gathering a map for simulation was created manually from scratch.

Figure 2.2 View from a recording of the top eastern intersection S5.

2.4 Data analysis

From the recorded videos it was possible to acquire very accurate time measures for each signal phase from every traffic light intersection that were

documented. Many data tables were created from these details. For example, table 2.1 that shows a data table over the traffic light program at intersection S1.

Video	@ (s)	Ns	PxE	←/←	↑/↑	→/→	PxE/PxE
0:25	-3	← ◉ →	◉	35/0	71/0	-3/34	74/0
0:28	0	← ◉ →	◉	38/0	74/0	0/0	77/0
1:26	58	← 9◉ →	◉	96/0	132/0	58/0	135/0
1:35	67	← ◉ →	◉	105/0	141/0	67/0	144/0
1:45	77	← ◉	◉	115/0	0/10	77/0	0/10
2:02	94	← 9◉	9◉	132/0	0/27	0/17	0/27
2:05	97	9← 6◉	6◉	135/0	0/30	0/20	0/30
2:11	103	3← ◉ →	◉	141/0	-3/36	-3/26	0/36
2:14	106	← ◉ →	◉	144/0	0/0	0/0	3/0
2:39	131	9← ◉ →	◉	0/25	25/0	25/0	28/0
2:48	140	◉ ◉ →	◉	-2/34	34/0	34/0	37/0
2:50	142	← ◉ →	◉	0/0	36/0	36/0	39/0
2:51	143	← ◉	◉	1/0	37/0	37/0	40/0
3:25	177	← ◉ →	◉	35/0	71/0	-3/34	74/0
3:28	180	← ◉ →		38/0	74/0	0/0	77/0

Table 2.1 Compilation of sampled signal phases for northbound traffic into intersection S1 and its eastern pedestrian crossing.

The motto in Hangzhou was “better safe than sorry”. It would be regretful to later find out that every little detail of the chosen area was not mapped accurately. As a result, there now exist data regarding signal phases for 11 intersections and three independent pedestrian crossings, intersection layouts, lane-configuration, street lengths, etc. In retrospect it can be established that the data gathering was too extensive. From the early start in this project, the approach was to find optimal signal phases with an analytic solution, like the one used for queueing systems. Then the idea of using SUMO to simulate different scenarios was introduced. This also meant that the authors, without any supervision or guidance, had to learn a completely new software and how to operate it. One benefit from that experience is an understanding of the principles of XML files which are used to feed SUMO.

First while back in Sweden again, it was decided to reduce the number of traffic light intersections in the theoretical models to base the calculations upon. Basically, that means that this approach ignores traffic signals at the smaller intersection, as well as at the three pedestrian crossings. This model

subtracts the street distances these other places occupy at a red-light signal phase. For each intersection 50 m is deducted and for pedestrian crossings 12 m.

2.5 Optimizing TLS phase times

In general terms road users are not interdependent of each other, whether they are motor vehicle drivers, bicyclists or pedestrians. Furthermore, it is normally not possible to predict at what time one road user will appear at a certain place in traffic. However, the number of all road users during a certain time period may be counted repeatedly, hence an average for any time period may be estimated. One could also argue that two members of the same group of road users, do not appear at the same place at the same time. Thus, vehicle arrivals in road traffic generally fulfills the criteria for a Poisson process.

The Poisson process does not apply for areas where traffic becomes denser, ultimately forming a traffic jam and making a lane change consequently impossible to perform. Vehicles in the traffic lanes are like disposable cups in a cup holder, where the cups will be removed in order. When traffic moves like this and rely on a traffic signal, telling the driver when to move his or her vehicle in a certain direction, the process in such area is predictable or with another term deterministic.

A jammed area with deterministic movements can grow, be stable or shrink. Outside such area there will be a zone to which vehicles arrive randomly like in a Poisson process and while slowing down, merge into the deterministic process. The circumference of this zone will typically grow or shrink over time to eventually cease to exist, has the jam been resolved.

2.5.1 Computer code

Most of the computer code is explained in the following sections. As a reference it is also found in the Appendix. However, all computer code is also found at: https://github.com/ada09aho/hangzhou.

2.5.2 Algorithm

An algorithm has been constructed in order to optimize dense traffic flow. MATLAB was chosen to perform that task. An analytical solution would probably result in an exact answer, but it is also more complex and much harder to grasp. Therefore, a numerical method was chosen to find out whether there exists a solution for evacuating traffic congested streets as smoothly and balanced as possible. What this algorithm does in short, is to create alternative settings of phase times, estimate the evacuation values of vehicles, compare all possible combinations of those values and their

respective variance value. From this the lowest possible value of standard deviation is found.

The hypothesis is that the setting with lowest standard deviation regarding evacuation of vehicles, would perform the overall smoothest traffic flow. There is of course an awareness of that there may exist settings with lower average evacuation value than in the case with the lowest standard deviation value, but this would also mean higher traffic flow for some streets at the expense of traffic on the other streets. The following MATLAB code describes the algorithm schematically.

```
Initialize matrices and parameters
least = [1, 1]
for s1 = 1:31
  ...
        ...
            for s11 = 1:31
              Z = (s1,s3,s5,s6,s9,s11)
              [avgX, varX] = variation(X, Z, S, P, CI, C)
              if varX < least(2) && avgX < 1
                least = [avgX, varX]
              end
        ...
  ...
end
disp('μ: ', least(1), 'd(μ): ', sqrt(least(2)))
```

The very simplified pseudo code above symbolically shows the main feature of the algorithm, i.e., the six nested for loops in which the search for lowest average, `avgX`, and variance, `varX`, takes part and finally displaying it as μ and as d(μ). The general term for average is μ, and the standard deviation, d(μ), is expressed as $d(\mu) = \sqrt{V(\mu)}$, where V(μ) is the variance. The algorithm is further discussed throughout Section 2.5 *Optimizing TLS phase times*.

All MATLAB code is shown in Section A.1 *MATLAB code*.

2.5.3 Scenarios

When simulating an alternative reality, a scenario with altered events plays out. A simulation scenario where the phase times from the G-matrix are left unaltered is here referred to as IRL, as *in real life*. *G* is also used as a notation

for IRL, which refers to a G-matrix containing all recorded phase times. G stands for green signal. The 100_33- and 50_150-scenario are alternatives with other variations of signal phases from the G-matrix. They are explained further in Section 2.5.11 *Sets of cases*.

2.5.4 PCUs

The concept of a passenger car unit, pcu is used. A pcu is of course the unit itself and is considered as an average car. A motorcycle is 0.4 - 0.5 pcu, being smaller than a car. A bus is approximately three pcu, since it is larger. This way one can model a system for pcus and make it valid for various combinations of all sorts of street vehicles.

2.5.5 Mass of cars

The starting scenario is all streets fully saturated by cars i.e., "bumper-to-bumper" traffic to represent congestion during rush hours. From the map service at www.gaode.com one can see how traffic intensity in certain city areas get affected during hours and days of an average week. Figure 2.3 shows such traffic intensity map, where symbols for traffic light signals and pedestrian crossings have been superimposed.

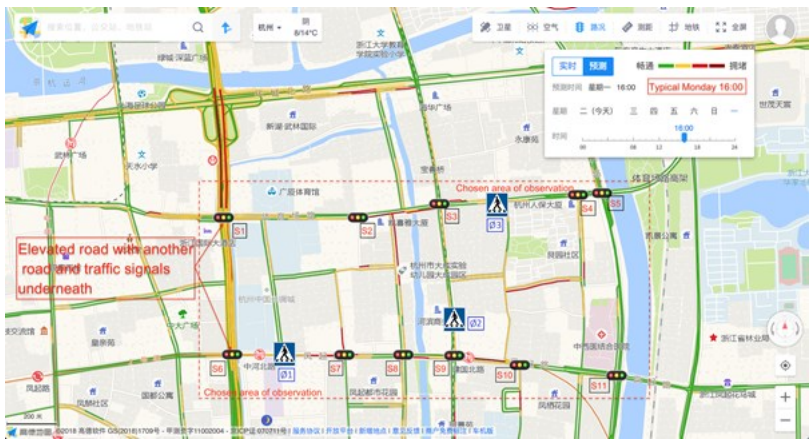


Figure 2.3. Road network of project area. Source: www.gaode.com

Each street usually has parallel lanes. The number of lanes coming out of an adjacent intersection is a bottleneck which decides the possible flow of cars coming into that street. To calculate the number of cars coming into the intersection, we need the intermediate street length d_i (see Figure 2.4), the number of lanes l_i at the street entry and a measurement of the vehicle's density v, $(vehicles/m)$. $C_i = d_i \cdot l_i \cdot v \ \ (vehicles)$. Here C_i is a 4x1-matrix, which represents vehicles on streets from 4 directions.

```
D =

   470   520   530   570   618   760
   840   658     0   803   640   305
   570   618   760   735   725   670
   395   840   658   230   803   640
```

Figure 2.4. The D-matrix with the distance of intermediate streets.

Note that there may exist more lanes along the streets than is the case with their entrances. This is ignored in the model. Figure 2.5 shows C, i.e., the calculated numbers of pcus that are possible to fit on each street. The values are not truncated, which they ought to be. The single 1 in column 3 was put there manually, to avoid the original zero as a denominator. The values in C corresponds to *prior vehicles* in Section 2.5.18 *Evacuation*.

```
C =

  156.6667  260.0000  265.0000  190.0000  206.0000  506.6667
  420.0000  329.0000    1.0000  401.5000  320.0000  101.6667
  285.0000  206.0000  380.0000  490.0000  241.6667  335.0000
  197.5000  280.0000  329.0000  115.0000  401.5000  320.0000
```

Figure 2.5. The C-matrix with maximum numbers of pcus per street.

Even if a street is fully saturated with vehicles, one still must consider some movements among cars and some safety space that each driver maintains around his or her vehicle. 150 % of the vehicle length may be a reasonable total road length used up by each vehicle, at least at a standstill and at slow speed. Some webpages gave information about Chinese passenger car lengths in the range 3.2 – 4.8 m, at the time the MATLAB algorithm was constructed. The average length tends to increase towards western standards over time. In January 2022 a rough estimate[4] of the average Chinese passenger car length would be 4.2-4.5 m. This does not include larger SUVs and pickups, but well all tiny minivans. However, those and any sizes of trucks and buses can all be considered within the pcu concept.

MATLAB calculations are based on an average car length of 4.0 m, which means every passenger car occupies a total of six m street lane length. For example, a three-lane 100 m long street would, with this reasoning, contain 50 passenger cars, i.e., pcus or any vehicle combination within this concept. This means 16 ⅔ cars per lane, which of course is difficult if each car is exactly four m long. But within this approach there are either 17 cars denser fitted in one lane or vehicles that have various lengths. Up to 21 cars may fit in a 100 m-lane within the pcu concept.

[4] https://en.wikipedia.org/wiki/Vehicle_size_class#China

2.5.6 Initial mass of cars

The green arrows in Figure 2.6 describes how the six TLIs are interconnected. Red arrows describe where traffic enters end exits the modelled system. This model is a rough estimation, since several minor streets exist as well.

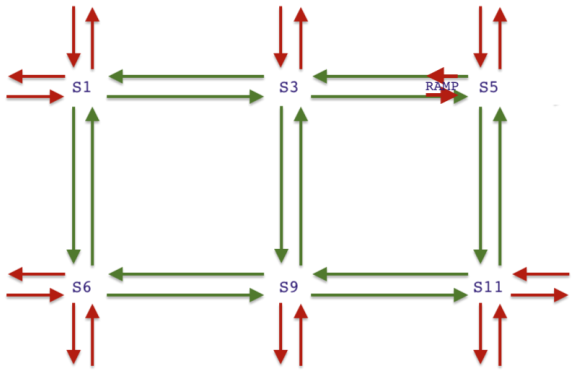


Figure 2.6. Seven intermediate bidirectional roads of the system.

The C-matrix symbolizes 14 street directions filled with cars. However, all streets have different lengths and number of lanes. Using the C_i-columns would result in unfair comparison of the TLIs. Therefore, an initial distance $d_i = 200\ m$ for all incoming directions is set. So, in addition to C_i there also is $CI_i = di_i \cdot l_i \cdot v$ where i is the TLI in question. Figure 2.7 shows the CI-matrix.

```
CI =

   66.6667  100.0000  100.0000   66.6667   66.6667  133.3333
  100.0000  100.0000    1.0000  100.0000  100.0000   66.6667
  100.0000   66.6667  100.0000  133.3333   66.6667  100.0000
  100.0000   66.6667  100.0000  100.0000  100.0000  100.0000
```

Figure 2.7. Matrix CI with initial numbers of pcus.

2.5.7 Probabilities

Figure 2.8 shows a layout of the TLI S11, which is situated in the south-east corner of the modelled area intended for road traffic simulation. Beige colored fields are traffic lanes reserved for buses. In this case buses do not turn, and they move only from north to south and vice versa respectively.

Using classic probability, based on the notion that a great number of vehicles pass through over time, the number of incoming traffic lanes from one direction can specify what shares of the vehicles probable will make a

left turn, a U-turn, a right turn and will continue straight ahead. All such probability values can be gathered in a 4 x 4-matrix, here named P.

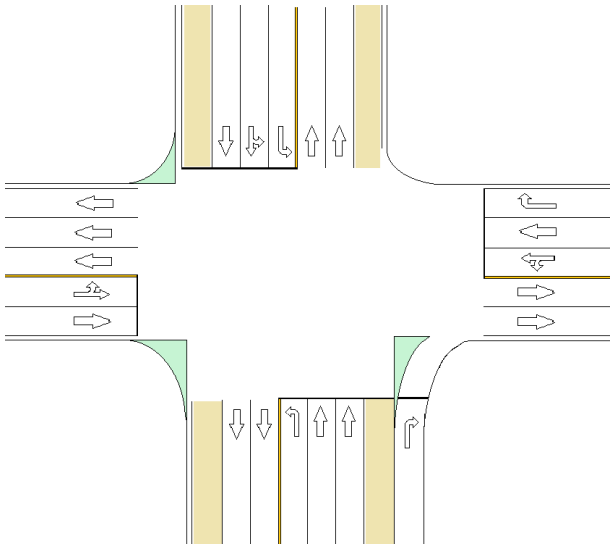


Figure 2.8. Traffic Light Intersection S11.

To be consistent all over the model, directions are specified with the index numbers 1 for north, 2 for east, 3 for south and 4 for west. The probability for traffic from north (1), going straight forward, i.e., south (3), will be found at position p_{31}in the matrix P and traffic from south (3) going to the west (4) will be in position p_{43} in P. U-turns are found on the diagonal of P. The formula for each index is:

$$p_{ij} = P(going\ to\ direction\ i|coming\ from\ direction\ j) = \frac{sum(lanes\ for\ direction\ i|from\ direction\ j)}{sum(lanes\ from\ direction\ j)}.$$

The probabilities for incoming traffic from north in S11 are explained as follows. Four lanes enter from north, i.e., index $j = 1$. No lane explicitly allows making U-turn, i.e., going back to the north or index $i = 1$. One lane is explicitly for left turns, which means $p_{21} = 1/4$. One lane allows both going straight to the south, i = 3, and turning left to the east, $i = 2$. This means that for this lane there are two values of probability, namely $p_{21} = 1/8$ and $p_{31} = 1/8$. Finally, there are two lanes from north that allow traffic going straight forward, a bus lane being one of them. For those two lanes the probability is $p_{31} = 2/4$. A right turn is not possible here, hence $p_{41} = 0$. Consequently, column 1 of the matrix P will be: $p_{11} = 0$, $p_{21} = 1/4 +$

$1/8 = 3/8$, $p_{31} = 1/8 + 2/4 = 5/8$ and $p_{41} = 0$. The complete P-matrix for S11 is shown in Figure 2.9.

```
P11=
  0    1/3   3/5   1/4
3/8     0    1/5   3/4
5/8   1/6     0     0
  0    3/6   1/5     0
```

Figure 2.9. The P-matrix for TLI S11.

2.5.8 Phase times

Traffic signals (TLS) usually consists of three different signal colors, red, yellow (or amber) and green for which the meaning may differ on various countries around the globe. The signals may also be combined in various ways. However, a single red light always means stop and a single green light always means go.

While compiling fieldwork data it became obvious that the duration of a traffic signal was a multiple of three seconds. The duration for yellow signals were always three seconds long. A typical signal time length is $27\ s$, $30\ s$, $57\ s$ or $60\ s$ but not $25\ s$, 31 s or $55\ s$ and so on. According to the Swedish regulation TRVMB [1] the shortest green light signal should be no less than $6\ s$, a rule which has been considered throughout the work.

A phase time is the time period during which a signal is, or a combination of signals are lit. One signal may also be divided into multiple signal phases, e.g., when it is permitted to make a right turn in different combinations with other signals. In the MATLAB code phase times are stored in the G-matrix.

The TLI S11 is a special case within the modelled system. The left matrix in Figure 2.14 represents the sampled phase times of S11. After the simulation process had begun and while correcting erroneous driving behaviors it was discovered that the traffic signals in this intersection must operate in a Round Robin style, i.e., where green signal is lit for traffic from only one direction at the time. The clue was given by the traffic lane arrows on the tarmac. This intersection is impossible to examine due to the fact it is an elevated non-pedestrian zone. Originally it was assumed S11 operates in a case 1-2-3-4-manner as explained in Section 2.5.9 *Cases*.

2.5.9 Cases

During a cycle time there are several signal phases, all of which occur at several different but chronologically ordered times. To minimize the number

of calculations, several signal phases were arranged according to these four different cases.

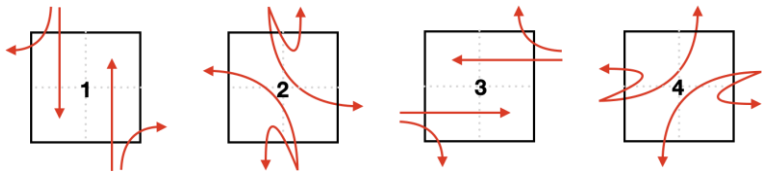


Figure 2.10. The four cases of a TLI.

Figure 2.10 shows four different cases where traffic from and to specific directions are represented by red arrows, each corresponding to a green signal phase in each case. It is possible to combine a few other directions with the fixed cases. But none of the left turning cases 2 and 4 can be combined with traffic going straight forward in any direction. However, many cases must be combined with a right turn signal because the phase time for turning right often is longer than its equivalence for going straight ahead.

2.5.10 Binary truth table

```
i =

     0     0     0     0
     0     0     0     1
     0     0     1     0
     0     0     1     1
     0     1     0     0
     0     1     0     1
     0     1     1     0
     0     1     1     1
     1     0     0     0
     1     0     0     1
     1     0     1     0
     1     0     1     1
     1     1     0     0
     1     1     0     1
     1     1     1     0
     1     1     1     1
```

Figure 2.11. A 4-bit binary truth table.

Have a look at the binary truth table in Figure 2.11. The values in column one represents case 1, i.e., traffic from north. Case 2, 3 and 4 are represented by column 2, 3 and 4 correspondingly. A truth table covers all possible configurations within its limits, which here is a 4-bit binary word. Having only two options per position, i.e., a 0 or a 1, the number of alternative configurations is 4^2 or 16 in total. When the truth table acts as a matrix, it can be multiplied by a factor and a scalar can be added to it. Row number six in

Figure 2.11 is highlighted, because it relates to the example in Section 2.5.11 *Sets of cases*.

2.5.11 Sets of cases

Suppose that each case will contain three different sets of times phases. From the phase times in the G-matrix, the cases for scenario 50_150 can be constructed by letting one truth table represent the transition from 50 % of each phase time to 100 % of such time respectively. In addition, a second truth table that represent the transition from 100 % to 150 % must complete the example. The 1111-alternative for the range 50-100 % will coincide with the 0000-alternative for the range 100-150 %, thus it may be omitted, resulting in $16 + 15 = 31$ unique combinations. For the scenario 50_150 MATLAB will basically produce a K-matrix with the following code: `K=[.5*i+.5; .5*i+1].`

```
>> [K(1:16,:), K(16:31,:)]

ans =

    0.5000    0.5000    0.5000    0.5000    1.0000    1.0000    1.0000    1.0000
    0.5000    0.5000    0.5000    1.0000    1.0000    1.0000    1.0000    1.5000
    0.5000    0.5000    1.0000    0.5000    1.0000    1.0000    1.5000    1.0000
    0.5000    0.5000    1.0000    1.0000    1.0000    1.0000    1.5000    1.5000
    0.5000    1.0000    0.5000    0.5000    1.0000    1.5000    1.0000    1.0000
    0.5000    1.0000    0.5000    1.0000    1.0000    1.5000    1.0000    1.5000
    0.5000    1.0000    1.0000    0.5000    1.0000    1.5000    1.5000    1.0000
    0.5000    1.0000    1.0000    1.0000    1.0000    1.5000    1.5000    1.5000
    1.0000    0.5000    0.5000    0.5000    1.5000    1.0000    1.0000    1.0000
    1.0000    0.5000    0.5000    1.0000    1.5000    1.0000    1.0000    1.5000
    1.0000    0.5000    1.0000    0.5000    1.5000    1.0000    1.5000    1.0000
    1.0000    0.5000    1.0000    1.0000    1.5000    1.0000    1.5000    1.5000
    1.0000    1.0000    0.5000    0.5000    1.5000    1.5000    1.0000    1.0000
    1.0000    1.0000    0.5000    1.0000    1.5000    1.5000    1.0000    1.5000
    1.0000    1.0000    1.0000    0.5000    1.5000    1.5000    1.5000    1.0000
    1.0000    1.0000    1.0000    1.0000    1.5000    1.5000    1.5000    1.5000
```

Figure 2.12. The truth table for scenario 50_150 split in two.

Figure 2.12 shows matrix K, separated into two 16x4-matrices and placed side by side. Note that position `K(16,:)` here appears twice. To understand the numbers, row 6 of this K-matrix is highlighted and taken as an example. Read the line such that all phase times in case 1 is to be multiplied by 0.5, in case 2 with 1.0, in case 3 with 0.5 and in case 4 with 1.0.

Bear in mind that each representation of a case contains a set with several phase times in four directions. Therefore, each unique case is represented by a 4x4-matrix, where each index represents the phase time for traffic from direction j to direction i. One TLI is thus represented by a 4x4x31-matrix. To make it even more complex, a 4x4x31x6-matrix X holds all the unique cases

for all six TLI for the MATLAB calculations. The MATLAB code constructs this and other matrices at start up and fetches and compares values during runtime. The observant reader may notice that there is a limitation within the same TLI, namely that the difference between phase times in a setting can never differ more than one step, i.e., either a combination of 0.5 and 1.0 or 1.0 and 1.5.

The share values which to multiply with the binary truth table is the crucial difference between scenario 50_150 and 100_33. For 100_33 the values in K are either $1/3$, $2/3$ or $1/1$, i.e., 33.3, 66.7 or 100 %. This of course results in completely different phase time values in the 31 comparable sets of cases for each TLI.

```
>> X(:,:,6,6)

ans =

         0   22.5000   22.5000   45.0000
   45.0000         0   22.5000   22.5000
   22.5000   45.0000         0         0
         0   22.5000   45.0000         0
```

Figure 2.13. Phase times setting 6 in S11 for 50_150.

When the MATLAB algorithm calls on `X(:,:,6,6)` it will get the matrix shown in Figure 2.13. This corresponds to the phase times in S11 and the four leftmost values on row 6, which were focused on in Figure 2.12. The four columns from left corresponds to traffic from north, from east, from south and from west. From the diagonal values there is no dedicated phase time for U-turn from any direction, which coincides with the intersection layout from Figure 2.8.

```
>> G(:, 21:24)                  >> case1

ans =                           case1 =

     0    45    45    45            0     0     1     0
    45     0    45    45            0     0     1     0
    45    45     0     0            1     0     0     0
     0    45    45     0            1     0     0     0
```

Figure 2.14. Phase times for S11 (left). Case 1 binary matrix (right).

To produce the phase time values in Figure 2.13 the following MATLAB operation is performed: `G(:,21:24).*case1*K(6,1).` The two

matrices in Figure 2.14 are multiplied elementwise and then with the scalar 0.5 found in column 1 in row 6 of K (Figure 2.12). There are four case-matrices that, when multiplied elementwise with a 4x4-portion of G, singles out the matrix values that corresponds to that case. When multiplied with its corresponding row and column value of K, a setting is compiled and put in the four-dimensional X-matrix. Values related to case 1 in Figure 2.13 and Figure 2.14 are highlighted.

The purpose of having basically 6 libraries with 31 case settings is to facilitate for the algorithm to compare every possible combination during search for a better traffic flow.

2.5.12 Cycle times

A traffic light signals controlled intersection (TLI) is operating according to a beforehand decided pattern or cycles. Some intelligent TLIs equipped with ground sensors, though may skip one or multiple phases, based on the local traffic situation. Accordingly, a cycle is completed just before the TLS starts to repeat its programmed pattern. The time period for this is the *cycle time*. In the MATLAB code's main algorithm cycle times are calculated and then put in S, a 31x6-matrix.

As there are 31 different case combinations per TLI, there are also different cycle times along with those. Therefore, the S-matrix is a 31x6-matrix containing all comparable cycle times. Figure 2.15 shows the general case of how to calculate the cycle time for each case setting (*cs*) and each TLI (*tli*) from a case matrix $X_{cs,tli}$. In short, the formula adds together the longest phase times for going straight ahead with ditto for making left turns.

$$X_{cs,tli} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \\ x_{41} & x_{42} & x_{43} & x_{44} \end{bmatrix}$$

$$S_{cs,tli} = \sum\big(max(x_{31}, x_{13}), max(x_{21}, x_{43}), max(x_{42}, x_{24}), max(x_{32}, x_{14})\big)$$

Figure 2.15. Formula for calculating cycle time from an X-matrix.

To find out the cycle time at case setting 6 for S11, which is examplified in Figure 2.13, the formula becomes:

$$S_{6,S11} = \sum\big(max(22.5,22.5), max(45,45), max(22.5,22.5), max(45,45)\big)$$
$$= \sum(22.5,45,22.5,45) = 135\text{s}$$

2.5.13 Comparisons

The method for finding best flow is a so called ‘brute force’ or ‘exhaustive search’ algorithm. As mentioned earlier the algorithm makes comparisons of estimated vehicle evacuations on system streets. 887,503,681 comparisons to be exact, which follows of it having 31^6 case settings. The modelled system has six TLIs, which in terms of computer code means implementing six nested for-statements, that each loop through its 31 indices. For each comparison all six indices are put in a 1x6-matrix, as $Z = [s1, s3, s5, s6, s9, s11]$, which is sent to the method for finding average and variance. One single run of the algorithm on a 10-year-old quad-core computer typically would consume 140,600 s, which means well over 39 h and roughly 6,300 comparisons per second.

If all phase times had been split into four parts, the algorithm would have to deal with 9.5 billion iterations, which would take some 17 days to calculate. Instead, another numerical calculation was designed. Hence there are two numerical calculations, one with the levels 33.3 %, 66.7% and 100 % (100_33) and one with 50 %, 100 % and 150 % (50_150) of observed phase times.

2.5.14 Ratios

Each traffic lane direction in a TLI has a certain ratio of the complete cycle time, that is a quota of the phase time of that lane and the cycle time. A ratio matrix R is received simply by dividing each element of an X-matrix with the scalar, s, from the cycle time formula or as $R = \frac{1}{s} \cdot X$.

This operation converts a time period of some seconds into a unitless fraction. However, the fraction represents the time from the case in question. To recreate the ratios for case 1 of the example in Figure 2.13 and Figure 2.14 with MATLAB code, the following command can be run:

```
R11_1=X(:,:,6,6).*case1/S(6,6).
```

Figure 2.16 shows the resulting ratios for that case. Position (4,1) in R11_1 matrix is zero because turning west from north is not possible in S11. Figure 2.12 the complete R11-matrix for the case setting 6 above.

```
R11_1 =

         0         0    0.1667         0
         0         0    0.1667         0
    0.1667         0         0         0
         0         0         0         0
```

Figure 2.16. Ratios of case 1, case setting 6, S11, scenario 50_150.

```
R11 =

         0    0.1667    0.1667    0.3333
    0.3333         0    0.1667    0.1667
    0.1667    0.3333         0         0
         0    0.1667    0.3333         0
```

Figure 2.17. All ratios in S11 at case setting 6 in scenario 50_150.

2.5.15 Incoming and outgoing

To obtain values for comparison, estimations of how many vehicles that can possibly enter and leave a street must be made. The main idea is to multiply the unitless probability, P, and ratio, R, with the initial mass of cars, CI. This gets a value, TF, of the number of vehicles that have left one street lane and entered another street lane after the intersection. The MATLAB command is `TF=P.*R.*CI`, which performs elementwise multiplications.

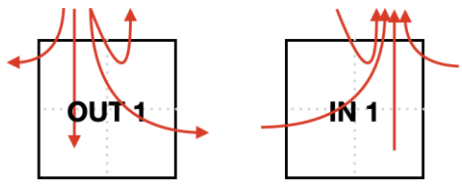


Figure 2.18. PCUs from north (left). PCUs to north (right).

The left part of Figure 2.18 shows traffic from north and the right part show traffic to the north. With respect to the intersection the sum of TF-values of incoming lanes of one street, gives the number of vehicles leaving that street. The sum of TF-values from lanes that all enters the same street gives the number of vehicles entering that street.

The whole operation is of course a rough estimation but being done in the same manner for all case settings. In the real world one must consider the law of inertia, driver's response time and of course random obstructions in the form of pedestrians, bicycles, mopeds, and other vehicles. All values could have been multiplied by a factor to take the mentioned considerations into account. However, this would just be a linear down-scaling, resulting in the same results in the end.

Since the ratio, R, comes from dividing a phase time with a cycle time, the TF-values apply for the current cycle time. An estimation of flow, i.e., vehicles per second, would be to divide TF-values with the current cycle time from the S-matrix. This would get the average pcus per second in general and over a specified time period. To find out how many pcus that actually moves per second, one has to divide the TF-value sum from each street lane with the

phase time of that same lane individually. Note that this neither do consider the law of inertia, driver's response time, and random obstructions.

```
PR11 =                                                  CI11 =

         0    0.0556    0.1000    0.0833                133.3333
    0.1250         0    0.0333    0.1250                 66.6667
    0.1042    0.0556         0         0                100.0000
         0    0.0833    0.0667         0                100.0000
```

Figure 2.19. PR-matrix (left) and CI-matrix (right) for S11.

By, for each TLI, multiplying the PR-matrix and the CI-matrix in two different ways the algorithm calculates *time flow in* (TFI) and *time flow out* (TFO). Figure 2.19 shows the PR- and CI-matrix for S11.

2.5.16 Time flow in

TFI is achieved by summing up each column of the PR-matrix to form a 1x4-matrix, transposing it and then multiply it elementwise with the CI-matrix. The result is a 4x1-matrix. Figure 2.20 shows a MATLAB command to create TFI and the resulting 4x1-matrix, regarding the previous examples from S11.

```
>> TFI11=sum(PR11)'.*CI11

TFI11 =

   30.5556
   12.9630
   20.0000
   20.8333
```

Figure 2.20. TFI for S11, case setting 6, scenario 50_150.

2.5.17 Time flow out

To get TFO, the PR-matrix and CI-matrix are multiplied. Figure 2.21 shows the MATLAB command and its resulting 4x1-matrix for the same example as with time flow in.

```
>> PR11*CI11

ans =

   22.0370
   32.5000
   17.5926
   12.2222
```

Figure 2.21. TFO for S11, case setting 6, scenario 50_150.

2.5.18 Evacuation

To estimate evacuation of vehicles, one must consider three amounts during a fixed time period, namely a) the number of vehicles on a street prior to a change, b) the number of vehicles going out of the street and c) the number of vehicles coming into the street. By dividing the sum of a, b and c with a, the quota will tell the degree of change and it helps to compare its evacuation with the other streets. A number less than 1 means an actual evacuation of vehicles and the opposite if the number is larger than 1. The rather simple formula for evacuation is:

$evacuation = \frac{(incoming+prior-outgoing)}{prior} \frac{(pcus)}{(pcus)}$, where pcu refers to a personal car unit. The quota is unitless. The average of all evacuation quotas, μ, makes it possible to estimate the deviation, $d(\mu)$.

TFI and TFO are 4x6-matrices built up from six consecutive calculations, which are done for every case setting. Next step is to estimate the evacuation. Given that TFO and TFI are ordered correct for the entire system, the formula for evacuation is in principle $evacuation = \frac{TFO+C-TFI}{C}$. However, that is not possible for several reasons, excessive information being one of them and matrix division rules being another reason. Therefore, firstly the TFO is rearranged into a new matrix, TFS, that can be reduced by TFI, i.e., where the street positions correspond to each other. Secondly a binary 4x6-matrix, SC, was used to single out the significant 14 streets. Thereafter it was possible for the algorithm to calculate an evacuation value by the following two rows of MATLAB code.

```
DIFF = (TFS - TFI) .* SC;
EVA = ((C + DIFF) ./ C) .* SC;
```

2.5.19 Average evacuation and variance

To find the average time flow and its variance the following four rows of MATLAB code was run before comparing it with the previous lowest value. The remarks are kept, due to their informative contents.

```
E = find(EVA); % finds indices of all nonzero elements
WS = EVA(E); % column vector with nonzero values of EVA
avgX = mean(WS); % mean value of WS
varX = mean((WS-avgX).^2); % Variance of WS
```

2.5.20 MATLAB plots

For the sake of plotting the behavior of the algorithm a plot lists were made for each scenario. The matrix PLOTLIST has 29,791 rows and two columns. To avoid plotting some 887.5 million values, estimations via the average values of evacuations and variances were used. 29,791 values of each evacuation and variance were summed up and then the average value of the evacuations and the square root of the average variance values were saved, which were repeated also 29,791 times. As mentioned before the algorithm performs 31^6 iterations. This can also be written as $31^3 \cdot 31^3$ or $29{,}791 \cdot 29{,}791$ iterations. Figure 2.22 shows the average deviation in which a diagram of all the new smallest deviation values has been imposed. Figure 2.23 shows the average evacuation values from PLOTLIST for scenario 100_33. The corresponding plots for scenario 50_150 are shown in Figure 2.24 and Figure 2.25. The reason why scenario 50_150 finds the lowest deviation value in an earlier iteration index, is because it is run from 50 to 150%, while the other scenario was run backwards, i.e., from 100 to 33.3%. However, both scenarios find the smallest deviation when $Z = [10,5,5,6,15,11]$.

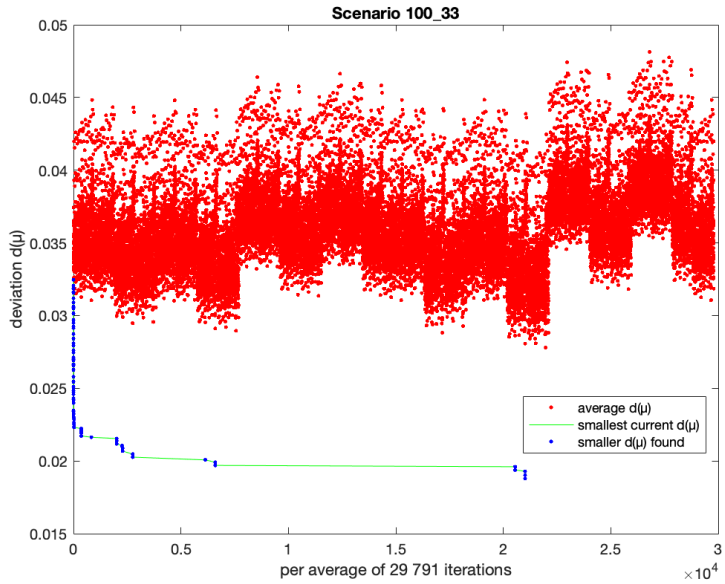


Figure 2.22. Lowest and average deviation for 100_33.

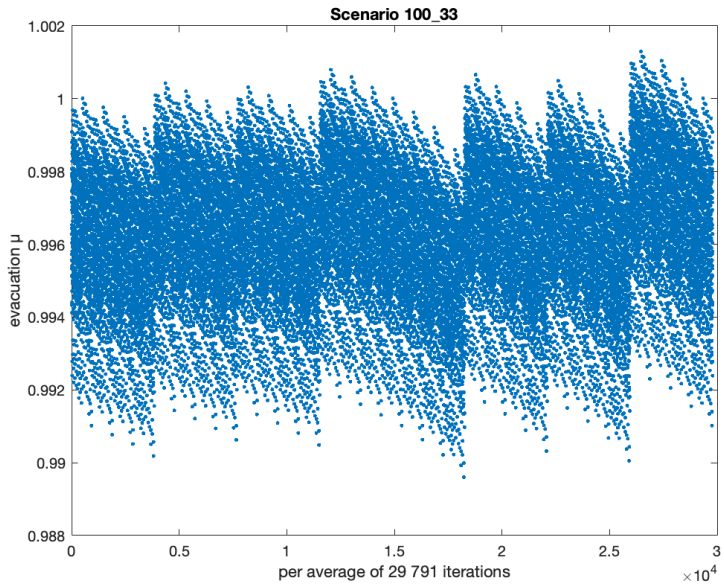


Figure 2.23. Average evacuation in scenario 100_33.

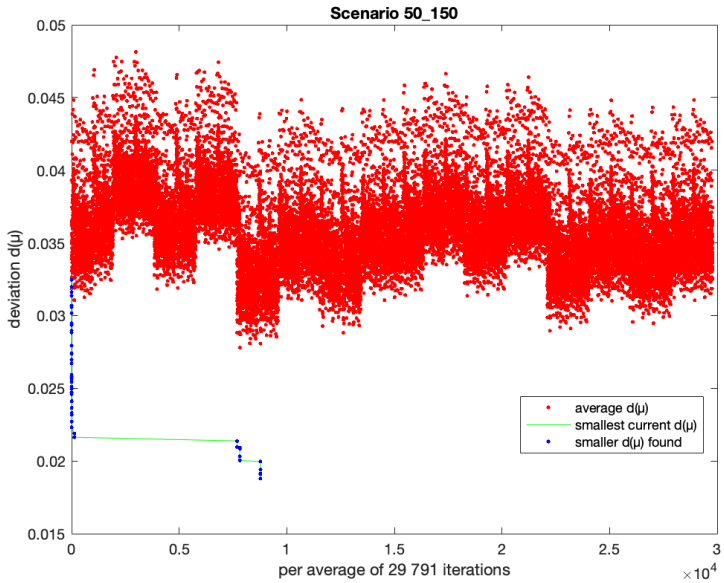


Figure 2.24. Lowest and average deviation for 50_150.

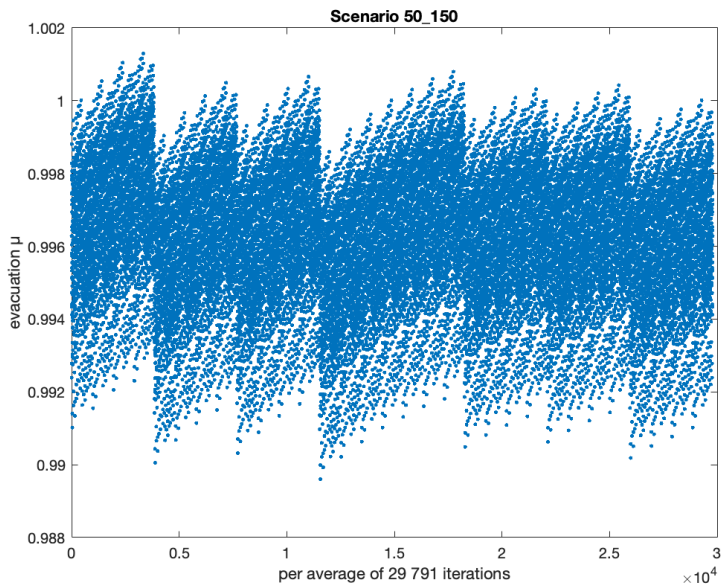


Figure 2.25. Average evacuation in scenario 50_150.

The matrix called TOP5DEV holds the top five lowest $d(X)$ and at what signal phase constellation. Below this paragraph are the TOP5DEV outputs for 100_to_33 and 50_to_150. The eight numbers on each row are from left: d(μ), Z and a plotindex. As earlier mentioned, Z contains the six index values of the nested for loops. Plotindex holds an x-value for plotting the found standard deviation values. Plotindex starts from zero and increases by one for every 31^3 comparison.

```
100_to_33:
0.0187926219180896 10  5 5 6 15 11 21013
0.0190185897535813 10  5 5 6 15 26 21013
0.0192886460232188 10  5 5 6 30 11 21013
0.0193846127856764 10 20 5 6 15 11 20548
0.0196039953793205 10 20 5 6 15 26 20548

50_to_150:
0.0187926219180896 10 5 5 6 15 11 8777
0.0190911123177668 10 5 5 6  5 11 8777
0.0191295710725981 10 5 5 5 15 11 8777
0.0194100142013391 10 5 5 5  5 11 8777
0.0199655055074297 10 5 5 5  5  1 8777
```

Note that despite what interval that is fed into the algorithm, it homes in on the same value of deviation, $d(X)$, and at the same constellation, i.e., when $Z = (10,5,5,6,15,11)$. The approach differs because 100_to_33 was run from top to bottom and 50_to_150 was run from bottom to top of the constellation values held in X. When this value of Z was passed, no lower value of $d(X)$ could be found - no matter in which direction the search was made. This is, however, logical. The 31 different constellations of each TLI are numbered from its lowest value and up. So, none of the values in Z are over the middle, meaning below 67 % in 100_to_33 and below 100 % in 50_to_150. By checking corresponding rows in the matrix K, one can interpret what the values in Z mean. In the case of 100_to_33, K here is:

```
10 - 0.67 0.33 0.33 0.67
 5 - 0.33 0.67 0.33 0.33
 5 - 0.33 0.67 0.33 0.33
 6 - 0.33 0.67 0.33 0.67
15 - 0.67 0.67 0.67 0.33
11 - 0.67 0.33 0.67 0.33
```

K for the rows 5, 6, 10, 11 and 15 are ordered and once repeated to accurately reflect the value of $Z = [10,5,5,6,15,11]$. The top row corresponds to intersection S1. In S1 case 1 should last for 67 % of observed time, case 2 for 33 %, case 3 for 33 % and case 4 for 67 %. In the same way intersections S3, S5, S6, S9 and S11 follow. This is true for both scenarios 100_to_33 and 50_to_150, since they both have the exact same lowest $d(X)$ at the exact same iteration. The two calculations are the exact same things, but with different start values that are scaled differently.

2.5.21 100_to_33 results

To get the corresponding signal phases the following calls in MATLAB are made.

```
OPTX_N = [X(:,:,10,1),X(:,:,5,2),X(:,:,5,3)];
OPTX_S = [X(:,:,6,4),X(:,:,15,5),X(:,:,11,6)];
```

```
OPTX_N =
 0.0 26.0 26.0 30.0  0.0  0.0 17.0 13.0  0.0  0.0 27.0 15.0
12.0  0.0 44.0 18.0 26.0 13.0 31.0 20.0  0.0  0.0  0.0  0.0
26.0 24.0  0.0 25.0 17.0 13.0 26.0 31.0 27.0  0.0 34.0 15.0
60.0 14.0 12.0 30.0 31.0 20.0 26.0 13.0 15.0  0.0 34.0 15.0

OPTX_S =
30.0 34.0 14.0 20.0  0.0 38.0 50.0 11.0  0.0 30.0 30.0 15.0
30.0 20.0 14.0 14.0 16.0  0.0 50.0 28.0 15.0  0.0 30.0 30.0
14.0 20.0  0.0 34.0 50.0 11.0  0.0 38.0 30.0 15.0  0.0  0.0
34.0 14.0 30.0  0.0 50.0 28.0 16.0  0.0  0.0 30.0 15.0  0.0
```

Three 4 x 4-matrices are put side by side and presented in one northern and one southern cluster, to make them more readable. However, the numbers are not factors of 3, so one final adjustment must be done, by calling the MATLAB function $adjGreen(XG, k)$. Here $k = 3$. This gives the below matrices. Again, presented as two matrices for better readability.

```
 0 27 27 30  0  0 18 12  0  0 27 15
12  0 45 18 27 12 30 21  0  0  0  0
27 24  0 24 18 12 27 30 27  0 33 15
60 15 12 30 30 21 27 12 15  0 33 15
```

```
30 33 15 21  0 39 51 12  0 30 30 15
30 21 15 15 15  0 51 27 15  0 30 30
15 21  0 33 51 12  0 39 30 15  0  0
33 15 30  0 51 27 15  0  0 30 15  0
```

As an example, look at intersection S3, which is the middle 4x4-matrix in the northern cluster. The phase times in the matrix are interpreted as the figure below shows. An * after a number means that this signal time immediately precedes the one in the following phase. Here this affects phase 2–3 and 4–1. Such adjustments must be made manually.

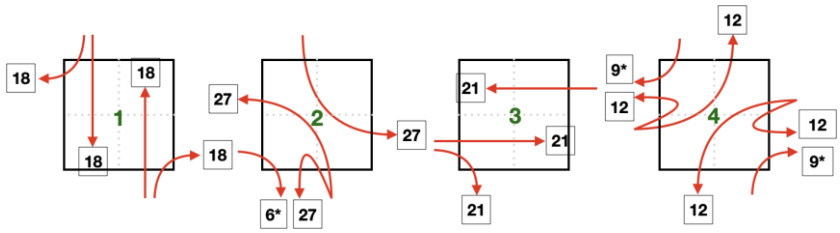


Figure 2.26 The four cases of S3 in scenario 100_33.

If the right turn phase time is longer than its equivalence for straight ahead, it will firstly begin in the previous case. If this is not enough, the signal for right turn, with respect to its own case, will span from the previous case until its subsequent case. A signal phase time that elongates like this in several cases, will stay in green mode until the last case, where it ends with a three second yellow signal. All cases, including phase times, are presented scenario-wise in Section A.1 *Graphic view of time phases used in all simulation*.

2.5.22 50_to_150 results

```
OPTX_N =
 0.0 39.0 39.0 45.0  0.0  0.0 25.5 19.5  0.0  0.0 40.5 22.5
18.0  0.0 66.0 27.0 39.0 19.5 46.5 30.0  0.0  0.0  0.0  0.0
39.0 36.0  0.0 37.5 25.5 19.5 39.0 46.5 40.5  0.0 51.0 22.5
90.0 21.0 18.0 45.0 46.5 30.0 39.0 19.5 22.5  0.0 51.0 22.5

OPTX_S =
45.0 51.0 21.0 30.0  0.0 57.0 75.0 16.5  0.0 45.0 45.0 22.5
45.0 30.0 21.0 21.0 24.0  0.0 75.0 42.0 22.5  0.0 45.0 45.0
21.0 30.0  0.0 51.0 75.0 16.5  0.0 57.0 45.0 22.5  0.0  0.0
51.0 21.0 45.0  0.0 75.0 42.0 24.0  0.0  0.0 45.0 22.5  0.0
```

After adjusting it to be nearest modulo 3, the matrices look like this:

```
 0 39 39 45  0  0 27 21  0  0 42 24
18  0 66 27 39 21 48 30  0  0  0  0
39 36  0 39 27 21 39 48 42  0 51 24
90 21 18 45 48 30 39 21 24  0 51 24


45 51 21 30  0 57 75 18  0 45 45 24
45 30 21 21 24  0 75 42 24  0 45 45
21 30  0 51 75 18  0 57 45 24  0  0
51 21 45  0 75 42 24  0  0 45 24  0
```

The intersection S3 is again used as an example of a graphical view of the cases and time phases

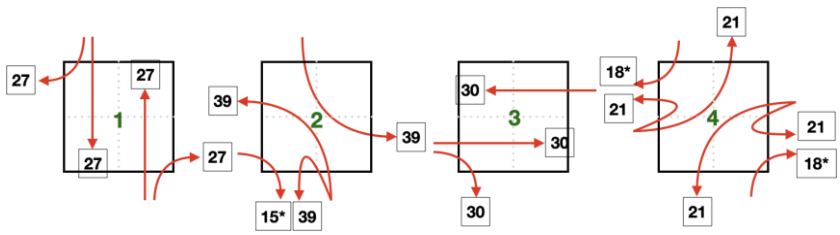


Figure 2.27 The four cases of S3 in scenario 50_150.

But take a look at S1, which is the 4x4-matrix to the left of S3. Figure 2.28 shows how extra right turns must be inserted in some of the four otherwise fixed cases.

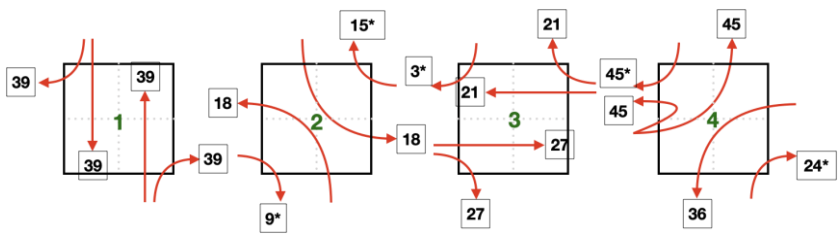


Figure 2.28. The four cases of TLI S1 in scenario 50_150.

If the right turn phase time is longer than its equivalence for straight ahead, it will firstly begin in the previous case. If this is not enough, the signal for right turn, with respect to its own case, will span from the previous case until its subsequent case. A signal phase time that elongates like this in several cases, will stay in green mode until the last case, where it ends with a three

second yellow signal. All cases, including phase times, are presented scenario-wise in Section A.1 *Graphic view of time phases in all simulations*.

2.6 Simulation with SUMO

Simulation of Urban MObility (SUMO) is an open-source microscopic traffic simulation program created by The Institute of Transportation Systems in Berlin, Germany. It was first released in 2001. It is a package program which includes documentation and tutorials, tools for running a simulation, building the traffic network, generating different traffic flows and tools for analyzing the results with different tools. It can visualize different types of vehicles and pedestrians. SUMO version 1.0.1 was used for building the model and version 1.9.0 for simulations.

To create a traffic model that resembles the real-world can be quite time-consuming. It is largely dependent on how big the model is supposed to be. In order to create the model and get the simulation running there are three key points to create or import.

The first key point is to outline the network, with roads and intersections and its corresponding number of lanes and distances. The second key point is to add the traffic infrastructure with the rules that apply to the model including traffic light signals programs, stop signs etc. The final key point is the traffic demand. One can import traffic matrices if available or create traffic files for the types of vehicles that are desired. For this purpose, the SUMO package has several accommodating programs.

SUMO can open already existing traffic maps when imported with the netconvert [8] command line application from websites such as OpenStreetMap [4]. This might lead to quick interaction between the simulation and the user, but it turned out not to give an accurate traffic model with respect to the number of lanes and intersection structure for the area that this thesis is about. Because of this it was easier to build the network from scratch in the program Netedit. Worth mentioning is that a map with geometrical shapes, area.poly.xml, was extracted from the www.openstreetmap.org with the help of SUMO's command line application polyconvert [9]. This was very helpful when building the road network in Netedit, as it could be loaded into the program and guide one when drawing the road network.

2.6.1 The selected area in Hangzhou

The area chosen is inside the Xiacheng District, one of Hangzhou's urban districts. The area has a connection to Zhong He Viaduct, an express- or

motorway, and is close to one of the city centers, with easy access to multiple tourist areas. It also contains schools, shopping district, sport hall, hospitals and is famous for its silk market. As many other urban areas in Hangzhou, it is packed with small shops and residential areas.

Figure 2.29. The network built with Netedit. TLIs are marked.

In the model there are six major intersections controlled by traffic light signals, numbers S1, S3, S5, S6, S9 and S11 as shown in Figure 2.29. In Appendix A.1 *General layouts of TLI*, the mapping of the different intersections is available. S1 being to the north-west, it is the biggest intersection with access to the expressway. S5 in north-east, is a three-way junction next to a river and lies below a main road connecting to another part of town. S6 in the south-west is a standard 4-way intersection. S11 in the south-east is a junction located up on a bridge, connecting traffic from four directions, with a small roundabout placed on the road below it. Traffic from north to west is directed to the roundabout, as well as traffic from west to south and back to the west. S3 and S9, were at the time of this thesis, building sites. The city was currently building a new metro line underground which caused some disturbances to the traffic network. The number of lanes were reduced in the south and north direction for S9, and only north incoming and outgoing lanes were reduced for S3. There are also seven other traffic signals present. Three of them are merely for the purpose of letting pedestrians and

cyclists cross the roads (see Ø1 - Ø3). The 4 others are smaller crossings, letting traffic enter and exit the residential areas. The main purpose of this thesis is to observe and optimize the traffic flow at the six major intersections. Therefore, the traffic light programs for the smaller intersections were noted and kept constant through all the simulations to maintain a realistic place-based simulation. The decision to ignore the smaller streets in the residential areas was made because the areas were mostly surrounded by fences, restricting passage and entries, as well as being too time-consuming to model.

2.6.2 Building the SUMO model

Building the street map was conducted in different steps with the Netedit program from the SUMO package. Through internet pages maps.google.com and maps.bing.com it was possible to measure the distances between the intersections and other adjacent roads. With the distances known it was easy to outline the road network. The major roads in the area had concrete barriers that carried grass and trees as two-way separators and/or roads that were separated by fencing. Therefore, when outlining the major roads in Netedit, effort was made not to draw them as two-way roads but to draw the road for each direction individually.

In Netedit roads are built with a tool for creating edges. An edge is created by drawing a line from point A to point B, where each edge segment represents a section of the road. As seen in Figure 2.30 an edge is represented by the black lanes between the red areas. The red dots seen there are called nodes, which are points where the x and y coordinates are stored. These nodes are automatically added when drawing the edge. The road itself can be represented by more than one edge. This is often the case when for example a road goes from three lanes to four lanes. If the road that is built passes an intersection, a junction will be introduced. This junction is a collection of nodes where multiple edges meet, as pictured by the dark red areas in Figure 2.30. An edge may also contain several lanes. But an increase or decrease requires a new edge to be created.

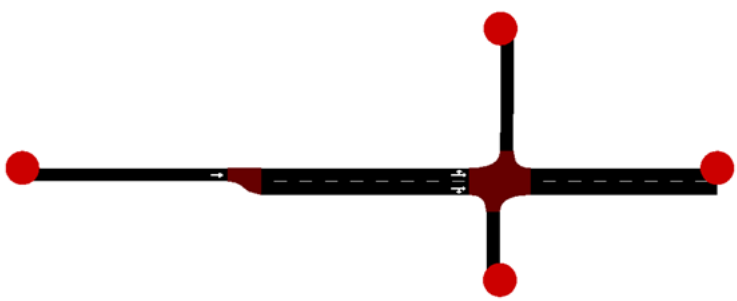


Figure 2.30. A one-way road with an intersection, built by several edges.

Netedit automatically adds connections between the incoming lanes at the junction to the outgoing lanes from the junction. These had to be altered to match the traffic rules on site for every junction. Figure 2.31 shows an example on how these connections can look like for the intersection in Figure 2.30. The connections between the incoming and outgoing lanes at the red nodes are visualized with white lines.

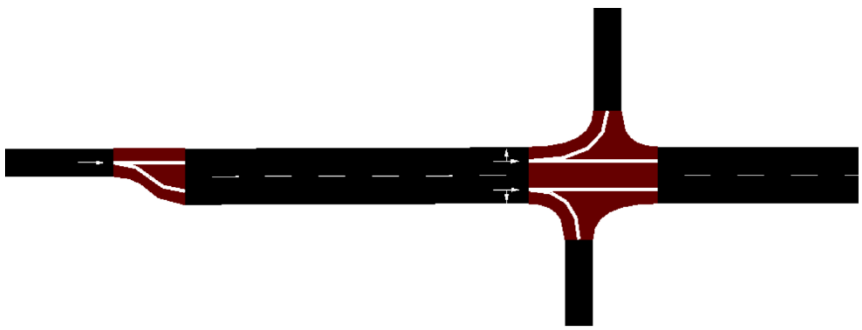


Figure 2.31. Illustration of how edges connect via nodes.

Restrictions were made as to not include the smaller streets inside the residential areas. All roads were given attributes to specify what kind of road it was, its priority rules, how many lanes it has, what kind of vehicles that are allowed and what traffic rules that apply. An attempt was made to include the pedestrian crossings existing in the area, but the attempts failed, which resulted in a few pedestrian crossings being visible in the model but of no use.

2.6.3 Creating traffic light signals (TLS)

When implementing a traffic light signal in Netedit the program automatically creates a TLS program suited for the number of lanes that goes into the intersection with regards to the turns that are allowed for each lane. The TLS also gets dedicated a name, a tlLogid id. The name is often related to the junction at which it is placed. By default, Netedit creates a static signal program that is 90 seconds long. To simply change the phases and times in the program one can save the TLS program and adjust it in a text editor. To program the TLS, one uses a combination of letters, one letter for each state or lane. G is a green signal without having to pay much attention to other vehicles, g for green but must yield for other traffic, y for yellow and r for red light. A TLS program for S9 is shown in the XML code below. By default

the programID for the TLS is set to zero. The programID for S9 is here set to two since an altered version was used.

```
<!-- s9 -->
<tlLogic id="gneJ666" type="static" programID="2" offset="0">
  <phase duration="72"
state="ggrrrrrrrrgggrgggrggrrrrrrrrgggrgggr"/>
  <phase duration="3"
state="ggrrrrrrrrggrryyyrggrrrrrrrrggrryyyr"/>
  <phase duration="12"
state="ggrrrrrrrrggrgrrrgggrrrrrrrrggrgrrrg"/>
  <phase duration="9"
state="ggrrgrrrrrggrgrrrgggrrgrrrrrggrgrrrg"/>
  <phase duration="3"
state="ggrrgrrrrrggrrrrryggrrgrrrrrggrrrrry"/>
  <phase duration="39"
state="gggrgggrrggrrrrrrrgggrgggrrggrrrrrr"/>
  <phase duration="3"
state="ggrryyyrrggrrrrrrrggrryyyrrggrrrrrr"/>
  <phase duration="30"
state="ggrgrrrrgggrrrrrrrggrgrrrggggrrrrrr"/>
  <phase duration="3"
state="ggrrrrrryyggrrrrrrrggrrrrryyggrrrrrr"/>
</tlLogic>
```

A complete TLS program for a simulation scenario contains programs for all the TLIs in the network. This includes the six major intersections, as well as the minor intersections, S2, S7, S8, S10 and the signal regulated pedestrian crossings o1, o2, o3-1 and o3-2 shown in Figure 2.3. A simplification of a complete TLS program is shown below, where the id names are changed to the TLI names for easier understanding, as well as showing the structure but omitting all the states and phase durations for all the TLIs.

```
<additional>
  <tlLogic id="s1" type="static" programID="2" offset="0">
</tlLogic>
  <tlLogic id="s2" type="static" programID="2" offset="0">
</tlLogic>
  <tlLogic id="s3" type="static" programID="2" offset="0">
</tlLogic>
  <tlLogic id="s5" type="static" programID="2" offset="0">
</tlLogic>
  <tlLogic id="s6" type="static" programID="2" offset="0">
</tlLogic>
  <tlLogic id="s7" type="static" programID="2" offset="0">
</tlLogic>
  <tlLogic id="s8" type="static" programID="3" offset="0">
</tlLogic>
  <tlLogic id="s9" type="static" programID="2" offset="0">
</tlLogic>
```

```
  <tlLogic id="s10" type="static" programID="2" offset="0">
</tlLogic>
  <tlLogic id="s11" type="static" programID="2" offset="0">
</tlLogic>
  <tlLogic id="o1" type="static" programID="2" offset="0">
</tlLogic>
  <tlLogic id="o2" type="static" programID="2" offset="0">
</tlLogic>
  <tlLogic id="o3-1" type="static" programID="2" offset="0">
</tlLogic>
  <tlLogic id="o3-2" type="static" programID="2" offset="0">
</tlLogic>
</additional>
```

It is crucial to have made all the choices regarding what way different lanes can turn beforehand, as a later change in the program will make the TLS program that was altered unable to run because of state changes. This thesis focuses on two calculated signal programs for the scenarios 100_33 and 50_150 as well as the existing traffic signal program on site, IRL. Netedit's own generated traffic signal program is also incorporated and tested. There have been no alterations done to Netedit's TLS. It is used as is.

2.6.4 Netedit default traffic light signal program

When a traffic light is placed in a junction, in Netedit, there will be an automated traffic light program created [10]. This program follows rules set by SUMO but may not be accurate to what is used in real life at that specific location. The general rules for the automated TLS program are based on a four-arm intersection, without regard to pedestrian crossings. There are four green phases that go as follows:

1. Straight forward for traffic from north and south, along with right turning traffic.
2. Left turns, for lanes that are dedicated to left turns specifically for traffic from north and south.
3. Straight for west and east bound traffic, along with right turning traffic.
4. Left turns for outgoing traffic from west and east.

The traffic light cycles have a default cycle time of 90 seconds, and all green phases are followed by a yellow phase. If it is a four-arm intersection the straightforward phase has a green light for 31 seconds. The speed limit for this area is below 70km/h which makes left turns allowed during the same time as oncoming traffic can drive straight. The vehicles that want to turn left must yield to oncoming traffic. If there is a specific lane for left turns it will

get an additional six seconds of green light after the traffic going straight from the same direction has turned red.

2.6.5 Detectors

When the overall traffic map is done it is time to implement the detectors that will collect the data that is wanted out of the simulation. In SUMO there exist three different types of detectors, with different values collected. In this case it was of importance to collect information about how many vehicles that passed a crossing given a certain time, as well as information about how long vehicles had to wait before they could pass. The detector that does this is called Detector_E2 [11], more specifically called the Lanearea detector. This detector can resemble a vehicle tracking camera. A lane area detector is given a length attribute, defined by a starting position, pos, and an end position, endPos. The detectors were placed on all incoming and outgoing lanes to the six main intersections of the system, which can be seen in Figure 2.33 as aqua blue rectangular shapes. On the incoming lanes they are set to begin slightly after where the road increases its number of lanes leading to the intersection and end at the stop line at the junction. If no such lanes are added, the detectors start between 20 to 50 meters away from the intersection depending on the overall layout around the intersection. The detectors on the outgoing lanes are placed so that they start at the beginning of the outgoing lanes and end a minimum of 10 meters away.

The detectors can be specified to count different kinds of vehicles. In this case there were detectors to count vehicles and detectors to count only buses. The E2-detector counts vehicles that touch the start position of the detector, vehicles that start or end their route on the detector and vehicles that pass through the detector. The output given of an E2-detector keeps track of all vehicles that are currently running in its area and has attributes to help measure queues.

2.6.6 Traffic Assignment Zone (TAZ)

Traffic assignment zones are a collection of edges used to make route designation simpler. They are valuable when creating traffic in and between areas of the road network. Here TAZs are placed on all incoming and outgoing fringes leading to the TLS junctions. A total of 22 TAZs for this system. A TAZ must be selected to be either a sink or a source. A sink being an edge from where vehicles will depart from the system and source being where vehicles will arrive to the system. The XML code in this section shows how one ingoing and one outgoing TAZ are structured.

Each TAZ has an id and a list of depart and destination edges. The edges are given a weight representing a probability, to determine how much traffic

that shall enter or leave the map from them. The shape and color parameters are optional.

```
<taz id="street_in" shape="x and y coorinates" color="green">
  <tazSource id="gneE###" weight="1.00"/>
  ... further source edges ...
  <tazSink id="gneE###" weight="0.00"/>
  ... further destination edges ...
</taz>
<taz id="street_out" shape="x and y coorinates" color="red">
  <tazSource id="gneE###" weight="0.00"/>
  <tazSink id="gneE###" weight="1.00"/>
</taz>
```

2.6.7 Creating traffic

When thinking of rush hour traffic, the first thought is of queues with a lot of vehicles and little movement. In this case it was attractive to focus on three different kinds of transportations when creating traffic for the network. The everyday drivers, the commuters driving to or from work and the public buses, with their set routes and timetables.

2.6.8 RandomTrips

SUMO has a predefined python script that helps create vehicles with random routes called randomTrips.py [12]. With this it was possible to generate a lot of vehicles that enter and leave the simulation randomly and with random routes. This randomness was useful to represent everyday drivers out and about in the network. RandomTrips can also be given a set of information to create traffic with a more desirable distribution. Even though the randomness was wanted it was still desirable to make traffic drive through the bigger streets that lead to the six TLS junctions.

By running the script with a set of selected attributes, see tables in Section A.3 *Attributes for XML code*, it is possible to create a route-file with many vehicles taking random routes through the network. The routes created are static. All edges from start to finish are stated in each route. Because of this it is possible that one later alteration of the network will make the routes invalid and the simulation unable to run. With the help of the attributes available for randomTrips it is possible to affect the distribution of the routes. One wanted behavior was to make vehicles choose routes along bigger roads as well as entering and exit the system through the fringes associated with the six big intersections.

Running the Python command `randomTrips.py`, will create routes where vehicles are required to drive a minimum of 500 meters in the system. To get a bigger number of vehicles to enter and/or exit the network from the

fringe the fringe-factor value was set to 40. This value increases the probability of creating routes that begin and/or end with an edge at the fringe 40 times. To make most vehicles drive through any of the major TLS intersections the three attributes, L, fringe-threshold, and speed-exponent were used to weigh the traffic to use roads with more lanes and with speed-limits equal to or above 40 km/h which translates to 11.11 m/s. To create more than one simulation for each scenario it was decided to create two other randomTrips files. The difference between them is the use of seeds. If not changed the routes created would be the same as the one first created. Seeds help generate pseudo-randomness to the distribution of vehicles. To keep it simple the seeds 33, 66 and 99 were chosen. Below is an example of the command used to generate vehicles with the randomTrips script.

```
python randomTrips.py -n v5_7.net.xml -r seed33Random.rou.xml
--seed=33 -b 0 -e 7200 --period 1.0 --binomial 400 --vclass
passenger --vehicle class passenger --min-distance 500.0 --
fringe-factor 40.0 -L --fringe-threshold 11.00 --speed-
exponent 11.11 --fringe-start-attributes "color=\"255,0,0\"" -
-trip-attributes="departLane=\"free\" departSpeed=\"random\"
departPos=\"random_free\" color=\"0,255,0\" length=\"4.00\"
minGap=\"2.00\" maxSpeed=\"25.00\" speedDev=\"0.1\"
accel=\"2.6\" decel=\"4.5\" sigma=\"0.2\" minGapLat=\"0.5\"
laneChangeModel=\"SL2015\""
```

The depart attributes following trip attributes decide where vehicles will be implemented on a road section. The values chosen as inputs are there to increase the number of vehicles that are inserted on roads with multiple lanes. Other attributes used are the ones for describing the vehicles. Their length, what type of vehicles they are, how fast they can accelerate, brake, as well as the maximum speed they can reach. The attributes sigma, minGapLat and laneChangeModel describe the behavior of the drivers. Sigma tells how likely the driver is to follow the rules, with 0 being always and 1 being never. The minGapLat decides the minimum distance for how close vehicles can be next to each other in meters. The laneChangeModel SL2015 [13] is an available script from the SUMO package that is created to resemble how Chinese drivers use and change lanes.

2.6.9 Flow for cars

Flow [14] is another way of implementing vehicles in the simulation. Instead of creating a set route for the vehicle, flows can be more dynamic. Here flows are created to drive between different traffic assignment zones (TAZ) [15], to represent commuters that travel across the network. At least one TAZ for incoming traffic and one TAZ for outgoing were created for each TLI. Every TAZ represents edges at the fringe in connection to the TLI. Some flows also have via-edges specified to better control which streets to use. Through trial

and error, a satisfactory flow of rush hour commuting traffic was constructed. SUMO has a Python script to make an XML file with flow from a so-called OD-matrix which basically is an origin-destination table. After getting used to the process, it became clear that vehicles in flows are set off by the help of classic probability. Hence there is no magic formula in the Python script. The XML file may be constructed much more easily using a spreadsheet and then let the built-in functions produce the XML code. The spreadsheet table that was used is shown in the appendix.

Below the XML code to produce commuter 9 is shown. It is one single line of code of almost 100 in total. These lines of code are wrapped inside tags just like in html code. Commuter 9 is a set of vehicles that starts, ends, and passes the same via edge between begin and end time values. So, its first vehicle starts at $t = 900\ s$ and enters the network from a fringe edge on the ramp north of S1 and exits at a fringe edge south of S6.

```
<flow id="commuter9" begin="900" end="8100"
probability="0.013889" type="commuter" fromTaz="taz_s1r_in"
via="gneE408.17" toTaz="taz_s6s_out" departLane="best"
departSpeed="random" color="orange" />
```

The attributes used for flows are primarily the same as for randomTrips with the exception that flows will choose its route by itself, with help of the start, end and via-edges given. The route it takes to get there is not specified. Flows are therefore more dynamic in the way that they will change their routes if for example, there are queues ahead. Therefore, flows are used to represent drivers that are regular commuters in the area.

2.6.10 Public transport

When creating the public buses flows were used again. Flows allowed different bus lines to traffic the roads and make stops according to their time schedule. All buses were given the same parameters for speed, acceleration, deceleration, length, gap and random speed variation. To guide the bus at least three or more edges had to be stated, i.e., edge to enter, edge to exit and edges where there are bus stops for that bus line. The duration for stops is set to 15 s for all buses. With <flow> each vehicle must individually choose its own path, which suggests there will be some variation of paths. The tendency to change lanes seems to be higher among flow vehicles compared to randomTrips-vehicles.

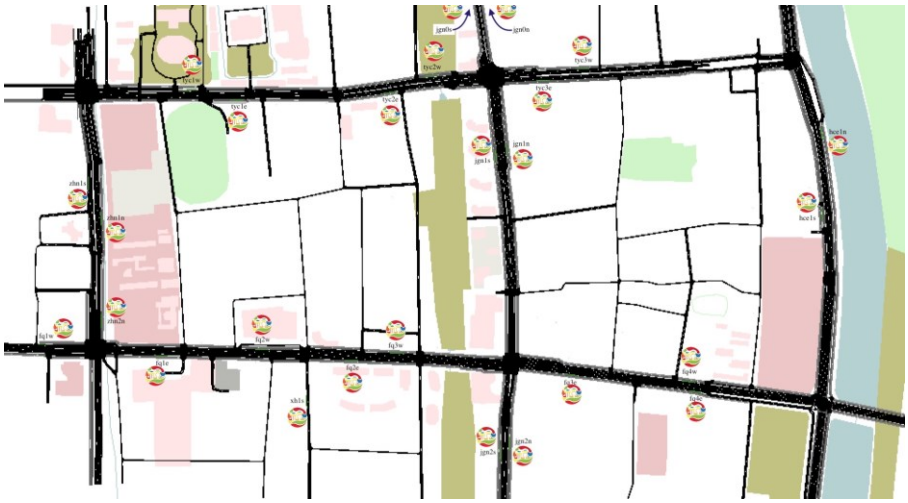


Figure 2.32. All bus stops, each marked with the Hangzhou logotype.

Figure 2.32 shows all bus stops within the modelled area. The example below shows bus line 68 which enters the network north of S3 and exits south of S7 and has green color. Line 68 has a total of 12 buses per hour between $t = 60\ s$ and $t = 3\,360\ s$.

```
<routes>
  <vType id="BUS" vClass="bus" accel="2.6" decel="4.5"
sigma="0" length="12" minGap="3" maxSpeed="70" guiShape="bus"
speedFactor="normc(1,0.1,0.2,2)" />
  <flow id="68s3s7" color="green" begin="60" end="3360"
number="12" type="BUS" from="gneE532" to="gneE510">
    <stop busStop="busStop_gneE532_0_22" duration="15" />
    <stop busStop="busStop_gneE629_0_24" duration="15" />
    <stop busStop="busStop_-gneE115_0_0" duration="15" />
    <stop busStop="busStop_gneE509_0_20" duration="15" />
  </flow>
</routes>
```

The final step before running the simulation is to create a main settings file, which is called the `sumocfg` which stands for sumo configuration file. In this file one specifies the road map to be used as well as the traffic-route files and additional files as can be seen in the XML code after this section. The additional files here used are a map of geometrical shapes in the area, `area.poly.xml`, a file where the busstops have been marked, `busstopsV5.add.xml`, as well as the TLS program and files with the detectors to be used. All files used are XML files.

```
<configuration>
  <input>
    <net-file value="v5_HZmap.net.xml"/>
    <route-files-value="busFlow7200sV5sorted.rou.xml,
commuters_flow.rou.xml, randomTrips.rou.xml"/>
    <additional-files value="area.poly.xml,
busStopsV5.add.xml, TLSfinal_100_33.add.xml,
AllDetectors.add.xml, v5_TAZ.taz.xml"/>
  </input>

  <time>
  <!-- Set start and end time -->
    <begin value="0"/>
    <end value="10800"/>
  </time>

</configuration>
```

2.7 TraCI

SUMO simulation can be controlled via a traffic control interface called TraCI [16]. By importing `traci` in Python the simulation can be controlled for example with the command `traci.simulationsStep()` inside a for-loop, which for every iteration advances the simulation process one second. Python's traci module has several methods to keep track of what kind of vehicles that have been on what street (set of edges) and even what lane.

As previously mentioned, traffic detectors were used during simulations, these can be seen in every direction close to the intersections in Figure 2.33. A detector keeps track of which vehicle currently is on it. Therefore, each detector must be represented by a set in which new vehicles are added on each timestep (i.e., second). On given intervals the total amount in each detector set must be saved for report and emptied for a new time interval.

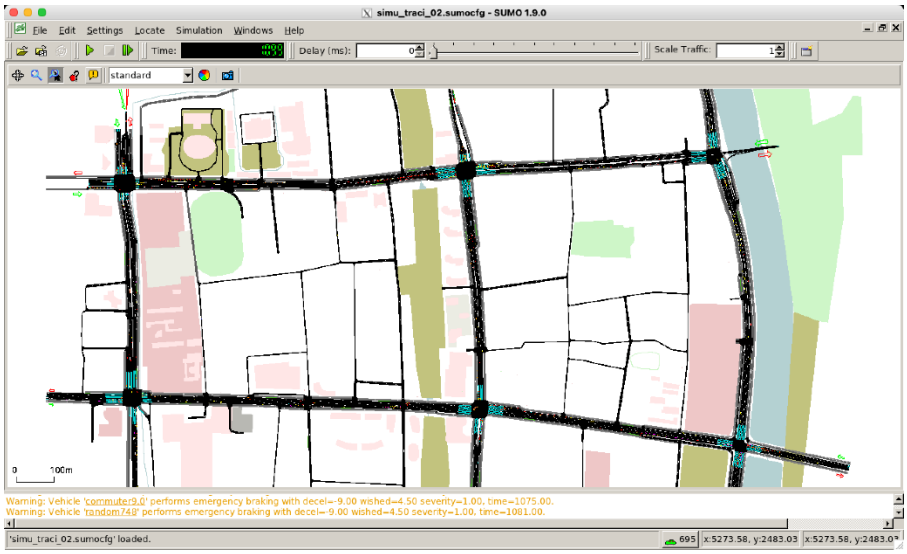


Figure 2.33 SUMO during run. Detectors show as turquoise field.

With a set of combined method calls, the control program could get instant information on a certain street's traffic saturation. There are methods to check which vehicle id's that are on a certain lane, the lane's length and through the given vehicle id also the vehicle length and what kind of vehicle it is. By adding all lanes on each edge on a street, a traffic density can be calculated. In a similar manner the `traci` module has methods for checking which specific vehicle id's that have not been able to be loaded into the system via a fringe edge. This gives information on the total number and what kind of vehicles that are queuing to enter onto which street, thus creating a hunch of the level of overload. However, all vehicles do not enter and exit through the fringe. Approximately 150 vehicles start and end inside the fringe.

The above methods and several new created ones were used to control both the simulation process and to compile its output. The outputs produced by the code were: a useful terminal text, a compiled copy of the terminal output written into a text file and a JSON[5] file representing the detector data.

[5] JavaScript Object Notation

3 Analysis

Running the simulations in SUMO revealed some troublesome behaviors. Here are some of them explained as well as possible solutions mentioned.

3.1 Several XML files vs TraCI JSON output

SUMO's default detector output is an XML file containing a lot of data with some of it being irrelevant for this report. The data in turn need to be converted to an Excel file to improve the reading comprehension. Each intersection had one detector file for incoming and one for outgoing traffic. In total 12 XML files had to be converted to 12 XLS files and thereafter compiled in order to calculate each intersection's specific traffic flow in each time interval. This is a time consuming and confusing process. If for some reason new simulations were to be undertaken, it will then be followed by such a time-consuming conversion and compiling process.

Consequently, it felt useful to invest more time on SUMO's traffic control interface and the Python code needed to put it into action. It can be discussed whether implementing Python code did reduce the total amount of time spent on simulation and compiling the results. However, both tasks were neatly automated, which made it possible to perform far more simulations than would otherwise be the case.

As previously mentioned, the Python code produced a JSON file. To compile the detector data in this JSON file, yet another Python code was constructed, which can read and compile from multiple JSON files and write the same number of JSON files as output in a few seconds. This part automated and saved time.

All Python code are shown in A.1 *Python code*.

3.2 Trouble with driving patterns.

When introducing many vehicles to the simulation there were several weird driving behaviors noted. The ones mentioned below were the ones that were most important to correct.

The use of randomTrips.py generated many vehicles driving on random routes. It is excellent when it is important to quickly introduce traffic to the simulation. But the routes are static and the routes that are created can cause trouble since they do not identify areas that might not be suited to make U-

turns at. A vehicle will not have a driver's sense in that matter. In this case many vehicles wanted to make U-turns on a road right after a traffic light intersection. This caused undesired queues because of the vehicle that stood still in its lane waiting for a gap to open to the next edge on its route. To work around this issue many vehicle routes were altered to make their turns at the nearest TLI.

A general problem that was noticed during the trial simulations was that even if two parallel roads could be utilized for a right turn in a junction, only one was used. This problem caused trouble specifically for the vehicles arriving at the S1 junction from the north fringe, since the queue that grew hindered other vehicles taking different routes from entering the system. Here the solution was to redirect several vehicles to use the otherwise unused road.

Since the road network built is quite complex, with multiple lanes and edges with their own set of regulations, it seems like all rules for use of way were not followed when routes were created. It could be seen a lot at the S1 junction where vehicles that did not enter the right lane could not continue their route because that specific lane did not allow turns to their desired direction. For flows it was possible to go around this issue by adding via-edges. A via-edge tells flows that if the vehicle wants to go from A to B it has to drive by this edge to get there. For vehicles created from the randomTrips-script the solution was to change the faltering edge in the XML file to the desired one.

3.3 Teleportation

Teleportation [17] is a built-in feature that is used to move vehicles that for some reason are stuck. It can be that there are queues that will not move because of a vehicle that blocks the front position in a lane or a scenario where a collision has occurred. Often the reason that prohibits a vehicle to continue its route is one of three scenarios:

- It has ended up on a lane that has no connection to the next edge on its route.
- If the vehicle is coming from a road that must yield before entering a higher prioritized road but cannot find a gap to do so.
- It is stuck in a traffic jam with no possibility to continue its route, as can be seen in Figure 3.1.

To keep the simulation running, SUMO will force a teleportation of the vehicle causing the block. This teleportation occurs when a vehicle has a velocity below 0.1 m/s for 300 seconds. When a vehicle teleports, it is

removed from the network only to be inserted again, when possible, at the next step of its route. Figure 3.1 shows two instances during a timestep in the simulation that traffic jams have occurred at intersection S3, the center picture in the top row, and S6, the left picture in the bottom row. These types of traffic jams with multiple vehicles involved can require vehicles to teleport to be able to solve the jam.

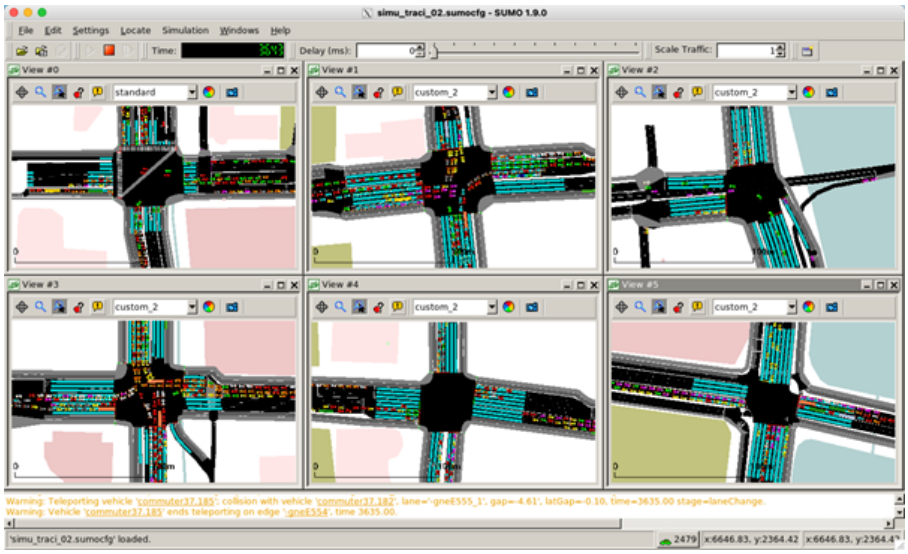


Figure 3.1 A SUMO run with two jams at t = 3,643 s.

3.4 SUMO versions update

After the initial network map was built and the creation of traffic began it was noted that SUMO was released in an updated version. This release was attractive because it enabled further options when for example implementing traffic.

3.5 Netedit's program not an option

Netedit creates a default traffic light program as soon as a TLS is implemented. At first this program was supposed to be used as one TLS scenario to be compared to the three scenarios previously mentioned. When looking at the TLS program Netedit created, it was evidently not a program that could be used in real life. At many of the TLS intersection it was evident that the TLS programs would cause conflicts between drivers and directions as can be seen for S1 in Figure 3.2. Therefore, the Netedit scenario was omitted.

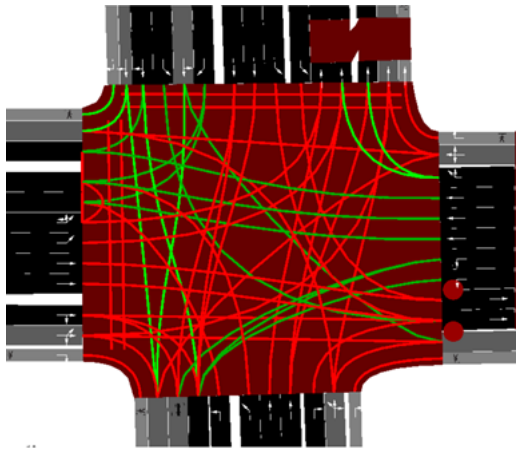


Figure 3.2 Netedit's TLS program overview for S1.

4 Results

Each simulation has been conducted three times, using the random pseudo seeds 33, 66 and 99. The only difference between the simulation scenarios IRL, 100_33 and 50_150 are their traffic light operating programs. To create traffic jams to be handled, intended overloads of the system were implemented in the form of loading too many vehicles. Despite the intersection parameter called *keep clear* being set to true, meaning that the junction should not allow vehicles to enter if there is a possibility for a queue inside the junction itself, cumbersome jams occurred inside intersection areas which led to complete stops. These were typically resolved by SUMO teleporting vehicles to the next available edge.

All results are an average of the three pseudo random simulations. Variances are calculated with Bessel's correction. Average of deviations, $d(\mu)$, are calculated according to the RMS formula.

4.1 Departed vehicles

Every simulation loads on average 20,439 vehicles. However not all of them get implemented, i.e., departed. Almost 6 800 of them end up in a queue outside a fringe entry. There is simply no available free space on which to enter in accordance with their dedicated routes. The 100_33 scenario has the highest value of departed vehicles, which may suggest the traffic light signals during this simulation worked more efficiently. It differs 2.9 percentage points, corresponding to 593 vehicles, between scenarios 100_33 and 50_150, which has the lowest level of actual departed vehicles. See Figure 4.1.

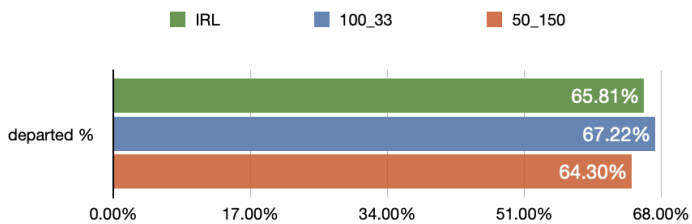


Figure 4.1. Departed vehicles as a percentage of loaded vehicles.

4.2 Arrivals, collisions and teleportations

Since all simulations were cut off at t = 10 800 s, some 26 - 28 % of the departed vehicles were still en route, i.e., had not yet reached their

destinations. While 465 vehicles (3.47 % of departed) in IRL are involved in collisions, the numbers for 100_33 and 50_150 are 432 (3.15 %) respective 451 (3.44 %). The number of teleports gives an idea of how cumbersome the traffic environment was. Figure 4.2 shows vehicles that have arrived at their destinations, the number of teleports and the number of collisions as a share of all departed vehicles. The scenario 100_33 simulation also excels with the least number of teleports. In real numbers it means 100_33 has 674 teleports less than IRL, which has the most occurrences.

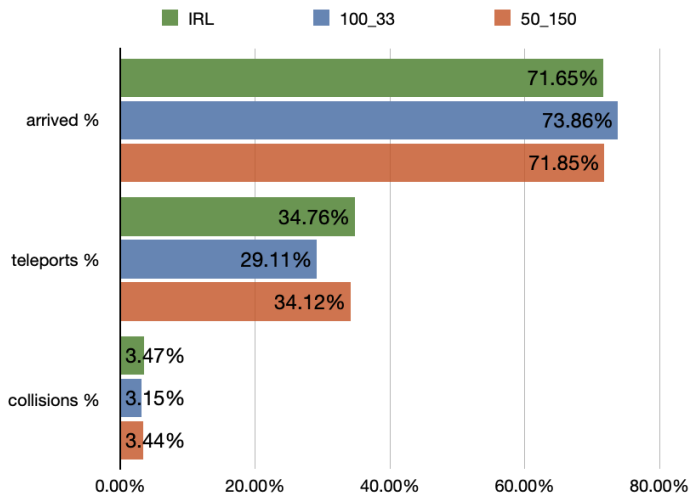


Figure 4.2. Vehicles that have arrived, been in collisions and teleported.

Note that all percentages in Figure 4.2. Vehicles that have arrived, been in collisions and teleported. are with respect to the departed number of vehicles in that simulation scenario.

4.3 Resolving traffic jams through street evacuation

Figure 4.3, Figure 4.4 and Figure 4.5 each shows the corresponding average street evacuation, μ, and deviation, d(μ), during each 900 s interval within the system during simulation. All values are the average of three simulations of each scenario.

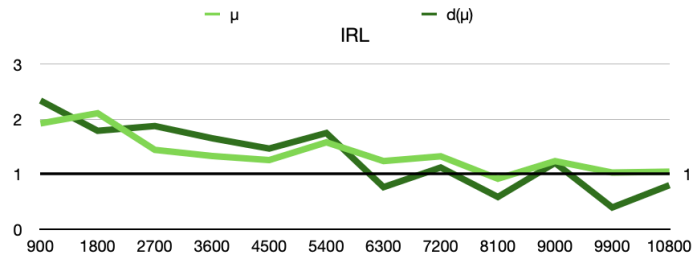


Figure 4.3. System evacuation and standard deviations of IRL.

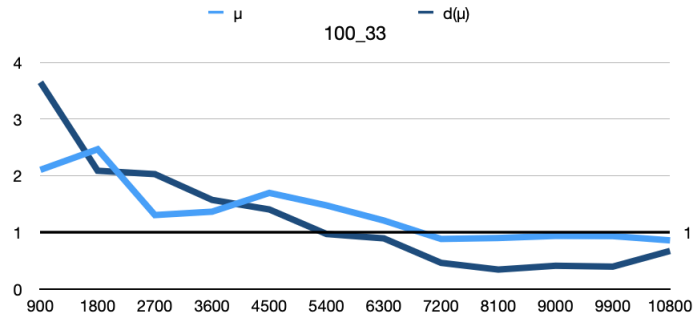


Figure 4.4. System evacuation and standard deviations of 100_33.

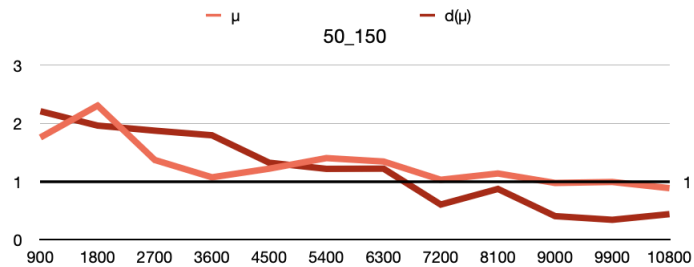


Figure 4.5. System evacuation and standard deviations of 50_150.

Scenario	Time	900	1800	2700	3600	4500	5400	6300	7200	8100	9000	9900	10800	Avg
IRL	μ	1.92	2.11	1.44	1.33	1.26	1.58	1.24	1.32	0.92	1.24	1.03	1.05	1.37
	d(X)	2.34	1.79	1.87	1.65	1.46	1.75	0.76	1.12	0.58	1.20	0.39	0.80	1.43
100_33	μ	2.10	2.47	1.31	1.37	1.70	1.48	1.21	0.88	0.90	0.94	0.94	0.86	1.35
	d(X)	3.65	2.09	2.03	1.57	1.40	0.97	0.90	0.46	0.34	0.41	0.40	0.67	1.56
50_150	μ	1.76	2.31	1.37	1.07	1.22	1.40	1.34	1.03	1.14	0.98	0.99	0.89	1.29
	d(X)	2.21	1.96	1.88	1.80	1.32	1.22	1.22	0.60	0.87	0.40	0.34	0.44	1.35

Table 4.1 E(μ) and d(μ) for each 900 s interval, on which figures 4.3 – 4.5 are based. The column 'Avg' refers to the overall average values of each entire simulation.

Here evacuation value is calculated as:

$$evacuation = \frac{incoming + existing - outgoing}{existing} \frac{(vehicles)}{(vehicles)}.$$

An evacuation value below 1 means actual evacuation. In the above evacuation formula, there is a risk of zero division if one street is completely empty. To prevent this, the factor $existing$ is an actual average of vehicle counts performed every 150 s interval within each 900 s interval. Exact values are found in Table 4.1. The interval counts also make evacuation values relative to average street occupancies rather than the full street capacities.

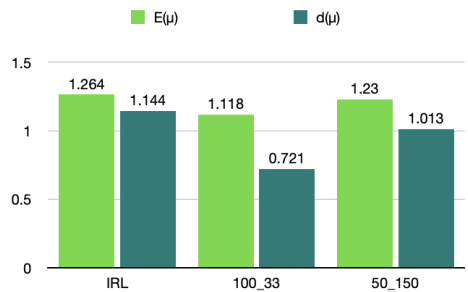


Figure 4.6. Average vehicle evacuation during t = 75 - 135 min.

Figure 4.6 shows average evacuation value and standard deviation during the last full hour with vehicles being loaded into the simulation. After 2 h 15 min there are no new vehicles put into the simulation, meaning that the ones already in place or pending at the fringe will be the ones that will run until all final destinations have been reached. During this time period the 100_33 simulation has the lowest average evacuation value, $E(\mu)$. None of the simulated scenarios at this time period has actual average evacuation, because the values are all above 1. Scenario 100_33 also has the lowest deviation $d(\mu)$, which mean it also have the smallest evacuation differences among the 14 streets.

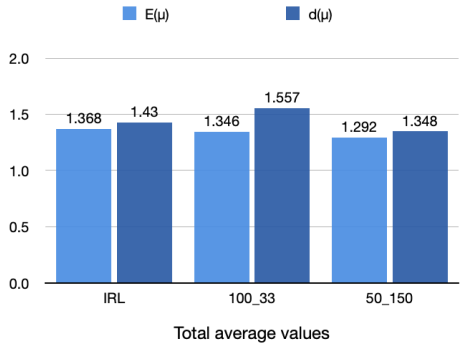


Figure 4.7. Average vehicle evacuation during t = 0 - 180 min.

If the entire simulation's average evacuation values are compared it can be seen, in Figure 4.7, that scenario 50_150 has the lowest value for $E(\mu)$ as well as for $d(\mu)$. The scenario 100_33 also has lower $E(\mu)$ compared to IRL, however with a higher $d(\mu)$. The latter is most likely due to the high initial value as is shown in Figure 4.4.

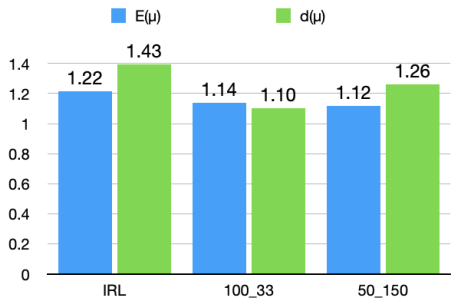


Figure 4.8.Average vehicle evacuation during t = 45 - 180 min.

In Figure 4.8 the first 45 min of simulation is deducted from the total simulations. In this case 50_150 still has the lowest overall average evacuation value. It also shows that 100_33 has the lowest deviation value.

4.4 Average street occupancy

Figure 4.9 shows that streets in scenario 100_33 on average fill up slightly faster, but then also empty earlier and a bit more than in the other scenarios. It is also seen that streets in scenario IRL fill up slower and for longer time, before they slowly are emptied - compared to both 100_33 and 50_150.

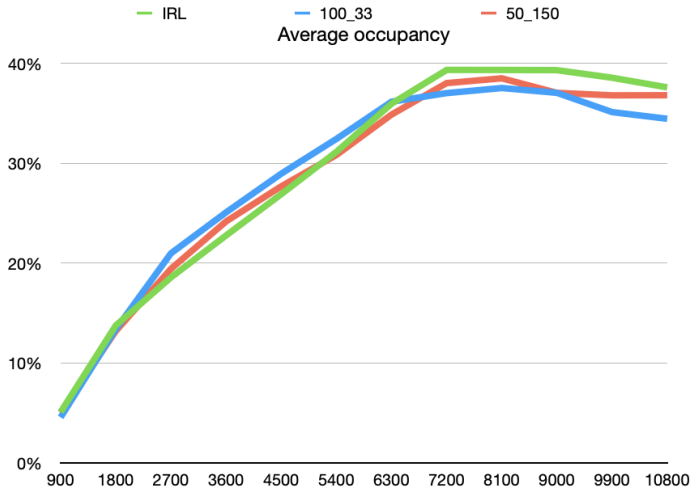


Figure 4.9. Average street occupancies during simulations.

4.5 Evacuation seen as a quota of incoming and outgoing vehicles

Another way to interpret evacuation is to, per intersection, calculate the quota

$$Q = \frac{incoming\ (vehicles)}{outgoing(vehicles)}.$$

Again, values below 1 mean actual evacuation. This interpretation is shown in Figure 4.10. The average quota from each simulation is seen in Figure 4.11, as an average per time interval of all intersections and per simulation scenario. This perspective does not seem to show any evacuations at all. Over a time period, more vehicles enter an intersection than leave the intersection, if viewed this way. We all know that in a sense this is not possible.

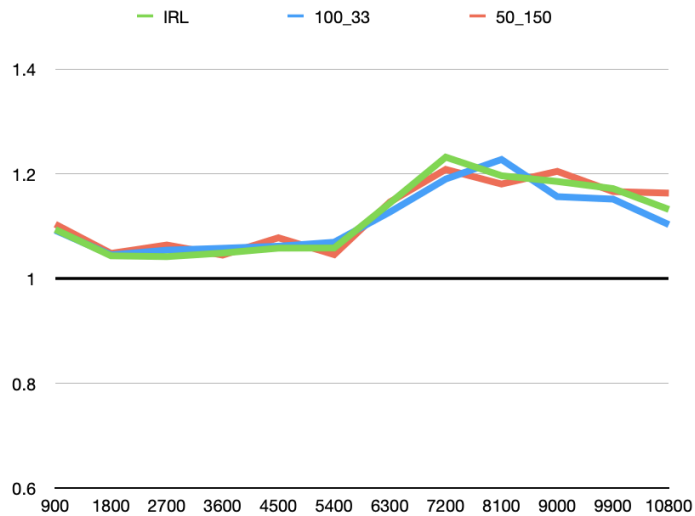


Figure 4.10. The average quota of incoming/outgoing traffic.

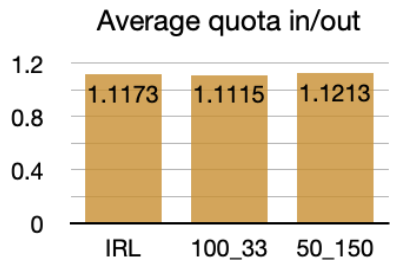


Figure 4.11. Average quota of the entire simulations.

4.6 Problematic streets

For some reason collisions in the simulations occur more frequently on some streets compared to other streets. Some very busy streets have “only” one or no collision. Figure 4.12 shows collisions per street and scenario. The sums per scenario are IRL: 180, 100_33: 164 and 50_150: 166 collisions. Each collision includes one or two vehicles. It i seen that the same pattern applies for all three scenarios. This may indicate some built-in flaws in either the model or in real traffic, since the model reflects traffic situations from the real city of Hangzhou. It is worth emphasizing that the collision results of the simulations do not correspond to reality.

In the simulations the vast number of teleports are not reported as detailed as the collisions. Nor are data on departures and arrivals compiled other than

as total sums. These data can, however, be examined with a few more lines of Python code.

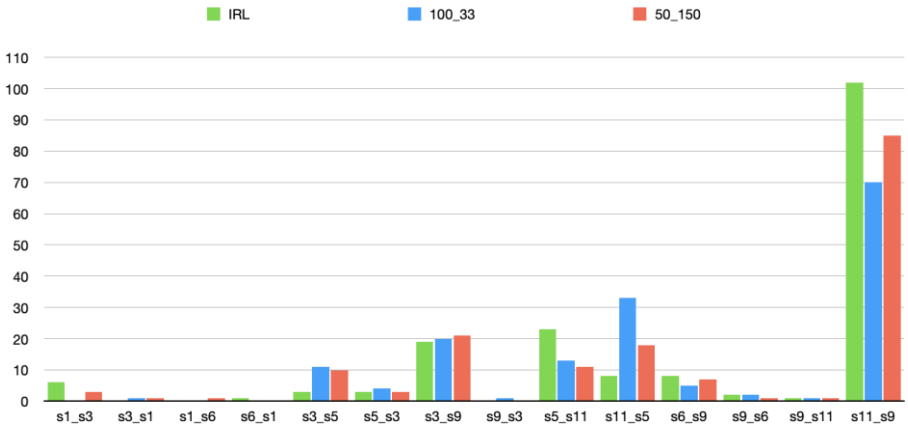


Figure 4.12. Number of collisions on each street.

5 Discussion

The results from the chosen simulation application, SUMO, is as close as they get to reality without changing all affected traffic signals, even with the flaws in SUMO. Thus, the simulation results are as realistic as is currently possible.

5.1 Are TLIs already optimized?

One obvious and possible problem with our trials is that we did only record the signal programs during rush hour traffic. It might be possible that the city has already applied specific signal phases at this time compared with other time periods during a day or a week.

5.2 Possible incorrect observation on site

The TLI S11 is elevated and restricted from pedestrians, which makes it hard to observe. We had to record signal lights with a tele zoom lens from a distance. This intersection's layout could be found mainly from satellite maps online. First after simulations and while writing this report, it became clear that the before mentioned four cases of TLI flow are not applicable in S11. The layout for S11 can be seen in Section A.1 General layouts of TLI. There exists one incoming lane, for three different directions, that allows traffic going straight and turning left at the same time. Hence it is not possible for traffic from opposite sides to have a green light at the same time. Traffic can thus only flow from one direction at a time e.g., like round-robin format. From the start this TLI should have a phase pattern of its own within the MATLAB calculation of optimal flow. However, it did not, and as such this may have an impact on the modelled traffic.

5.3 Dismissal of SUMO's built-in solution

During early analyses of simulations, it was discovered that the traffic signal patterns made automatically by SUMO software, Netedit, were unrealistic and cannot be put into real action. In just one phase there are several conflicting vehicle paths. So even if this simulation performed the best flow at the time in terms of numbers, it had to be disqualified from the comparisons of these trials. The reason for the chaotic traffic light subprograms that Netedit came up with are believed to be caused by the non-standard four-way intersections that are present in the model. It might have been better if the network map was further simplified. But the aim was to build a network map that represented the real and present structures that were observed on location.

Hence the logic for Netedit could not be properly applied here, see Section 2.6.4 *Netedit default traffic light signal program*.

5.4 Strange traffic behavior

The SUMO simulations produce strange traffic behavior repeatedly. If this strange behavior occurs due to inexperience with the program or if the program currently does not have a way to handle these behavior patterns are unknown. What happens often is that a vehicle stops and as a result blocks other vehicles, who's drivers in turn are not inventive enough to steer pass the obstructor.

A common blocking is for example when a driver intends to change lanes and there is not an available space gap. As a result, the driver simply stands still and waits for a gap, which of course effectively stops all traffic on the vehicle's current lane. Another example is when all vehicles in the same direction choose the same lane, despite the existence of available empty adjacent lanes. If vehicles were to spread out more even among available lanes, queues would in a sense be shorter, which in turn would allow more vehicles to enter the street. In many cases a teleport will resolve an obstruction due to excessive wait. A smoother driver's behavior would be desirable.

Some of the problems experienced may have to do with behavior settings for driving in intersection, lane changes and overtaking. Perhaps the authors do not master these settings fully. By chance it was discovered that setting all traffic in intersection in yield mode, prevents and faster resolves otherwise blocking conflicts. There may also exist solutions for obstructive driving behavior along street lanes, which are yet to be discovered.

5.5 Yielding behavior at TLI's

During the SUMO simulations another strange behavior was noted. At the intersections many vehicles stopped inside the TLI even if the signal showed green. A lot of james occurred and did not resolve themselves until a teleportation was don even if there was enough space for the vehicles to undo the jams themselves. A multitude of changes were done to try to solve these problems. In the final hour a a solution was found. By making changes in the TLS programs it was seen that the bavior changed. First, all the g's were changed to G, meaning that all vehicles driving into the crossing had the right of way. This change made the problem worse, but it had an effect. Therefore another approach was made. By changing the G in the TLS program to a g, many traffic jams and odd stand stills were resolved. That is, by making all

the drivers have to look out for each other none took the right of way for granted.

5.6 Empty streets

The evacuation formula is based on streets fully saturated with vehicles. It may produce negative values if there are no, or few vehicles and a larger number of vehicles leave than enter the street. The traffic density on some streets were close to zero at some time intervals during all simulations. The reason seems to be both jammed intersections and that the script randomTrips.py did a poor job spreading out random cars and routes everywhere. Therefore, another measure called quota was calculated. This is simply the quota of incoming and outgoing vehicles.

As mentioned in Section 4.3 *Resolving traffic jams through street evacuation*, the variable existing, which acts both as a term and a denominator, has been calculated on each street during each entire simulation at every 150 s interval within the larger 900 s interval. Each count, except for all zero counts, were put in a list from which the average was finally used as the variable existing. The probability of all existing counts being zero is also close to zero. This way divisions with zero were avoided.

5.7 SUMO script inadequacy

The SUMO simulation software package also has software to occupy the model with desired forms of traffic. The Python code randomTrips.py gave us a notion that it would be able to deploy cars randomly across the model and as such having various destinations. We tweaked the existing parameters, to make it as random as needed. But many vehicles with roughly the same origin have the same destination and at the same time. In one perspective this is what forms digested traffic, but it may as well be that a certain area will be forming a hub of digested traffic merely because too many drivers at the same time chooses to pass through that same area.

5.8 High numbers of collisions and teleports

Finding the reason for the relative high number of collisions and teleports, is beyond the scope of this report. Still, it raises many questions that one does want to find the answer to. So far it is only known that collisions are overrepresented on at least five of the 14 streets. If the model were to be further refined, surely the high number of collisions on those streets were to be investigated and if possible, resolved with other settings. The high number

of teleports are also worth investigating, in order to further enhance the model and the simulation of traffic.

5.9 Determine what resolves congested traffic

The level of departures and arrivals ought to show the efficiency of the service nodes, i.e., the traffic signal intersections. Comparatively 100_33 has the best throughput, followed by IRL and then 50_150. The difference between 100_33 and IRL is more than 5%, in terms of number of vehicles, so 100_33 has a significant advantage over IRL.

The number of collisions and teleports point toward complex problems, which are likely to worsen the congestions that arise. All scenarios show the same tendencies for on which streets collisions occur. Still 100_33 has significantly lower occurrences of collisions ($\Delta > 7$ %) as well as teleports ($\Delta > 14$ %) compared to IRL.

The vehicle saturation (occupancy) on streets from one time interval to another shows if an increase or decrease has occurred. During these simulations a pcu (personal car unit) is thought to occupy 150 % of its length during standstills, i.e., including a longitudinal safety gap. The quota 100/150 shows that a 66.7 % occupancy can be considered as full saturation. The average occupancy exceeds 35 % at some point in all simulations. Some streets have occupancy levels close to 60 % at the time of count, while some barely makes it over 10 %.

The street evacuation formula, shown and explained in Section 4.3 *Resolving traffic jams through street evacuation,* indeed gives a value to whether the number of vehicles is increasing or decreasing on a street. All simulation scenarios mostly show an increase of vehicle amounts. At some point after t = 2 h 15 min (8100 s) a decrease is expected, since no new vehicles are loaded into the simulation. Comparing the three figures 4.3, 4.4 and 4.5, shows that the 100_33 scenario has a strict decrease from t = 1 h 15 min and until t = 2 h from which it actually evacuates streets of vehicles throughout the remainder of the simulation. The deviation also follows this trend and stays steadily low until last 15 min where deviation of average evacuations in all scenarios slightly increase.

If the total average evacuations are compared it will show that 50_150 performs the lowest evacuation value and lowest deviation. Even if the first simulation hour is deducted, shown in Figure 4.8, 50_150 will show the lowest average evacuation value, but in this case 100_33 has the lowest deviation value. This can be explained by the high initial values of evacuation, μ, and deviation, $d(\mu)$, in scenario 100_33.

When focusing on the last full hour with vehicles still being loaded into the simulation model, as Figure 4.6 does, scenario 100_33 undoubtedly has the lowest evacuation and deviation values. This also coincides with the time interval when 100_33 starts to increase evacuation and at the same time decrease deviation among its 14 streets. The difference is significant from both IRL and 50_150. This hour may be regarded as the peak pressure period.

5.10 SUMO as a simulation software

In earlier sections there have already been a few points regarding the troubles we faced with SUMO during the simulations. It is easy to conclude that the software doesn't work perfectly. But there are many things to take into consideration here. None of us have worked with traffic simulation programs before hence we have nothing to compare SUMO with. That said, it might be well possible that the network map would have worked better if more simplifications were done. Since the network map was built to accurately represent the traffic network seen on location it requires more processing and more choices to be made during simulation. The complexity of the network map that we used may be the cause of some of the traffic jams that occur as well as collisions and teleportations. It might also be one of the reasons that the Netedit default TLS did not manage to create a program that would work for the individual intersections.

Working in Netedit to build the network was straight forward when outlining the roads. The area we chose to replicate were a small part of Hangzhou, but it was still very time-consuming to build. Much time was spent on adding the traffic rules to follow on each edge as well as what types of vehicles that were allowed. The newer version of SUMO made it possible to create flows and trips without having to manually write the XML file. However, there still existed the need to add some of the attributes to get the desired behavior and driving patterns. Our knowledge here is limited which resulted in the usage of randomTrips. It might be possible with further knowledge to create traffic that reacts to the environment. Still, the traffic behavior observed in the simulations were similar to one owns experience in Hangzhou with quick lane changes, accelerations and deaccelerations.

Looking at the simulations and the overall performance we would say that SUMO works well as a simulation program. The results that we got were easy to understand and simulations gave believable results.

6 Conclusion

In this report we got acquainted with the software Simulation of Urban MObility, SUMO, and did a case study for a specific area in Hangzhou, China. For this case study we gathered TLS phase times on location during the afternoon rush hours and used videorecording to get accurate measurements. We created an algorithm with the intent to create TLS programs that optimized the traffic flow, by reducing the standard deviation D(X) and evaluating the expected value E(X). The algorithm used the TLS recordings as a base for calculations which created two possible TLS program scenarios. We built a network model representing the selected area in Hangzhou with the Netedit program included in the SUMO program package. Further we created traffic for the simulation and used TraCI to control, evaluate and gather data from the simulations. In total three simulations per scenario were performed. Each TLS scenario had the same setup for each of its simulations.

The simulation results shows that it is possible to find a significant improvement of traffic throughput by shortening several green signal phases. The shorter signal phases also seem to have a soothing effect on deviation values. To recapitulate the beforementioned problems, they are once again repeated, but also answered orderly.

Is it possible to increase traffic flow through the system? Yes, it is. The results in Section 4.1 *Departed vehicles*, and Sections 4.2 *Arrivals, collisions and teleportations*, show that scenario 100_33 has a higher number of departed vehicles and a higher number of arrived vehicles. In addition, this scenario has lower number of teleports, which indicates a smoother running traffic in general. Scenario 100_33 even has a slightly lower degree of recorded collisions. In terms of simulations results, scenario 100_33 performs better than both the scenarios IRL and 50_150.

Is it possible to reduce idling time for traffic in this area? Yes. Since shorter green signal phases result in shorter total phase cycles, each individual signal phase is occurring more often. This means shorter time in standstill. The vehicle also moves more often, making better use of a combustion engine that is otherwise kept on idle throughout the traffic congestion.

The lower levels of deviation in the 100_33 scenario, translates into less differences among the 14 interacting streets, which ought to decrease frustrations and feelings of having chosen the wrong street.

Will the algorithm come up with a better signal-regulation solution for optimizing the flow of traffic than the current traffic signal program? The

short answer is yes. The long answer is that there may exist a better solution for traffic jams, if realistic simulation scenarios show it.

It should also be mentioned that no matter how alternative realities are constructed, whether they are numerical or analytic, there may not exist a better prediction than the outcome from a simulation. Of course, the quality of the simulation determines the granularity of such an outcome.

The findings serve several purposes. Efficiency and throughput increase, which in turn lower emissions due to less time in standstill. In turn it is possible to decrease the drivers frustration by implementing a system with signal phases that provide a more even traffic flow, since the feeling of steady movement would be enhanced.

6.1 Algorithm

Brute force was chosen because an analytic solution seemed too complex. A recursive algorithm would be likely to shorten the calculation time notably, given the computer has enough working memory. Such gain in speed may open for a larger comparison table, which would make an increased phase time diversity within each TLI possible. This could also be used for dividing current signal phases into even smaller sections. Another improvement may be to use Gray Code ordered truth tables, to change only one value in each comparison. For comparison reasons all phase time values ought to be adjusted to be a factor of 3 s already from start. This fact may or may not have affected which combination of case settings that were seen to perform optimal flow. In retrospect it is obvious that many math operations are repeated multiple times, which should be able to resolve with even more precalculated value matrices where these repeated values are stored. Surely there exists a better algorithm to find optimized signal times. It is yet to be proven by someone.

6.2 Combustion engines

Given the scale of the global carbon dioxide emission problem, in which combustion engines of cars among other emissions play an important role, one may predict a future where combustion engines running on idle in traffic jams, will merely be a memory. Vehicles in the future are supposed not to emit any dangerous levels of carbon dioxide, be electric or use a fuel where the carbon dioxide is kept as a recycled constant. So, if there would exist time consuming traffic jams, one can expect very few people who would pay attention to or arguing about idling engines spewing out emissions. ▢

7 Bibliography

[1] Trafikverket, "TRVMB Kapacitet och framkomlighetseffekter (TRV 2013:64343)," 04 2014. [Online]. Available: https://www.trafikverket.se/contentassets/32ce05ecc3ac458bb8ecb802e8e2da54/trvmb_kapacitet_och_framkomlighetseffekter.pdf. [Accessed 17 09 2019].

[2] Federal Highway Administration of the U.S Department of Transportation, "Adeptive Signal Control Technology," 2017. [Online]. Available: https//:www.fhwa.dot.gov/innovation/everydaycounts/edc-1asct.cfm. [Accessed 09 2019].

[3] "Simulation of Urban MObility," German Aerospace Center, 2001. [Online]. Available: https://sumo.dlr.de/about. [Accessed 09 2019].

[4] "OpenStreetMap," 09 08 2004. [Online]. Available: https://www.openstreetmap.org. [Accessed 09 2019].

[5] Department of Scientific and Industrial Research, "Traffic Signal Settings," 1957. [Online]. Available: https://www.sinaldetransito.com.br/artigos/traffic_signals_webster.pdf. [Accessed 14 09 2019].

[6] Y. D. J. W. a. S. Y. Guo Min, "Simulation Study of Mixed Traffic in China- a Practice in Beijing.," 12 10 2008. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4732680. [Accessed 14 09 2019].

[7] "Modeling Signalized Intersection Using Queueing Theory," 2009. [Online]. Available: https://pdfs.semanticscholar.org/9a9e/9d9d49c28d93fdfbcbbf657567ea414a368d.pdf. [Accessed 14 09 2019].

[8] "netconvert," German Aerospace Center, 2001. [Online]. Available: https://sumo.dlr.de/docs/netconvert.html. [Accessed 09 2019].

[9] "polyconvert," German Aerospace Center, 2001. [Online]. Available: https://sumo.dlr.de/docs/polyconvert.html. [Accessed 09 2019].

[10] "Traffic Lights," German Aerospace Center, 2001. [Online]. Available: https://sumo.dlr.de/docs/Simulation/Traffic_Lights.html . [Accessed 09 2019].

[11] "Lanearea Detectors(E2)," German Aerospace Center, 2001. [Online]. Available: https://sumo.dlr.de/docs/Simulation/Output/Lanearea_Detectors_(E2).html. [Accessed 10 2019].

[12] "Trip," German Aerospace Center, 2001. [Online]. Available: https://sumo.dlr.de/docs/Tools/Trip.html. [Accessed 10 2019].

[13] "Lane-Changing," German Aerospace Center, 2001. [Online]. Available: https://sumo.dlr.de/docs/Simulation/SublaneModel.html#lane-changing. [Accessed 01 2020].

[14] "Definition of Vehicles, Vehicle Types, and Routes," German Aerospace Center, 2001. [Online]. Available: https://sumo.dlr.de/docs/Definition_of_Vehicles%2C_Vehicle_Types%2C_and_Routes.html#repeated_vehicles_flows. [Accessed 11 2019].

[15] "Describing the TAZ," German Aerospace Center, 2001. [Online]. Available: https://sumo.dlr.de/docs/Demand/Importing_O/D_Matrices.html#describing_the_taz. [Accessed 01 2020].

[16] "TraCI," German Aerospace Center, 2001. [Online]. Available: https://sumo.dlr.de/docs/TraCI.html. [Accessed 01 2021].

[17] "Why Vehicles are teleporting," German Aerospace Center, 2001. [Online]. Available: :https://sumo.dlr.de/docs/Simulation/Why_Vehicles_are_teleporting.html. [Accessed 01 2020].

Appendix A

All computer code that has been used in this work can be found at the address https://github.com/ada09aho/hangzhou. This appendix contains crucial bases with the aim for successful traffic simulations. Some tables are long and perhaps not meaningful to study in a book format. Such table has been shortened and is better read online at the above URL.

The appendix contains the following sections.

A.1 MATLAB code

Originally the MATLAB code contained these remark snippets, to help understand the context in which the code should work.

```
% Hangzhou Traffic Signal Intersections (S) in a system with 6
intersections.


% Directions are to/from; North=1, East=2, South=3, West=4.
%           1
%           |
%      4 -- + -- 2
%           |
%           3
% TLS: traffic light signal, TLI: traffic light intersection


% ************** System Map *************
%    |               |               |
% --S1--------S2----S3----Ø3-----S4--S5
%    |               |               |
%    |               |               |
%    |               Ø2              |
%    |               |               |
% --S6-Ø1---S7--S8--S9-----S10------S11--
%    |               |               |
%
% *** S = TLI, Ø = ped crossing, S4 is not in function ***
```

A.1.1 Main algorithm

Below is the main algorithm with start-up values and the 6 nested for-loops. For every iteration the subroutine variation is called. It takes 6 arguments. The code for variation follows below. In this case it goes backwards from 150% to 50 % of original phase times.

But how are the arguments for variation being produced? A closer look at the main method will tell. It is also good to know, while reading the MATLAB code, that there exist 11 TLIs and three pedestrian crossings within the boundaries of the system. Out of these 11, only six were chosen to base

calculations on. These TLIs have initially been numbered 1, 3, 5, 6, 9 and 11, which are kept when forming the matrices.

```
001: timerVal=tic;
002: pcu=4.0;
003: d=1.5;
004: vpm=1/(d*pcu); % vehicles per meter (vpm=1/6 as initial
value)
005: % Start and stop of gren phases as percentage of real
sampled time value
006: percBase = 50; percTop = 150;
007: % Iteration range
008: start = 31; step = -3; stop = 1;
009: % Variable for calculation and display purposes
010: roof = (max(start, stop)-1)/abs(step) + 1;
011: % Lowest deviation value / Highest index in LIST
012: least = 1; last = 200;
013: % Table with values of deviation and indices of X, i.e.
each green phase
014: LIST = zeros(last, 8);
015: % Initial inbound distance for each intersection, di
016: di=200;
017:
018: % D is a vector with distance to/from neighbour TLI (S).
019: D1=[470; 890; 570; 395];
020: D3=[520; 670; 630; 890];
021: D5=[530; 0; 760; 670];
022: D6=[570; 915; 735; 230];
023: D9=[630; 690; 725; 915];
024: D11=[760; 305; 670; 690];
025: % Deduction of intermediate TLI (50 m) and pedestrian
crossings (12 m)
026: D1=D1-[0; 50; 0; 0];
027: D3=D3-[0; 12; 12; 50];
028: D5=D5-[0; 0; 0; 12];
029: D6=D6-[0; 12+50+50; 0; 0];
```

```
030: D9=D9-[12; 50; 0; 12+50+50];
031: D11=D11-[0; 0; 0; 50];
032: D=[D1 D3 D5 D6 D9 D11];
033:
034: % DI is a vector with initializing distance di to/from
neighbour TLI (S).
035: DI1=di*[1; 1; 1; 1];
036: DI3=di*[1; 1; 1; 1];
037: DI5=di*[1; 0; 1; 1];
038: DI6=di*[1; 1; 1; 1];
039: DI9=di*[1; 1; 1; 1];
040: DI11=di*[1; 1; 1; 1];
041: DI=[DI1 DI3 DI5 DI6 DI9 DI11];
042:
043: % L is a vector with the number of lanes inbound from
neighbour TLI.
044: L1=[2; 3; 3; 3];
045: L3=[3; 3; 2; 2];
046: L5=[3; 0; 3; 3];
047: L6=[2; 3; 4; 3];
048: L9=[2; 3; 2; 3];
049: L11=[4; 2; 3; 3];
050: L=[L1 L3 L5 L6 L9 L11];
051:
052: % C is a "mass"-vector with vehicles going into each
intersection.
053: C1=vpm*(D1.*L1);
054: C3=vpm*(D3.*L3);
055: C5=vpm*(D5.*L5); C5(2)=1; % To avoid later division by
zero.
056: C6=vpm*(D6.*L6);
057: C9=vpm*(D9.*L9);
058: C11=vpm*(D11.*L11);
059: C=[C1 C3 C5 C6 C9 C11];
060:
061: % CI is an initial "mass"-vector with vehicles going into
each intersection.
```

```
062: CI1=vpm*(DI1.*L1);
063: CI3=vpm*(DI3.*L3);
064: CI5=vpm*(DI5.*L5); CI5(2)=1; % To avoid later division by
zero.
065: CI6=vpm*(DI6.*L6);
066: CI9=vpm*(DI9.*L9);
067: CI11=vpm*(DI11.*L11);
068: CI=[CI1 CI3 CI5 CI6 CI9 CI11];
069:
070: % P is a matrix with directional ratio relative to
inbound number of lanes.
071: P1=[0 1/6 2/5 3/10; 3/7 0 1/5 5/10; 5/14 2/6 0 1/10; 3/14
3/6 2/5 1/10];
072: P3=[0 0 2/5 1/10; 1/6 1/10 1/5 3/5; 3/6 3/10 1/10 1/5;
2/6 3/5 3/10 1/10];
073: P5=[0 0 3/4 9/8; 0 0 0 0; 5/8 0 1/8 1/4; 3/8 0 1/8 1/8];
074: P6=[1/10 1/5 5/8 2/5; 3/10 1/10 1/8 2/5; 2/5 3/10 0 1/5;
1/5 2/5 1/4 0];
075: P9=[0 1/5 1/3 2/5; 1/3 0 1/3 2/5; 3/6 2/5 0 1/5; 1/6 2/5
1/3 0];
076: P11=[0 1/3 3/5 1/4; 3/8 0 1/5 3/4; 5/8 1/6 0 0; 0 3/6 1/5
0];
077: P=[P1 P3 P5 P6 P9 P11];
078:
079: % G is the green light time (incl 3 s of yellow) in each
direction per TLI
080: G1=[0 78 39 45; 36 0 66 54; 39 36 0 75; 90 42 36 45];
081: G3=[0 0 51 39; 39 39 93 60; 51 39 39 93; 93 60 39 39];
082: G5=[0 0 81 45; 0 0 0 0; 81 0 51 45; 45 0 51 45];
083: G6=[45 102 42 30; 45 30 42 42; 42 30 0 102; 102 42 45 0];
084: G9=[0 57 75 33; 24 0 75 42; 75 33 0 57; 75 42 24 0];
085: G11=[0 45 45 45; 45 0 45 45; 45 45 0 0; 0 45 45 0];
086: G=[G1 G3 G5 G6 G9 G11];
087:
088: % i is a 4-bit truth table, i.e. a 16x4 binary matrix,
multiplied by ...
089: i=((percTop-percBase)/200)*[0 0 0 0; 0 0 0 1; 0 0 1 0; 0
0 1 1; 0 1 0 0; 0 1 0 1; 0 1 1 0; 0 1 1 1;
```

```
090:   1 0 0 0; 1 0 0 1; 1 0 1 0; 1 0 1 1; 1 1 0 0; 1 1 0 1; 1
1 1 0; 1 1 1 1];

091: % K contains all possible combinations in one TLI, given
the 4 phases below.

092: K=zeros(31,4); % To avoid repetitive values the original
K(16) are reduced below

093: K(1:16,:)=(percBase/100)*ones(16,4)+i;

094: K(16:31,:)=(((percTop-
percBase)/2+percBase)/100)*ones(16,4)+i;

095: % A case is 1 of 4 green light phases. There are more
phases IRL.

096: case1=[0 0 1 0; 0 0 1 0; 1 0 0 0; 1 0 0 0];

097: case2=[1 0 0 0; 1 0 0 0; 0 0 1 0; 0 0 1 0];

098: case3=[0 1 0 0; 0 0 0 1; 0 0 0 1; 0 1 0 0];

099: case4=[0 0 0 1; 0 1 0 0; 0 1 0 0; 0 0 0 1];

100: % X contains all various G as X=[G(:,:), i(m)*cases,
TLI(s)]

101: X=zeros(4,4,31,6);

102: % S contains the calculated cycle time of every TLI and
actual case as of i

103: S=zeros(31,6);

104: % X(in,out,scenario,TLI) contains all scenarios for all
TLIs. It's based on

105: % 4 cases. All combinations of cases decide every cycle
time, S. To avoid

106: % car crashes, too long right turns are adjusted before
saving to X.

107: for m=1:31 % number of scenarios

108:    for s=0:5 % number of TLIs

109:      Gs=G(:,s*4+1:s*4+4); % selection of G

110:
x=Gs.*(K(m,1)*case1+K(m,2)*case2+K(m,3)*case3+K(m,4)*case4);

111:      tempS =
max(x(3,1),x(1,3))+max(x(2,1),x(4,3))+max(x(4,2),x(2,4))+max(x
(3,2),x(1,4));

112:      S(m,s+1)= tempS;

113:      corrR = tempS - [x(1,3); x(2,4); x(3,1); x(4,2)];

114:      x(1,2) = min( x(1,2), corrR(1) );

115:      x(2,3) = min( x(2,3), corrR(2) );
```

```
116:      x(3,4) = min( x(3,4), corrR(3) );
117:      x(4,1) = min( x(4,1), corrR(4) );
118:      X(:,:,m,s+1) = x;
119:    end
120: end
121:
122: plotIndex=0;
123: avgSum=0;
124: varSum=0;
125: index = 1; % of LIST
126: updates = 1; % LIST updates
127: s2Round = roof^4;
128: plotRound = roof^3;
129: LOWAVG = zeros(10,8);
130: lowestAvg = 0.996;
131: avgIndex = 1;
132: PLOTLIST=zeros(plotRound,2);
133: disp(['This run starts at ', num2str(K(start,1)*100), ' %
and ends at ', num2str(K(stop,4)*100), ' % of the measured
phasetimes.']);
134: for s1 = start:step:stop
135:    for s2 = start:step:stop
136:      tic
137:      for s3 = start:step:stop
138:        for s4 = start:step:stop
139:          for s5 = start:step:stop
140:            for s6 = start:step:stop
141:              Z = [s1, s2, s3, s4, s5, s6];
142:              [var, avg] = variation(X, Z, S, P, CI, C);
143:              avgSum = avgSum + avg;
144:              varSum = varSum + var;
145:              if avg < lowestAvg
146:                row = mod(avgIndex,10)+1;
147:                LOWAVG(row,:) = [avg Z avgIndex];
148:                disp(['E(X) = ', num2str(avg), ' at Z = [
', num2str(Z), ' ].']);
```

```
149:                lowestAvg = avg;
150:                avgIndex = avgIndex + 1;
151:              end
152:              if var < least && avg < 1
153:                dev = sqrt(var);
154:                if index < last
155:                  LIST(index,:) = [dev Z plotIndex];
156:                  least = var;
157:                  index = index + 1;
158:                else
159:                  LIST(last,:) = [dev Z plotIndex];
160:                  least = var;
161:                  sortrows(LIST);
162:                end
163:                disp(['Update: ', num2str(updates),'. E(X)
= ',num2str(avg),', d(X) = ', num2str(dev), '. Z = [ ',
num2str(Z), ' ].']);
164:                updates = updates + 1;
165:              end
166:            end
167:          end
168:        end
169:        plotIndex = plotIndex + 1;
170:        PLOTLIST(plotIndex,1) = sqrt(varSum / plotRound);
171:        PLOTLIST(plotIndex,2) = avgSum / plotRound;
172:        avgSum = 0;
173:        varSum = 0;
174:      end
175:      disp(['This round with s1/s2 = ', num2str(s1), '/',
num2str(s2), ' and ', num2str(s2Round), ' comparisons took: ',
num2str(toc), ' s. E(X) = ', num2str(PLOTLIST(plotIndex,2)),
', d(X) = ', num2str(PLOTLIST(plotIndex,1))]);
176:    end
177: end
178:
179: TOP5DEV = sortrows(LIST(find(LIST(:,1)),:));
180: disp(TOP5DEV(1:5,:))
```

```
181: TOP5AVG = sortrows(LOWAVG(find(LOWAVG(:,1)),:));
182: disp(TOP5AVG(1:5,:));
183: toc(timerVal)
184:
185: plot (LIST(:,8),LIST(:,1),'b');
```

A.1.2 Variation function

The MATLAB code in this section is the help function which calculates a street average of vehicle evacuation values and their variance value. The main algorithm calls this function continuously to compare every possible case setting against each other. Among the arguments only Z has a different value for each call. Z is a 1x6-matrix that hold the indices of the six nested for-loops. The other arguments are prewritten matrices.

```
01: function [varX, avgX] = variation (X, Z, S, P, CI, C)
02: % Computation of variance as of how many vehicles that are
either a
03: % decrease or an increase of the amount of cars of each
street within the system.
04: % In: X is the green phase time of every TLI. S holds
cycle times.
05: % Z has the particular index for each TLI-composition in
X.
06: % Out: varX is the variance of the average evacuation of
street cars.
07:
08: TFO = zeros(4,6);
09: TFI = zeros(4,6);
10: for i=0:5
11:    R(:,i*4+1:i*4+4) = X(:,:,Z(i+1),i+1) / S(Z(i+1), i+1); %
creates green phase ratio of full cycle
12:    PR(:,i*4+1:i*4+4) = P(:,i*4+1:i*4+4) .*
R(:,i*4+1:i*4+4); % total probability for each direction
13:    TFO(:,i+1) = PR(:,i*4+1:i*4+4) * CI(:,i+1); % number of
pcus going out of TLI
14:    TFI(:,i+1) = sum(PR(:,i*4+1:i*4+4))' .* CI(:,i+1); %
number of pcus going into TLI (out of streets)
15: end
16:
```

```
% SC is a binary control matrix to decide which streets
belong to system.
SC=[0 0 0 1 1 1;
   1 1 0 1 1 0;
   1 1 1 0 0 0;
   0 1 1 0 1 1];

% TFS is a rearranged TFO (onto streets) in order to fit
subtraction with TFI
TFS =   [0 0 0 TFO(3) TFO(6) TFO(11);
   TFO(8) TFO(12) 0 TFO(20) TFO(24) 0;
   TFO(13) TFO(17) TFO(21) 0 0 0;
   0 TFO(2) TFO(6) 0 TFO(14) TFO(18)];

DIR = (TFS - TFI) .* SC;
EVA = ((C + DIR) ./ C) .* SC; % EVAcuation of streets
respectively

E = find(EVA); % finds indices of all nonzero elements
WS = EVA(E); % column vector with nonzero values of EVA
avgX = mean(WS); % mean value of WS
varX = mean((WS-avgX).^2); % Variance of WS

return
end
```

A.1.2.1 Explanation of variation function

X is a 4-dimensional array which contains 31 beforehand calculated TLS scenario for each of the 6 TLIs, thus giving a matrix with the dimensions 4x4x31x6.

Z is an array that holds each index number of the 6 for-loops. Consequently, Z will have 31^6 different configurations. By calling X(:,:,Z(i),i) the function will have the exact phase time of each TLI at that precise scenario.

S is a 31x6-matrix with beforehand calculated phase cycle times for each TLI scenario. The cycle time is reached by calling S(Z(i)).

P consists of 6 joined 4x4-matrices which with classical probability holds the probability in each direction through the TLI. If for example a TLI has 5 incoming lanes from one direction, where 1 lane is for left turns, 2 goes straight forward, 1 is both for going straight and to the right and 1 is for right turns only, then the probability for those possibilities is calculated as 1/5, 5/10 and 3/10. If the left lane would also be holding a U-turn possibility, then U-turn and left turn would have a probability of 1/10 each.

CI consists of 6 joined column arrays that each hold the number of possible cars within 200 m from the 4 directions of each TLI. This is for priming the algorithm with a certain number of cars in order to have a measurement of the capacity of each TLI.

C similarly consists of 6 joined column vectors, which each hold the total amount of cars on each of the 14 intermediate directions within the system that is modelled.

The function variation uses a 6-iteration loop - one for each TLI - to calculate the following variables.

R contains the ratios of green light with respect to the certain cycle time of this particular state.

PR is an element-wise product of P and R (P.*R).

TFO contains 6 joined column vectors with the time-flow out of a TLI and the product of PR and CI.

TFI is the time-flow into a TLI and is also calculated with the help of PR and CI, but in a manner to capture incoming traffic. Each column in PR is summed up, which produces a row vector. This is transposed to get a column vector which is element-wise multiplied with CI.

The matrix SC is a 4x6 binary matrix that tells which streets (and corresponding values) that are essential for these calculations. The column index decides the TLI in question and the row index decides what direction traffic flow is taken into consideration. Row index 1 corresponds to going north, 2 to east, 3 to south and 4 to west. Notice that there are 14 ones, which coincides with the number of street directions the model considers.

Thereafter TFO (flow out of TLI, onto street) will be reduced by TFI (flow into TLI, from street). But TFO must first be rearranged to a TFS-matrix, so both it and TFI have the same arrangement as the SC matrix.

DIR is the difference TFS - TFO multiplied with SC, to just contain essential directions. If TFS < TFO then DIR will be negative.

The degree of evacuation per street is put in EVA by elementwise dividing the sum of DIR and C with C.

E is a vector with all indices in EVA that contains nonzero values.

WS is a vector corresponding to the indices kept in E.

E(X) and V(X) for 1 of 31^6 iterations can now be calculated and returned as avgX and varX to the main method.

A.1.3 Phase times adjusting function

Before using the MATLAB result all phase times must be adjusted to be a factor of 3. This is done by setting k = 3 in the below MATLAB function.

```
% Makes each position in G-matrix a multiple of k
function [aG] = adjGreen (XG, k)
[m, n] = size(XG);
aG = zeros(m, n);
for i = 1:m
  for j = 1:n
    rest = mod(XG(i,j),k);
    if rest >= k/2
      aG(i,j) = XG(i,j) - rest + k;
    else
      aG(i,j) = XG(i,j) - rest;
    end
  end
end
```

A.2 Graphic view of simulation time phases

The following section's simple layouts present the four phases used both in MATLAB calculations and in SUMO simulations. The numbers represent the time period with green signal including a final 3 s period with yellow signal. Many of the time periods for right turns extend into one to two other phases. If a green signal period seamlessly extends into next phase, it is indicated with an asterisk * after the time number for that time period. Hence there is no final 3 s of yellow signal before extending into next phase.

While concluding data from Hangzhou field work, multiple green signal periods during one cycle were accumulated to one long time signal. Most of these green signal periods had a final 3 s of yellow signal. Therefore, concatenated green signal periods have as a rule an extra 3 s yellow signal period, i.e., a total of 6 s.

In Figure A2.1 the MATLAB matrix with turn duration times is shown. To easier compare right turn durations for S1 with the equivalence in section A.2.1 IRL, the right turn values has been circled.

```
G1 =

     0    78    39    60
    36     0    66    60
    39    36     0    99
    90    42    36    60
```

Figure A2.1 Sampled phase times from intersection S1.

All intersections were calculated as a standard 4 phase cycle, i.e., where a green signal phase would have a corresponding green signal from the opposite direction. Since S11 was changed to a round robin pattern, late in the process, its TLS matrix does not comply with its signal phases.

A.2.1 Scenario IRL phase times

Figure A2.2 – Figure A2.7 shows all signal phases within the four cases during scenario IRL in intersections S1, S3, S5, S6, S9, and S11 respectively.

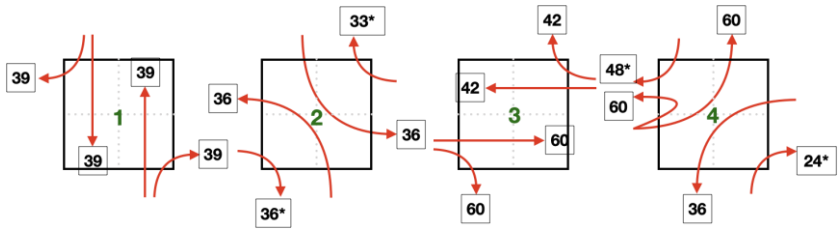


Figure A2.2. Signal phases at intersection S1 for scenario IRL.

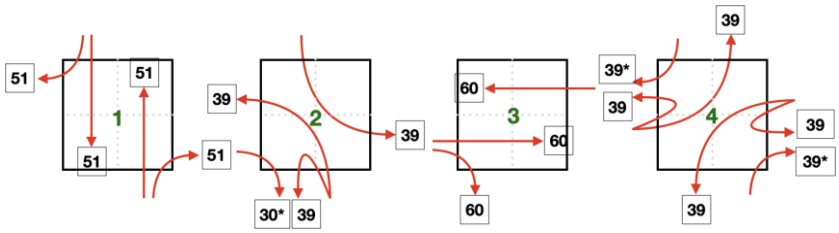


Figure A2.3. Signal phases at intersection S3 for scenario IRL.

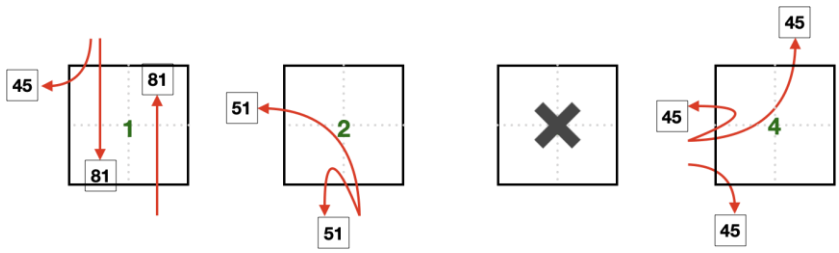


Figure A2.4. Signal phases at intersection S5 for scenario IRL.

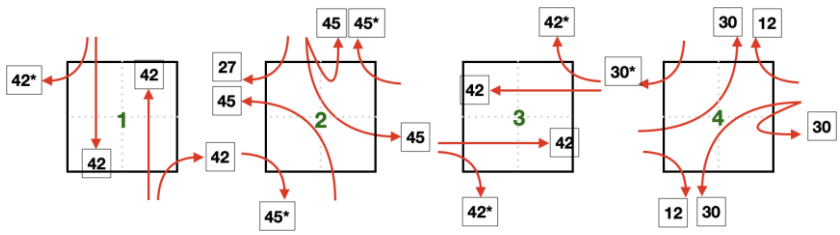


Figure A2.5. Signal phases at intersection S6 for scenario IRL.

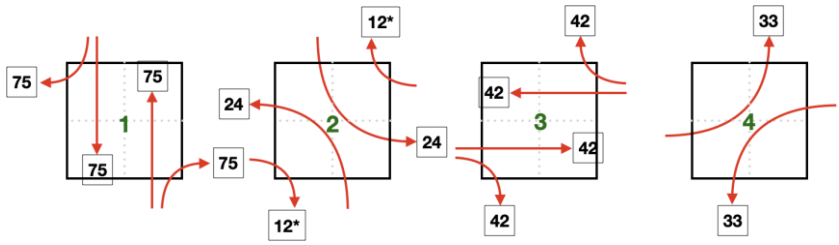


Figure A2.6. Signal phases at intersection S9 for scenario IRL.

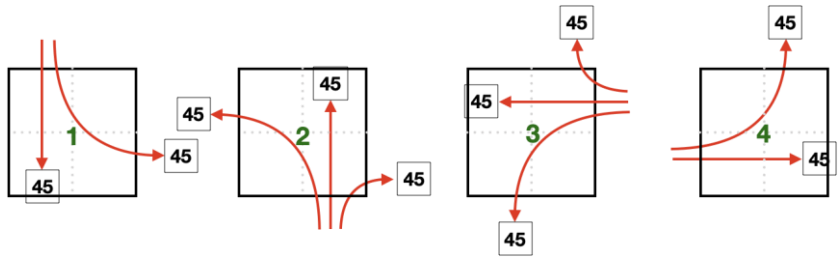


Figure A2.7. Signal phases at intersection S11 for scenario IRL.

A.2.2 Scenario 100_33 phase times

Figure A2.8 – Figure A2.13 shows all signal phases within the four cases during scenario 100_33 in intersections S1, S3, S5, S6, S9, and S11 respectively.

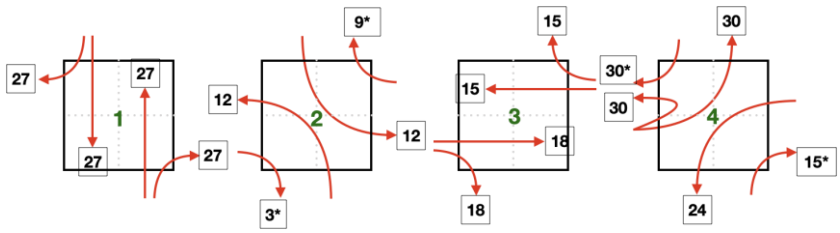


Figure A2.8. Signal phases at intersection S1 for scenario 100_33.

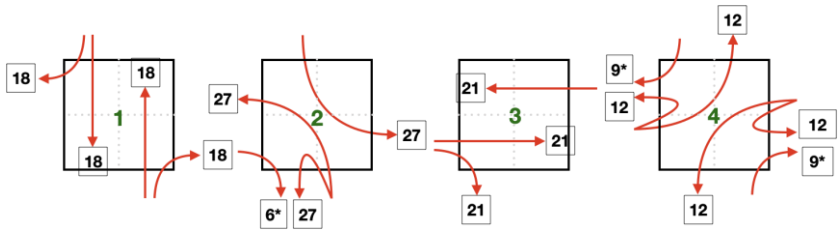


Figure A2.9. Signal phases at intersection S3 for scenario 100_33.

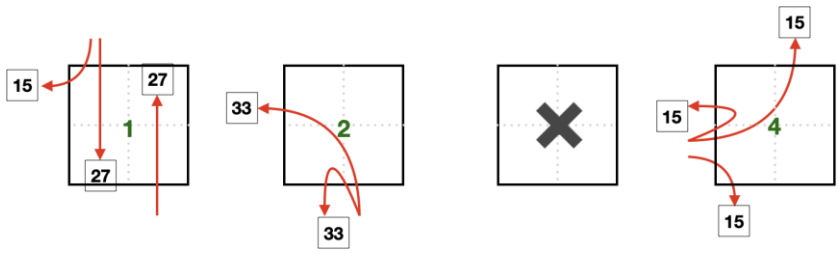


Figure A2.10. Signal phases at intersection S5 for scenario 100_33.

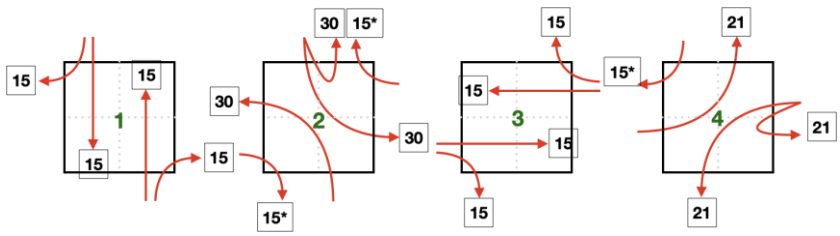


Figure A2.11. Signal phases at intersection S6 for scenario 100_33.

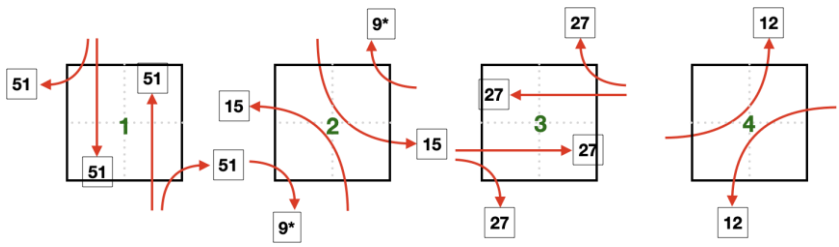


Figure A2.12. Signal phases at intersection S9 for scenario 100_33.

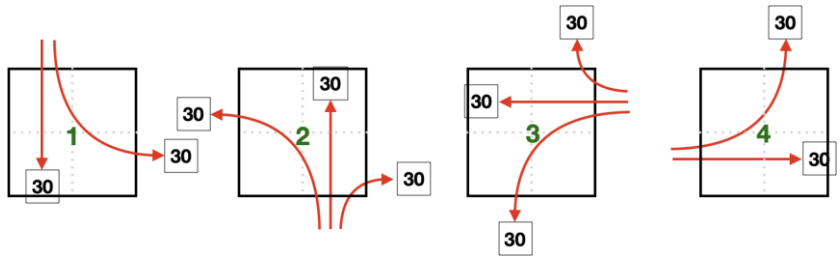


Figure A2.13. Signal phases at intersection S11 for scenario 100_33.

A.2.3 Scenario 50_150 phase times

Figure A2.14 - Figure A2.19 shows all signal phases within the four cases during scenario 50_150 in intersections S1, S3, S5, S6, S9, and S11 respectively.

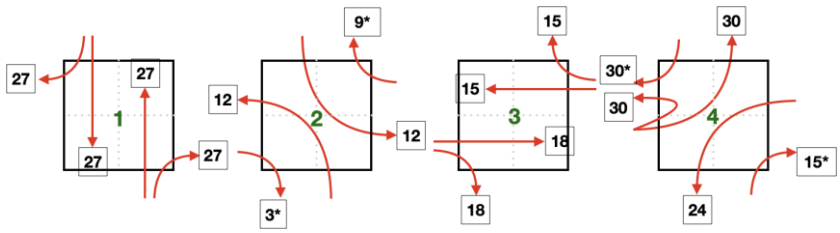


Figure A2.14. Signal phases at intersection S1 for scenario 50_150.

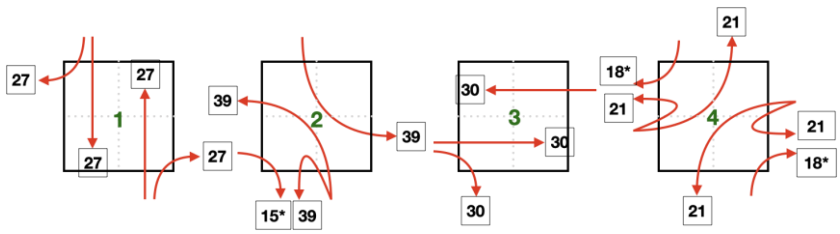


Figure A2.15. Signal phases at intersection S3 for scenario 50_150.

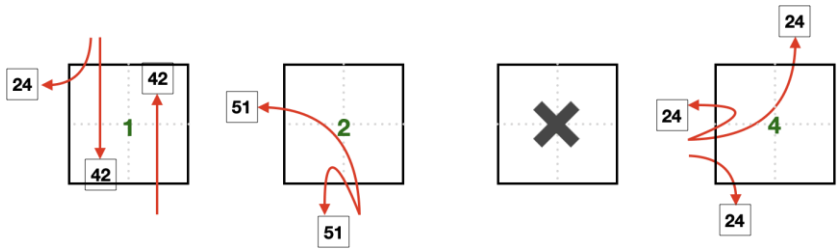


Figure A2.16. Signal phases at intersection S5 for scenario 50_150.

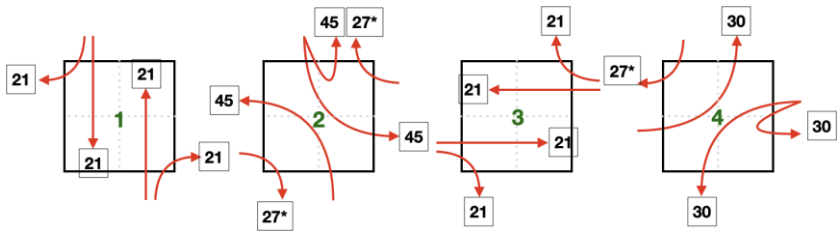


Figure A2.17. Signal phases at intersection S6 for scenario 50_150.

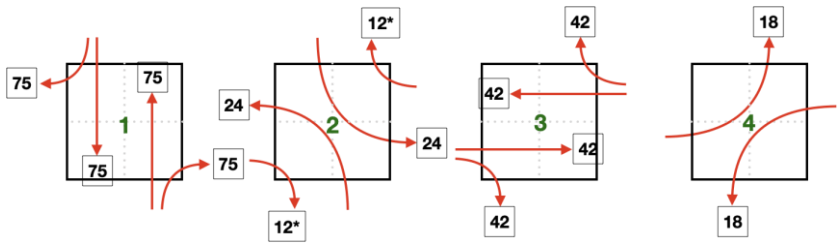


Figure A2.18. Signal phases at intersection S9 for scenario 50_150.

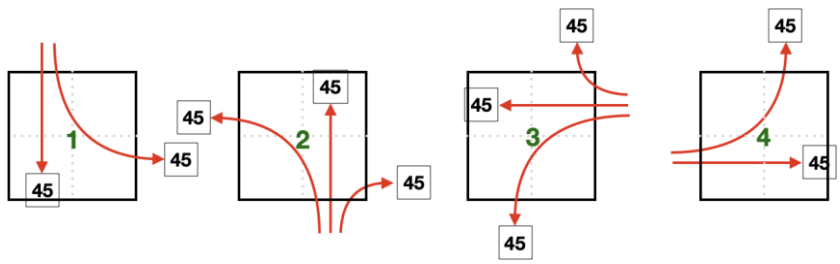


Figure A2.19. Signal phases at intersection S11 for scenario 50_150.

A.3 Attributes for XML code

The application SUMO requires several XML files gathered in a configuration script, in order to run a simulation. Besides the map upon which to simulate traffic, the traffic itself is required. This work has mainly used vehicles in the form of passenger cars and buses. Different types of scripts are used to create the various vehicle types. The following subsections show tables with attributes that specifies the vehicles and their driver's behavior. The attributes are used as terminal commands, which then creates and writes an XML file to disc. The first five tables show attributes used for passenger cars and the last section shows two tables with the attributes used to create commute vehicles across the system.

A.3.1 Insertions and choice of way

name	**used value**	**description**
n	v5_7.net.xml	specifies the map that the routes will be created for.
b	0	vehicle routes will be created from timestep 0.
e	7200	no more vehicle routes will be created after 7200.
r	seed33random.rou.xml	name of route- file created.
min-distance	500	the route from start to finish-edge must be at least 500 meters long.
fringe-factor	40.0	the probability that vehicles will enter/exit the system via the fringe.
L	n/a	increases the probability that roads with multiple lanes are chosen.
fringe-threshold	11.00	considers edges with speed above value[m/s] as fringes.
speed-exponent	11.11	increases the probability that edges with this speed [m/s] are chosen.

A.3.2 Creation numbers and distribution.

name	**used value**	**description**
seed	33, 66 & 99	generates a repeatable pseudo-randomness to the distribution of vehicles.
period	1.0	generates vehicles with equally distanced departure times, if combined with binomial it will set the arrival rate to 1/period.
binomial	400	will create a binomial distribution for the number of departures per second from the number given and the value given for the argument period.

A.3.3 Trip-attributes

name	**value**	**description**
departLane	free	the lane that is least occupied is chosen.
departSpeed	random	a depart speed between 0 and the maxSpeed will be applied. Can be adapted to maintain a safe distance to vehicles in front.
departPos	random_free	ten random positions are tried. If they are unsuccessful a free space will be searched for and used on that lane.

A.3.4 Descriptions

name	**value**	**description**
vclass	passenger	specifies the type of vehicles that will be created.
length	4.00	vehicle length in meters.
minGap	2.00	minimum distance in meters to vehicle in front.
maxSpeed	25.0	maximum speed vehicle can drive in meter per second.
speedDev	0.1	speed deviation where default = 0.1.
accel	2.6	acceleration ability for the vehicle type in m/s^2.

decel	4.5	deceleration ability for the vehicle type in m/s^2.

A.3.5 Driving behaviors

name	**value**	**description**
sigma	0.2	how good the driver is. 0 indicates perfect driving.
minGapLat	0.5	the minimum sideway distance between vehicles in meters.
laneChangeModel	SL2015	the lane change model used.

A.3.6 Commuter flow

Commuting vehicles were created from the values in the table as parameters.

Type	**Start time**	**End time**	**Run time**	**Factor**			
$OR;D2	900	8100	7200	100			
fromTaz	**toTaz**	**nbr**	**probability**	**commuter**	**via edges**	**via**	**color**
taz_s1n_in	taz_s1n_out	0.00	0.000000				white
taz_s1n_in	taz_s1r_out	0.00	0.000000				white
taz_s1n_in	taz_s1w_out	0.00	0.000000				white
taz_s1n_in	taz_s3n_out	0.00	0.000000				cyan
taz_s1n_in	taz_s5n_out	0.00	0.000000				magenta
taz_s1n_in	taz_s5r_out	0.25	0.003472	commuter0			magenta
taz_s1n_in	taz_s6w_out	0.50	0.006944	commuter1			orange
taz_s1n_in	taz_s6s_out	1.00	0.013889	commuter2			orange
taz_s1n_in	taz_s9s_out	1.00	0.013889	commuter3	"gneE25"	s6	gray
taz_s1n_in	taz_s11e_out	1.00	0.013889	commuter4	"gneE25 gneE128"	s6-s9	yellow
taz_s1n_in	taz_s11s_out	0.50	0.006944	commuter5	"gneE25 gneE128"	s6-s9	yellow
taz_s1r_in	taz_s1n_out	0.00	0.000000				white
taz_s1r_in	taz_s1r_out	0.00	0.000000				white
taz_s1r_in	taz_s1w_out	0.00	0.000000		"gneE409"		white
taz_s1r_in	taz_s3n_out	0.00	0.000000				cyan
taz_s1r_in	taz_s5n_out	0.25	0.003472	commuter6	"gneE408.17"		magenta
taz_s1r_in	taz_s5r_out	0.50	0.006944	commuter7	"gneE408.17"		magenta
taz_s1r_in	taz_s6w_out	0.50	0.006944	commuter8	"gneE408.17"		orange
taz_s1r_in	taz_s6s_out	1.00	0.013889	commuter9	"gneE408.17"		orange
taz_s1r_in	taz_s9s_out	1.00	0.013889	commuter10	"gneE408.17 gneE628"	s3	gray
taz_s1r_in	taz_s11e_out	1.00	0.013889	commuter11	"gneE408.17 gneE25 gneE128"	s6-s9	yellow

taz_s1r_in	taz_s11s_out	0.50	0.006944	commuter12	"gneE408.17 gneE25 gneE128"	s6-s9	yellow
taz_s1w_in	taz_s1n_out	0.25	0.003472	commuter13			white
taz_s1w_in	taz_s1r_out	0.25	0.003472	commuter14			white
taz_s1w_in	taz_s1w_out	0.00	0.000000				white
taz_s1w_in	taz_s3n_out	0.00	0.000000				cyan
taz_s1w_in	taz_s5n_out	1.00	0.013889	commuter15			magenta
taz_s1w_in	taz_s5r_out	1.00	0.013889	commuter16			magenta
taz_s1w_in	taz_s6w_out	1.00	0.013889	commuter17			orange
taz_s1w_in	taz_s6s_out	3.00	0.041667	commuter18			orange
taz_s1w_in	taz_s9s_out	2.50	0.034722	commuter19	"gneE628"	s3	gray
taz_s1w_in	taz_s11e_out	1.50	0.020833	commuter20	"gneE25 gneE128"	s6-s9	yellow
taz_s1w_in	taz_s11s_out	1.50	0.020833	commuter21	"gneE25 gneE128"	s6-s9	yellow
taz_s3n_in	taz_s1n_out	0.25	0.003472	commuter22			white
taz_s3n_in	taz_s1r_out	0.25	0.003472	commuter23			white
taz_s3n_in	taz_s1w_out	0.50	0.006944	commuter24			white
taz_s3n_in	taz_s3n_out	0.00	0.000000				cyan
taz_s3n_in	taz_s5n_out	0.50	0.006944	commuter25			magenta
taz_s3n_in	taz_s5r_out	1.00	0.013889	commuter26			magenta
taz_s3n_in	taz_s6w_out	2.50	0.034722	commuter27	"-gneE121"	s9	orange
taz_s3n_in	taz_s6s_out	2.50	0.034722	commuter28	"gneE413"	s1	orange
taz_s3n_in	taz_s9s_out	6.00	0.083333	commuter29			gray
taz_s3n_in	taz_s11e_out	3.50	0.048611	commuter30	"gneE128"	s9	yellow
taz_s3n_in	taz_s11s_out	2.50	0.034722	commuter31	"gneE331"	s5	yellow
taz_s5n_in	taz_s1n_out	0.00	0.000000				white
taz_s5n_in	taz_s1r_out	0.00	0.000000				white
taz_s5n_in	taz_s1w_out	0.50	0.006944	commuter32			white
taz_s5n_in	taz_s3n_out	0.00	0.000000				cyan
taz_s5n_in	taz_s5n_out	0.00	0.000000				magenta
taz_s5n_in	taz_s5r_out	0.00	0.000000				magenta
taz_s5n_in	taz_s6w_out	0.50	0.006944	commuter33	"gneE256 gneE413"	s3-s1	orange
taz_s5n_in	taz_s6s_out	0.75	0.010417	commuter34	"gneE551 -gneE121"	s11-s9	orange
taz_s5n_in	taz_s9s_out	3.50	0.048611	commuter35	"gneE551"	s11	gray
taz_s5n_in	taz_s11e_out	6.00	0.083333	commuter36			yellow
taz_s5n_in	taz_s11s_out	6.00	0.083333	commuter37			yellow
taz_s5r_in	taz_s1n_out	0.50	0.006944	commuter38			white
taz_s5r_in	taz_s1r_out	0.50	0.006944	commuter39			white
taz_s5r_in	taz_s1w_out	0.75	0.010417	commuter40			white
taz_s5r_in	taz_s3n_out	0.00	0.000000				cyan
taz_s5r_in	taz_s5n_out	0.50	0.006944	commuter41			magenta
taz_s5r_in	taz_s5r_out	1.00	0.013889	commuter42			magenta
taz_s5r_in	taz_s6w_out	1.00	0.013889	commuter43	"gneE256 gneE413"	s3-s1	orange
taz_s5r_in	taz_s6s_out	0.75	0.010417	commuter44	"gneE628 -gneE121"	s3-s9	orange
taz_s5r_in	taz_s9s_out	2.50	0.034722	commuter45	"gneE628"	s3	gray

taz_s5r_in	taz_s11e_out	0.00	0.000000				yellow
taz_s5r_in	taz_s11s_out	0.00	0.000000				yellow
taz_s6w_in	taz_s1n_out	2.00	0.027778	commuter46			white
taz_s6w_in	taz_s1r_out	2.00	0.027778	commuter47			white
taz_s6w_in	taz_s1w_out	1.50	0.020833	commuter48			white
taz_s6w_in	taz_s3n_out	1.00	0.013889	commuter49	"-gneE406"	s1	cyan
taz_s6w_in	taz_s5n_out	1.25	0.017361	commuter50	"-gneE406 gneE262"	s1-s3	magenta
taz_s6w_in	taz_s5r_out	1.25	0.017361	commuter51	"gneE128 gneE555"	s9-s11	magenta
taz_s6w_in	taz_s6w_out	0.00	0.000000				orange
taz_s6w_in	taz_s6s_out	0.25	0.003472	commuter52			orange
taz_s6w_in	taz_s9s_out	4.50	0.062500	commuter53			gray
taz_s6w_in	taz_s11e_out	2.00	0.027778	commuter54			yellow
taz_s6w_in	taz_s11s_out	2.00	0.027778	commuter55			yellow
taz_s6s_in	taz_s1n_out	2.00	0.027778	commuter56			white
taz_s6s_in	taz_s1r_out	2.00	0.027778	commuter57			white
taz_s6s_in	taz_s1w_out	1.50	0.020833	commuter58			white
taz_s6s_in	taz_s3n_out	1.00	0.013889	commuter59	"gneE131"	s9	cyan
taz_s6s_in	taz_s5n_out	1.25	0.017361	commuter60	"-gneE406 gneE262"	s1-s3	magenta
taz_s6s_in	taz_s5r_out	1.25	0.017361	commuter61	"gneE128 gneE555"	s9-s11	magenta
taz_s6s_in	taz_s6w_out	0.25	0.003472	commuter62			orange
taz_s6s_in	taz_s6s_out	0.00	0.000000				orange
taz_s6s_in	taz_s9s_out	3.50	0.048611	commuter63			gray
taz_s6s_in	taz_s11e_out	2.00	0.027778	commuter64			yellow
taz_s6s_in	taz_s11s_out	2.00	0.027778	commuter65			yellow
taz_s9s_in	taz_s1n_out	1.00	0.013889	commuter66	"gneE20"	s6	white
taz_s9s_in	taz_s1r_out	0.50	0.006944	commuter67	"gneE256"	s3	white
taz_s9s_in	taz_s1w_out	0.50	0.006944	commuter68	"gneE20"	s6	white
taz_s9s_in	taz_s3n_out	5.00	0.069444	commuter69			cyan
taz_s9s_in	taz_s5n_out	2.50	0.034722	commuter70	"gneE555"	s11	magenta
taz_s9s_in	taz_s5r_out	5.00	0.069444	commuter71	"gneE555"	s11	magenta
taz_s9s_in	taz_s6w_out	0.25	0.003472	commuter72			orange
taz_s9s_in	taz_s6s_out	0.25	0.003472	commuter73			orange
taz_s9s_in	taz_s9s_out	0.00	0.000000				gray
taz_s9s_in	taz_s11e_out	2.00	0.027778	commuter74			yellow
taz_s9s_in	taz_s11s_out	0.25	0.003472	commuter75			yellow
taz_s11e_in	taz_s1n_out	0.50	0.006944	commuter76	"-gneE121 gneE20"	s9-s6	white
taz_s11e_in	taz_s1r_out	0.50	0.006944	commuter77	"-gneE121 gneE20"	s9-s6	white
taz_s11e_in	taz_s1w_out	0.50	0.006944	commuter78	"gneE322 gneE256"	s5-s3	white
taz_s11e_in	taz_s3n_out	1.25	0.017361	commuter79	"gneE131"	s9	cyan
taz_s11e_in	taz_s5n_out	2.00	0.027778	commuter80			magenta

taz_s11e_in	taz_s5r_out	0.50	0.006944	commuter81			magenta
taz_s11e_in	taz_s6w_out	1.25	0.017361	commuter82			orange
taz_s11e_in	taz_s6s_out	1.25	0.017361	commuter83			orange
taz_s11e_in	taz_s9s_out	0.25	0.003472	commuter84			gray
taz_s11e_in	taz_s11e_out	0.00	0.000000				yellow
taz_s11e_in	taz_s11s_out	0.25	0.003472	commuter85			yellow
taz_s11s_in	taz_s1n_out	0.50	0.006944	commuter86	"-gneE121 gneE20"	s9-s6	white
taz_s11s_in	taz_s1r_out	0.50	0.006944	commuter87	"-gneE121 gneE20"	s9-s6	white
taz_s11s_in	taz_s1w_out	0.50	0.006944	commuter88	"gneE322 gneE256"	s5-s3	white
taz_s11s_in	taz_s3n_out	1.25	0.017361	commuter89	"gneE131"	s9	cyan
taz_s11s_in	taz_s5n_out	2.00	0.027778	commuter90			magenta
taz_s11s_in	taz_s5r_out	2.00	0.027778	commuter91			magenta
taz_s11s_in	taz_s6w_out	1.25	0.017361	commuter92			orange
taz_s11s_in	taz_s6s_out	1.25	0.017361	commuter93			orange
taz_s11s_in	taz_s9s_out	0.00	0.000000				gray
taz_s11s_in	taz_s11e_out	0.25	0.003472	commuter94			yellow
taz_s11s_in	taz_s11s_out	0.00	0.000000				yellow

A.3.7 Bus flow

Public transport that run on schedule and on specified routes is represented by an XML file with the name busFlow7200sV5sorted.rou.xml. The code snippet below is shortened and can be found in its entirety online at the URL mentioned in the Appendix' introduction.

```
<routes>

<vType id="BUS" vClass="bus" accel="1.3" decel="2.3"
sigma="0.1" length="12.0" minGap="4.0" maxSpeed="19.5"
speedFactor="normc(1,0.1,0.2,2)" laneChangeModel="SL2015"
maxSpeedLat="1.0" minGapLat="0.6" latAlignment="nice"/>

<flow id="bus3s1s6" color="255,127,80" begin="0" end="6900"
number ="12" type="BUS" from="gneE660" to="gneE53">
</flow>
<flow id="bus13s6s6w" color="255,127,80" begin="0" end="6480"
number ="5" type="BUS" from="-gneE10" to="gneE53">
</flow>
<flow id="bus21s6s11" color="255,127,80" begin="0" end="6600"
number ="6" type="BUS" from="-gneE10" to="gneE553">
<stop busStop="busStop_gneE26_0_13" duration="15"/>
<stop busStop="busStop_gneE79_0_25" duration="15"/>
<stop busStop="busStop_-gneE136_0_3" duration="15"/>
<stop busStop="busStop_gneE142_0_8" duration="15"/>
</flow>
<flow id="bus151s1s6" color="255,215,0" begin="0" end="6600"
number ="6" type="BUS" from="gneE412" to="gneE53">
<stop busStop="busStop_gneE469_1_19" duration="15"/>
</flow>
<flow id="bus187s3s9" color="255,215,0" begin="0" end="6900"
number ="12" type="BUS" from="gneE532" to="gneE133">
<stop busStop="busStop_gneE532_0_22" duration="15"/>
</flow>
<flow id="bus208s3s9" color="176,196,222" begin="0" end="6900"
number ="12" type="BUS" from="gneE532" to="gneE133">
<stop busStop="busStop_gneE532_0_22" duration="15"/>
<stop busStop="busStop_gneE629_0_24" duration="15"/>
<stop busStop="busStop_gneE133_0_7" duration="15"/>
</flow>
<flow id="bus274s1s6" color="176,196,222" begin="0" end="6600"
number ="6" type="BUS" from="gneE660" to="gneE53">
<stop busStop="busStop_gneE469_1_19" duration="15"/>
</flow>

</routes>
```

A.4 Python code

The SUMO application was controlled by using Traffic Control Interface (TraCI). By using the add-on traci, Python code was used to build a simulation control program and a data output compiler.

A.4.1 Simulation control code

```
001: from pathlib import Path
002: from numpy import mean as mean
003: import traci, os, datetime, json, sys, csv
004: programstart = datetime.datetime.now()
005: filesuffix = programstart.isoformat().replace('-',
'_').replace(':', '_').replace('.', '_').replace('/', '_')
006:
007: sumoBinary = "/usr/local/opt/sumo/share/sumo/bin/sumo-
gui"
008: sumoCmd = [sumoBinary, "-c", "simu_traci_02.sumocfg"]
#simu_traci_01.sumocfg, simu_v5-6orig2-1.sumocfg
009: folder = 'REPORT/'
010: reportFile = "summary" + filesuffix + ".txt"
011: detectorFile = "pcuReport" + filesuffix + ".json"
012: dataFile = "data" + filesuffix + ".csv"
013:
014: start, end = 0, 8100
015: process_time = [int(arg) for arg in sys.argv[1:]]
016: if len(process_time) > 1:
017:   start = min(process_time)
018:   end = max(process_time)
019: print(start, end)
020: warning = False
021:
022: streets = {
023: "s1_s3" : ["-gneE406", "-gneE290", "-gneE291", "-
gneE293", "-gneE294", "-gneE295", "-gneE296", "gneE252",
"gneE253", "gneE255", "gneE297", "gneE396", "gneE611"],
024: "s1_s6" : ["gneE17", "gneE413", "gneE440", "gneE443",
"gneE469", "gneE470", "gneE471"],
025: "s3_s1" : ["-gneE297", "gneE256", "gneE257", "gneE261",
"gneE290", "gneE291", "gneE293", "gneE294", "gneE295",
"gneE296", "gneE406", "gneE425", "gneE428", "gneE430",
"gneE434", "gneE618", "gneE621"],
026: "s3_s5" : ["gneE262", "gneE270", "gneE274", "gneE279",
"gneE320", "gneE321"],
027: "s3_s9" : ["-gneE131", "gneE248", "gneE374", "gneE375",
"gneE628", "gneE629"],
028: "s5_s3" : ["gneE263", "gneE268", "gneE269", "gneE271",
"gneE272", "gneE273", "gneE311", "gneE322", "gneE688"],
```

```
029: "s5_s11" : ["-gneE555", "gneE210", "gneE331", "gneE332",
"gneE333", "gneE334", "gneE336", "gneE648", "gneE649",
"gneE651"],
030: "s6_s1" : ["-gneE413", "-gneE440", "-gneE441", "gneE20",
"gneE414", "gneE415", "gneE436", "gneE444"],
031: "s6_s9" : ["-gneE71", "gneE115", "gneE116", "gneE120",
"gneE121", "gneE25", "gneE26", "gneE70", "gneE72", "gneE73",
"gneE79"],
032: "s9_s3" : ["gneE131", "gneE376", "gneE377", "gneE378",
"gneE626", "gneE627"],
033: "s9_s6" : ["-gneE115", "-gneE116", "-gneE120", "-
gneE121", "-gneE25", "-gneE26", "-gneE70", "-gneE72", "-
gneE73", "-gneE79", "gneE71"],
034: "s9_s11" : ["-gneE136", "gneE128", "gneE137", "gneE140",
"gneE142", "gneE552"],
035: "s11_s5" : ["-gneE210", "gneE337", "gneE343", "gneE344",
"gneE346", "gneE399", "gneE400", "gneE404", "gneE555"],
036: "s11_s9" : ["-gneE128", "-gneE137", "-gneE140", "-
gneE142", "-gneE552", "gneE136"]
037: }
038:
039: detectors = ["e2Detector_s1eOut1", "e2Detector_s1eOut2",
"e2Detector_s1en", "e2Detector_s1es1", "e2Detector_s1es2",
"e2Detector_s1ew1", "e2Detector_s1ew2", "e2Detector_s1ew3",
"e2Detector_s1nOut1", "e2Detector_s1nOut2",
"e2Detector_s1nOut3", "e2Detector_s1ne1", "e2Detector_s1ne2",
"e2Detector_s1ne3", "e2Detector_s1ns1", "e2Detector_s1ns2",
"e2Detector_s1nw1", "e2Detector_s1nw2", "e2Detector_s1sOut1",
"e2Detector_s1sOut2", "e2Detector_s1se", "e2Detector_s1sn1",
"e2Detector_s1sn2", "e2Detector_s1sw1", "e2Detector_s1sw2",
"e2Detector_s1wOut1", "e2Detector_s1wOut2",
"e2Detector_s1wOut3", "e2Detector_s1we1", "e2Detector_s1we2",
"e2Detector_s1wn1", "e2Detector_s1wn2", "e2Detector_s1ws",
"e2Detector_s3eOut1", "e2Detector_s3eOut2",
"e2Detector_s3eOut3", "e2Detector_s3es1", "e2Detector_s3es2",
"e2Detector_s3ew1", "e2Detector_s3ew2", "e2Detector_s3ew3",
"e2Detector_s3nOut", "e2Detector_s3ne", "e2Detector_s3ns1",
"e2Detector_s3ns2", "e2Detector_s3ns3", "e2Detector_s3nw1",
"e2Detector_s3nw2", "e2Detector_s3sOut1",
"e2Detector_s3sOut2", "e2Detector_s3se", "e2Detector_s3sn1",
"e2Detector_s3sn2", "e2Detector_s3sw1", "e2Detector_s3sw2",
"e2Detector_s3wOut1", "e2Detector_s3wOut2",
"e2Detector_s3wOut3", "e2Detector_s3we", "e2Detector_s3we1",
"e2Detector_s3we2", "e2Detector_s3we3", "e2Detector_s3ws",
"e2Detector_s5busnws", "e2Detector_s5nOut1",
"e2Detector_s5nOut2", "e2Detector_s5nbusOut",
"e2Detector_s5ns1", "e2Detector_s5ns2", "e2Detector_s5nw",
"e2Detector_s5sOut1", "e2Detector_s5sOut2",
"e2Detector_s5sOut3", "e2Detector_s5sbusOut",
"e2Detector_s5sn1", "e2Detector_s5sn2", "e2Detector_s5sn3",
"e2Detector_s5sw", "e2Detector_s5wOut1", "e2Detector_s5wOut2",
"e2Detector_s5wOut3", "e2Detector_s5wn1", "e2Detector_s5wn2",
```

```
"e2Detector_s5wn3", "e2Detector_s5ws", "e2Detector_s6eOut1",
"e2Detector_s6eOut2", "e2Detector_s6eOut3", "e2Detector_s6en",
"e2Detector_s6es1", "e2Detector_s6es2", "e2Detector_s6ew1",
"e2Detector_s6ew2", "e2Detector_s6nOut1",
"e2Detector_s6nOut2", "e2Detector_s6nOut3",
"e2Detector_s6ne1", "e2Detector_s6ne2", "e2Detector_s6ns1",
"e2Detector_s6ns2", "e2Detector_s6nw", "e2Detector_s6sOut1",
"e2Detector_s6sbusOut", "e2Detector_s6se", "e2Detector_s6sn1",
"e2Detector_s6sn2", "e2Detector_s6sw1", "e2Detector_s6sw2",
"e2Detector_s6wOut1", "e2Detector_s6wOut2",
"e2Detector_s6wOut3", "e2Detector_s6we1", "e2Detector_s6we2",
"e2Detector_s6wn1", "e2Detector_s6wn2", "e2Detector_s6ws",
"e2Detector_s9eOut1", "e2Detector_s9eOut2",
"e2Detector_s9eOut3", "e2Detector_s9en", "e2Detector_s9es1",
"e2Detector_s9es2", "e2Detector_s9ew1", "e2Detector_s9ew2",
"e2Detector_s9nOut1", "e2Detector_s9nOut2", "e2Detector_s9ne",
"e2Detector_s9ns", "e2Detector_s9nw", "e2Detector_s9sOut1",
"e2Detector_s9sOut2", "e2Detector_s9se", "e2Detector_s9sn",
"e2Detector_s9sw", "e2Detector_s9wOut1", "e2Detector_s9wOut2",
"e2Detector_s9wOut3", "e2Detector_s9we1", "e2Detector_s9we2",
"e2Detector_s9wn1", "e2Detector_s9wn2", "e2Detector_s9ws1",
"e2Detector_s11nwBus", "e2Detector_s11eOut1",
"e2Detector_s11eOut2", "e2Detector_s11en", "e2Detector_s11es",
"e2Detector_s11ew", "e2Detector_s11nBusOut",
"e2Detector_s11nOut1", "e2Detector_s11nOut2",
"e2Detector_s11ne", "e2Detector_s11ns1", "e2Detector_s11ns2",
"e2Detector_s11sBusOut", "e2Detector_s11sOut1",
"e2Detector_s11sOut2", "e2Detector_s11se",
"e2Detector_s11sn1", "e2Detector_s11sn2",
"e2Detector_s11snbus", "e2Detector_s11sw",
"e2Detector_s11wOut1", "e2Detector_s11wOut2",
"e2Detector_s11wOut3", "e2Detector_s11we1",
"e2Detector_s11we2", "taz_s1n0", "taz_s1n1", "taz_s1n3",
"taz_s1r0", "taz_s1w0", "taz_s1w1", "taz_s1w2", "taz_s1w3",
"taz_s1w4", "taz_s3n0", "taz_s3n1", "taz_s5n0", "taz_s5n1",
"taz_s5n2", "taz_s5n3", "taz_s5r0", "taz_s6w0", "taz_s6w1",
"taz_s6w2", "taz_s6s0", "taz_s6s1", "taz_s6s2", "taz_s9s0",
"taz_s9s1", "taz_s9s2", "taz_s11s0", "taz_s11s1",
"taz_s11s2" , "taz_s11e0", "taz_s11e1", "taz_s11e2"]
040:
041: fringeEdges = {'s1n': ["taz_s1n0", "taz_s1n1",
"taz_s1n3"], 's1r': ["taz_s1r0"], 's1w': ["taz_s1w0",
"taz_s1w1", "taz_s1w2", "taz_s1w3", "taz_s1w4"], 's3n':
["taz_s3n0", "taz_s3n1"], 's5n': ["taz_s5n0", "taz_s5n1",
"taz_s5n2", "taz_s5n3"], 's5r': ["taz_s5r0"], 's6w':
["taz_s6w0", "taz_s6w1", "taz_s6w2"], 's6s': ["taz_s6s0",
"taz_s6s1", "taz_s6s2"], 's9s': ["taz_s9s0", "taz_s9s1",
"taz_s9s2"], 's11s': ["taz_s11s0", "taz_s11s1", "taz_s11s2"],
's11e': ["taz_s11e0", "taz_s11e1", "taz_s11e2"]}
042: pcuOnStreets = {'s1_s3': [[], []], 's1_s6': [[], []],
's3_s1': [[], []], 's3_s5': [[], []], 's3_s9': [[], []],
's5_s3': [[], []], 's5_s11': [[], []], 's6_s1': [[], []],
```

```
's6_s9': [[], []], 's9_s3': [[], []], 's9_s6': [[], []],
's9_s11': [[], []], 's11_s5': [[], []], 's11_s9': [[], []]}
043: collisions = {'s1_s3': 0, 's1_s6': 0, 's3_s1': 0,
's3_s5': 0, 's3_s9': 0, 's5_s3': 0, 's5_s11': 0, 's6_s1': 0,
's6_s9': 0, 's9_s3': 0, 's9_s6': 0, 's9_s11': 0, 's11_s5': 0,
's11_s9': 0}
044: statistics = {'sumLoaded': 0, 'sumDeparted': 0,
'sumArrived': 0, 'sumTeleports': 0, 'sumCollisions': 0,
'collisions': []}
045: csv_header = {'fringe' : True, 'pending' : True,
'occupancy' : True}
046: map, intervals, streetLanes = {}, {}, {} # [pcu,
avg_occupancy], {t: {detector: pcu, ...}, ...}, [laneID, ...]
047: csv_fringe, csv_pending, csv_occupancy = {}, {}, {}
048: fringe, laneStreets = {}, {}
049: old = {'t': start}
050: for entry in fringeEdges.keys():
051:   fringe[entry] = 0
052:
053: def resetDetected(detected):
054:   for detector in detectors:
055:     detected.update({detector: set()})
056:   return detected
057:
058: def resetDictionary(iterable, value):
059:   for item in iterable:
060:     iterable[item] = value
061:
062: def countVehicleTypes(idList, count_pcu):
063:   randoms, commuters, buses = [0, 0, 0]
064:   if count_pcu:
065:     busFactor = 3
066:   else:
067:     busFactor = 1
068:   for id in idList:
069:     if 'bus' in id:
070:       buses += busFactor
071:     elif 'commuter' in id:
072:       commuters += 1
073:     else:
074:       randoms += 1
075:   return [randoms, commuters, buses]
076:
077: def countPCUsOnDetector(vhcLeftDetector):
078:   PCU = {}
079:   for detector in vhcLeftDetector:
080:     PCU[detector] =
sum(countVehicleTypes(vhcLeftDetector[detector], True))
081:   return PCU
082:
083: def setStreetLanes(streetLanes):
084:   for street in streets:
```

```
085:      streetLanes[street] = []
086:      for edge in streets[street]:
087:        lanes = traci.edge.getLaneNumber(edge)
088:        for lane in range(lanes):
089:          laneID = edge + '_' + str(lane)
090:          allowed = traci.lane.getAllowed(laneID)
091:          unprotected = ['moped', 'bicycle', 'pedestrian']
092:          if not any(category in allowed  for category in
unprotected):
093:            streetLanes[street].append(laneID)
094:
095: def setLaneStreets(streetLanes, laneStreets):
096:    for street in streetLanes:
097:      for laneID in streetLanes[street]:
098:        laneStreets[laneID] = street
099:
100: def update_pcuOnStreets(streetLanes):
101:    for street in streetLanes:
102:      vehicles = []
103:      occupancy = []
104:      for laneID in streetLanes[street]:
105:
vehicles.extend(traci.lane.getLastStepVehicleIDs(laneID))
106:
occupancy.append(traci.lane.getLastStepOccupancy(laneID))
107:      if sum(occupancy) != 0:
108:
pcuOnStreets[street][0].append(sum(countVehicleTypes(vehicles,
True)))
109:        pcuOnStreets[street][1].append(mean(occupancy))
110:
111: def pop_pcuOnStreets(street):
112:    vhc, occ = pcuOnStreets[street]
113:    pcuOnStreets[street] = [[], []]
114:    return int(mean(vhc)), mean(occ)
115:
116: def getFringeVehicles(intervals, t):
117:    vhcAtFringe = {}
118:    entriesAtFringe = ''
119:    period = t - old['t']
120:    for entry in fringeEdges:
121:      nbr = 0
122:      for detector in fringeEdges[entry]:
123:        nbr += intervals[detector]
124:      vhcAtFringe[entry] = nbr
125:      fringe[entry] += nbr
126:    values = vhcAtFringe.values()
127:    entriesAtFringe += 'Number of PCUs that entered thru
fringe from {} s to {} s ({} s).\n'.format(old['t'], t,
period)
```

```
128:     entriesAtFringe += '{0:^5s} {1:^5s} {2:^5s} {3:^5s}
{4:^5s} {5:^5s} {6:^5s} {7:^5s} {8:^5s} {9:^5s} {10:^5s}
{11:^5s}\n'.format('sum', *vhcAtFringe.keys())
129:     entriesAtFringe += '{0:^5.0f} {1:^5.0f} {2:^5.0f}
{3:^5.0f} {4:^5.0f} {5:^5.0f} {6:^5.0f} {7:^5.0f} {8:^5.0f}
{9:^5.0f} {10:^5.0f} {11:^5.0f}\n'.format(sum(values),
*values)
130:     old['t'] = t
131:     if csv_header['fringe']:
132:       csv_fringe[0] = ['time', 'sum', *vhcAtFringe.keys()]
133:       csv_header['fringe'] = False
134:     csv_fringe[t] = [t, sum(values), *values]
135:     return entriesAtFringe
136:
137: def findConfig(sumoCfg):
138:     configFiles = {}
139:     values = ['<net-file', '<lateral-resolution', '<gui-
settings-file', '<route-files', '<additional-files']
140:     with open(sumoCfg, 'r', encoding='utf-8') as aFile:
141:       cfgText = aFile.read()
142:     for value in values:
143:       i = cfgText.find(value)
144:       if i > 0:
145:         j = cfgText.find('/>', i)
146:       configFiles[value[1:]] = cfgText[i+1:j]
147:     return configFiles
148:
149: def getPendings(t):
150:     origins = {
151:     's1n':['gneE660'], 's1ramp':['-gneE659'], 's1w':['-
gneE411', 'gneE412'],
152:     's3n':['gneE532'], 's5n':['gneE666', 'gneE667'],
's5ramp':['-gneE675'],
153:     's6w':['-gneE10'], 's6s':['-gneE657'], 's9s':['-
gneE133'],
154:     's11s':['-gneE669'], 's11e':['-gneE553']
155:     }
156:     M = [[0] * 12 for i in range(3)]
157:     col = 1
158:     thisMap = {}
159:     for entry in origins:
160:       for edge in origins[entry]:
161:         idList = traci.edge.getPendingVehicles(edge)
162:         thisMap[entry] = len(idList)
163:         M[0][col], M[1][col], M[2][col] =
countVehicleTypes(idList, False)
164:       col +=1
165:     map[t].update(thisMap)
166:     headline = '{0:^5s} {1:^5s} {2:^5s} {3:^5s} {4:^5s}
{5:^5s} {6:^5s} {7:^5s} {8:^5s} {9:^5s} {10:^5s} {11:^5s}\n'
```

```
167:    resultline = '{0:^5s} {1:^5.0f} {2:^5.0f} {3:^5.0f}
{4:^5.0f} {5:^5.0f} {6:^5.0f} {7:^5.0f} {8:^5.0f} {9:^5.0f}
{10:^5.0f} {11:^5.0f} \n'
168:    sumOfPenders = [sum(M[0]), sum(M[1]), sum(M[2])]
169:    pendings = 'At t = {} min {} vehicles are pending to
enter via the fringe. \n'.format(t/60, sum(sumOfPenders))
170:    pendings += headline.format('vTyp', 's1n', 's1r',
's1w', 's3n', 's5n', 's5r', 's6w', 's6s', 's9s', 's11s',
's11e')
171:    pendings += resultline.format('rand', *M[0][1:12])
172:    pendings += resultline.format('comm', *M[1][1:12])
173:    pendings += resultline.format('bus', *M[2][1:12])
174:    if csv_header['pending']:
175:      csv_pending[0] = ['time', 'vTyp', 's1n', 's1r',
's1w', 's3n', 's5n', 's5r', 's6w', 's6s', 's9s', 's11s',
's11e']
176:      csv_header['pending'] = False
177:    csv_pending[t] = [[t, 'rand', *M[0][1:12]], [t, 'comm',
*M[1][1:12]], [t, 'bus', *M[2][1:12]]]
178:    return pendings
179:
180: def getSystemOccupancy(streetLanes):
181:    streetKeys = streets.keys()
182:    systemOccupancy = 'System occupancy:\n'
183:    systemOccupancy += '{0:5s} {1:5s} {2:5s} {3:5s} {4:5s}
{5:5s} {6:5s} {7:6s} {8:5s} {9:5s} {10:5s} {11:5s} {12:6s}
{13:6s} {14:6s}\n'.format('avg', *streetKeys)
184:    streetOccupancy = {}
185:    for street in streetLanes:
186:      streetOccupancy[street] = pop_pcuOnStreets(street)
187:      if streetOccupancy[street][0] == 0:
188:        warning = True
189:    map[t].update(streetOccupancy)
190:    occupancy_format = [mean([v[1] for v in
streetOccupancy.values()]), *[streetOccupancy[street][1] for
street in streetKeys]]
191:    systemOccupancy += '{0:^5.1%} {1:^5.1%} {2:^5.1%}
{3:^5.1%} {4:^5.1%} {5:^5.1%} {6:^5.1%} {7:^6.1%} {8:^5.1%}
{9:^5.1%} {10:^5.1%} {11:^5.1%} {12:^6.1%} {13:^6.1%}
{14:^6.1%}\n'.format(*occupancy_format)
192:    if csv_header['occupancy']:
193:      csv_occupancy[0] = ['time', 'avg', *streetKeys]
194:      csv_header['occupancy'] = False
195:    csv_occupancy[t] = [t, mean([v[1] for v in
streetOccupancy.values()]), *[streetOccupancy[street][1] for
street in streetKeys]]
196:    return systemOccupancy
197:
198: def updateStatistics(t, summarize):
199:    statistics['sumLoaded'] +=
traci.simulation.getLoadedNumber()
```

```
200:   statistics['sumDeparted'] +=
traci.simulation.getDepartedNumber()
201:   statistics['sumArrived'] +=
traci.simulation.getArrivedNumber()
202:   statistics['sumTeleports'] +=
traci.simulation.getEndingTeleportNumber()
203:   statistics['sumCollisions'] +=
traci.simulation.getCollidingVehiclesNumber()
204:
statistics['collisions'].extend(traci.simulation.getCollisions
())
205:   if summarize:
206:     for collision in statistics.pop('collisions'):
207:       laneID = str(collision.lane)
208:       street = laneStreets.get(laneID)
209:       if street:
210:         collisions[street] += 1
211:     intervals[t].update(statistics)
212:     resetDictionary(statistics, 0)
213:     statistics['collisions'] = []
214:
215: # Run code:
216: detected =  resetDetected({}) # Dictionary of sets
{detector: {vehicle, ...}, ...}
217: traci.start(sumoCmd)
218: summary = 'Start = {} s. Stop = {} s.\n'.format(start,
end)
219: print(summary)
220: setStreetLanes(streetLanes)
221: setLaneStreets(streetLanes, laneStreets)
222: checkframe = [900, 1800, 2700, 3600, 4500, 5400, 6300,
7200, 8100, 9000, 9900, 10800, end] # time in s
223: all_pending_cars = ''
224: for t in range(start, end+1):
225:   traci.simulationStep()
226:   if t % 150 == 0 and t > 0:
227:     update_pcuOnStreets(streetLanes)
228:   if t not in checkframe:
229:     updateStatistics(t, False)
230:     for detector in detected:
231:       vehiclesOnDetector =
traci.lanearea.getLastStepVehicleIDs(detector)
232:       detected[detector].update(vehiclesOnDetector)
233:   else:
234:     vehiclesLeftDetector = {}
235:     map[t], intervals[t] = {}, {}
236:     updateStatistics(t, True)
237:     for detector in detected:
238:       vehiclesOnDetector =
traci.lanearea.getLastStepVehicleIDs(detector)
239:       detected[detector].update(vehiclesOnDetector)
```

```
240:       vehiclesLeftDetector[detector] =
set(detected[detector].difference(vehiclesOnDetector))
241:       detected[detector] = set(vehiclesOnDetector)
242:
intervals[t].update(countPCUsOnDetector(vehiclesLeftDetector))
243:     fringeVehicles = getFringeVehicles(intervals[t], t)
244:     pending_cars = getPendings(t)
245:     systemOccupancy = getSystemOccupancy(streetLanes)
246:     all_pending_cars += '\n' + fringeVehicles + '\n' +
pending_cars + '\n' + systemOccupancy + '\n'
247:     print(fringeVehicles)
248:     print(pending_cars)
249:     print(systemOccupancy)
250: vehicles = {}
251: system_occupancy = []
252: sums = [0, 0, 0, 0, 0, 0, 0] # car, pcu, sum-%, random,
commuter, bus, occupancy
253: occupancy_report = ''
254: for street in streetLanes:
255:   random_cars, commuters, buses, pcu = 0, 0, 0, 0
256:   idList, occupancy = [], []
257:   for laneID in streetLanes[street]:
258:
idList.extend(traci.lane.getLastStepVehicleIDs(laneID))
259:
occupancy.append(traci.lane.getLastStepOccupancy(laneID))
260:   random_cars, commuters, buses =
countVehicleTypes(idList, False)
261:   carsum = commuters + random_cars + buses
262:   pcu = commuters + random_cars + 3 * buses
263:   avg_occupancy = mean(occupancy)
264:   vehicles[street] = [carsum, pcu, 0, random_cars,
commuters, buses, avg_occupancy]
265:   sums = [sums[i] + vehicles[street][i] for i in
range(len(sums))]
266: sums[-1] = sums[-1] / len(streets) # create average
occupancy
267: endtime = 'Time = {} min.'.format(t/60)
268: summary += endtime + '\n'
269: print(endtime)
270: header ='{0:7s} {1:>7s} {2:>7s} {3:>7s} {4:>7s} {5:>7s}
{6:>7s} {7:>7s} {8:>7s}'.format('Street', 'CarSum', 'PCU',
'Sum-%', 'Random', 'Commute', 'Bus', 'Occ-%', 'Coll')
271: summary += header + '\n'
272: print(header)
273: for street in vehicles:
274:   carsum, pcu, perc, ran, com, bus, occ =
vehicles[street] # nbrList
275:   if sums[0] > 0:
276:     perc = float(carsum)/sums[0] # percentage
277:     sums[2] += perc
278:   else:
```

```
279:     perc = 0
280:   coll = collisions[street]
281:   line = '{0:7s} {1:7.0f} {2:7.0f} {3:7.1%} {4:7.0f}
{5:7.0f} {6:7.0f} {7:7.1%} {8:7.0f}'.format(street, carsum,
pcu, perc, ran, com, bus, occ, coll)
282:   print(line)
283:   summary += line + '\n'
284: sumline = '{0:7s} {1:7.0f} {2:7.0f} {3:7.1%} {4:7.0f}
{5:7.0f} {6:7.0f} {7:7.1%} {8:7.0f}'.format('TOTAL', *sums,
sum(collisions.values()))
285: summary += sumline + '\n'
286: print(sumline)
287: programend = datetime.datetime.now()
288: runtime = programend - programstart
289: print('Program runtime:', runtime, '\n')
290: sumFromFringe = 'Total number of PCUs that entered thru
fringe during entire simulation ({} s).\n'.format(t - start)
291: sumFromFringe += '{0:^5s} {1:^5s} {2:^5s} {3:^5s} {4:^5s}
{5:^5s} {6:^5s} {7:^5s} {8:^5s} {9:^5s} {10:^5s}
{11:^5s}\n'.format('sum', *fringe.keys())
292: sumFromFringe += '{0:^5.0f} {1:^5.0f} {2:^5.0f} {3:^5.0f}
{4:^5.0f} {5:^5.0f} {6:^5.0f} {7:^5.0f} {8:^5.0f} {9:^5.0f}
{10:^5.0f} {11:^5.0f}\n'.format(sum(fringe.values()),
*fringe.values())
293: # filePath = Path(os.getcwd() + '/REPORT/')
294: if not os.path.exists(folder):
295:   os.makedirs(folder)
296: json_configFiles = json.dumps(findConfig(sumoCmd[2]),
indent=4)
297: with open(folder+reportFile, 'w') as aFile:
298:   aFile.write(json_configFiles + '\n')
299:   aFile.write('Program runtime:' + str(runtime) + '\n')
300:   aFile.write(summary)
301:   aFile.write(all_pending_cars)
302:   aFile.write(sumFromFringe)
303: print('Above report has been written to
{}\n'.format(os.path.abspath(folder + reportFile)))
304: print(sumFromFringe)
305: if warning:
306:   print('------ *** WARNING! Check for zeros. *** ------
')
307: with open(folder+detectorFile, 'w') as bFile:
308:   bFile.write(json_configFiles + '\n\n')
309:   bFile.write(json.dumps(collisions, indent=2) + '\n\n')
310:   bFile.write(json.dumps(map, indent=2) + '\n\n')
311:   bFile.write(json.dumps(intervals, indent=2))
312: with open(folder+dataFile, 'w') as cFile:
313:   for data in [csv_fringe, csv_pending, csv_occupancy]:
314:     header = data[0]
315:     writer = csv.writer(cFile)
316:     writer.writerow(header)
317:     for t in data:
```

```
318:       if t > 0:
319:         if len(data[t]) < 4:
320:           for row in data[t]:
321:             writer.writerow(row)
322:         else:
323:           writer.writerow(data[t])
324:     writer.writerow(['' for x in data[t]])
325: traci.close()
```

A.4.2 Simulation output process code

```
001: import json, csv, sys
002: from numpy import mean as mean
003:
004: folder = 'REPORT'
005: filename = 'pcuReport2021_06_06T17_26_26_836291.json' #
default filename
006:
007: # Call file with optional arguments: filename.json -junc
--> filename.csv
008: # Run as: <codename>.py <filename1>.json
[<filename2>.json ,…] [-junc]
009: # Argument -junc will sort table in junction order.
Default: period order.
010:
011: filesToProcess = sys.argv[1:]
012: sortJunction = False
013: maxDivisor = False
014: if '-junc' in filesToProcess:
015:   filesToProcess.remove('-junc')
016:   sortJunction = True
017: if '-maxdiv' in filesToProcess:
018:   filesToProcess.remove('-maxdiv')
019:   maxDivisor = True
020: nbrOfFiles = len(filesToProcess)
021: if not nbrOfFiles > 0:
022:   filesToProcess.append(filename)
023:
024: detectors = {
025: 's1' : {'fromNorth' : ["e2Detector_s1ne1",
"e2Detector_s1ne2", "e2Detector_s1ne3", "e2Detector_s1ns1",
"e2Detector_s1ns2", "e2Detector_s1nw1", "e2Detector_s1nw2"],
'fromEast' : ["e2Detector_s1en", "e2Detector_s1es1",
"e2Detector_s1es2", "e2Detector_s1ew1", "e2Detector_s1ew2",
"e2Detector_s1ew3"], 'fromSouth' : ["e2Detector_s1se",
"e2Detector_s1sn1", "e2Detector_s1sn2", "e2Detector_s1sw1",
"e2Detector_s1sw2"], 'fromWest' : ["e2Detector_s1we1",
"e2Detector_s1we2", "e2Detector_s1wn1", "e2Detector_s1wn2",
"e2Detector_s1ws"], 'toNorth' : ["e2Detector_s1nOut1",
"e2Detector_s1nOut2", "e2Detector_s1nOut3"], 'toEast' :
["e2Detector_s1eOut1", "e2Detector_s1eOut2"], 'toSouth' :
["e2Detector_s1sOut1", "e2Detector_s1sOut2"], 'toWest' :
```

```
["e2Detector_s1wOut1", "e2Detector_s1wOut2",
"e2Detector_s1wOut3"]},
026: 's3' : {'fromNorth' : ["e2Detector_s3ne",
"e2Detector_s3ns1", "e2Detector_s3ns2", "e2Detector_s3ns3",
"e2Detector_s3nw1", "e2Detector_s3nw2"], 'fromEast' :
["e2Detector_s3es1", "e2Detector_s3es2", "e2Detector_s3ew1",
"e2Detector_s3ew2", "e2Detector_s3ew3"], 'fromSouth' :
["e2Detector_s3se", "e2Detector_s3sn1", "e2Detector_s3sn2",
"e2Detector_s3sw1", "e2Detector_s3sw2"], 'fromWest' :
["e2Detector_s3we", "e2Detector_s3we1", "e2Detector_s3we2",
"e2Detector_s3we3", "e2Detector_s3ws"], 'toNorth' :
["e2Detector_s3nOut"], 'toEast' : ["e2Detector_s3eOut1",
"e2Detector_s3eOut2", "e2Detector_s3eOut3"], 'toSouth' :
["e2Detector_s3sOut1", "e2Detector_s3sOut2"], 'toWest' :
["e2Detector_s3wOut1", "e2Detector_s3wOut2",
"e2Detector_s3wOut3"]},
027: 's5' : {'fromNorth' : ["e2Detector_s5busnws",
"e2Detector_s5ns1", "e2Detector_s5ns2", "e2Detector_s5nw"],
'fromEast' : [], 'fromSouth' : ["e2Detector_s5sn1",
"e2Detector_s5sn2", "e2Detector_s5sn3", "e2Detector_s5sw"],
'fromWest' : ["e2Detector_s5wn1", "e2Detector_s5wn2",
"e2Detector_s5wn3", "e2Detector_s5ws"], 'toNorth' :
["e2Detector_s5nOut1", "e2Detector_s5nOut2",
"e2Detector_s5nbusOut"], 'toEast' : [], 'toSouth' :
["e2Detector_s5sOut1", "e2Detector_s5sOut2",
"e2Detector_s5sOut3", "e2Detector_s5sbusOut"], 'toWest' :
["e2Detector_s5wOut1", "e2Detector_s5wOut2",
"e2Detector_s5wOut3"]},
028: 's6' : {'fromNorth' : ["e2Detector_s6ne1",
"e2Detector_s6ne2", "e2Detector_s6ns1", "e2Detector_s6ns2",
"e2Detector_s6nw"], 'fromEast' : ["e2Detector_s6en",
"e2Detector_s6es1", "e2Detector_s6es2", "e2Detector_s6ew1",
"e2Detector_s6ew2"], 'fromSouth' : ["e2Detector_s6se",
"e2Detector_s6sn1", "e2Detector_s6sn2", "e2Detector_s6sw1",
"e2Detector_s6sw2"], 'fromWest' : ["e2Detector_s6we1",
"e2Detector_s6we2", "e2Detector_s6wn1", "e2Detector_s6wn2",
"e2Detector_s6ws"], 'toNorth' : ["e2Detector_s6nOut1",
"e2Detector_s6nOut2", "e2Detector_s6nOut3"], 'toEast' :
["e2Detector_s6eOut1", "e2Detector_s6eOut2",
"e2Detector_s6eOut3"], 'toSouth' : ["e2Detector_s6sOut1",
"e2Detector_s6sbusOut"], 'toWest' : ["e2Detector_s6wOut1",
"e2Detector_s6wOut2", "e2Detector_s6wOut3"]},
029: 's9' : {'fromNorth' : ["e2Detector_s9ne",
"e2Detector_s9ns", "e2Detector_s9nw"], 'fromEast' :
["e2Detector_s9en", "e2Detector_s9es1", "e2Detector_s9es2",
"e2Detector_s9ew1", "e2Detector_s9ew2"], 'fromSouth' :
["e2Detector_s9se", "e2Detector_s9sn", "e2Detector_s9sw"],
'fromWest' : ["e2Detector_s9we1", "e2Detector_s9we2",
"e2Detector_s9wn1", "e2Detector_s9wn2", "e2Detector_s9ws1",
"e2Detector_s11nwBus"], 'toNorth' : ["e2Detector_s9nOut1",
"e2Detector_s9nOut2"], 'toEast' : ["e2Detector_s9eOut1",
"e2Detector_s9eOut2", "e2Detector_s9eOut3"], 'toSouth' :
```

```
["e2Detector_s9sOut1", "e2Detector_s9sOut2"], 'toWest' :
["e2Detector_s9wOut1", "e2Detector_s9wOut2",
"e2Detector_s9wOut3"]},
030: 's11' : {'fromNorth' : ["e2Detector_s11ne",
"e2Detector_s11ns1", "e2Detector_s11ns2"], 'fromEast' :
["e2Detector_s11en", "e2Detector_s11es", "e2Detector_s11ew"],
'fromSouth' : ["e2Detector_s11se", "e2Detector_s11sn1",
"e2Detector_s11sn2", "e2Detector_s11snbus",
"e2Detector_s11sw"], 'fromWest' : ["e2Detector_s11we1",
"e2Detector_s11we2"], 'toNorth' : ["e2Detector_s11nBusOut",
"e2Detector_s11nOut1", "e2Detector_s11nOut2"], 'toEast' :
["e2Detector_s11eOut1", "e2Detector_s11eOut2"], 'toSouth' :
["e2Detector_s11sBusOut", "e2Detector_s11sOut1",
"e2Detector_s11sOut2"], 'toWest' : ["e2Detector_s11wOut1",
"e2Detector_s11wOut2", "e2Detector_s11wOut3"]}
031: }
032:
033: pcu_capacity = {'s1_s3': 0, 's1_s6': 0, 's3_s1': 0,
's3_s5': 0, 's3_s9': 0, 's5_s3': 0, 's5_s11': 0, 's6_s1': 0,
's6_s9': 0, 's9_s3': 0, 's9_s6': 0, 's9_s11': 0, 's11_s5': 0,
's11_s9': 0}
034:
035: def findDivisor(pcuOnStreets, direction):
036:   div = []
037:   for t in pcuOnStreets:
038:     div.append(t[direction][0])
039:   return int(mean(div))
040:
041: def getDeviation(N, pcuOnStreets):
042:   dev_order = {
043:   's1_s3' : {'s1' : 'toEast', 's3' : 'fromWest'},
's3_s1' : {'s3' : 'toWest', 's1' : 'fromEast'},
044:   's1_s6' : {'s1' : 'toSouth', 's6' : 'fromNorth'},
's6_s1' : {'s6' : 'toNorth', 's1' : 'fromSouth'},
045:   's3_s5' : {'s3' : 'toEast', 's5' : 'fromWest'},
's5_s3' : {'s5' : 'toWest', 's3' : 'fromEast'},
046:   's3_s9' : {'s3' : 'toSouth', 's9': 'fromNorth'},
's9_s3' : {'s9' : 'toNorth', 's3' : 'fromSouth'},
047:   's5_s11' : {'s5' : 'toSouth', 's11' : 'fromNorth'},
's11_s5' : {'s11' : 'toNorth', 's5' : 'fromSouth'},
048:   's6_s9' : {'s6' : 'toEast', 's9' : 'fromWest'},
's9_s6' : {'s9' : 'toWest', 's6' : 'fromEast'},
049:   's9_s11' : {'s9' : 'toEast', 's11' : 'fromWest'},
's11_s9' : {'s11' : 'toWest', 's9' : 'fromEast'}
050:   }
051:   efficiency = {}
052:   headerMuStdev = '{0:>5s} {1:^4s} {2:^4s} {3:^11s}
{4:^11s} {5:^11s}  {6:^11s} {7:^11s} {8:^11s} {9:^11s}\n'
053:   muAndStdev = headerMuStdev.format('time', '¬μ', 'd(X)',
's1/s3', 's3/s5', 's1/s6', 's3/s9', 's5/s11', 's6/s9',
's9/s11')
054:   for t in N:
```

```
055:      evac = {}
056:      streets = N[t]
057:      pcu_occupancy = pcuOnStreets[t]
058:      divisors = []
059:      for direction, parts in dev_order.items():
060:        divisor = int(pcu_occupancy[direction][0])
061:        if maxDivisor:
062:          divisor = int(pcu_occupancy[direction][0] /
pcu_occupancy[direction][1])
063:        if divisor == 0:
064:          warning = True
065:          divisor = findDivisor(pcuOnStreets, direction)
066:          print('divisor corrected to', divisor, '@ t=', t,
'on', direction)
067:        divisors.append(divisor)
068:        startNode, endNode = parts.keys()
069:        toWhere, fromWhere = parts.values()
070:        toAdd = streets[startNode].get(toWhere)
071:        toSubtract = streets[endNode].get(fromWhere)
072:        evac[direction] = round((divisor+toAdd-
toSubtract)/divisor, 6)
073:      mu = sum(evac.values())/len(evac)
074:      stdev = (sum([(x-mu)**2 for x in
evac.values()])/(len(evac.values())-1))**0.5
075:      evac['¬μ'] = round(mu, 6)
076:      evac['d(X)'] = round(stdev, 6)
077:      efficiency[t] = evac
078:      valueMuStdev = '{0:>5s} {1:4.2f} {2:4.2f}
{3:5.2f}/{4:5.2f} {5:5.2f}/{6:5.2f} {7:5.2f}/{8:5.2f}
{9:5.2f}/{10:5.2f} {11:5.2f}/{12:5.2f} {13:5.2f}/{14:5.2f}
{15:5.2f}/{16:5.2f}\n'
079:      muAndStdev += valueMuStdev.format(t, mu, stdev,
evac['s1_s3'], evac['s3_s1'], evac['s3_s5'], evac['s5_s3'],
evac['s1_s6'], evac['s6_s1'], evac['s3_s9'], evac['s9_s3'],
evac['s5_s11'], evac['s11_s5'], evac['s6_s9'], evac['s9_s6'],
evac['s9_s11'], evac['s11_s9'])
080:    return efficiency, muAndStdev
081:
082: def getStatistics(intervals):
083:    statistics = 'Statistics:\n'
084:    S = []
085:    tStats = [0, 0, '', 0, '', 0, '', 0, '']
086:    header2 = ['time', 'loaded', 'departed', 'dep-%',
'arrived', 'arr-%', 'teleports', 'tele-%', 'collisions',
'coll-%']
087:    statistics +=
'{0:6s}{1:>7s}{2:>9s}{3:>7s}{4:>8s}{5:>7s}{6:>11s}{7:>8s}{8:>1
1s}{9:>7s}\n'.format(*header2)
088:    statLine =
'{0:6s}{1:7.0f}{2:9.0f}{3:7s}{4:8.0f}{5:7s}{6:11.0f}{7:8s}{8:1
1.0f}{9:7s}\n'
```

```
089:    sumLine =
'{0:6s}{1:7.0f}{2:9.0f}{3:7.2%}{4:8.0f}{5:7.2%}{6:11.0f}{7:8.2
%}{8:11.0f}{9:7.2%}\n'
090:    S.append(header2)
091:    for time in intervals:
092:      interval = intervals[time]
093:      stats = [interval['sumLoaded'],
interval['sumDeparted'], '', interval['sumArrived'], '',
interval['sumTeleports'], '', interval['sumCollisions'], '']
094:      S.append([time, *stats])
095:      statistics += statLine.format(time, *stats)
096:      for i in range(len(stats)):
097:        if stats[i] != '':
098:          tStats[i] += stats[i]
099:    tStats[2], tStats[4], tStats[6], tStats[8] =
tStats[1]/tStats[0], tStats[3]/tStats[1], tStats[5]/tStats[1],
tStats[7]/tStats[1]
100:    statistics += sumLine.format('sum',*tStats)
101:    S.append(['sum', *tStats])
102:    print(statistics)
103:    return S
104:
105: def getTotalCollisions(collisions):
106:    header3 = ['totColl', 'sum', 'avg', 's1_s3', 's3_s1',
's1_s6', 's6_s1', 's3_s5', 's5_s3', 's3_s9', 's9_s3',
's5_s11', 's11_s5', 's6_s9', 's9_s6', 's9_s11', 's11_s9']
107:    allCollisions = ['', sum(collisions.values()),
round(mean([*collisions.values()]),1), *collisions.values()]
108:    coll_headerline =
'{0:>7s}{1:>6s}{2:>6s}{3:>6s}{4:>6s}{5:>6s}{6:>6s}{7:>6s}{8:>6
s}{9:>6s}{10:>6s}{11:>6s}{12:>6s}{13:>6s}{14:>6s}{15:>7s}{16:>
7s}\n'
109:    coll_line =
'{0:7s}{1:6.0f}{2:6.1f}{3:6.0f}{4:6.0f}{5:6.0f}{6:6.0f}{7:6.0f
}{8:6.0f}{9:6.0f}{10:6.0f}{11:6.0f}{12:6.0f}{13:6.0f}{14:6.0f}
{15:7.0f}{16:7.0f}\n'
110:    print('Total nbr of collisions:\n' +
coll_headerline.format(*header3) +
coll_line.format(*allCollisions))
111:    csv_collisions = [header3, allCollisions]
112:    return csv_collisions
113:
114: # Run code
115: warning = False
116: for jsonFile in filesToProcess:
117:    fileToRead = folder + '/' + jsonFile
118:    if maxDivisor:
119:      jsonFile = 'maxdiv_' + jsonFile
120:    fileToWrite = folder + '/' + jsonFile.split('.')[0] +
'.csv'
121:    summary = ''
122:    M = []
```

```
123:    N = {}
124:    with open(fileToRead) as aFile:
125:      text = aFile.read()
126:      config, collision_data, pcu_data, detector_data =
text.split('\n\n', 3)
127:    collisions = json.loads(collision_data)
128:    pcuOnStreets = json.loads(pcu_data)
129:    intervals = json.loads(detector_data)
130:    header = ['junc', 't-per', 'start', 'end', 'fr N', 'fr
E', 'fr S', 'fr W', 'sumIN', 'to N', 'to E', 'to S', 'to W',
'sumOUT', 'I/O']
131:    summary +=
'{0:>6s}{1:>6s}{2:>6s}{3:>6s}{4:>6s}{5:>6s}{6:>6s}{7:>6s}{8:>6
s}{9:>6s}{10:>6s}{11:>6s}{12:>6s}{13:>7s}{14:>8s}\n'.format(*h
eader)
132:    period = 0
133:    periods = [0 , *[int(key) for key in intervals.keys()]]
134:    for time, interval in intervals.items():
135:      Junc = {}
136:      for junction, directions in detectors.items():
137:        results = {}
138:        m = ['', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0.0]
139:        m[0:4] = [junction, period, periods[period],
periods[period+1]]
140:        for direction, detector in directions.items():
141:          pcu = 0
142:          for aDetector in detector:
143:            pcu += interval[aDetector]
144:          results[direction] = pcu
145:        m[4:8] = [results['fromNorth'],
results['fromEast'], results['fromSouth'],
results['fromWest']]
146:        m[8] = sum(m[4:8])
147:        m[9:13] = [results['toNorth'], results['toEast'],
results['toSouth'], results['toWest']]
148:        m[13] = sum(m[9:13])
149:        m[14] = round(m[8] / m[13], 6)
150:        M.append(m)
151:        Junc[junction] = results
152:      N[time] = Junc
153:      period += 1
154:    if sortJunction:
155:      M.sort(key=lambda x: int(x[0][1:])) # Sort in
Junction order instead of period order
156:    for row in M:
157:      summary +=
'{0:>6s}{1:6.0f}{2:6.0f}{3:6.0f}{4:6.0f}{5:6.0f}{6:6.0f}{7:6.0
f}{8:6.0f}{9:6.0f}{10:6.0f}{11:6.0f}{12:6.0f}{13:7.0f}{14:8.4f
}\n'.format(*row)
158:    print(summary)
159:    for direction in pcu_capacity:
```

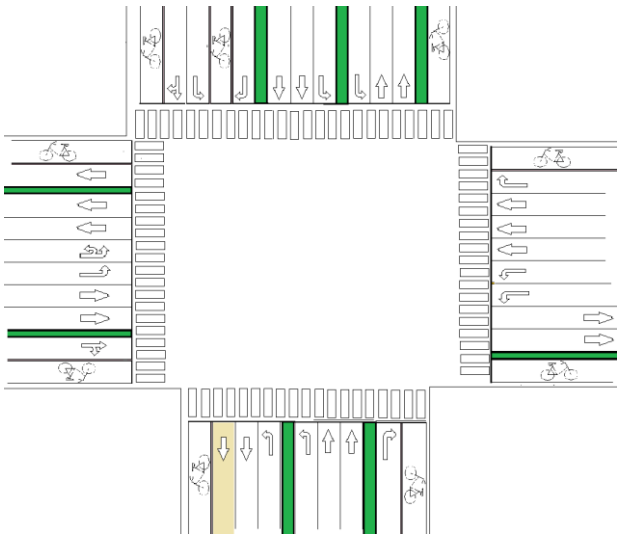
```
160:     nbrOfVehicles = []
161:     occupancies = []
162:     for t in pcuOnStreets:
163:       vehicles, occupancy = pcuOnStreets[t][direction]
164:       if vehicles != 0:
165:         nbrOfVehicles.append(vehicles)
166:       if occupancy != 0:
167:         occupancies.append(occupancy)
168:     pcu_capacity[direction] = round(mean(nbrOfVehicles) /
mean(occupancies), 0)
169:   efficiency, muAndStdev = getDeviation(N, pcuOnStreets)
170:   print(muAndStdev)
171:   fieldnames1 = ['time', '¬μ', 'd(X)', 's1_s3', 's3_s1',
's1_s6', 's6_s1', 's3_s5', 's5_s3', 's3_s9', 's9_s3',
's5_s11', 's11_s5', 's6_s9', 's9_s6', 's9_s11', 's11_s9']
172:   with open(fileToWrite, 'w', encoding='UTF8',
newline='') as bFile:
173:     writer = csv.writer(bFile)
174:     writer.writerow(header)
175:     writer.writerows(M)
176:     writer.writerow(['' for x in fieldnames1])
177:     writer = csv.DictWriter(bFile, fieldnames1)
178:     writer.writeheader()
179:     evac = []
180:     for t in efficiency:
181:       efficiency[t].update({'time' : t})
182:       evac = efficiency[t]
183:       writer.writerow(evac)
184:     writer = csv.writer(bFile)
185:     writer.writerow(['' for x in fieldnames1])
186:     writer.writerows(getTotalCollisions(collisions))
187:     writer.writerow(['' for x in fieldnames1])
188:     writer.writerows(getStatistics(intervals))
189: if warning:
190:   print('----- ***** WARNING! Check divisors and
occupancies. ***** -----')
```

A.5 General layouts of TLIs

These layouts were in fact crucial for developing this entire work. Thus, overlooking important details were prevented. As a convenience for the reader matrices with turn probability `P` and green light times `G` are also shown for the six TLIs where alternative phase times were simulated. The implementations of each item in Netedit and SUMO are shown underneath each layout. The location of every intersection and pedestrian crossing is shown on the map in Figure A 5.1. The layouts are not exactly to scale.

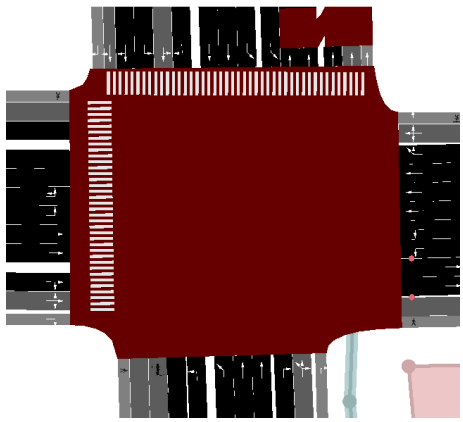
Figure A 5.1 The TLIs and pedestrian crossings across the system.

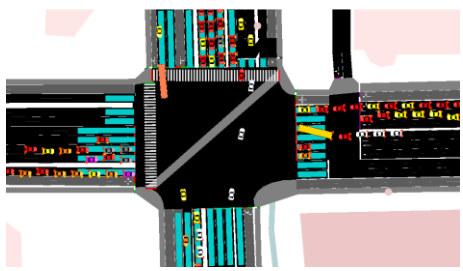
A.5.1 Layout of traffic light intersection S1



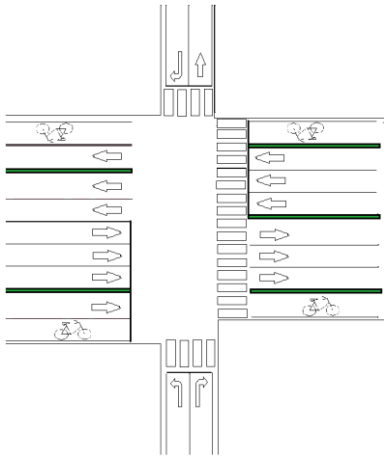
```
P1=
 0    1/6 2/5 3/10
3/7    0  1/5 5/10
5/14 2/6  0   1/10
3/14 3/6 2/5 1/10
```

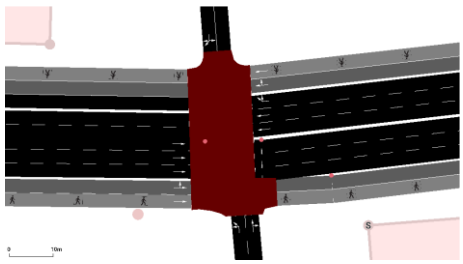
```
G1=
 0 78 39 45
36  0 66 54
39 36  0 75
90 42 36 45
```

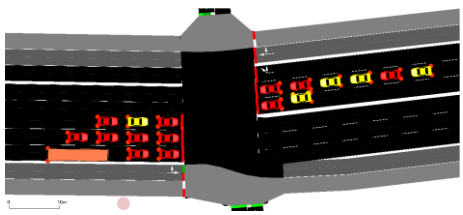




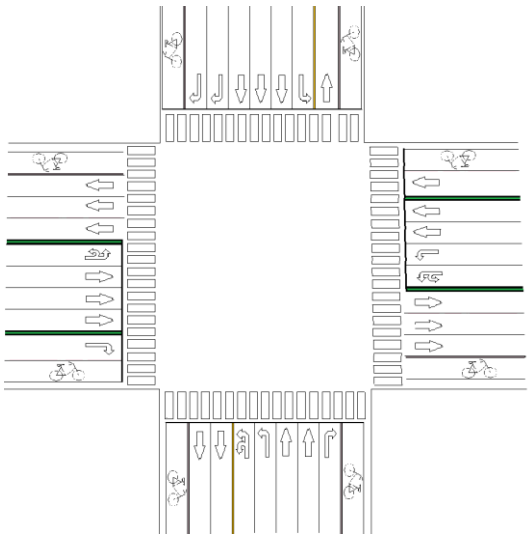
A.5.2 Layout of traffic light intersection S2





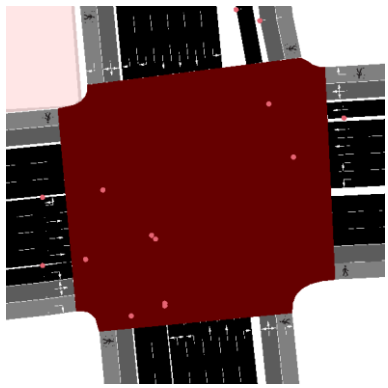


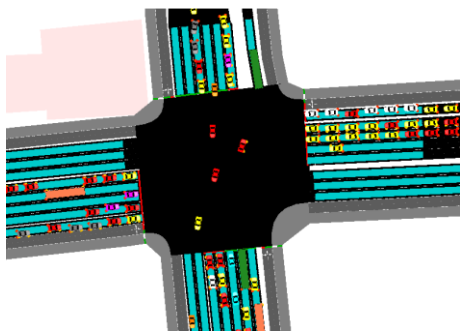
A.5.3 Layout of traffic light intersection S3



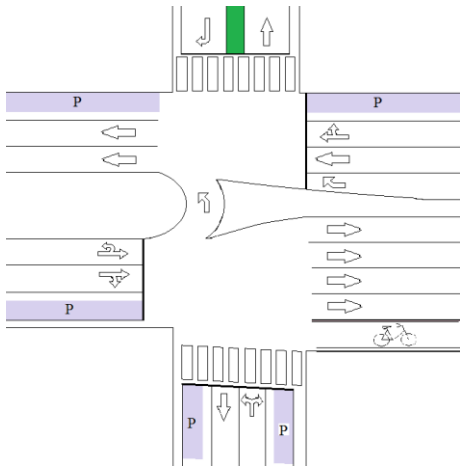
```
P3=
 0    0    2/5   1/10
1/6  1/10  1/5   3/5
3/6  3/10  1/10  1/5
2/6  3/5   3/10  1/10
```

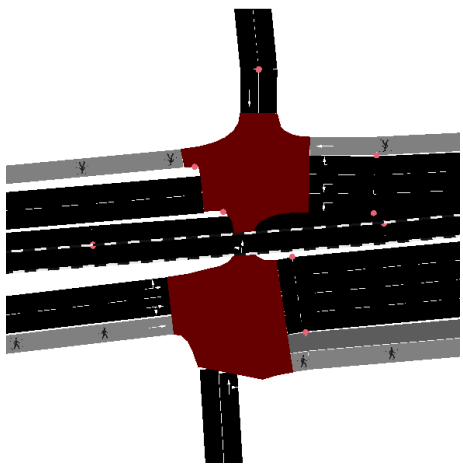
```
G3=
 0   0  51  39
39  39  93  60
51  39  39  93
93  60  39  39
```





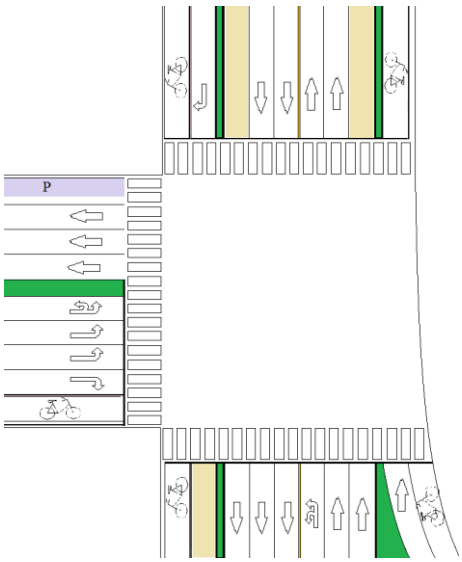
A.5.4 Layout of traffic light intersection S4





This TLS was not active.

A.5.5 Layout of traffic light intersection S5

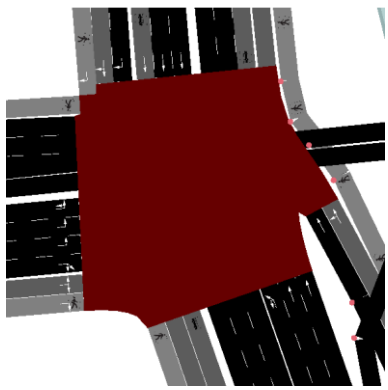


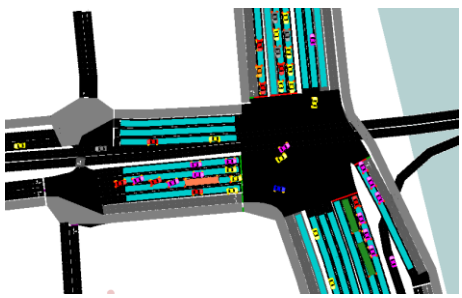
P5=

0	0	3/4	9/8
0	0	0	0
5/8	0	1/8	1/4
3/8	0	1/8	1/8

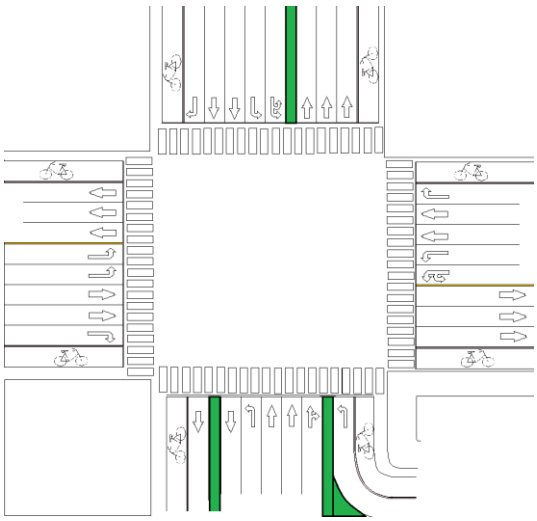
G5=

0	0	81	45
0	0	0	0
81	0	51	45
45	0	51	45





A.5.6 Layout of traffic light intersection S6

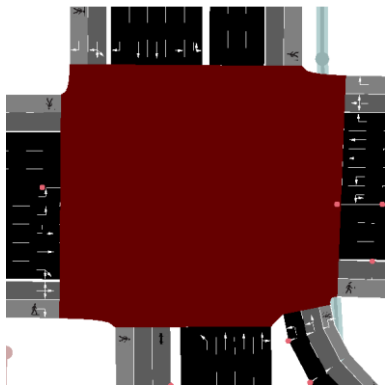


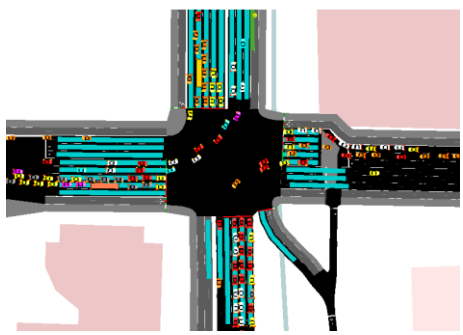
P6=

1/10	1/5	5/8	2/5
3/10	1/10	1/8	2/5
2/5	3/10	0	1/5
1/5	2/5	1/4	0

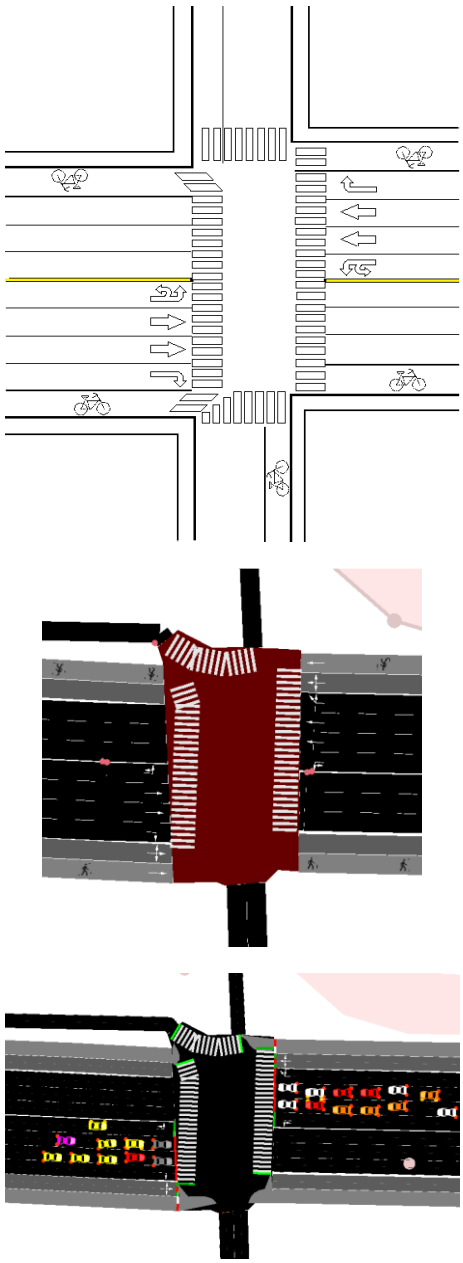
G6=

45	102	42	30
45	30	42	42
42	30	0	102
102	42	45	0

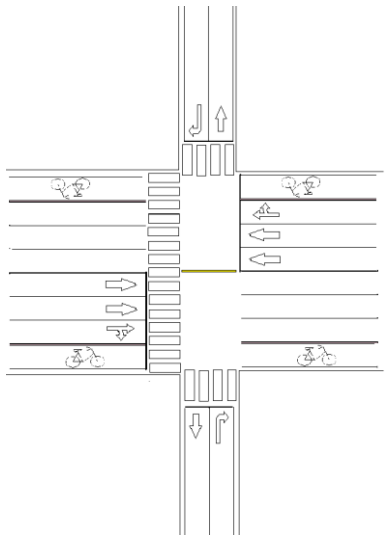


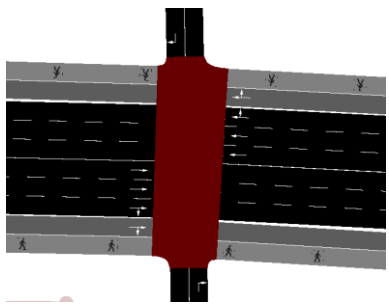


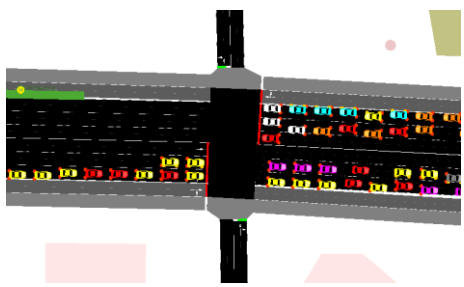
A.5.7 Layout of traffic light intersection S7



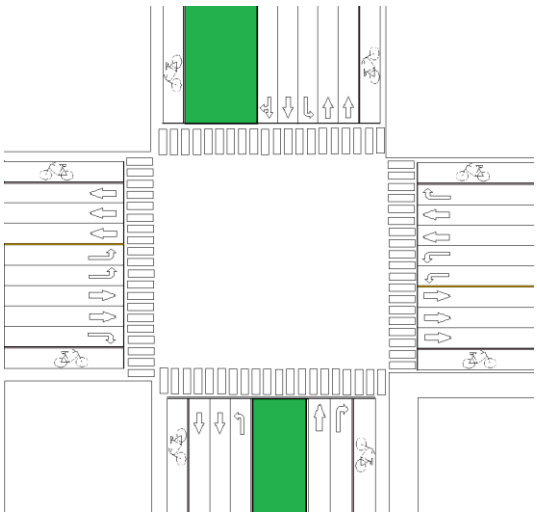
A.5.8 Layout of traffic light intersection S8







A.5.9 Layout of traffic light intersection S9

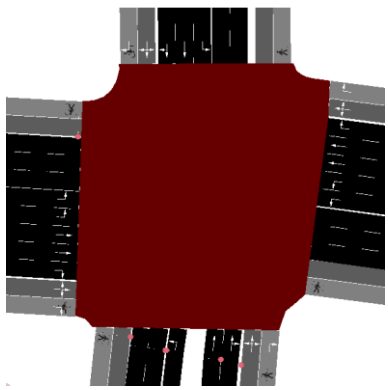


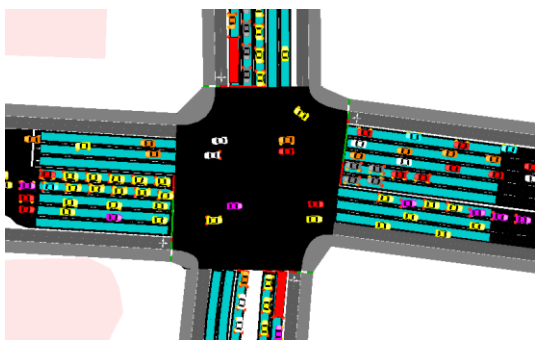
P9=

0	1/5	1/3	2/5
1/3	0	1/3	2/5
3/6	2/5	0	1/5
1/6	2/5	1/3	0

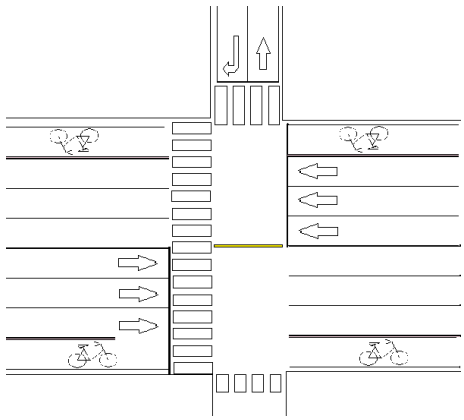
G9=

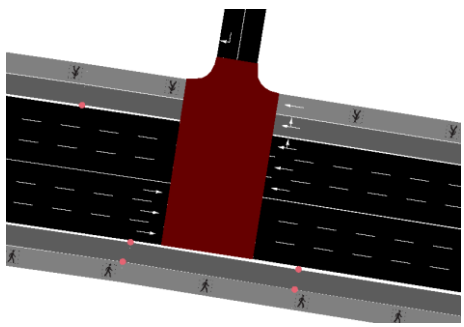
0	57	75	33
24	0	75	42
75	33	0	57
75	42	24	0

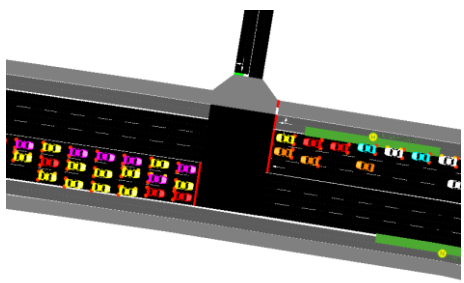




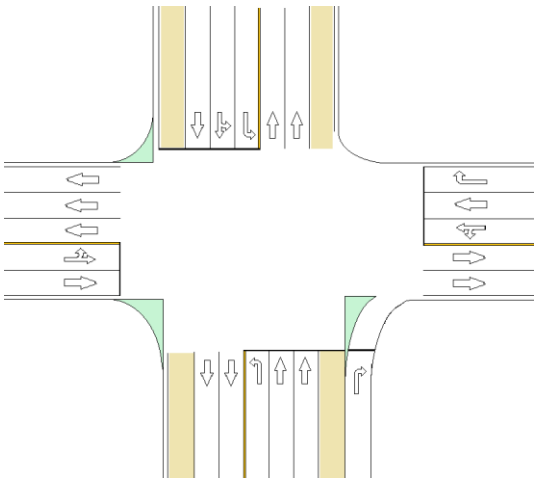
A.5.10 Layout of traffic light intersection S10





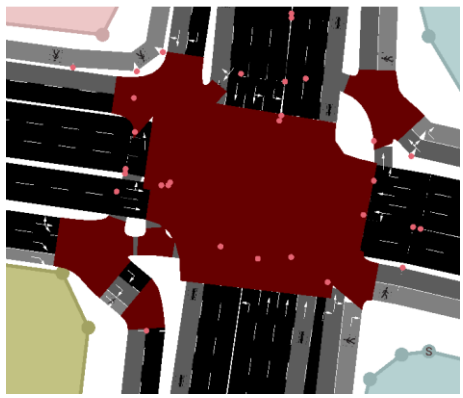


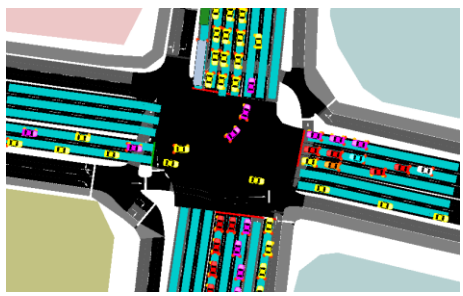
A.5.11 Layout of traffic light intersection S11



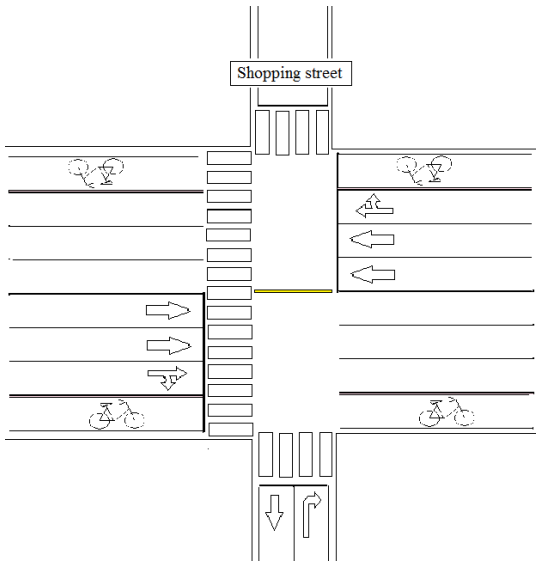
```
P11=
 0    1/3   3/5   1/4
3/8    0    1/5   3/4
5/8   1/6    0     0
 0    3/6   1/5    0
```

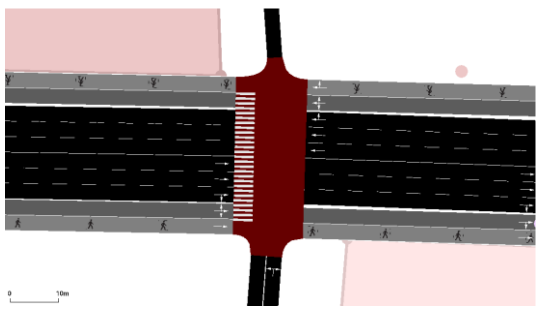
```
G11=
 0 45 45 45
45  0 45 45
45 45  0  0
 0 45 45  0
```

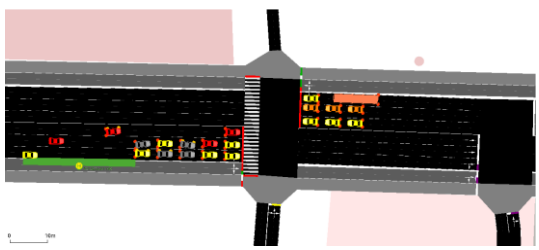




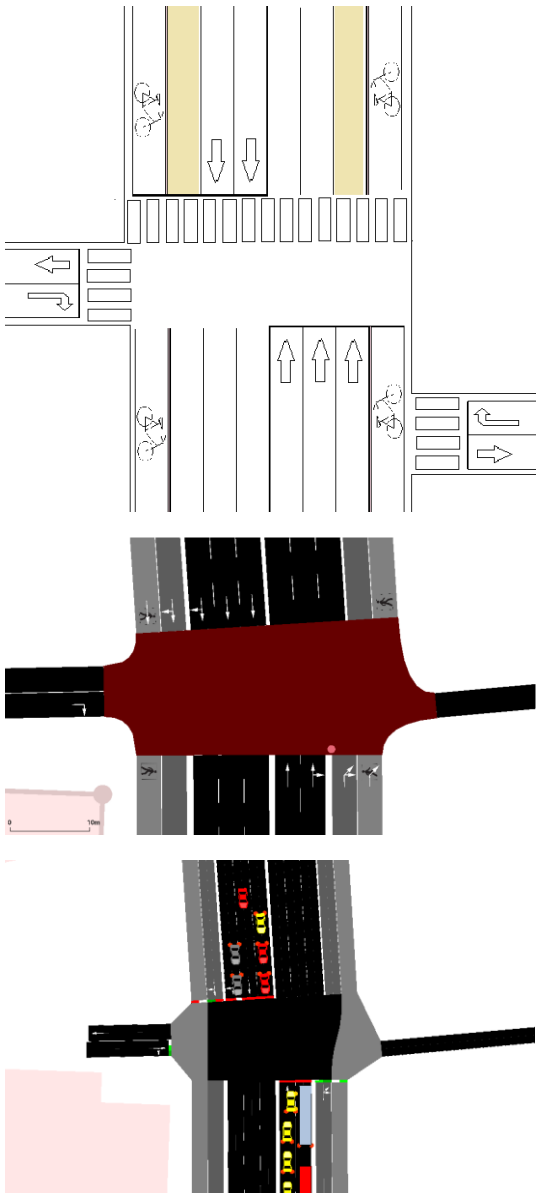
A.5.12 Layout of regulated pedestrian crossing Ø1







A.5.13 Layout of regulated pedestrian crossing Ø2



A.5.14 Layout of regulated pedestrian crossing Ø3