A new simulation model for urban traffic networks
MSc. thesis

By:
Edwin van Endhoven (717019)

Supervisor company: ir. W. van Nifterick
Supervisor university: dr.ir. M.A.A. Boon

Department of Mathematics and Computer Science
Eindhoven University of Technology

August 12, 2016
Contents

1 Introduction 2
  1.1 ARS T&TT 2
  1.2 Project description 2
    1.2.1 Problem background 2
    1.2.2 Our project 3
  1.3 Existing simulators 4
    1.3.1 SUMO and other microscopic models 5
    1.3.2 Macroscopic models 5
    1.3.3 Mesoscopic models 5

2 Framework for our simulation model 7
  2.1 Introduction 7
  2.2 Assumptions 7
  2.3 Network representation 9
  2.4 Traffic representation 10
    2.4.1 Concept of traffic blocks 11
    2.4.2 Queues 12
  2.5 External Arrivals 12
  2.6 Links 13
  2.7 Approaches 14
  2.8 Direction Lanes 15
  2.9 Signalized intersections 15

3 Implementation of the simulation model 17
  3.1 Forward movement of traffic blocks 17
  3.2 Generating new traffic blocks 19
    3.2.1 Determining block size distribution 20
  3.3 Setting the traffic lights 21
  3.4 Traffic blocks crossing intersections 23
  3.5 The modelling of the queue in Direction Lanes 26
    3.5.1 Model 1 27
    3.5.2 Model 2 30
    3.5.3 Model 3 31
  3.6 Transferring blocks from approaches to direction lanes 39
    3.6.1 Determining whether a direction lane is blocked 39
    3.6.2 The splitting of an arriving traffic block 41
3.6.3 Determining distribution fractions $p_i$ .......................... 43
3.7 Modelling the queue in approach $l$ ............................... 45
3.8 Performance measures .............................................. 47

4 Implemented traffic light control algorithms 48
4.1 Fixed traffic light algorithm ........................................ 49
4.2 Isolated VA traffic light algorithm ................................. 49
4.3 Synchronized VA traffic light algorithm .......................... 50
4.4 Synchronized VA traffic light algorithm with reduction ........ 51
4.5 Back pressure ........................................................ 51

5 Testing our simulation model 54
5.1 Comparing with SUMO ............................................. 54
  5.1.1 Average velocity .............................................. 54
  5.1.2 Capacity of system ............................................ 57
5.2 Runtime performance ............................................... 59
  5.2.1 Network ....................................................... 60
  5.2.2 Performance of traffic wave model ......................... 63
  5.2.3 Runtime performance of different traffic light algorithms . 65
5.3 Stability tests on the expected waiting time .................... 66
5.4 Impact of the offset on queue dynamics ........................ 69
5.5 Impact of different signal strategies on the network ........ 70

6 Conclusions 74
Summary

This document is a masters thesis on a newly developed traffic simulation model for inner city networks with signalized intersection. This main purpose of this model is to simulate traffic significantly faster than existing microscopic models. This is done by using a heavily simplified representation of both the road network and the traffic using it.

The network is represented by a set of nodes (representing intersections) and links between them. These links represent streets, and are built up of two parts: a part at which all traffic drives together, and a part at which traffic heading for different directions is separated (or prefiltered). The advantage of this structure is that long queues of traffic heading for some direction (left heading traffic, for example) can block traffic heading for other directions, while short queues cannot. This method is fast and relatively easy to implement.

Traffic is represented by blocks, with a specified volume, representing the amount of traffic inside the block. These blocks moves forward over links with a fixed speed, until they hit a queue or face a red traffic light and stop driving. We do not simulate complicated interactions between individual vehicles, which heavily speeds up our model.

Each intersection contains traffic lights. These traffic lights can be controlled by multiple algorithms: not only fixed cycle traffic light settings are possible, but also vehicle actuated control algorithms are implemented.

In Chapter 1, we introduce the background of the problem and existing traffic simulators. In Chapter 2, we introduce our representation of traffic, traffic networks and signalized intersections. Chapter 3 contains an extensive explanation of our simulation algorithm. In Chapter 4, we present our implemented traffic light algorithms. Chapter 5 contains some simulation studies we have done, to demonstrate its functionality and to compare it with SUMO, an existing microscopic simulation model. In Chapter 6 contains the conclusions and recommendations of our work.
Chapter 1

Introduction

This research has been conducted in the context of a final project of the Industrial and Applied Mathematics master of Eindhoven University of Technology, and the work has been done under the supervision of ARS T&TT. This report contains the results of this research.

1.1 ARS T&TT

ARS T&TT is a company founded in 1997, specialized in innovative solutions for the world of transport and traffic. They provide strategic, tactical and operational consulting for traffic issues for the public and private sector, and develop and operate systems to support services. Examples of solutions are:

- Systems providing dynamic travel information for traffic and public transport
- Traffic related enforcement systems (like distance based speed enforcement, environmental zone access enforcement)
- Programs motivating commuters to avoid travelling in rush hour
- Planning systems for logistics sector
- Systems for traffic data collection, processing and warehousing

The company is founded in The Netherlands, but expanded to India, a country with much IT knowledge, and many challenges in the transport sector.

ARS has employees from various disciplines, such as traffic engineers, IT professionals, civil engineers, and many more. In addition, ARS has many interns and graduates from engineering studies, thinking about new ideas and technology for ARS with the knowledge from their studies. This makes ARS a place with much knowledge, used for developing and maintaining new traffic solutions.

1.2 Project description

1.2.1 Problem background

ARS is involved in a project which must lead to better traffic flow in Patna, a rapidly growing city in Northeastern India which is coping with huge traffic congestion. The basic element
of their approach is placing traffic signals at around 100 intersections, to be able to control traffic streams. To achieve effective traffic control, settings should be created for the traffic signal system. The settings of traffic signal systems determine, for each time instance, which traffic signals show green light, which ones show red light and which ones show yellow light. They determine which directions get right of way, and how long this right of way will last.

Globally, we can distinguish two kinds of settings:

- **Fixed settings**: The system runs a program in which all direction at some point get right of way. When the program (or cycle) has ended, it starts over again.

- **Adaptive settings**: The system receives information about the current traffic situation in the network, and tries to adapt its signal settings in a way that maximizes traffic flow, minimizes queue lengths, minimizes lost times, etcetera.

The settings of the system that ARS want to install shall be adaptive. The information the system receives is provided by cameras, pointing at stop lines of intersections. They can detect presence of vehicles, and when these vehicles are moving, they measure gap lengths between them.

The settings of the lights are of great importance: they determine how well traffic flows over an intersection and, on a higher level, through a city. When developing new adaptive settings for a city network like the Patna area, we want to know how well they will perform before taking them into use. Testing settings in real life is almost impossible: when these settings perform very bad, the city may become completely congested, which lead to enormous economical losses. It is, however, difficult to check this performance analytically. Mathematical analysis becomes extremely difficult when the size of the network is large, and when the start times and lengths of green phases are no longer known beforehand.

### 1.2.2 Our project

The most commonly used way of evaluating traffic light settings is by simulation. Evaluation is then done by imitating the system, making use of computers. It takes as input the network layout, information about the traffic intensities and turning fractions, and the proposed signal settings. With this information, the software mimics the behaviour of traffic and traffic lights in the traffic network for a given amount of time. As output, it gives results of the performance of the system, which can be expected values and distributions of queue lengths, overflow queue lengths, lost times, etcetera.

Of course, these results of simulation runs are only estimates, since stochasticity plays a big role in these simulations. These estimates become more accurate when the simulated time is longer. When the simulated time is too short, results are more sensible to extreme situations, like a very busy period that may take a significant part of the simulated time. To avoid getting too high (or low) estimated queue lengths and lost times, we need to increase the simulated period. An obvious drawback of long simulated periods is that the simulation will take more time.

In this project, we will focus on developing a simulation model for big city networks. The size of these networks is a huge challenge, since a bigger network leads to a slower simulator. Microscopic simulators, like SUMO and VISSIM, which simulate the behaviour of each car individually, become very slow when complete city networks are analyzed. Another challenge
is the Indian traffic, since we want it to be useful for cities like Patna. Indian is highly heterogeneous, which is traffic that consists of various types of vehicles, with highly differing characteristics like (maximum) speed, acceleration, deceleration, size, etcetera. The traffic is also non-lane-based: it does not make use of lanes like western traffic does, each vehicles just picks their preferred position on the road. SiMTraM is a microscopic simulator which is able to take into account heterogeneous, non-lane-based traffic, but it is even slower than SUMO and VISSIM and it contains many errors. Our solution can not be microscopic, since other microscopic solutions have shown to be too slow to be useful.

The simulator we will develop represents traffic in a more simplified manner, to highly increase its speed.

1.3 Existing simulators

Simulation is a widely used method for evaluating traffic networks. Many methods and software packages are developed to do this. These methods vary heavily: when using a simulator for research purposes, the choice for a method or software package should depend on the research that is to be done. Examples of criteria that influence this choice are:

The size of the network

The size of the network heavily influences the speed of simulation runs. For small networks (one or two intersections) very detailed simulations can be used. When looking on such a small scale, you might be interested in individual car behavior, which desires microscopic simulators like SUMO (1.3.1) or VISSIM.

When, on the other hand, city road networks or country-wide networks are analyzed, microscopic analysis turn out to be far too slow to use. Luckily, the microscopic behavior of individual vehicles may play a less important role in such a big-scale analysis. For these problems, macroscopic analysis can be used, where traffic is represented as a fluid instead of a set of individual cars.

The composition of the network

Research on traffic behavior at highways differs heavily from research on city networks with signalized intersections. Highway analysis asks for flow simulations, where there is a one-to-one relation between speed and traffic density. This cannot be used when traffic signals are playing a role, since flows can not be stopped completely.

The composition of the traffic

Simulators like SUMO and VISSIM are based on traffic in developed countries, where traffic consists mainly of cars (at least 80%) and vehicles follow lanes. In this situation, traffic moves forward inside a lane and the only sideways movement that can be made is switching to the next lane (if available and empty).

Traffic in developing countries like India is heterogeneous. It is a mix of cars, trucks, motorized rickshaws, motorcycles, bicycles, etcetera. Because most of the vehicles are smaller than cars, lane-based traffic would be very inefficient. In these countries, drivers do not pay attention to lanes, they can take any position on the road. A microscopic simulation on this
traffic would be even more time-consuming than on homogeneous traffic, because far more movements are possible.

As a result of these different criteria, many different simulators are developed. In the following paragraphs, we will give a brief overview of existing simulation tools.

1.3.1 SUMO and other microscopic models
SUMO [1] (Simulation of Urban MObilility) is an open source simulation tool developed by the Institut für Verkehrssystemtechnik (Institute of Transportation Systems), located in Braunschweig, Germany. SUMO is able to perform microscopic simulations of lane based traffic, which means that it simulates the behavior of each vehicle in the system individually. SUMO can simulate cars, trucks and motorcycles, but also pedestrians, emergency vehicles and public transport.

SUMO simulates traffic in time steps of one second. After each second, SUMO determines what the traffic light settings for next step should be. SUMO has some built-in strategies for traffic lights, but with the TraCI tool (which is an abbreviation for Traffic Control Interface) it is also possible to retrieve values of simulated object on-line (like vehicles, detectors, etcetera), and use these values to determine the traffic light settings for next time step. With TraCI, we have almost unlimited freedom in programming adaptive traffic light settings.

Since SUMO is a microscopic simulation tool, it becomes very slow when used on big city network. It is more useful to simulate smaller networks, when you want to focus on detailed individual car behaviour.

Further on in this report, we will use SUMO as an benchmark to compare our new simulation approach.

Another commonly used microscopic simulation tool is VISSIM [2], which is a commercial simulation package. VISSIM is the global market leader in traffic simulation, and is well known for its graphics. Our choice for SUMO was based on the fact that VISSIM is commercial and quite expensive, while SUMO is open source and free to use.

1.3.2 Macroscopic models
In macroscopic models, traffic on a road is represented as a fluid with a density $\rho(x) \text{ (veh/m)}$, which depends on the position $x$ on the road. There is a one-to-one relation between traffic density and traffic flow (veh/s). The road also has a capacity $C(x)$, which is the maximum flow that can cross the point $x$. When the flow raises to values higher than the capacity, traffic will get stopped at the bottleneck and a queue will appear.

Macroscopic models are mainly used for highway simulations, since traffic can not get completely stopped in macroscopic models. This makes using then for traffic light simulations very difficult.

More information about macroscopic models can be found in [3].

1.3.3 Mesoscopic models
The simulation model we built can be classified as a mesoscopic model. A mesoscopic model does not model each vehicle individually, but it still keeps information about the vehicles in
the system. For example, it can keep blocks or platoons of traffic, where the composition of vehicles in the blocks is kept in memory.

Mesoscopic models are able to gain vehicle delays and other vehicle-dependant data, but they do this much faster than microscopic models. The speed of a mesoscopic model depends on its level of detail, which can still vary heavily per model.

We have not tested existing mesoscopic models, since these models were not easily accessible. An example of a mesoscopic that has some similarities with our model is CONTRAM [6]. In this model, packages of vehicles (which can have different amounts of vehicles in them) travel through a network, represented as a system with nodes and links. Traffic is described by an OD-matrix, and is assigned over the network dynamically. Its output will not just consists of travel times, but also of network flows and routing choices.
2.1 Introduction

In the previous chapter, we discussed many drawbacks of existing traffic simulation tools. In this chapter, we will explain our simulation model in a detailed way. We will show which choices we made and why we made those choices.

The simulation model we built should be able to simulate traffic in Indian city networks with around 100 intersections. This means that it should be able to simulate these systems over long periods to gain accurate results, without the simulation taking too much time. Simulations are used to check whether traffic signal settings provide sufficient traffic flow or not, so it should be flexible when it comes to implementing signal settings. We should be able to implement fixed signal settings, but also adaptive signals settings which collect data from detectors.

The results of the simulator should make clear to us whether the signal strategy is sufficient for the available traffic or not. It should also point out at which link traffic load is too high, especially when links are often completely occupied, blocking the upstream intersection.

Before we start explaining how our simulation model is constructed, we will make some (simplifying) assumptions for our model. This is essential to make a simplified model that is fast enough to simulate big systems.

2.2 Assumptions

Assumption 1: traffic moves over links in platoons

Since a red traffic light mostly results in a queue of vehicles, these vehicles will depart together when the light turns green. We assume these vehicles travel forward together in a group, called a platoon. Instead of simulating all vehicles individually, we only keep these platoons in memory.

Platoons will only be splitted when:

- vehicles make different routing choices at intersections, or
• a traffic light turns red while the platoon crosses its stop line.

We will refer to these platoons as “traffic blocks”.

**Assumption 2: all traffic blocks on the same link travels at the same speed**

Each link (or road) \( l \) has a speed \( v_l \) which represents the driving speed of all traffic blocks. Blocks can only stand still or drive at speed \( v_l \). These speeds may be different for each link, depending on the maximum allowed speed on that link.

A result of assumptions 1 and 2 is that interactions inside a traffic block are neglected. Although traffic inside a traffic block can overtake each other, this will not lead to splitting traffic blocks or varying speeds of blocks.

Another result is that, if multiple blocks are driving on a link, they will keep the same distance to each other. Because of this, these traffic blocks will not interfere with each other as long as they are driving. This obviously changes when they meet each other at a queue.

**Assumption 3: near the stop line of a link, traffic heading for different directions does not interact with each other**

At some point on the link, we will split traffic heading for different directions. After this splitting point, \( x \) meters from the stop line, traffic is only interacting with traffic heading for the same direction.

For each direction, we keep a separate queue. As long as none of these queues is longer than \( x \), traffic is not affected by queues for other directions. If one of the queues grow longer than \( x \), it will block traffic from all directions.

**Assumption 4: intersections will never be blocked by queues of downstream links**

If the queue on a link grows excessively, it may grow longer than the length of the link itself. In practice, the queue would block the intersection upstream of the link. If we actually simulate this, and we prohibit traffic to cross this intersection, this would rigorously slow down our simulation.

Instead of actually modelling the fact that the intersection gets blocked, we only keep track of whether intersections were blocked or not. We permit the queue to grow longer than the length of its link. The philosophy of this approach is that the model would already perform very bad in this case, and when we model intersection blocking, it would even perform worse. Good performing signal settings should not be affected by this assumption.

**Assumption 5: the destination of traffic on a link is independent from its origin**

At the splitting point \( x \), mentioned in Assumption 3, we distribute traffic over the possible directions traffic can take at the intersection. This distribution process is assumed to be independent from the path the traffic has traveled before entering this link.

The assumptions mentioned above are essential for building our model. In the following paragraphs, we will explain step by step how our simulation model is built up.
2.3 Network representation

In the simulation model, the network consists of several building blocks. These building blocks are:

- Intersections
- Approaches
- Direction Lanes
- External Arrivals

In the following section, we will discuss all these elements. Links (or roads) between intersection are modelled as an Approach, followed by several Direction Lanes, so no element Link is used.

In Figure 2.2 we see the schematic representation of the model for the road (sub)network in Figure 2.1. At the left end, traffic enters the network, and it leaves the network at the right end. The network continues beyond the exits below.

![Figure 2.1: Example of real life (sub)network](image1)

![Figure 2.2: Modelled network of Figure 2.1](image2)

In Figure 2.2, the elements represented as ovals will contain traffic blocks, while the elements represented as rectangles will not. The arrows show how traffic moves through the
network. These arrows show that, although the Intersection elements will have a big influence on the traffic streams by providing signal settings to each incoming Direction Lane, they will not receive traffic themselves. Instead of that, traffic will be move directly from the Direction Lanes to their downstream Approaches.

The structure consisting of Intersections, Direction Lanes, Approaches and External Arrivals is able to represent a complete road network. This structure, with all the mentioned elements, is contained in an object Network.

The Network object has lists of all Intersections, Direction Lanes, Approaches and External Arrivals, which makes it easy to iterate over all of them when running the simulation. It also contains some parameters that hold for the complete network, like the (average) size of cars and the distance they keep to each other. In Table 2.1, we give an overview of the elements and parameters in the Network element.

<table>
<thead>
<tr>
<th>parameter</th>
<th>symbol</th>
<th>unit</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car space width</td>
<td>$csw$</td>
<td>m</td>
<td>The width of the space a non-moving car needs</td>
</tr>
<tr>
<td>Car space length</td>
<td>$csl$</td>
<td>m</td>
<td>The length of the space a non-moving car needs</td>
</tr>
<tr>
<td>Driving distance</td>
<td>$d_d$</td>
<td>s</td>
<td>The distance cars keep to traffic in front</td>
</tr>
<tr>
<td>Acceleration</td>
<td>$acc$</td>
<td>m/s$^2$</td>
<td>The acceleration of cars when light turns green</td>
</tr>
<tr>
<td>Deceleration</td>
<td>$dec$</td>
<td>m/s$^2$</td>
<td>The acceleration of cars when light turns yellow</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contained element</th>
<th>sort of element</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>${DL}_i$</td>
<td>List of direction lanes</td>
<td>All direction lanes in the network</td>
</tr>
<tr>
<td>${App}_i$</td>
<td>List of approaches</td>
<td>All approaches in the network</td>
</tr>
<tr>
<td>${Int}_i$</td>
<td>List of intersections</td>
<td>All intersections in the network</td>
</tr>
<tr>
<td>${ExtArr}_i$</td>
<td>List of external arrivals</td>
<td>All external arrivals in the network</td>
</tr>
</tbody>
</table>

Table 2.1: Parameters of Intersection elements

2.4 Traffic representation

In the previous chapter, we pointed out that microscopic simulation is not a satisfying technique, since it is too slow to use it for big city networks. We need to represent traffic in a way that does not require separate calculations for each vehicle. As stated in Assumption 1, we will leave out individual vehicles completely, so that no extremely specific information of the composition of traffic is needed (i.e. which part of the vehicles is car, which part is motorcycle, etc), since this data is not always available for each link.
For this thesis we do this, by expressing traffic in PCU’s (Passenger Car Units). Each vehicle represents an amount of PCU’s, according to their size and speed. An average car should represent 1 PCU, while motorcycles (which are much smaller) have a much lower value (for example 0.4). Trucks, on the other hand, can be worth 3 or even 4 PCU’s, because of their big size and slow acceleration.

2.4.1 Concept of traffic blocks

We represent roads in the network as links. Traffic travels over links as blocks with a position and a volume. These blocks travel forward with a speed \( v \) (which only depends on the link \( l \) where the block moves, and is equal for each block), until they hit a queue or pass the end of the link (the intersection where the links leads to) without delay. Obviously, there can be multiple moving blocks on a link, but as long as they are not part of the queue, they do not interfere with each other, as stated in Assumption 2. The model is represented in Figure 2.3.

![Figure 2.3: Impression of blocks on a link](image)

The position of the block, denoted by \( x \), is defined as the distance in meters of the front of the block towards the stopping line of the link. The volume \( V \) of the blocks denotes how many PCU’s are present in the block. A block also has a width (m), which is equal to the width of the link it travels on, and a density (PCU/m\(^2\)), which is equal for all driving blocks on the same link. The length of the block follows from the volume, the width and the density. Because of this, the only parameters a block needs to contain are its position and its volume.

In the simulation, we call these blocks traffic blocks. In Table 2.2, it is shown how traffic blocks are composed.

<table>
<thead>
<tr>
<th>TrafficBlock parameter</th>
<th>symbol</th>
<th>unit</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volume</td>
<td>( V )</td>
<td>PCU</td>
<td>Amount of traffic in block</td>
</tr>
<tr>
<td>Position</td>
<td>( x )</td>
<td>m</td>
<td>Position of block on the link, as distance from the stop line</td>
</tr>
</tbody>
</table>

Table 2.2: Parameters of TrafficBlock elements

Traffic blocks are generated at the start of a link. If the link has an upstream intersection that is also in the network, the blocks are transferred from links approaching the upstream intersection when these links see a green traffic light. If the link has no upstream intersection in the network, traffic is coming from outside the network. In this case, blocks are stochastically generated. Each second, a new traffic block can be generated with a stochastic size. If this size turns out to be zero, the block will not be added to the system.
The size (length and width) depends on the link the block travels on, and the density (PCU/m²) of the blocks. We consider two densities. These are the density of blocks driving at full speed, called \( \rho_d \) (driving density), and the density of queued traffic, called \( \rho_q \) (queueing density). The queueing density is equal for all links in the network, and only depends on the size of an average car (or the size of one PCU). The driving density also depends on the speed \( v_l \) on link \( l \), so this value may be different for different links. How these values will be calculated will be shown in Section 3.4.

We have only discussed driving traffic in this section. The properties of queues will be explained in the following section.

### 2.4.2 Queues

When building a realistic simulation model, a good representation of queues is essential. Queues will appear when traffic can not cross a road section, because of a red traffic light or blockades by other traffic.

A queue on link \( l \) has two main properties: its length (\( QL_l \), in meters) and its volume (\( QV_l \), in PCU). These two values may seem directly proportional, but this is no realistic assumption. When a queue of traffic is standing still in front of a red traffic light, and the light turns green, the front vehicles will start driving directly, causing the queue volume \( QV_l \) to drop immediately. However, the traffic at the back of the queue is still waiting, so the queue length \( QL_l \) has not dropped yet. This explains why we keep both values.

Each link \( l \) has a width \( w_l \). The density of the queue (\( QD_l \)) is calculated as follows:

\[
QD_l = \frac{QV_l}{QL_l \cdot w_l}.
\]

We assume that the queue is completely standing still when \( QD_l = \rho_q \), and that it completely drives when \( QD_l = \rho_d \). Because of this, \( \rho_q \) and \( \rho_d \) are boundary values for \( QD_l \).

Each lane \( l \) contains a queue, even if no vehicles are waiting at all. In this case, we just set \( QV_l = QL_l = 0 \).

### 2.5 External Arrivals

To simulate traffic in a network, we need to generate this traffic at the entrances of the network. Each entrance of the network will be represented by an External Arrival.

An External Arrival will generate traffic blocks, and places them into the Approach they are linked to. The External Arrival will not contain blocks itself, it immediately passes them forward.

We chose to use External Arrivals, instead of generating traffic directly in the Approaches. If we did the latter, we created a fundamental difference between approaches from outside the
network and approaches from an internal intersection. We preferred to keep these approaches identical.

In the simulation, the object is called **ExternalArrival**. In Table 2.3, it is shown how external arrivals are composed.

<table>
<thead>
<tr>
<th>parameter</th>
<th>symbol</th>
<th>unit</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival distribution</td>
<td>$D_{arr}(\bar{\lambda})$</td>
<td>(blocks/s)</td>
<td>Distribution of arriving blocks per second</td>
</tr>
<tr>
<td>Block size distribution</td>
<td>$D_{size}(\bar{\mu})$</td>
<td>(PCU)</td>
<td>Distribution of size of blocks</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contained element</th>
<th>sort of element</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incomingApproach</td>
<td>Approach</td>
<td>The approach to which blocks are transferred</td>
</tr>
</tbody>
</table>

Table 2.3: Parameters of ExternalArrival elements

### 2.6 Links

We refer to a Link as a one direction road between two intersections, or a road between the outside of the network and an intersection. Two-way roads are simply divided in two separate one-way roads. A link starts as an **Approach**. This approach represents the road, with its width, length, speed limit, etcetera, until a certain point.

At this point, we assume that traffic will prefilter: traffic that wants to go right will use the right side of the road, etcetera. From this point, the road separates into multiple lanes, all containing traffic heading for the same direction at the downstream intersection. These lanes are referred to as **Direction Lanes**. The number of direction lanes is equal to the number of directions that can be chosen on the intersection. The sum of the length of the approach, plus the length of one of the direction lanes, should be equal to the length of the link.

The splitting of traffic at some point on the link is essential for our model. If we do this directly at the start of the link, we assume that traffic for different directions do not interact with each other on the rest of the link, which is very unrealistic. If we do the opposite, splitting at the stop line, we would have only one queue which is affected by multiple traffic lights, which is very difficult to model.

In Figure 2.6 a visualisation of our model is given. It is based on the real-life link shown in Figure 2.5.

![Figure 2.5: Example of real life link between two intersections](image)

We chose this model to make the transfer of traffic through intersections easier. Instead of splitting the traffic streams for each direction at the stop line, we will do this earlier. This
Figure 2.6: Link model of Figure 2.5

will particularly help us when only a part of the directions will face green signals. In this case, it is not clear which part of the traffic can cross the intersection, and which part needs to wait for a red signal.

2.7 Approaches

As shortly described in the previous section, approaches are defined as the first part of the link. We have three kinds of approaches:

Approaches between two intersections
An approach that lies between two intersections both contained in the network, starts at the upstream intersection $I_{up}$. They will be filled with traffic by direction lanes that lead to $I_{up}$, and they will feed a set of direction lanes $\{DL\}_i = \{DL_0, DL_1, ..\}$ themselves.

Approaches from outside the network
An approach coming from the outside is not filled by direction lanes, but by an external arrival element. It will still feed a list of direction lanes.

Approaches leaving the network
These approaches are filled by direction lanes arriving at $I_{up}$, but they do not lead to direction lanes. Instead of that, traffic disappears when it reaches the end of the approach.

The approach element is built in such a way that we can use the same element for each case. The approach does not need to know whether it is fed from direction lanes or from an external arrival.

When the list of downstream direction lanes $\{DL_i\}$ is larger than one, the approach needs to share traffic between the lanes. This can be done by just providing a list $\{q_i\}$ of proportions, where $\sum q_i = 1$, and provide $q_i$ of total traffic to $DL_i$. It is also possible to make the distribution of traffic more stochastic, so that the $q_i$’s are only expected proportions. We will do the latter, which is described in Section 3.6.3.

The parameters and elements described in Table 2.4 are needed to specify an approach:
### Approach

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>$l$</td>
<td>m</td>
<td>Length of approach</td>
</tr>
<tr>
<td>width</td>
<td>$w$</td>
<td>m</td>
<td>Width of approach</td>
</tr>
<tr>
<td>velocity</td>
<td>$v$</td>
<td>m/s</td>
<td>Speed of driving blocks</td>
</tr>
<tr>
<td>traffic distribution</td>
<td>${q_i}$</td>
<td>List of fractions</td>
<td>Parameters defining which part of traffic flows to which direction lane</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contained element</th>
<th>sort of element</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{up}$</td>
<td>Intersection</td>
<td>The upstream intersection</td>
</tr>
<tr>
<td>${DL_i}$</td>
<td>List: DirectionLane</td>
<td>A list of downstream direction lanes</td>
</tr>
</tbody>
</table>

Table 2.4: Parameters of Approach elements

### 2.8 Direction Lanes

As mentioned earlier, direction lanes form a link between an approach and its downstream intersection. It contains traffic that will head to one specific direction on the intersection. Because of this, the direction lane is also connected to the Approach to which the traffic will be transferred.

An important part of the direction lane is its queue. During the simulation, the queue length (in meters) and queue volume (in PCU’s) are kept. These are not linearly dependent of each other, since the density of traffic in the queue will be variable. The way in which this value varies will be discussed later on. Apart from the queue, the direction lane can also contain driving traffic blocks.

The direction lane does not determine the traffic signal state, this is done by the intersection. If the signal is reported as green, there should be determined how much traffic departs this second. This is done by the direction lane element. This value can be fixed, but it can also be a stochastically determined value. In both situations, a departure intensity $\mu$ should be set, defined as the expected amount of traffic departing per second. In the rest of this paper, we will not determine $\mu$ stochastically.

In Table 2.5, we see which parameters and elements should be specified for a Direction Lane:

### 2.9 Signalized intersections

The main task of the Intersection element is determining the signal setting for each time step, and communicating these settings to the corresponding direction lanes. The Intersection should contain the algorithm to determine these settings: if fixed settings are used, a simple list of setting per direction lane is sufficient, but for adaptive settings an algorithm should be implemented, which should use traffic information from the coupled Direction Lanes and/or Approaches.

Intersections are coupled to all their incoming Direction Lanes, and their outgoing Ap-
DirectionLane

<table>
<thead>
<tr>
<th>parameter</th>
<th>unit</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>l</td>
<td>m</td>
</tr>
<tr>
<td>width</td>
<td>w</td>
<td>m</td>
</tr>
<tr>
<td>velocity</td>
<td>v</td>
<td>m/s</td>
</tr>
<tr>
<td>departure intensity</td>
<td>μ</td>
<td>PCU/s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contained element</th>
<th>sort of element</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_{down}$</td>
<td>Intersection</td>
<td>The downstream intersection</td>
</tr>
<tr>
<td>$App_{up}$</td>
<td>Approach</td>
<td>The upstream approach (source of traffic)</td>
</tr>
<tr>
<td>$App_{down}$</td>
<td>Approach</td>
<td>The downstream approach (destination of traffic)</td>
</tr>
</tbody>
</table>

Table 2.5: Parameters of DirectionLane elements

approaches. The outgoing approaches are saved in a list \{App\}_i, the incoming Direction Lanes are saved in a two-dimensional array \{DL\}_{ij}. In this array, Direction Lanes coming from the same direction get the same index \(i\), and Direction lanes with the same destination get the same index \(j\). This mostly results in an array with some empty elements (there will be no Direction Lane for making a U-turn), but it leads to more structured code.

In Table 2.6, we depict which parameters and elements should be specified for an Intersection:

Intersection

<table>
<thead>
<tr>
<th>parameter</th>
<th>unit</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SignalSetting</td>
<td>algorithm</td>
<td>Determines which direction gets green light at which time epoch</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contained element</th>
<th>sort of element</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{DL}_{ij}</td>
<td>Array of direction lanes</td>
<td>All direction lanes entering the intersection. Index (i) represents their origin, index (j) represents their destination.</td>
</tr>
<tr>
<td>{App}_i</td>
<td>List of approaches</td>
<td>All {App}_0, {App}_1, ... leaving the intersection</td>
</tr>
</tbody>
</table>

Table 2.6: Parameters of Intersection elements
Chapter 3

Implementation of the simulation model

In this chapter, the algorithm will be explained extensively. We implemented this algorithm in Java.

We will split this chapter in several sections, each explaining a specific part of the simulation algorithm. The following sections will explain how:

- Traffic blocks are moved forward
- New traffic blocks are created
- Traffic lights are being set
- Traffic blocks cross intersections
- The queue is modelled in the direction lanes
- Traffic blocks are transferred from approach to direction lane
- The queue is modelled in the approaches
- The performance of the traffic light settings is measured

The algorithm operates by increasing time in fixed steps of one second. In each time step, all the tasks above are executed. Throughout the rest of the section, the variable $t$ represents the current time (s).

3.1 Forward movement of traffic blocks

Traffic blocks are portions of traffic driving on an approach or a direction lane (from now on called the lane $l$). We refer to the end of lane $l$ as the downstream exit of $l$. A block $b$ is represented by a position $x_b$ (which represents the distance between the front of the block and the end of the lane) and a volume $V_b$. However, in our implementation, we replaced the position $x_b$ by the departure time $t_b$, which is the time at which the front of the block has
passed the start of the lane. We did this because the value $t_b$ does not have to be adjusted every time step. $x_b$ can be calculated easily when $t_b$ is known:

$$x_b = l_l - (t - t_b)v_l,$$

where $l_l$ and $v_l$ are respectively the length and the driving speed of the lane.

Traffic blocks are always driving at speed $v_l$, defined on the lane $l$ it is driving on. If a traffic block needs to stop (because it joins a queue or faces a red traffic light), the traffic of the block is added to the queue ($Queue_l$) of the lane. If the queue is empty, the traffic block becomes the new queue. More details on adding blocks to queues can be found in Section 3.5.3.

We save all blocks $b$ that are driving on $l$ in an ArrayList called $blockList_l$. Blocks are ordered by departure time; the block with the lowest departure time is in first position. This is the front block ($fb$), since all other blocks entered $l$ at a later moment of time.

For each time step in which $blockList_l$ is not empty, we check the front block position $x_{fb}$. This would be position of the block if it has not joined the queue or reached the end of $l$ in last time step. We should check whether this is the case or not.

The queue length $QL_l$ plays an important role in this process, because the process depends on whether there is a queue ($QL_l > 0$) or not ($QL_l = 0$). If $QL_l > 0$, we check whether $x_{fb} < QL_l$. If this is the case, we add the front block to the queue.

If $QL_l = 0$ we check whether $x_{fb} < 0$, so whether the block has reached to the end of the lane. If this is the case, the procedure depends on whether $l$ is an approach or a direction lane. On an approach, we transfer the block to the downstream direction lanes. On a direction lane, we check whether the signal is red or green. If it is red, the traffic block is stopped at $x = 0$, so we turn $fb$ into a new queue on $l$. If it is green, we check which part of the block has yet crossed the line. This part is transferred to the next approach, the rest is still turned into a queue. How we calculate this part is handled in Section 3.4.

If the block is transferred from Approach to Direction Lane, it can also occur that one of the Direction Lanes is (almost) completely filled by traffic. When we transfer a traffic block from an approach to direction lanes, we always check the capacity of these direction lanes. If the capacity is lower than the size of the traffic block, the block can not be added completely to the direction lanes, and a queue will appear on the approach. How we calculate this capacity, and how we handle this case, is explained in Section 3.6.

If the block is deleted from $blockList_l$, we should continue with checking the next front block (as long as the list is not empty).

\[
\begin{array}{c}
\text{l}_l \\
\text{blockList}_l = \{b_0, b_1, b_2\} \\
\end{array}
\]

Figure 3.1: Example of blocks on a link (with no queue)

\textbf{Algorithm} Moving blocks forward on Approach($fb = \{blockList_l\}_0$)
Figure 3.2: Example of blocks on a link (with queue)

1. while blockList₀ is not empty and $x_{fb} < QL$
2.   if $QL > 0$
3.     Add $fb$ to Queue₀
4.   else
5.     Add $fb$ to direction lanes
6. Delete $fb$ from blockList

Algorithm Moving blocks forward on Direction Lane($fb = \{blockList₀\}_0$)
1. while blockList₀ is not empty and $x_{fb} < QL$
2.   if $QL > 0$
3.     Add $fb$ to Queue₀
4.   else
5.     if traffic light is green
6.         Add first part of $fb$ to destination approach
7.         Turn rest of $fb$ into Queue₀
8.     else
9.         Add $fb$ to Queue₀
10. Delete $fb$ from blockList

3.2 Generating new traffic blocks

At positions where roads enter the network, blocks should be generated. Generating blocks is done by External Arrival elements, which are coupled to incoming Approaches.

Generating new blocks is a stochastic process. Each time step, the simulation has to determine:

- Whether a block is generated.
- How many traffic the blocks contain

The distributions used to determine these values should be chosen to give a good representation of the real traffic input. Input processes can vary, traffic can for example arrive in a constant stream (modelled by a high arrival intensity of blocks with small sizes) or in big platoons (modelled by a low arrival intensity of huge blocks).

When a block $b$ is generated, it is added to the block list of its approach $l$. It is added to the last position, since it is the backmost blocks of the approach. Its arrival time $t_{b}$ is set to $t$, the current time.
Let $D_{l,t}(.)$ be the distribution of the block size arriving at time $t$ on approach $l$. Then blocks are generated according to the following algorithm:

**Algorithm Generate blocks for Approach $l$(time $t$)
1. Generate random variable $V \sim D_{l,t}(.)$
2. If $V > 0$
3. Generate new block $b$
4. Set $t_b \leftarrow t$, $V_b \leftarrow V$
5. Add $b$ to blockList$_l$

### 3.2.1 Determining block size distribution

The block size distribution $D_{l,t}(.)$ represents the distribution of the volume $V_t$ of the arriving block at time $t$. Since we have one arriving block per second, $D_{l,t}(.)$ is also the distribution of arriving traffic volume per second.

We want our block size distribution to be as realistic as possible. For our incoming approaches $l$, we assume we have the following information:

- All block size distributions $\{D_{l,t}(.)|t=0,1,2...\}$ are i.i.d., which leads to a general block size distribution $D_l(.)$.
- We know the expected amount of traffic volume per second on $l$, denoted as $\lambda_l$.
- We know the composition of the traffic.

To achieve a realistic arrival process, we will generate individual vehicles. The last assumption states that we know how traffic is composed. This means that we know which part of the vehicles consists of motorcycles, which part consists of cars, etcetera. In order to do this, we define vehicle classes. Each vehicle class $vc$ has the following characteristics:

- The probability $p_{vc}$ that an arriving vehicle is of class $vc$.
- The minimum size $s_{vc}^{min}$ and maximum size $s_{vc}^{max}$ (in PCU) of a vehicle of class $vc$.

We assume that the size of a single vehicle $s_{vc}$ of class $vc$ is uniformly distributed between the values $s_{vc}^{min}$ and $s_{vc}^{max}$. Let $VC = \{vc_0, vc_1, \ldots\}$ be the set of vehicle classes.

For example, assume we have three kinds of vehicles: motorcycles, trucks and cars. We have $VC = \{motorcycle, truck, car\}$. We can define the class car with $p_{car} = 0.6$ (so 60% of the vehicles is a car), $s_{car}^{min} = 0.9$ and $s_{car}^{max} = 1.1$. The size of a generated car $s_{car}$ is uniformly distributed between 0.9 and 1.1, so $E(s_{car}) = 1.0$. We should define these variables for motorcycles and trucks in a similar way. It is clear that it should hold that $p_{car} + p_{motorcycle} + p_{truck} = 1$.

The arrival intensity of vehicles ($\lambda_{veh}$) per second is not equal to $\lambda$, the expected amount of traffic volume per second. However, it can be deduced easily:

$$\lambda_{veh} = \frac{\lambda}{\sum_{vc \in VC} p_{vc} \cdot 0.5(s_{vc}^{min} + s_{vc}^{max})}$$
The arrival intensity of vehicle class $vc$ is now equal to $\lambda_{veh}p_{vc}$.

We will determine the size of the block as follows. For each vehicle class $vc$, we determine the amount of vehicles arriving at this time step. We assume this value is Poisson distributed with parameter $\lambda_{veh}p_{vc}$. For each vehicle of class $vc$, we will determine its size, which we will draw from a Uniform distribution with parameters $s_{vc}^{min}$ and $s_{vc}^{max}$. The sum of all these vehicle sizes will be the size of our new traffic block $b$. The distribution $D(t)$ is now as follows:

$$D(t) \sim \sum_{vc \in VC} \sum_{i=0}^{\Lambda_{vc}} S_{vc}^{i}$$

$$\Lambda_{vc} \sim \text{Pois}(\lambda_{veh}p_{vc})$$

$$S_{vc}^{i} \sim \text{Unif}(s_{vc}^{min}, s_{vc}^{max})$$

We use the following algorithm to determine the volume $V_b$ of a block $b$:

**Algorithm** Determine block size $V_b$ of block $b$

1. $V \leftarrow 0$
2. for vehicle class $vc \in VC$
3.     Generate random vehicle $n \sim \text{Pois}(\lambda_{veh}p_{vc})$
4.     for $i \in \{1, 2, ..., n\}$
5.         Generate random variable $S \sim \text{Unif}(s_{vc}^{min}, s_{vc}^{max})$
6.         $V \leftarrow V + S$
7. Return $V$

### 3.3 Setting the traffic lights

An Intersection object does not contain traffic itself, it only contains a set of incoming direction lanes that can contain approaching traffic. Its main task is providing the right traffic light setting to each incoming direction lane. It contains a twodimensional array $DL$ of direction lanes $DL_{ij}$ (where index $i$ represents their origin and index $j$ represents their destination). Each second, it passes to each direction lane in $DL$ the current state of its traffic light, which is a boolean representing green (True) or red (False).

A traffic light setting describes which direction lane faces green light at which time step. At first, we define signal phases $\sigma_k$: time periods in which a specific subset $S_k$ of all direction lanes in $DL$ faces a green light. We make a sequence $\Sigma = \{\sigma_0, \sigma_1, ..\}$ of signal phases, which defines in which order the traffic lights become green. We should make sure that each direction lane $DL_{ij}$ is in at least one of the subsets $S_k$, otherwise this direction lane will never face a green light. An example of a sequence of signal phases is given in Figure 3.3.

For each phase $\sigma_k$, we define a maximum phase length $l_{\sigma_k}^{max}$ and a minimum phase length $l_{\sigma_k}^{min}$, where clearly should hold that $l_{\sigma_k}^{min} \leq l_{\sigma_k}^{max}$.

We also define a yellow length $l_y$, representing the length of the period traffic signals show yellow light at the end of the green period. When a signal phase $\sigma_k$ is ended, the transition to the next phase is made by a period of length $l_y$ in which for each direction lane $DL_{ij}$ it holds that:

- if $DL_{ij} \in S_k$ and $DL_{ij} \notin S_k$, its light is yellow
• if $DL_{ij} \in S_k$ and $DL_{ij} \in S_k$, its light is green

• otherwise, the light is red.

An example of such a yellow phase is given in Figure 3.4.

In the simple case of a fixed cycle traffic light setting, we use the maximum phase lengths $l_{\sigma_k}^{\text{max}}$ as fixed phase lengths. The cycle length $c$ is the sum of these values: $c = \sum_k l_{\sigma_k}^{\text{max}}$. The timing of the signal settings are specified using an offset $o$ represents the time gap between the start of the simulation $t = 0$ and the start of the current cycle. If $o = 0$, the simulation starts at the beginning of the first phase. If $o = 5$, the signal cycle started at $t = -5$. We determine the signal settings at time $t$ as follows:

• If $\sum_{i=0}^{k-1} l_{\sigma_k}^{\text{max}} \leq (t + o) \mod c < \sum_{i=0}^{k} l_{\sigma_k}^{\text{max}} - l_y$, we are in phase $k$. 
\[ \sum_{i=0}^{k} \sigma_k - l_y \leq (t+o) \mod c < \sum_{i=0}^{k} \sigma_k, \]

we are in the yellow period between phases \( k \) and \( k+1 \).

If the light settings are adaptive, the algorithm needs extra information as input. This information can be a check whether traffic is present at a specific Direction Lane, or a check whether queues at downstream roads are not blocking the upstream intersection. We will explain the working of several adaptive signal settings in Section 4.

As mentioned earlier, Intersections do not handle traffic. After the signal settings are provided, traffic from Direction Lanes is transferred directly to their destination Approach.

### 3.4 Traffic blocks crossing intersections

Obviously, traffic can only pass the intersection if the signal is green. There are two criteria to ensure traffic from direction lane \( l \) crosses the intersection:

- The traffic light is green for \( l \).
- There is traffic available on \( l \).

Assumption 4 tells us that traffic could never get blocked by excessive long queues on downstream approaches.

To determine the maximum amount of traffic that can cross the stop line in one second, we look at how much traffic can pass a point in one second. As shown in Figure 3.6, this depends on the speed \( v_l \), the width \( w_l \) of \( l \) and the density of traffic when driving. The first two variables are known, the last one depends on the speed, the size of one PCU (which is equal to the space an average car needs), and the distance vehicles keep to each other.

The space a car (or a PCU) needs when standing still is determined by the Car Space Width (\( cs w \)) and Car Space Length (\( cs l \)), representing the width and length of this needed space. These values are defined in the Network object, so they are equal all over the network. The values \( cs w \) and \( cs l \) are not exactly the size of a car, because a car needs a little more space around it in practice. The queueing density \( \rho_q \) (PCU/m²) is completely determined by these values:

\[ \rho_q = \frac{1}{cs w \cdot cs l}. \] (3.1)

The space a driving car needs is determined by \( cs w \) and \( cs l \), but also by the driving distance \( d_d \) (in seconds) and the speed \( v_l \). When driving, a car keeps a space of length \( v_l \cdot d_d \) in front of it. The width of this area is taken as \( cs w \). The total area a driving car uses is equal to \( cs w \cdot cs l + cs w \cdot v_l \cdot d_d \), as shown in Figure 3.5. The driving density \( \rho_{d_d} \) is determined as follows:

\[ \rho_{d_d} = \frac{1}{cs w \cdot (cs l + v_l \cdot d_d)}. \] (3.2)

The amount of traffic that is able to cross the line in one second can now be determined: \( v_l \cdot w_l \cdot \rho_{d_d} \). This is visualized in Figure 3.6.

The departure rate \( \mu_l \) of the direction lane (in PCU/s) is equal to \( v_l \cdot w_l \cdot \rho_{d_d} \) when traffic is able to cross the stop line at free flow. However, when traffic needs to make a turn, it has
to slow down a bit, which lowers the departure rate. For this reason, an extra parameter \( f_l \) is added to the direction lane, which is a parameter between 0 and 1 which determines how heavily traffic is slowed down at the intersection. We now have:

\[
\mu_l = v_l \cdot w_l \cdot \rho_d \cdot f_l.
\]

When traffic can pass the stop line at free flow, \( f_l \) should be set to 1.

Another thing that can influence the departure rate is the fact that traffic needs to accelerate when light turns from red to green. At the moment the light turns green, traffic is still standing still, so assuming traffic crosses the line at free flow is very inaccurate.

We will edit the departure intensity \( \mu_l \) to \( \bar{\mu}_l(t) = \mu_l \cdot v_{av}(t)/(f_l \cdot v_l) \), where \( v_{av}(t) \) is the average speed during the time step. When traffic drives at full speed during the complete time step, \( \bar{\mu}(t) \) will result in the old value \( \mu_l \).

Now, we need to calculate \( v_{av}(t) \). Assume that the time \( t_{acc} \) that traffic needs to fully accelerate is equal to \( f_l \cdot v_l/\text{acc} \). Let \( t_g \) be the time at which the light turns green, such that \( t_g + t_{acc} \) is the moment traffic drives at full speed. When we want to calculate \( v_{av}(t) \) for time step \( t \) (which lasts from \( t \) to \( t + 1 \)), we can have three situations:

**Situation 1:** \( t \geq t_g + t_{acc} \)

Since traffic is fully accelerated, we have \( v_{av}(t) = f_l \cdot v_l \).

**Situation 2:** \( t + 1 \leq t_g + t_{acc} \)

Traffic is accelerating during the complete time step. The average speed is equal to the speed at \( t + 1/2 \), so we have \( v_{av}(t) = (t + 1/2 - t_g)\text{acc} \).
**Situation 3:** $t < t_g + t_{acc} < t + 1$

Traffic accelerates from $t$ to $t_g + t_{acc}$. From $t_g + t_{acc}$ to $t + 1$, it drives at speed $f_l \cdot v_l$. In this case, $v_{av}(t) = f_l \cdot v_l - (t_g + t_{acc} - t)^2 \cdot 1/2 \cdot acc$.

On the other hand, when traffic faces yellow light, we assume that the speed decreases. Traffic has a deceleration of $dec$ m/s$^2$. When light turns yellow (at time $t_y$), traffic is still driving at speed $v_l \cdot f_l$. The time $t_{dec}$ the traffic needs to stop completely is equal to $f_l \cdot v_l/dec$.

When we want to calculate $v_{av}(t)$ for time step $t$ (which lasts from $t$ to $t + 1$), we can have three situations:

**Situation 1:** $t \geq t_y + t_{dec}$

Since traffic is completely decelerated, we have $v_{av}(t) = 0$.

**Situation 2:** $t + 1 \leq t_y + t_{dec}$

Traffic is decelerating during the complete time step. The average speed is equal to the speed at $t + 1/2$, so we have $v_{av}(t) = f_l \cdot v_l - (t + 1/2 - t_y)dec$.

**Situation 3:** $t < t_g + t_{dec} < t + 1$

Traffic decelerates from $t$ to $t_g + t_{acc}$. From $t_g + t_{acc}$ to $t + 1$, it stands still. In this case, $v_{av}(t) = (t_g + t_{dec} - t)^2 \cdot 1/2 \cdot dec$.

After these calculations, we know our adjusted departure intensity $\bar{\mu}_l(t)$.

If traffic is present and the traffic light is green or yellow, we have two cases in which traffic can pass the stop line:

- A queue is present, so traffic leaves the queue
- No queue is present, but a driving block crosses the stop line.

If a driving block crosses the stop line, we have the situation discussed in Section 3.1. The front block position $x_{fb}$ (which is a negative number) and the volume $V_{fb}$ determines how many traffic has crossed the stop line: the maximum amount that can cross the line is $-x_{fb} \cdot w_l \cdot \rho_d$, but if $V < -x_{fb} \cdot w_l \cdot \rho_d$ the complete block crosses the line. In both cases, we add a new block $b$ to the destination approach $a$. The position $x_b$ is set to $l_a + x_{fb}$, the volume $V_a$ is equal to the departed volume.

When only a part of the arriving block is departed, we remember the new block $b$ on $a$ as feeding block ($b_{feed,l}$), so we can put more traffic in it in later time steps. When the complete block passed, there is no more traffic to add to the block, so no $b_{feed,l}$ is kept track of.

**Algorithm** *Driving block fb crosses stop line of l(time t, destination approach a)*

1. $C \leftarrow -x_{fb} \cdot w_l \cdot \rho_d$
2. if $C < V_{fb}$
3. Generate new block $b$
4. $x_b \leftarrow l_a + x_{fb}$, $V_b \leftarrow C$
5. add $b$ to blockList$_a$
6. $b_{feed,l} \leftarrow b$
7. add volume $V_{fb} - C$ to the queue of $l$
8. delete $fb$ from blockList$_l$
9. 
else
10. Generate new block $b$
11. $x_b \leftarrow l_a + x_{fb}$, $V_b \leftarrow V_{fb}$
12. add $b$ to blockList$_a$
13. delete $fb$ from blockList$_l$

If $QV_i$ is the queue volume (the amount of traffic in the queue), the amount of transferred traffic is the minimum of $\bar{\mu}_l(t)$ and $QV_l$. If a feeding block $b_{feed}$ is remembered, we add the traffic to the feeding block. If no feeding block is in memory, we generate a new block $b$ with departure time $t$. We remember the block as $b_{feed,l}$ when the queue is not yet empty, otherwise we forget it.

**Algorithm** *Part of queue crosses stop line of l(time t, destination approach a)*
1. if $b_{feed,l}$ exists
2. 
else
4. $b$ is new empty Traffic Block
5. $t_b \leftarrow t$, $V_b \leftarrow 0$
6. add $b$ to blockList$_a$
7. if $\bar{\mu}_l(t) < QV_l$
8. $V_b \leftarrow V_b + \bar{\mu}_l(t)$
9. distract volume $\bar{\mu}_l(t)$ from the queue of $l$
10. $b_{feed,l} \leftarrow b$
11. 
else
12. $V_b \leftarrow V_b + QV_l$
13. empty the queue of $l$
14. $b_{feed,l} \leftarrow$ null

Note that we have mentioned the queue length $QL_l$ (m) in Section 3.1, while we used the queue volume $QV_l$ (PCU) in this section. We use both variables, because they are not linearly dependent: the density of the queue can vary. We will discuss the queue dynamics in next section.

There is one more case in which we should discard $b_{feed,l}$: when the traffic signal turns to red. At this moment, the block will no longer be fed. This is also a step in the simulation: when the signal is red, while it was green last time step, then set $b_{feed,l}$ to null.

### 3.5 The modelling of the queue in Direction Lanes

Queue modelling is one of the most important tasks of our simulation model. Queue lengths and waiting times determine whether signal settings are well functioning. It is essential to model the queues in a realistic way, but we also have to keep it as simple as possible, to avoid creating queue algorithms that make the algorithm inefficient.

Queues are, as mentioned earlier, represented by a length $QL_l$ in meters and a volume $QV_l$ in PCU’s. These are definitely depending on each other, since a long queue is very likely
to contain many PCU’s, but a completely linear dependency between $QL_l$ and $QV_l$ is very unrealistic. If a long queue is present, and the light turns from red to green, cars start to depart immediately (so $QV_l$ decreases), but the back of the queue will not move yet (so $QL_l$ does not decrease). When, on the other hand, light turns from green to red when a queue is still present, cars will stop crossing the line immediately, but the queue will still get shorter since cars are still moving.

We have implemented three models to represent queues, each with their own specific benefits and disadvantages. We will explain and discuss them in more detail now, and make a choice between them afterwards.

### 3.5.1 Model 1

The relation between the queue length $QL_l$ and the queue volume $QV_l$ can be represented as the queue density $QD_l$, the average amount of PCU’s on a square meter in the queue. The relation, as stated earlier in Section 2.4.2 is as follows:

$$QD_l = \frac{QV_l}{QL_l \cdot w_l},$$

so the density of the queue is its volume divided by its surface.

We defined the queueing density $\rho_q$ (3.1) and the driving density $\rho_d$ (3.2), which we will from now on use as boundary values for $QD_l$:

$$\rho_d \leq QD_l \leq \rho_q.$$ 

If the complete queue is standing still, its density will be $\rho_q$, while if the complete queue is driving at speed $v_l$, its density will be $\rho_d$.

Our first model is based on two assumptions. If a queue gets right of way after a period of red light, the front of the queue starts to drive, while the back of the queue still has to wait. We assume that the back of the queue starts driving when the density $QD_l$ is reduced to $\rho_d$. As long as $QD_l \geq \rho_d$, the back of the queue will not move forward. This is illustrated in Figure 3.7. The boundary between the driving traffic in the front and the queueing traffic in the back moves backward with a speed $v_{tw}$, which is not necessarily equal to $v$. This boundary is called a traffic wave $tw$. Since $v_{tw}$ is not relevant for this algorithm, we will not calculate its value.

In each time step with a queue present and a green signal, we decrease $QV_l$ with $\tilde{\mu}_l(t)$ (until $QV_l = 0$). We set $QL_l$ as the minimum of the old value of $QL_l$ and $QV_l / (\rho_d \cdot w_l)$, which is the queue length when the queue density is $\rho_d$. This leads to the following algorithm:

**Algorithm** Edit queue of direction lane $l$ when signal is green (Model 1)

1. Transfer $\min\{\tilde{\mu}_l(t), QV_l\}$ to destination approach $a$
2. $QV_l \leftarrow \max\{QV_l - \tilde{\mu}_l(t), 0\}$
3. $QL_l \leftarrow \min\{QL_l, QV_l / (\rho_d \cdot w_l)\}$

The second assumption is made for the situation with a red signal. When the signal is red, the back of the queue keeps moving forward with speed $v_l$, until queue density $QD_l$ is equal to $\rho_q$. In each time step, we decrease $QL_l$ with $v$, until $QL_l = QV_l / (\rho_q \cdot w_l)$.

**Algorithm** Edit queue of direction lane $l$ when signal is red (Model 1)
\[ QD_l = \rho_q \]
\[ \rho_d \leq QD_l \leq \rho_q \]
\[ QD_l = \rho_d \]

\[ QD_l = \rho_q \]
\[ \mu_l(t) \]
\[ \mu_l(t) \]
\[ \mu_l(t) \]

Figure 3.7: Illustration of green light queue behaviour (Model 1)

Figure 3.8: Illustration of red light queue behaviour

1. \[ QL_l \leftarrow \max\{QL_l - v, QV_l / (\rho_q, w_l)\} \]

The algorithm is illustrated in Figure 3.8.

We also need to define how the queue length is affected when a driving block (the front block \( fb \) of the blockList) reaches the back of the queue. We know the position of the front of this block \( x_{fb} \) as well as its volume \( V_{fb} \), and the length of the queue \( QL_l \). Because \( fb \) reached the queue, we know \( x_{fb} \leq QL_l \).

The position of the back of \( fb \) plays a role in this process. If the front of \( fb \) hits the queue in the current time step, it is not sure the back of \( fb \) also hits the queue in that time step. In both cases, we can calculate the position of the back of \( fb \):

- If the back of the \( fb \) is also stopped, the complete block is at queueing density \( \rho_q \). In this case, the length of the queue is increased with the length of \( fb \) at queueing density: \( V_{fb}((w_1, \rho_q)) \). This is visualized in Figure 3.9.

- If the back of the \( fb \) is not stopped, it is still at the same distance from \( x_{fb} \) as it was before the block hit the queue. In this case, the complete block enters the queue, but not at queueing density. The queue length is set to \( x_{fb} + V_{fb}((w_1, \rho_d)) \). This is visualized in Figure 3.10.

Algorithm Block \( fb \) enters queue (Model 1)

1. \[ QL_l \leftarrow \max\{QL_l + V_{fb}/(w_1, \rho_q), x_{fb} + V_{fb}/(w_1, \rho_d)\} \]
The way the blocks enter the queue is modelled quite realistically. There is, however, one drawback of this model. If a big block joins the queue when the light is green and $QD_l > \rho_d$, the queue length $QL_l$ will stay equal until $QD_l = \rho_l$ or until the light turns red, since the model assumes that the front of the queue drives and the back of the queue stands still. This is unrealistic, since the back of the queue should drive along in a realistic representation. The problem is visualized in Figure 3.11.

Model 1 implicitly makes the following two assumptions:

- If the light is green and $\rho_d < QD_l < \rho_q$, the back of the queue is not moving.
- If the light is red and $\rho_d < QD_l < \rho_q$, the back of the queue moves.

As shown, these assumptions do not have to hold. This is a big disadvantage of Model 1. The big advantage of this model is that it is fast, since the traffic waves (the borders between driving and waiting parts of the queue) are not kept in memory. In more difficult models, we should keep these waves in memory. A model which does keep traffic waves in memory is Model 3, a model which is slightly more difficult and requires more calculations, but is more realistic. In Model 2, we use the same assumptions for queue behaviour as in Model 1, but we try to avoid the unnecessarily long queues by editing the block entering procedure, without making the simulation procedure very slow.
3.5.2 Model 2

As mentioned in previous subsection, Model 1 sometimes makes unrealistic assumptions when long blocks enter the queue while the traffic light is green. We want to avoid this by changing the block entering procedure.

When the back of the queue drives during an entire time step, no moving blocks will join the queue, since they move at the same speed as the back of the queue. They can only enter the queue if its back is standing still. Because of this, we are able to let all blocks enter the queue at queueing density. In Figure 3.12 we show why this is the case. When Model 1 is used, the backmost block would enter the queue too early.

A drawback of this model is that the queue length is still not represented correctly. For a totally realistic representation of the queue, we should remember exactly which part of the queue drives and which does not. In Model 3, we will present a model using a more detailed...
representation of the queue.

### 3.5.3 Model 3

Model 3 will represent queues in a highly realistic way. To achieve this, it will use traffic waves, which are borders between driving traffic and non-driving traffic inside the queue.

For this model, we assume that a queue exists of parts of two different densities, known as the formerly mentioned queueing density $\rho_q$ and driving density $\rho_d$. We know that (when a queue is present) the front of the queue drives when the light is green, while the front of the queue stands still when the light is red.

Assume that a queue is completely standing still, and the light is red. When the light turns green, the front of the queue starts driving. Now, we have a part of the queue that moves, and a part that is still waiting. Between them, there is a border, called a traffic wave $tw$. Every time step, the part of the queue that drives gets bigger, so the traffic wave moves backwards.

When the light becomes red while the queue is not empty yet, another traffic wave will be generated, as border between the part in front that is standing still and the part behind it that drives. This wave also moves backwards, since the part of the traffic that stands still is growing each time step.

![Figure 3.13: Illustration of traffic wave speed determination](image)

An example of a queue with waves is given in Figure 3.13. We store the traffic waves $tw$ in a list called `waveList`. An element $tw$ contains its position $x_{tw}$ and a volume $V_{tw}$, which is the volume that is in front of the wave. We order the waves in `waveList` by distance to the stop line. We do this in a descending order: the behindmost wave is in position 0 of the list.

**Determining backward speed of traffic waves**

Now, we want to determine at which speed the waves are running backwards. We distinguish three kinds of waves, shown in Figure 3.13:

- Traffic waves where traffic stops driving ($tw_2$). These waves have queueing traffic in front of them and driving traffic behind them.

- The front wave when the light is green ($tw_1$). This wave has driving traffic in front of it, and queueing traffic behind it.
• Other traffic waves where traffic starts driving ($tw_3$).

For each of the traffic waves, we will determine their backward speed $v_{tw}$. We will show that all traffic waves have the same backward speed (which is $v_{tw} = (v_l \cdot \rho_d)/(\rho_q - \rho_d)$), except for the front traffic wave when the traffic light is green.

We should note that a queue can have more traffic waves than shown in 3.13, but these waves show the same behaviour as $tw_2$ (when traffic stops driving) or $tw_3$ (when traffic starts driving).

**Traffic waves where traffic stops driving**

A traffic wave $tw$ where traffic stops driving ($tw_2$ in the example) is followed by a part of driving traffic that will be stopped in the next time step. The traffic that is exactly at the position of $tw$ ($x_{tw}$) is stopped exactly at time $t$. Now, we want to determine $v_{tw}$, the speed of the wave.

We know the position of the wave at time $t+1$ is $x_{tw} + v_{tw}$. All traffic between $x_{tw}$ and $x_{tw} + v_{tw}$ has been stopped between $t$ and $t+1$, and the density of this traffic is $\rho_q$. We define $V$ as the volume of this part of traffic.

We also know that traffic positioned exactly on $tw$ at time $t+1$ has traveled over $v_l$ meters since time $t$, while traffic on $x_{tw}$ is standing still since time $t$. This means the traffic volume $V$ was distributed between positions $x_{tw}$ and $x_{tw} + v_{tw} + v_l$ at time $t$. The density of this traffic was $\rho_d$, while the volume of this traffic was also equal to $V$. Now we have two expressions for $V$:

\[ V = (x_{tw} + v_{tw} - x_{tw}) \cdot w_l \cdot \rho_q = v_{tw} \cdot w_l \cdot \rho_q \]
\[ V = (x_{tw} + v_{tw} + v_l - x_{tw}) \cdot w_l \cdot \rho_d = (v_{tw} + v_l) \cdot w_l \cdot \rho_d \]

Now that we have two expressions for $V$, both containing $v_{tw}$, we can rewrite the following equation to get an expression for $v_{tw}$:

\[ v_{tw} \cdot w_l \cdot \rho_q = (v_{tw} + v_l)w_l \cdot \rho_d \]
\[ v_{tw} \cdot \rho_q = (v_{tw} + v_l) \cdot \rho_d \]
\[ v_{tw} \rho_q - \rho_d = v_l \cdot \rho_d \]
\[ v_{tw} = \frac{v_l \cdot \rho_d}{\rho_q - \rho_d} \]

We illustrated the situation in Figure 3.14, where $y_a$ represents the length of traffic block $V$ while it was driving, and $y_b$ represents the length of $V$ while it stopped moving. It is shown that $y_a = v_l + v_{tw}$ and $y_b = v_{tw}$.

Now we know how to calculate the backward traffic wave speed $v_{tw}$. To gain more insight in this value, we will rewrite its expression further by expressing $\rho_q$ and $\rho_d$ in terms of $csw$, 

32
Figure 3.14: Illustration of traffic wave speed determination

csl and \( d_d \):

\[
v_{tw} = \frac{\frac{v_l \cdot \rho_d}{\rho_q - \rho_d}}{\frac{csw \cdot csl + csw \cdot \rho_q \cdot d_d}{csw \cdot csl + csw \cdot \rho_q \cdot d_d}} = \frac{\frac{v_l \cdot csl}{csl + v_l \cdot d_d}}{\frac{csl}{csl + v_l \cdot d_d}} = \frac{\frac{v_l}{csl + v_l \cdot d_d}}{\frac{csl}{csl + v_l \cdot d_d}} - 1 = \frac{v_l}{\frac{csl}{d_d}}
\]

This is an intuitive result: \( d_d \) is the driving distance between two cars. When a car stops, it takes \( d_d \) seconds for the next car to reach the back of its predecessor and stop, which means that each \( d_d \) seconds, the back of the waiting part of the queue shifts one car backwards.

Since the length of a car is defined as \( csl \), the results is that each \( d_d \) seconds, the back of the queue shifts \( csl \) meters backwards.

Traffic waves where traffic starts driving

A traffic wave \( tw \) where traffic starts driving which is not the front wave (so \( tw_1 \) in the example) requires a quite similar approach for calculating \( v_{tw} \), compared to the waves where traffic stops.

A traffic wave \( tw \) where traffic starts driving is followed by a part of standing still traffic that will start driving in the next time step. The traffic that is exactly at \( x_{tw} \) starts driving exactly at time \( t \). This traffic is at position \( x_{tw} - v_l \) at time \( t + 1 \). Traffic that is at position \( x_{tw} + v_{tw} \) at time \( t \) is still there at time \( t + 1 \), but then it starts driving immediately.

At time \( t \), a volume \( V \) is positioned between \( x_{tw} \) and \( x_{tw} + v_{tw} \) with density \( \rho_q \). At time \( t + 1 \), the same volume is positioned between \( x_{tw} - v_l \) and \( x_{tw} + v_{tw} \) with density \( \rho_d \). This leads to the following expressions for \( V \):

\[
V = ((x_{tw} + v_{tw}) - x_{tw}) w_l \cdot \rho_q = v_{tw} \cdot w_l \cdot \rho_q
\]

\[
V = ((x_{tw} + v_{tw}) - (x_{tw} - v_l)) \cdot w_l \cdot \rho_d = (v_{tw} + v_l) \cdot w_l \cdot \rho_d
\]
These expressions are equal to the expressions for $V$ in previous section. This means that if we have a standing still part of the queue, the amount of traffic that departs each time step is equal to the amount of traffic that joins it. Since both the departing and the arriving traffic drives at the same speed $v$, this is intuitive.

We conclude:

$$v_{tw} = \frac{v_l \cdot \rho_d}{\rho_q - \rho_d} \left( = \frac{csl}{d_d} \right)$$

Just as in the previous section, the result for the backward wave speed $v_{tw}$ is intuitive. When a car departs, it takes $d_d$ seconds until the next car can depart if it wants to keep the correct distance. The result is that each $d_d$ seconds, a car of length $csl$ departs.

An illustration of this situation is given in Figure 3.15. From now on, we will use $v_{wav}$ as parameter for the backward speed of all traffic waves.

![Figure 3.15: Illustration of traffic wave speed determination](image)

This result means both kinds of waves move at same speed, which is a nice result which heavily simplifies the implementation, since it implies that traffic waves can not pass each other.

**Front traffic wave when light is green**

The speed of the front wave $f_w$ can be different compared to the other waves due to the speed at which cars pass the stop line. If the cars have to make a sharp left turn, the speed is lower than $v_l$. For this case, we introduced the variable $f_l$ in Section 3.4, which represents the factor with which the speed $v_l$ is lowered when traffic crosses the stop line.

To determine the backward wave speed $v_{fw}$, we use the same reasoning as in previous section, but we replace $v_l$ by $v_l.f_l$ and $v_{tw}$ by $v_{fw}$. This is illustrated in Figure 3.16.

![Figure 3.16: Illustration of traffic wave speed determination](image)
The resulting expressions for $V$ are:

$$V = ((x_{fw} + v_{fw}) - x_{fw}).w_l.\rho_q = v_{fw}.w_l.\rho_q$$

$$V = ((x_{fw} + v_{fw}) - (x_{fw} - v_l.f_l)).w_l.\rho_d = (v_{fw} + v_l.f_l).w_l.\rho_d$$

These expressions give the following result:

$$v_{fw} = \frac{v_l.f_l.\rho_d}{\rho_q - \rho_d} = f_l.v_{wav}$$

This means that the backward speed of the front wave is also reduced by the factor $f_l$.

The queue movement algorithm

Now we will define the algorithm that describes the queue dynamics. Each time step at which a queue is present, this algorithm needs to take the following steps:

- Generate a new wave if light turns red or green this time step
- Move all waves backwards
- Determine queue length
- Delete wave that passed the end of the queue
- Let driving blocks join the queue

First, we give the algorithm that moves all blocks backwards. We know how to edit the position $x_{tw}$ of a wave $tw$, but we also need to edit the volume $V_{tw}$ in front of $tw$. We will distinguish two cases: editing the volumes when the light is red and when the light is green.

Before we do this, we define $fw$ as the front wave, which is the traffic wave closest to the stop line. We also define the second wave $sw$, which is the closest wave behind $fw$.

Updating $V_{tw}$ when light is red

First, note that all traffic waves move backwards with the same speed. A result of this fact is that the volume between the waves stays equal, so for all waves $tw$, we have the same increment for $V_{tw}$.

The front wave $fw$ goes backwards with speed $v_{wav} = \frac{v_l.\rho_d}{\rho_q - \rho_d}$ when the light is red. Since the traffic density in front of $fw$ is $\rho_q$, the volume in front of $fw$ increases with $v_{wav}.w_l.\rho_q$. This automatically leads to an increment of $v_{wav}.w_l.\rho_q$ for the value $V_{tw}$ of all traffic waves $tw$, since no changes in volume occur between the traffic waves.

Updating $V_{tw}$ when light is green

In this case, the volume between the front wave and the second front wave can also change, because of the difference in speed of these traffic waves. The volumes between the other waves stay equal because their backward speed is equal.
The front wave $fw$ moves backwards with speed $f_l v_{wav}$. Since the density of traffic in front of $fw$ is $\rho_d$, we see that $V_{fw}$ increases with $f_l v_{wav} \cdot \rho_d$.

For the second wave $sw$ we have that it moved backwards with speed $v_{wav}$. Since $fw$ moved backwards with $f_l v_{wav}$, the distance between these waves increased with $v_{wav} - f_l v_{wav} = (1 - f_l) v_{wav}$. The volume between $sw$ and $fw$ increases with $(1 - f_l) v_{wav} \cdot w_l \cdot \rho_q$, since the traffic has density $\rho_q$. The total increment of volume in front of $sw$ is the sum of the increment between $sw$ and $fw$ and the increment in front of $fw$, so $(1 - f_l) v_{wav} \cdot w_l \cdot \rho_q + f_l v_{wav} \cdot w_l \cdot \rho_d = v_{wav} \cdot w_l (\rho_d + (1 - f_l) (\rho_q - \rho_d))$.

For the waves behind $sw$, we have the same volume increment as for $sw$, since the volumes between the waves remain equal.

The algorithm that sets all traffic wave positions and volumes works as follows:

**Algorithm** Determine $x_{tw}$ and $V_{tw}$ for all traffic waves in $waveList_l$
1. $n \leftarrow$ the length of $waveList_l$
2. if $n > 0$
3.   if The traffic light is red
4.     for all $tw$ in $waveList_l$
5.         $x_{tw} \leftarrow x_{tw} + v_{wav}$
6.         $V_{tw} \leftarrow V_{tw} + v_{wav} \cdot w_l \cdot \rho_q$
7.     else
8.         $fw \leftarrow \{waveList_l\}_{n-1}$
9.         $x_{fw} \leftarrow x_{tw} + f_l v_{wav}$
10.        $V_{fw} \leftarrow V_{tw} + f_l v_{wav} \cdot w_l \cdot \rho_d$
11.       for all $tw$ in $waveList_l$ where $tw \neq fw$
12.          $x_{fw} \leftarrow x_{tw} + v_{wav}$
13.          $V_{fw} \leftarrow V_{tw} + v_{wav} \cdot w_l \cdot (\rho_d + (1 - f_l) (\rho_q - \rho_d))$

**Determining queue length $QL_l$**

Determining the queue length is done by using the following algorithm (with the queue volume $QV_l$ as a known value). This algorithm also removes traffic waves that have passed the back of the queue from $waveList_l$. The algorithm checks for every wave (starting at the backmost wave) if the volume in front of the wave ($V_{tw}$) is bigger than the queue volume $QV_l$. If this is the case, the wave is no longer part of the queue and should be removed.

The first wave $tw$ for which holds that $V_{tw} \leq QL_l$ is the backmost traffic wave of the queue. We use this wave to determine the position of the back of the queue. Because we know the position of $tw$, we only have to determine the distance between $tw$ and the back of the queue. This value depends on the traffic volume and the traffic density of the traffic in this part of the queue (from now on called ‘back of queue density’).

The volume between $tw$ and the back of the queue is $QV_l - V_{tw}$. The back of queue density is $\rho_q$ or $\rho_d$, depending on the amount of traffic waves left in $waveList_l$ and the current traffic light setting. We distinguish the following four cases:

- If the light is green and the amount of traffic waves in $waveList_l$ is even, the back of queue density is $\rho_d$
• If the light is green and the amount of traffic waves in \(\text{waveList}_l\) is odd, the back of queue density is \(\rho_q\)

• If the light is red and the amount of traffic waves in \(\text{waveList}_l\) is even, the back of queue density is \(\rho_q\)

• If the light is red and the amount of traffic waves in \(\text{waveList}_l\) is odd, the back of queue density is \(\rho_d\)

If we take \(\rho\) as the back of queue density, \(QL_l\) should be set to \(x_{tw} + (QV_l - V_{tw})/(w_l.\rho)\).

It can also occur that there is no traffic wave left in \(\text{waveList}_l\). In this case, the queue has exactly one density, \(\text{rho}_d\) if the light is green and \(\rho_q\) if the light is red. If \(\rho\) is the queue density, \(QL_l\) should be set to \(QV_l/(w_l.\rho)\).

**Algorithm**

**Determine** \(QL_l\) and erase traffic waves that passed \(QL_l\)

1. **while** \(\text{waveList}_l\) is not empty
2. \(tw \leftarrow \{\text{waveList}_l\}_0\)
3. **if** \(V_{tw} > QV_l\)
4. delete \(tw\) from \(\text{waveList}_l\)
5. **else**
6. **if** (Traffic light is green and \(n \mod 2 \equiv 1\) or (Traffic light is red and \(n \mod 2 \equiv 0\))
7. \(QL_l \leftarrow x_{tw} + (QV_l - V_{tw})/(w_l.\rho_q)\)
8. **else**
9. \(QL_l \leftarrow x_{tw} + (QV_l - V_{tw})/(w_l.\rho_d)\)
10. **stop**
11. **if** (Traffic light is green)
12. \(QL_l \leftarrow QV_l/(w_l.\rho_d)\)
13. **else**
14. \(QL_l \leftarrow QV_l/(w_l.\rho_q)\)

**Add traffic blocks to queue**

The last process that has to be explained is how traffic blocks that reach the queue are added to the queue. The addition of blocks to the queue is done after the algorithms mentioned before. We add a block \(tb\) to the queue if \(x_{tb} \leq QL_l\). If a block reaches the queue we can have two different cases:

• The back of the queue is driving

• The back of the queue stands still

If the back of the queue drives, we assume that \(tb\) does not have to stop. We just add the block to the queue with density \(\rho_d\), so we add \(V_{tb}\) to \(QV_l\) and we add \(V_{tb}/(w_l.\rho_d)\) to \(QL_l\).

If the back of the queue is standing still, we should have a more detailed look at what happens with the added traffic block. We know \(x_{tb} < QL_l\). The front of the block has joined the queue, so it is at position \(QL_l\). We can calculate where the back of \(tb\) should be if it is still driving. We call this value \(y_{tb}\), and it is equal to \(x_{tb} + V_{tb}/(w_l.\rho_d)\).

We can distinguish two cases:
• The back of the block still drives after it is added to the queue

• The back of the block is also standing still after being added to the queue

Assume that the complete block \( tb \) is added to the queue with queueing density \( \rho_q \). In this case, we add \( V_b \) to \( QV_l \) and \( V_{tb}/(w_l.\rho_q) \) to \( QL_l \). Now we have the following information about traffic at the back of \( tb \):

• If the back of \( tb \) still drives, it is at position \( y_{tb} \).

• If the back of \( tb \) is stopped, it is at position \( QL_l + V_{tb}/(w_l.\rho_q) \).

Because of this, we know that the back of the block is stopped when \( QL_l + V_{tb}/(w_l.\rho_q) \geq y_{tb} \).

If \( QL_l + V_{tb}/(w_l.\rho_q) < y_{tb} \), the back of the queue still drives. In this case, a new traffic wave \( tw \) will be generated, since the front of \( tb \) stands still and the back of \( tb \) drives. We visualized this in Figure 3.17. In Figure 3.17 we see that \( tb \) is divided in two parts:

\[
\begin{align*}
\text{Figure 3.17: Illustration of traffic block reaching queue} \\
\end{align*}
\]

• A part between \( QL_l \) and \( x_{tw} \) with density \( \rho_q \)

• A part between \( x_{tw} \) and \( y_{tb} \) with density \( \rho_d \)

Because the volume of \( tb \) is known as \( V_{tb} \), we have the following equation to determine the position of \( x_{tw} \):

\[
V_{tb} = w_l.((x_{tw} - QL_l).\rho_q + (y_{tb} - x_{tw}).\rho_d)
\]

which leads to

\[
x_{tw} = \frac{V_{tb} + QL_l.\rho_q - y_{tb}.\rho_d}{\rho_q - \rho_d}.
\]

Summarized, what we do in this case is:

• We add \( V_{tb} \) to \( QV_l \)

• We add a traffic wave \( tw \) to index 0 of \( \text{waveList}_l \), with \( x_{tw} \) as in equation 3.3 and \( V_{tw} \) as \( QV_l - (y_{tb} - x_{tw}).\rho_d. w_l \).
• We set $QL_l$ to $y_{tb}$.

The algorithm runs as follows

**Algorithm** Add traffic block $tb$ to queue

1. if the back of the queue drives
2. $QL_l \leftarrow QL_l + V_{tb}/(w_l, \rho_d)$
3. $QV_l \leftarrow QV_l + V_{tb}$
4. else
5. Set $y_{tb} \leftarrow x_{tb} + V_{tb}/(w_l, \rho_d)$
6. if $QL_l + V_{tb}/(w_l, \rho_q) \geq y_{tb}$
7. $QL_l \leftarrow QL_l + V_{tb}/(w_l, \rho_q)$
8. $QV_l \leftarrow QV_l + V_{tb}$
9. else
10. $QV_l \leftarrow QV_l + V_{tb}$
11. add new traffic wave $tw$ to $waveList_l$ at index 0
12. $x_{tw} \leftarrow (V_{tb}/w_l + QL_l\rho_q - y_{tb}\rho_d)/(\rho_q - \rho_d)$
13. $V_{tw} \leftarrow QV_l - (y_{tb} - x_{tw})\rho_d w_l$
14. $QL_l \leftarrow y_{tb}$

Model 3 is a model which is much closer to the realistic behaviour of queues than Model 1 and 2. It is, however, slower, since extra calculations have to be done to calculate the positions of waves. The difference in performance speed will be tested in Section 5.2.2. From now on, we will use model 3 as our queue model, but we will also define algorithms for Model 1 and 2.

### 3.6 Transferring blocks from approaches to direction lanes

The transfer of blocks from approaches to direction lanes has a lot of similarities with the earlier discussed transferring from direction lanes to approaches. The main differences are:

- It is not affected by a traffic light
- In many cases, traffic should be distributed over multiple direction lanes
- We consider that traffic on the approach can get blocked when one of the direction lanes is completely filled with traffic.

These differences make the transfer of blocks from approach to direction lane is slightly more complicated.

#### 3.6.1 Determining whether a direction lane is blocked

What we will do before transferring blocks to direction lanes is checking how much space is left for incoming traffic. Each direction lane $l$ has a maximum amount of traffic that fits inside of it. We call this the volume $V_l$ of $l$, and it is defined as the size of $l$ (length times width) multiplied with the queueing density:

$$V_l = l_l \cdot w_l \cdot \rho_q.$$
If no traffic is present in the direction lane, an amount of $V_l$ will fit inside the direction lane. However, this space will become lower when traffic is present. There are two kinds of traffic that can be inside the direction lane $l$:

- Traffic inside the queue
- Traffic in driving blocks from $\text{blockList}_l$

All traffic would be stopped if it reaches the back of the queue $QL_l$. This means only the space between the back of direction lane $l$ and $QL_l$ is available for driving traffic. This space is called the capacity of $l$, and has a length of $l_l - QL_l$, and a volume of $(l_l - QL_l).w_l.\rho_q$. For calculating the capacity, we need to take into account that there already may be traffic that drives in $l$. The available capacity is equal to:

$$ (l_l - QL_l).w_l.\rho_q - \sum_{tb \in \text{blockList}_l} V_{tb} $$

We visualized the capacity in Figure 3.18. In model 1 (Section 3.5.1) and model 2 (Section 3.5.2) we will use the capacity mentioned in 3.3. When using model 3 (Section 3.5.3), we can make a more detailed estimation of the capacity, since we know whether the back of the queue drives or not.

If the back of the queue is standing still, the calculation of the capacity is equal to the calculation for Model 1 and 2. In this case, traffic has to stop driving at $QL_l$.

If the back of the queue drives, traffic is not stopped at $QL_l$, since traffic at that position can still drive. Traffic will be stopped at the backmost traffic wave $bw$, which is the first position on the link where traffic is actually stopped. This is visualized in Figure 3.19. In this case, the capacity is equal to:

$$ (l_l - x_{bw}).w_l.\rho_q - \sum_{tb \in \text{blockList}_l} V_{tb} - (QL_l - x_{bw}).w_l.\rho_d $$

Figure 3.18: Illustration of the capacity in Model 1 and 2

Figure 3.19: Illustration of the capacity in Model 3 (when back of queue drives)
If \( \text{waveList}_l \) is empty, the complete queue drives and traffic can proceed towards the stop line. In this case, the capacity is equal to:

\[
l_l.w_l.r_q - \sum_{tb \in \text{blockList}_{\text{List}_l}} V_{tb} - QL_l.w_l.r_d.
\]

Now that we have calculated the capacity, we can check whether the traffic we want to put from the approach into the direction lane fits or not. Assume the direction lane has space \( S \) (in PCU’s) and we want to transfer an amount of \( V \) PCU’s to the direction lane. We can distinguish between the following situations:

- All traffic fits in the direction lane, so \( V < S \).
- Only a part of \( V \) fits in the direction lane \((0 < S < V)\)
- The direction lane is completely occupied \((S = 0)\)

We will determine the space \( S_i \) for each direction lane \( dl_i \) that is downstream of the approach. These values together determines whether a traffic block fits completely, partially or does not fit inside the direction lanes.

### 3.6.2 The splitting of an arriving traffic block

When a traffic block \( tb \) arrives at the end of approach \( l \) without delay, we try to distribute its traffic over the downstream direction lanes. Let \( dl \) be a list of direction lanes: \( dl = \{dl_1, dl_2, \ldots\} \). First, we have to determine how the volume \( V_{tb} \) distributes over the direction lanes, so we have to obtain a list \( p = \{p_1, p_2, \ldots\} \) where \( p_i \) represents the fraction of \( V_{tb} \) that will be transferred to \( dl_i \). We need to have that \( \sum p_i = 1 \). We will discuss how we determine \( p \) in Section 3.6.3.

The distribution of traffic is visualized in Figure 3.20. Note that the front position of the new blocks lies inside of their direction lanes, but the position of the back is not necessarily inside of it.

![Figure 3.20: Illustration of traffic block reaching queue](image)

Now we know, for each direction lane \( dl_i \), the volume we want to put inside of it \((p_i.V_{tb})\). Now, we need to check whether this volume actually fits inside of \( dl_i \). Define \( S_i \) as the space left in \( dl_i \) for traffic. If \( S_i > p_i.V_{tb} \), the complete volume fits inside the direction lane. If this is not the case, we have that only a fraction \( f_i \) of the volume fits inside \( dl_i \), where \( f_i = S_i/(p_i.V_{tb}) \). We will define \( f_i \) as follows:

\[
f_i = \min\{1, S_i/(p_i.V_{tb})\}
\]
Our model states that, if one of the direction lanes of $dl$ has no capacity left, all traffic from the approach that wants to enter one of the direction lanes will get blocked.

- If, for one of the direction lanes $dl_i$, we have that $f_i = 0$ (which means that $S_i = 0$, so the complete direction lane is filled), no traffic will be transferred at all. The complete volume $V_{tb}$ will be added to the queue in approach $l$.

- If, for all direction lanes $dl_i$, we have that $f_i = 1$, we transfer the complete volume $p_i.V_{tb}$ to direction lane $dl_i$.

In all other cases, there is at least one direction lane $dl_i$ for which $0 < f_i < 1$. In this case, one of the direction lanes gets completely filled while adding traffic block $tb$. In this case, we take $f$ as the minimum of all fractions $f_i$, so $f = \min f_i$, and add $f.p_i.V_{tb}$ to each direction lane $dl_i$. This is visualized in Figure 3.21. Note that the volume $V_{tb}$ of $tb$ is adjusted to $(1 - f).V_{tb}$.

![Figure 3.21: Illustration of traffic block reaching queue](image)

The front position $x_{tb}$ is adjusted to $x_{tb} + f.V_{tb}/(w_l.\rho_d)$. If we still have that $x_{tb} \leq 0$, we add traffic block $tb$ to the queue. The algorithm is as follows:

**Algorithm** Transfer traffic block $tb$ from approach $l$ to direction lanes $dl(p = \{p_1, p_2, \ldots\})$

1. for $dl_i$ in $dl$
2. Determine space $S_i$
3. $f_i \leftarrow \min\{1, S_i/(p_i.V_{tb})\}$
4. $f \leftarrow \min f_i$
5. if $f = 0$
6. Add $V_{tb}$ to queue of $l$
7. delete $tb$ from blockList
8. if $f = 1$
9. for $dl_i$ in $dl$
10. Generate new TrafficBlock $nb$
11. $x_{nb} \leftarrow x_{tb} + l_{dl_i}, V_{nb} \leftarrow p_i.V_{tb}$
12. add $nb$ to blockList

42
13. delete \(tb\) from \(blockList_l\)
14. if \(0 < f < 1\)
15. for \(dl_i\) in \(dl\)
16. Generate new TrafficBlock \(nb\)
17. \(x_{nb} \leftarrow x_{tb} + l_{dl_i}, V_{nb} \leftarrow f.p_i.V_{tb}\)
18. add \(nb\) to \(blockList_{dl_i}\)
19. \(x_{tb} \leftarrow x_{tb} + f.V_{tb}/(w_l.\rho_d), V_{tb} \leftarrow (1 - f).V_{tb}\)
20. if \(x_{tb} < 0\)
21. Add \(V_{tb}\) to queue of \(l\)
22. delete \(tb\) from \(blockList_l\)

3.6.3 Determining distribution fractions \(p_i\)

Before we can distribute traffic over the direction lanes, we have to determine the fractions \(p_i\). Assume we want to distribute a traffic volume \(V\) over the direction lanes. In approach \(l\), we defined \(q_i\) as the expected fraction of traffic going from \(l\) to \(dl_i\), where \(\sum q_i = 1\). If \(q_i = 0\) or \(q_i = 1\), we know \(dl_i\) receives respectively no traffic or all traffic. In all other cases, \(dl_i\) will receive a part of the traffic from approach \(l\). In this case, we have to determine the value \(p_i\).

The easiest approach we can use, is just setting \(p_i = q_i\) for all direction lanes \(dl_i\). This may, however, lead to unrealistic behaviour, since no stochasticity is included. We have created another approach, which leads to a more variable distribution of traffic over the direction lanes.

Assume every single vehicle has a probability \(q_i\) to choose for direction lane \(dl_i\). If \(n\) vehicles are distributed over \(k\) direction lanes, and the variables \(X_i (i \in \{0, ..., k\})\) represent the amount of vehicles that took direction lane \(dl_i\), the \(X_i's\) follow a Multinomial distribution with \(n\) trials and event probabilities \(\{q_1, ..., q_k\}\). For \(X_i\) we have:

\[
\begin{align*}
\mathbb{E}[X_i] &= n.q_i \\
Var[X_i] &= n.q_i.(1 - q_i)
\end{align*}
\]

The fraction of vehicles that entered direction lane \(i\) is \(p_i\). In this example, \(p_i\) is defined as \(X_i/n\). For \(p_i\), we have:

\[
\begin{align*}
\mathbb{E}[p_i] &= q_i \\
Var[p_i] &= q_i.(1 - q_i)/n
\end{align*}
\]

Since our traffic blocks \(tb\) do not have an integer size \(n\) but a non-integer size \(V_{tb}\), we want a continuous distribution for \(p_i\) with the same expected value and variance. A useful distribution is the gamma distribution. For stochast \(X \sim \Gamma(\alpha, \beta)\) we have:

\[
\begin{align*}
\mathbb{E}[X] &= \alpha/\beta \\
Var[X_i] &= \alpha/\beta^2.
\end{align*}
\]

If we have two stochastic variables \(X_1\) and \(X_2\), where \(X_1 \sim \text{Gamma}(\alpha_1, \beta)\) and \(X_2 \sim \text{Gamma}(\alpha_2, \beta)\), it is generally known that \(X_1 + X_2 \sim \text{Gamma}(\alpha_1 + \alpha_2, \beta)\). We can extend
this to a more general case: if we have \( m \) variables \( \{X_1, \ldots, X_m\} \) with \( X_i \sim \text{Gamma}(\alpha_i, \beta) \), we know that \( \sum_{i=1}^{m} X_i \sim \text{Gamma}(\sum_{i=1}^{m} \alpha_i, \beta) \).

Another generally known property is that the variable \( X_1/(X_1 + X_2) \) is Beta distributed with parameters \((\alpha_1, \alpha_2)\), so \( X_1/(X_1 + X_2) \sim \text{Beta}(\alpha_1, \alpha_2) \). For a stochastic \( Y \sim \text{Beta}(\alpha, \beta) \) we have

\[
\mathbb{E}[Y] = \frac{\alpha}{\alpha + \beta} \quad \text{and} \quad \text{Var}[Y] = \frac{\alpha \beta}{(\alpha + \beta)^2 (\alpha + \beta + 1)}.
\]

We have \( k \) direction lanes \( \{dl_1, \ldots, dl_k\} \), where we want to transfer a traffic volume \( V \) towards. Direction lane \( dl_i \) has an expected fraction \( q_i \) of traffic from \( l \) entering \( dl_i \). We will determine the fractions \( p_i \) as follows:

- We determine stochastic variables \( p'_i \sim \text{Gamma}(q_i \cdot V, V) \)
- We set \( p_i = \frac{p'_i}{\sum_{j=1}^{k} p'_j} \)

Now, we will calculate the distribution of \( p_i \). We have that

\[
p_i = \frac{p'_i}{\sum_{j=1}^{k} p'_j} = \frac{p'_i}{\frac{1}{\sum_{j\neq i} p'_j}}.
\]

For all elements in this expression, we know the distribution:

\[
p'_i \sim \text{Gamma}(q_i \cdot V, V)
\]

and

\[
\sum_{j=1, j\neq i}^{k} p'_j \sim \text{Gamma}(\sum_{j=1, j\neq i}^{k} q_j \cdot V, V) = \text{Gamma}((1-q_i) \cdot V, V).
\]

As a result, we know that

\[
p_i = \frac{p'_i}{\frac{1}{\sum_{j\neq i} p'_j}} \sim \text{Beta}(q_i \cdot V, (1-q_i) \cdot V).
\]

The first two moments of \( p_i \) are:

\[
\mathbb{E}[p_i] = \frac{q_i \cdot V}{q_i \cdot V + (1-q_i) \cdot V} = q_i
\]

(3.6)

\[
\text{Var}[p_i] = \frac{q_i \cdot V \cdot (1-q_i) \cdot V}{(q_i \cdot V + (1-q_i) \cdot V)^2 (q_i \cdot V + (1-q_i) \cdot V + 1)} = \frac{q_i (1-q_i) \cdot V^2}{V^2 (V+1)} = \frac{q_i (1-q_i)}{V+1}.
\]

(3.7)

We see that the mean of \( p_i \) (equation 3.6) is equal to its mean in case of a multinomial distribution (equation 3.3). If we assume that the parameter \( n \) in the multinomial distribution can be replaced by the block size \( V \), we see that the variance of \( p_i \) (equation 3.7) converges to the variance in the multinomial case (equation 3.5).

The algorithm which determines the fractions \( p_i \) for a volume \( V \) that should be transferred from approach \( l \) to direction lanes \( dl_1, \ldots, dl_k \) works as follows
Algorithm Determine fractions $p_i$ for block $tb$

1. for $i \in \{1, \ldots, k\}$
2. if $0 < q_i < 1$
3. $\alpha \leftarrow q_i.V_{tb}$
4. $\beta \leftarrow V_{tb}$
5. $p'_i \leftarrow \text{sample of } \Gamma(\alpha, \beta)$
6. else
7. $p'_i \leftarrow q_i$
8. $P \leftarrow \sum p'_i$
9. for $i \in \{1, \ldots, k\}$
10. $p_i \leftarrow p'_i / P$

3.7 Modelling the queue in approach $l$

For queues in approaches, we use the same model as for the direction lanes. The main difference is that traffic is not stopped by a red traffic light, but by a completely filled downstream direction lane. When a queue is present, we assume we can transfer an amount of $v.w_l.\rho_d$ traffic to the direction lanes per time step. We define $\mu_l = v.w_l.\rho_d$. If $QV_l \geq \mu_l$, we try to transfer a block with size $\mu_l$ to the direction lanes, otherwise we try to transfer a block with size $QV_l$. The transferring of blocks to the direction lanes is discussed in Section 3.6.

Adding blocks to the queue is done in the same way as in the direction lanes. This is explained in Section 3.5.3.

The behaviour inside the queue slightly differs from the queue behaviour inside the direction lanes. If we use model 1 (Section 3.5.1) or model 2 (Section 3.5.1), we distinguish two cases:

- Some traffic departed from queue
- No traffic departed from the queue (one of the direction lanes is blocked)

Let $d$ be the amount of traffic that departed from the queue:

Algorithm Determine new queue length and volume when $d$ PCU’s departed from the queue

1. if $d = 0$
2. $QL_l \leftarrow \max\{QV_l/(w_l.\rho_q, QL_l - v)\}$
3. else
4. $QV_l \leftarrow QV_l - d$
5. $QL_l \leftarrow \min\{QV_l/(w_l.\rho_d, QL_l)\}$

If we use model 3 (Section 3.5.3), the traffic waves in the queue move backwards with speed $v_{tw} = v_l.\rho_d/(\rho_q - \rho_d) = v_{wav}$. We have three different kinds of waves, visualized in Figure 3.13.

- Traffic waves where traffic stops driving ($tw_2$)
- The front traffic wave when the traffic light is green ($tw_3$)
- Other traffic waves where traffic starts driving ($tw_1$)

45
The calculations for the backward speed of waves in Section 3.5.3 (which was for the queue in the direction lane) also hold for the queue in the approach. For $tw_3$, we calculated that $v_{tw} = f_l v_{wav}$, but since traffic always departs with maximum speed, we have that $f_l = 1$, so all waves travel backwards with $v_{tw} = v_{wav}$.

We also need to determine when traffic waves should be added to this queue. We have two kinds of traffic waves:

- Waves where traffic starts driving
- Waves where traffic stops driving

When traffic starts driving at the stop line, we send a traffic wave backwards, and when traffic gets stopped we do the same. We have to determine when these events exactly happen. We only send waves backwards if there is still a queue left, so all cases in which the complete queue departs are left out. We can distinguish three cases:

- Case 1: no traffic departs.
- Case 2: The maximum amount of traffic departs ($v_l w_l \rho_d$).
- Case 3: An amount of $f v_l w_l \rho_d$ departs, where $0 < f < 1$.

To determine whether a wave departs, we need to know if the front of the queue was driving at the end of the previous time step. This can be done by introducing a boolean $driving$. If $driving = True$, traffic was driving at the end of the previous time step.

In case 1, traffic is stopped during the complete time step. If $driving = True$, traffic was driving, so we send a traffic wave backwards, starting at time $t$.

In case 2, traffic drives during the complete time step. If $driving = False$, traffic was standing still, so we send a traffic wave backwards, starting at time $t$.

In case 3, we assume that traffic drives from the start of the time step ($t$) until the moment the traffic gets blocked. This means that traffic stops driving at time $t + f$, which is visualized in Figure 3.22. We send a traffic wave backwards, starting at time $t + f$. If $driving = False$, we also create a new traffic wave starting at time $t$.

The complete algorithm for model 3 takes the following steps:

- Let traffic depart from the queue.
- Add new traffic waves at the front of the queue.
- Move all traffic waves backwards.
- Calculate the new queue length.
- Let new blocks join the queue.

Since this algorithm is very similar to the algorithm for queue behaviour in the direction lanes, we do not describe the complete algorithm here.
3.8 Performance measures

Before this simulation model can be useful, we have to develop useful performance measures. Since we do not represent vehicles as individual entities, we cannot see how long their trip through the system takes. The information we do have is:

- The queue lengths for each time step
- The queue volumes for each time step
- How many time steps traffic from approaches to direction lanes was blocked
- How many time steps the queue in the approach was longer than the length of the approach. In this case, the intersection should be blocked (which we neglected in our model).

In chapter 5, we will do multiple tests to show which insights our model can give.
Chapter 4

Implemented traffic light control algorithms

As mentioned in Section 3.3, we are able to implement multiple traffic light control algorithms. We will briefly explain the algorithms we implemented, after (re)introducing the parameters that are essential in the various algorithms we implemented. This is done in Table 4.1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DL$</td>
<td>Collection of all incoming direction lanes</td>
</tr>
<tr>
<td>$DL_{ij}$</td>
<td>Direction Lane with origin $i$ and destination $j$</td>
</tr>
<tr>
<td>$\Sigma = {\sigma_0, \sigma_1, \ldots}$</td>
<td>Sequence of signal phases</td>
</tr>
<tr>
<td>$\sigma_k$</td>
<td>The $k$'th signal phase</td>
</tr>
<tr>
<td>$S_k$</td>
<td>Collection of direction lanes with green light in phase $\sigma_k$</td>
</tr>
<tr>
<td>$A_k$</td>
<td>Collection of relevant direction lanes of $\sigma_k$</td>
</tr>
<tr>
<td>$l_{\sigma_k}^{\text{max}}$</td>
<td>Maximum length of phase $\sigma_k$</td>
</tr>
<tr>
<td>$l_{\sigma_k}^{\text{min}}$</td>
<td>Minimum length of phase $\sigma_k$</td>
</tr>
<tr>
<td>$l_y$</td>
<td>Length of yellow period between two phases (included in phase lengths)</td>
</tr>
<tr>
<td>$o$</td>
<td>Offset, or time since start of cycle at $t = 0$</td>
</tr>
</tbody>
</table>

Table 4.1: List of used parameters in the signal settings

As mentioned in Table 4.1, we implemented the yellow period between two phases $\sigma_k$ and $\sigma_{k+1}$ as a part of phase $\sigma_k$. This means that each phase is finished with $l_y$ time steps of yellow lights. Because of this, the maximum green period of $\sigma_k$ is not equal to the phase length $l_{\sigma_k}^{\text{max}}$, but to the remaining phase length $l_{\sigma_k}^{\text{max}} - l_y$. The minimum green length is equal to $l_{\sigma_k}^{\text{min}} - l_y$. The reason for this implementing choice is that, in this case, the cycle time is simply the sum of the phase lengths, which we see as the most intuitive option.

We will explain the following algorithms:

- Fixed traffic light algorithm
- Isolated vehicle actuated (VA) traffic light algorithm
- Synchronized VA traffic light algorithm
- Synchronized VA traffic light settings with algorithm
- Back pressure algorithm
4.1 Fixed traffic light algorithm
This algorithm is explained in Section 3.3.

4.2 Isolated VA traffic light algorithm
The previous algorithm is the easiest possible algorithm: we just define phase lengths, and when the phase $\sigma_k$ lasted its phase length $l_{\sigma_k}^{\max} - l_y$, the signals turn yellow and move on to phase $\sigma_{k+1}$. When queues are short, it will occur that all available traffic on the direction lanes in $S_k$ is served before the end of the green period. In this case, a part of phase $\sigma_k$ is not used by traffic from direction lanes in $S_k$, while traffic from other directions is unnecessarily waiting for their turn. In this case, the throughput of the intersection could be improved by finishing $\sigma_k$ before its maximal green time.

This strategy takes into account whether traffic is still present at the direction lanes that receive green light during the current phase $\sigma_k$. In real life, this information is gained by using detectors at the stop line. These (induction loop) detectors give signals to the traffic light controller when a vehicle crosses the stop line, or when it is waiting at the stop line. In our simulation, we assume that each direction lane has a detector.

We define a set of relevant direction lanes $A_k$, which is a subset of $S_k$. The phase $\sigma_k$ will be ended when the queue of all direction lanes of $A_k$ has been empty during $\sigma_k$, or when the phase has lasted for $l_{\sigma_k}^{\max} - l_y$ time steps. After this, the yellow period will be started. In many cases, we set $A_k = S_k$, but it may be the case that we do not want to take a direction lane $DL_{ij} \in S_k$ into account. For example, this can be the case when $DL_{ij}$ is also contained in $S_{k+1}$. In this case, it is not a problem when there is still a queue at $DL_{ij}$ when moving on to the next phase.

The phase $\sigma_k$ also has a minimum length $l_{\sigma_k}^{\min}$. For safety reasons, the yellow period of $\sigma_k$ will not be started before the phase lasted for at least $l_{\sigma_k}^{\min} - l_y$ time steps.

We need to define how we measure availability of traffic in a direction lane $DL_{ij}$. After each time step $t$, we check whether traffic was present at the stop line during the time step. This is the case if:

- A queue is present in $DL_{ij}$ at the end of time step $t$
- If no queue is present, but traffic crossed the stop line of $DL_{ij}$ during $t$.

The algorithm works as follows:

- At the start of phase $\sigma_k$, set $A \leftarrow A_k$.
- At the end of each time step $t$, check for all direction lanes $DL_{ij} \in A$ if traffic was present.
- If no traffic was present during $t$, delete $DL_{ij}$ from $A$.
- If $A = \emptyset$ and the phase lasted for at least $l_{\sigma_k}^{\min} - l_y$ time steps, move on to the yellow phase.
- If the phase lasted for $l_{\sigma_k}^{\max} - l_y$ time steps, move on to the yellow phase.
- At the end of the yellow phase, move on to the next phase.
We can make the algorithm even more efficient by skipping phases when queues are not present. When a phase $\sigma_k$ is ended, we normally move on to phase $\sigma_{k+1}$, but now we first check for all $DL_{ij} \in A_{k+1}$ whether traffic is present. If there is no traffic available, we move on to phase $\sigma_{k+2}$, and check for available traffic. As long as no traffic is available, we move on to the next phase. If we returned to phase $\sigma_k$ and we still have not found any traffic, we set all traffic lights to red.

When all traffic lights are red, we will check after each time step $t$ if traffic has arrived at any of the direction lanes. The first phase $\sigma_k$ for which one of the direction lanes in $A_k$ contains a queue will be the next green phase.

One of the features of this algorithm is that the cycle time is no longer constant. When using fixed settings, the cycle time $c$ was the sum of the (maximum) cycle lengths ($\sum_l l^\text{max}_{\sigma_k}$). Since the length of the phases can vary now (and can even be zero), there is no longer a fixed cycle length. This also means that it is no longer possible to synchronize the signal settings of neighboring intersections by choosing the best possible offset, since offsets makes no sense when intersections run on different cycle lengths. This problem will be fixed in the next traffic light algorithm.

### 4.3 Synchronized VA traffic light algorithm

In many networks, it is needed to maintain a fixed cycle length. If we preempt phases before their maximum length is reached, the length of the cycle is variable. We can fix this by defining main phases. If a phase is preempted, we no longer use the saved time to get a shorter cycle time, but we provide the saved time to the next main phase. We describe the main phases as phases that are extended if needed to maintain a fixed cycle length.

Let $M_S \subset \Sigma$ be the set of main phases in $\Sigma$. For each phase which is not a main phase, we do the same as in the Isolated VA algorithm: we end them when all queues of the direction lanes in $A_k$ are completely served. If we start the yellow phase of $\sigma_k$ after $l^\text{max}_{\sigma_k} - l_y$ time steps, we do not have to edit anything to keep the cycle length constant. If we stop the phase earlier, say after $t$ time steps, the cycle time is reduced by $l^\text{max}_{\sigma_k} - l_y - t$ time steps. Since we do not want this, we keep this time in a variable $t_s$, which we call the saved time. We will use this saved time to extend main phases. After each non-main phase, we add the saved time to $t_s$.

When the current phase $\sigma_k$ is a main phase, so $\sigma_k \in M_S$, we do not preempt the phase anymore. Instead of that, we end the green period after $l^\text{max}_{\sigma_k} - l_y + t_s$ time steps. After we finish this phase, we set the saved time $t_s$ to zero. The result is that the cycle length is maintained, and we are able to synchronize neighboring intersections with each other. The main phase is often chosen to be the phase which handles most traffic, or the phase which is part of a green wave through multiple intersections.

When a phase is skipped, its complete (maximal) phase length will be added to the lost time. The main phase will never be skipped: if no traffic is available at the intersection when a phase is ended, the algorithm will skip phases until the main phase is reached.

This algorithm is very suitable for regulating a green wave on an arterial, since it skips unused time in non-main phases and adds it to the main phases, providing extra green time to a busy stream. A drawback of this system is that, especially for light traffic situations, too
much time may be added to the main phase. For example, it can occur that when the cycle length is 90 seconds, 70 seconds are provided to the main phase. Since it never preempts this phase, traffic arriving from other directions may have to wait for an unnecessary long period.

### 4.4 Synchronized VA traffic light algorithm with reduction

Assume we have a group of \( n \) intersections \( I_1, I_2, \ldots, I_n \), running on the synchronized VA algorithm with the same cycle length \( c \). We define the saved time of intersection \( I_i \) as \( t_{s,i} \). If all intersections have skipped some non-main-phase periods since the end of their last main phase, we have that \( t_{s,i} > 0 \) for \( i = 1, 2, \ldots, n \).

If we reduce all saved times \( t_{s,i} \) with one, we reduce all cycle times with one. This results in a cycle length of \( c - 1 \) for all intersections, which means the intersections still run on the same cycle lengths, so they are still synchronized.

If we apply reduction on the group of intersections \( I_1, I_2, \ldots, I_n \), we will check the size of their saved times every time step. If we have that \( t_{s,i} > 0 \) for all intersections, we reduce all lost times with \( \min \{ t_{s,i} \} \). The result is that the group of intersections is still synchronized, but we have reduced the amount of extra green time for the main phases. This leads to possible waiting time reductions for direction lanes receiving green light in non-main phases.

In our simulation model, we are able to define multiple groups of intersections on which we apply Synchronized VA with Reduction.

Research on the performance of the fixed algorithm, the Isolated VA algorithm, the Synchronized VA algorithm and the Synchronized VA algorithm with reduction is done earlier by Van Endhoven [4]. In this report, performance study is done on small networks in SUMO, which generally shows that:

- Isolated VA only works well in networks with very light traffic, where less than 20% of the capacity is used. It performs very bad in heavy traffic situations.

- Busy arterials profit heavily on well synchronized signal settings, better known as green waves. The Synchronized VA settings without reduction are the best option for these arterial, since they will receive the most green time in that case.

- Side roads of this arterial are not making profit of Synchronized VA without reduction, since it may lead to excessive long green periods for traffic on the arterial. The reduction reduces their waiting times significantly.

- Fixed times can always be programmed in a way that they are not performing bad, but they will never be the best option.

### 4.5 Back pressure

The back pressure model is slightly different from the earlier described models. The model uses estimates of queue lengths during a cycle to determine the lengths of the phases of the next cycle. We will briefly explain the model in this section, the model is described extensively in [5].
Queue length measurements

Although our simulation model has the ability to measure queue lengths and volumes, this is very difficult to do in real life. Because of this, we deliberately do not use this information.

We will estimate the queue length of a direction lane \( l \) at the start of a green period by using historical information. At each green period, we measure how long it takes until the queue is served completely. This time \( \tilde{T} \) is then multiplied by \( \mu_l \), the saturation flow of the direction lane \( l \), to obtain the estimated queue length \( \hat{Q}_l \).

Determining phase weights

Assume we have \( n \) phases \( \sigma_1, \sigma_2, \ldots, \sigma_n \). Each phase \( \sigma_k \) has a set of \( n_k \) direction lanes \( l_{k1}^k, \ldots, l_{nk}^k \) which get green light during phase \( \sigma_k \). Each of these direction lanes \( l_i^k \), on their turn, have a set of \( n_{ki} \) downstream direction lanes \( l_{ki1}^{ki}, \ldots, l_{kini}^{ki} \) to which traffic in \( l_i^k \) will be transferred.

Let \( p_{ji}^{ki} \) be the probability that a vehicle in \( l_i^k \) will be transferred to \( l_{ji}^{ki} \).

Now, we can define the phase weight \( w_{\sigma_k} \) of phase \( \sigma_k \):

\[
 w_{\sigma_k} = \sum_{i=1}^{n_k} \mu_{l_i^k} (\hat{Q}_{l_i^k} - \sum_{j=1}^{n_{ki}} p_{ji}^{ki} \hat{Q}_{l_{ji}^{ki}}).
\]

As we can see, the weights of the phases not only depend on the queue lengths at the intersection itself, but also on the queue lengths at the downstream intersections. This is done because when a downstream queue is very long, you might want to reduce the amount of traffic entering this queue.

Determining phase lengths

Out of the phase weights, we can determine the actual phase lengths. The back pressure algorithm uses a fixed cycle length \( C \). We also have a part of the cycle which is classified as lost time \( L \), which is composed of the yellow and red periods between the phases. The remaining cycle time \( C - L \) can be distributed over the phases.

First, we determine the proportion \( P_{\sigma_k} \) of the cycle time that should be provided to phase \( \sigma_k \):

\[
 P_{\sigma_k} = \frac{e^{\eta w_{\sigma_k}}}{\sum_{m=1}^{n} e^{\eta w_{\sigma_m}}},
\]

where \( \eta \) is a parameter which determines how heavily the system prefers to provide much green time to the phases \( \sigma_k \) with high weight \( w_{\sigma_k} \). The provided time to \( \sigma_k \) is then equal to \( P_{\sigma_k} \cdot (C - L) \).

If one of the phases is shorter than a specified minimum phase length, we have to edit the phase lengths slightly to fix this. We also have to round the phase lengths to integers, since our simulation model works with time steps of one second.

Smoothing of queue length estimates

The system tends to be very unstable when the queue length estimation \( \hat{Q}_l \) is only based on the previous green phase. To make it less sensible for fluctuations, we have to use smoothing techniques. Two possible techniques are:
• Moving average: the estimated queue length is the average of the last \( n \) queue length measurements.

• Exponential smoothing: the estimated queue length is defined as the sum of \( \alpha \) times the last queue length measurement, and \( (1 - \alpha) \) times the queue length estimation of the previous cycle.

The performance of the algorithm depends heavily on the choice of \( \eta \), and on the choice of the smoothing technique (and its parameter). Research on this topic has been done by Tommi Sisso [5].
Chapter 5

Testing our simulation model

In this chapter, we will do several simulation runs to check the performance of our simulation model. This performance depends on several criteria, and we will try to answer the following questions:

- Does it model city traffic in a realistic way?
- Is the model fast enough to simulate big networks?
- Is the system stable?
- Does it provide useful information about the signal settings?

5.1 Comparing with SUMO

Since we have a lack of real life traffic data, we cannot compare our simulation results directly with real life data. Since it is believed that microscopic simulation models can represent traffic in a relatively realistic way, we will use the microscopic simulation model SUMO to test whether our results are correct.

In the following simulation runs, SUMO is always used with its standard settings, unless stated otherwise.

Before we start comparing, we need to state that SUMO simulates lane-based traffic. The width of a road is specified by the amount of lanes the road contains. When we simulate a road with \( n \) lanes in SUMO, its equivalent in our simulation model is a road \( l \) where \( n \) cars would fit next to each other. Since we know the width of a car (the parameter \( csw \)), we set the width of the road to \( n \cdot csw \).

5.1.1 Average velocity

In our simulation model, we define for each link \( l \) a driving speed \( v \), and assume that all traffic blocks move forward with speed \( v \). In SUMO, we also define a maximum speed \( v \) on a link, but the driving speed of individual cars can vary heavily. SUMO simulation runs showed us that the actual average driving speed of cars turn out to be significantly lower than \( v \). If we set the driving speed to the same value (\( v \)) in both systems, the results turn out to be uncomparable because of the lower average driving speed in SUMO.
What we will do for every simulation run where we compare our simulation model to SUMO, is that we set the maximum speed in SUMO to a chosen value $v'$, and run the simulation in SUMO. We will calculate the average driving speed $v$ of this simulation, and use $v$ as the link speed in our simulation model.

To show the difference between the maximum speed $v$ and the average driving speed $v'$ in SUMO, we will do a couple of simulation runs. We will use the network shown in Figure 5.1, a simple three-lane road with a fixed cycle traffic light on it. The cycle length is set to 60 seconds: the cycle starts with 37 seconds of green light, followed by three seconds of yellow light and 20 seconds of red light. We will measure the average speed of vehicles on a point 1900 meters after this traffic light, such that traffic has enough time to accelerate to their maximum speed.

Since we are not able to simulate heterogeneous traffic in SUMO, we will simulate a network with only cars. Since we only simulate at relatively low speeds $v$, we assume all cars want to drive at maximum speed $v$. We will set their maximum acceleration $a$ to 4 m/s$^2$ and their maximum deceleration $d$ to 6 m/s$^2$. The cars have a length of 4 meter. When queueing, the cars will keep a minimum gap of 2.5 meter to each other, and when driving, their minimum distance is one second (which clearly depends on the driving speed of the cars) plus the minimum gap. The last two settings are standard settings of SUMO.

We choose the arrival intensity such that the system is oversaturated, and the complete green phase is used. This does not affect our speed measure. The results are presented in Figure 5.2 and Table 5.1.

<table>
<thead>
<tr>
<th>$v$</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v'$</td>
<td>4.01</td>
<td>4.99</td>
<td>5.97</td>
<td>6.96</td>
<td>7.95</td>
<td>8.94</td>
<td>9.94</td>
<td>10.93</td>
<td>11.92</td>
<td>12.91</td>
<td>13.91</td>
</tr>
</tbody>
</table>

Table 5.1: Relation between maximum and average speed in SUMO

Simulations with a similar network with only 2 lanes or 1 lane show similar results, which show that the relation between $v$ and $v'$ can be assumed independent of the width of the road.

If we choose to vary the preferred velocity per vehicle, we may get slightly different results. Instead of giving each vehicle a fixed preferred velocity $v$, we give them a preferred velocity $Xv$, where $X$ is normally distributed with mean 1 and standard deviation 0.1 ($X \sim \mathcal{N}(1, 0.1)$). In this situation, around 95% of the vehicles has a preferred speed between $0.8 \cdot v$ and $1.2 \cdot v$.

We will simulate for the same values $v$ as in the previous simulation, but since more vehicles will tend to overtake each other, we expect that the amount of lanes also has its influence on the average speed. For this reason, we will run this simulation for three lanes, two lanes and one lane (where overtaking is impossible). The results are presented in Table 55.
The average speed $v'$ is heavily influenced by the difference in preferred speed of the
5.1.2 Capacity of system

One of the most important features of a traffic network is its capacity. Since the capacity of the network highly depends on the traffic light settings, we need to make sure that the process of traffic crossing the stop is modelled correctly.

We will compare this process by reporting the amount of traffic that is able to cross the stop line during a green period. We will use the same simulation settings as in the previous section, where we compared the maximum and average speed on a three lane road with no speed differences between vehicles (see Figure 5.2 and Table 5.1).

In SUMO, we measured how many vehicles cross the stop line on average during a green period. In our model, this value is deterministic, which means we can calculate it without running the simulation. Our simulation settings are the following:

- Car space width: $csw = 2.2$ m
- Car space length: $csl = 6.5$ m
- Driving distance: $d_d = 1$ s
- Road width: $w_l = 6.6$ m
- Acceleration: $acc = 4$ m/s$^2$
- Deceleration: $dec = 6$ m/s$^2$
- Velocity: $v_l = v'$ m/s
- Green time: $g = 37$ s
• Yellow time: $y = 3$ s

We determine the driving density $\rho_d$ and the amount of traffic $\mu_l$ that can cross a green traffic light per time step ($\mu_l = v_l \cdot \rho_d \cdot w_l$).

We assume that traffic starts to accelerate immediately when the light turns green. It takes traffic $t_{acc} = v_l/a$ seconds to gain full speed. From time $t_a$ to time $g$, traffic can cross the stop line at full speed. When light turns yellow, traffic needs $t_{dec} = v_l/d$ seconds to slow down.

During the acceleration and deceleration period, traffic has an average speed of $0.5v$, so the average departure intensity is $0.5\mu_l$. During the rest of the green phase, the average departure intensity is $\mu_l$. This leads to the following amount of traffic $\mu_c$ crossing the stop line per cycle:

$$\mu_c = 0.5\mu_l(t_{acc} + t_{dec}) + \mu_l(g - t_{acc}).$$

Now we can compare the results obtained from simulations in SUMO with the deterministic value $\mu_c$ of our model. For the network where all vehicles want to maintain the same speed, we obtained the following result shown in Figure 5.7.

![Figure 5.6: Relation between maximum and average speed in SUMO](image)

We see that our model highly overestimates the cycle capacity, compared to SUMO. It is very likely that we made some different assumptions when adapting the settings of our model to the settings in SUMO.

One of the settings that may be unrealistic is the driving distance $d_d$. In SUMO, this distance is set to one second, which we also did in our model, but the actual distances between vehicles in SUMO seem to be bigger. The setting of the driving distance in SUMO should be seen as a lower bound instead of a fixed value, which means the average driving distance is a bigger value.

We want to set the driving distance $d_d$ in our model equal to the average driving distance in SUMO. Measuring this average driving distance in SUMO is almost impossible, but we can easily calculate the cycle capacity $\mu_c$ of our model for any possible value of $d_d$. 

58
Let \( \mu(v, d_d) \) be the cycle capacity of our model with link speed \( v \) and driving distance \( d_d \). Let \( \mu'(v) \) be the (simulated) cycle capacity of SUMO at speed \( v \). Let \( V' \) be the set of average speeds \( v' \) in Table 5.1. We want to find the driving distance \( d'_d \) for which the difference between \( \mu(v, d'_d) \) and \( d_d \) is minimized for all realistic values of \( v \). We define \( d'_d \) as follows:

\[
d'_d = \min_{d_d} \sum_{v' \in V'} |\mu'(v') - \mu(v', d_d)|
\]

We increased \( d_d \) with 0.01 until we reached the minimum value of equation 5.1. This result is that \( d'_d = 1.28 \). We showed the new cycle capacities for \( d_d = d'_d \) in Figure 5.7.

![Figure 5.7: Relation between maximum and average speed in SUMO](image)

Although there is still a difference between the values of SUMO and our model, we see that the values for \( \mu_c \) follow roughly the same pattern.

We will do the same for the networks in Section 5.1.1 where vehicles have different speeds. For the case of one lane, we found that \( d'_d = 1.26 \). The results of SUMO and our model (Figure 5.8) seem to match almost perfectly.

![Figure 5.8: Relation between maximum and average speed in SUMO](image)

The results for two lanes (\( d'_d = 1.33 \), Figure 5.9) and three lanes (\( d'_d = 1.35 \), Figure 5.10) are less consistent. In these cases, other factors seem to play a role. It seems likely that the real average driving distances are lower than the calculated values of \( d'_d \) (which should lead to a higher capacity \( \mu_c \)), but traffic is slowed down further by the fact that traffic is switching lanes, causing a lower capacity.

### 5.2 Runtime performance

One of the main reasons for our representation of traffic as blocks was to heavily increase our simulation speed, so that we are able to simulate big city networks. In this section, we will
test this performance extensively. We will do this on a network of 80 intersections, which we will introduce now.

All simulations runs took place on the same system: a HP ZBook 15, with an Intel Core i7-4700MQ processor (2.40 GHz).

5.2.1 Network

The network we will use is shown in Figure 5.11. The network consists of 80 intersections, positioned in a 9 by 9 grid. On the middle position of the grid, there is no intersection but a ‘sink’ which represents a destination (or origin) of traffic in the system. We will call this the city center. Each intersection has four entries and exits (north, west, south and east), traffic can enter or leave the system on the 36 entries/exits on the edge of the network and the four entries/exits on the sink.

We divided the intersections in 8 groups, which can be seen in Figure 5.11. The groups N,W,S and E represent intersections which are part of arterials, wide roads running from the edge of the network to the city center. The roads between these intersections have the following features:

- Length of approach: 300 meters
- Width of approach: 7.5 meters
- Amount of direction lanes: 3 (left, straight ahead, right)
- Length of direction lanes: 200 meters
- Width of left direction lane: 2.5 meters
- Width of right direction lane: 2.5 meters
- Width of straight ahead direction lane: 7.5 meters.

Figure 5.9: Relation between maximum and average speed in SUMO

Figure 5.10: Relation between maximum and average speed in SUMO
• Driving speed: 12 m/s

The incoming and outgoing roads of the arterials at the edge of the network and the city center have the same features. A visualisation of these roads is given in Figure 5.12.

All other roads have the following features:

• Length of approach: 300 meters
• Width of approach: 7.5 meters
• Amount of direction lanes: 3 (left, straight ahead, right)
• Length of direction lanes: 200 meters
• Width of all direction lanes: 2.5 meters
• Driving speed: 12 m/s

A visualisation is given in Figure 5.13. Other parameters of the network are:

![Figure 5.13: Non-arterial road in the network](image)

• Car space length: 6.5 meters
• Car space width: 2.5 meters
• Acceleration: 4 m/s²
• Deceleration: 6 m/s²
• Driving distance: 1.25 seconds

The chosen driving distance of 1.25 seconds is chosen to match the results in figure 5.8, since most links in the system will be one-lane. The choice for 1.25 seconds instead of 1.26 seconds is made because 1.25 is a more intuitive value.

The traffic consists of vehicle coming from three vehicle classes: motorcycles, trucks and cars. For each class $v_c$, we define the probability $p_{v_c}$ that a vehicle is of class $v_c$ and the uniform distribution boundaries of their size, $s_{v_c}^{\text{min}}$ and $s_{v_c}^{\text{max}}$:

- $p_{\text{motorcycle}} = 0.58$, $s_{\text{motorcycle}}^{\text{min}} = 0.45$, $s_{\text{motorcycle}}^{\text{max}} = 0.55$
- $p_{\text{car}} = 0.4$, $s_{\text{car}}^{\text{min}} = 0.9$, $s_{\text{car}}^{\text{max}} = 1.1$
- $p_{\text{truck}} = 0.02$, $s_{\text{truck}}^{\text{min}} = 2.5$, $s_{\text{truck}}^{\text{max}} = 3.5$

We also define turning fractions for all approaches in the network. For each approach $l$, we have the following turning fraction parameters:

- $p_{\text{right}}$: probability that a vehicle turns right.
- $p_{\text{straight}}$: probability that a vehicle goes straight ahead
- $p_{\text{left}}$: probability that a vehicle turns left
- $p = (p_{\text{right}}, p_{\text{straight}}, p_{\text{left}})$
We generate a network in which the arterial going towards the city center is getting more and more busy. We will divide the links in the system in groups, and define $p$ for each group separately.

1. Links that are entirely inside groups N, W, S or E (arterial roads). For these roads, we let the majority of the traffic go straight ahead: $p = (0.125, 0.75, 0.125)$

2. Incoming side links of the arterials, for which the city center is on the left side. These are the roads with
   - Their origin in NW and destination in W,
   - Their origin in SW and destination in S,
   - Their origin in SE and destination in E,
   - Their origin in NE and destination in N.

   For these links, we have $p = (0.15, 0.5, 0.35)$

3. Incoming side links of the arterials, for which the city center is on the right side. These are the roads with
   - Their origin in NW and destination in N,
   - Their origin in SW and destination in W,
   - Their origin in SE and destination in S,
   - Their origin in NE and destination in E.

   For these links, we have $p = (0.35, 0.5, 0.15)$

4. For all other links, we have $p = (0.25, 0.5, 0.25)$.

After defining this complete network, we only have the following degrees of freedom left:
   - The traffic light settings
   - The arrival intensities at all 40 entries of the network

### 5.2.2 Performance of traffic wave model

In this section, we will compare the run time of the simulation model with traffic waves (Model 3 in Section 3.5.3) with a simpler model (Model 2 in Section 3.5.2). We will use the network stated in Section 5.2.1.

#### Signal settings

For all intersections, we will use the same fixed signal settings. We use the phase plan showed in Figure 5.14. The phases $\sigma_1$ and $\sigma_3$ will last 29 seconds, while the phases $\sigma_1$ and $\sigma_3$ will last 16 time seconds. The offsets between all intersections is 0. This is definitely not the best possible offset, since travel times at full speed between two intersections are around 42 seconds. There will be no green waves for traffic going straight ahead. This is not a bad thing, since we want to compare queue models, which can be done best when longer queues are present.
Arrival intensities

We will vary the arrival intensities to gain insight in the influence of the load of the system on the runtime of the simulation. We will do this on both the wave and the non wave model. We will take the following arrival intensities:

- For all external arrivals on the arterials (which are eight external arrivals: four from outside the network and four from the city center), the arrival intensity is $\lambda$.
- For all other 32 external arrivals, the arrival intensity is $0.5 \cdot \lambda$.

We will vary $\lambda$ between 0.06 and 0.6, with increments of 0.06. At $\lambda = 0.6$, the system is almost at full capacity, but not oversaturated.

We will run the simulation for queue model 2, which we call the non-wave model, and model 3, the wave model. Each simulation run will simulate a time span of 100,000 seconds, plus 3,600 seconds of warm-up time. Since the runtime can vary heavily, we will run the simulation 100 times for each setting, and take the average. The results can be found in Figure 5.15 (where it should be mentioned that the runtime-axis is shortened).

These results show both expected and unexpected results. The expected result is that the wave model is more expensive than the non-wave model, but the difference is relatively small. The difference increases when the load gets higher, which is a natural result since the number of waves in the system increases. Since the wave model is far more realistic than the non-wave model and the difference is not very big, we choose to use the wave model for the rest of the simulation runs.
Figure 5.15: Relation between runtime and traffic load

An unexpected result is that the runtime decreases when the system gets busier. In microscopic simulation software, runtime heavily increases when the amount of vehicles increase, but our model is not a discrete event simulation in which we simulate individual vehicles. In this model a higher traffic load leads to bigger blocks instead of more blocks. Since the processing time a traffic block demands does not depend on its size, the number of events in the simulation will not increase when the size of the blocks increase.

This does not explain why the run time decreases when the load increases. This may be caused by the distribution of blocks from approaches over multiple direction lanes (described in Section 3.6.2. The algorithm for splitting blocks samples values from a Gamma-distribution, but the used algorithm for drawing random gamma variables tend to be very slow when the inserted parameter is very small.

5.2.3 Runtime performance of different traffic light algorithms

Since we implemented multiple traffic light algorithms, we want to compare their runtime performances. We will do this by using the same settings as in the previous simulation run, with the same phases $\sigma_1, \sigma_2, \sigma_3, \sigma_4$, but now we vary the traffic light algorithm. The arrival intensity parameter $\lambda$ is set to 0.6. We will test the following algorithms:

1. Fixed traffic light algorithm
2. Isolated VA traffic light algorithm
3. Synchronized VA traffic light algorithm In this case, we choose the following phases as main phases:
   - For intersections in groups W and E, $\sigma_1$ is the main phase.
   - For intersections in groups N and S, $\sigma_3$ is the main phase.
   - For all other intersections, $\sigma_1$ and $\sigma_3$ are both main phases.
<table>
<thead>
<tr>
<th>Algorithm name</th>
<th>Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed</td>
<td>3.156</td>
</tr>
<tr>
<td>Isolated VA</td>
<td>3.669</td>
</tr>
<tr>
<td>Synchronized VA</td>
<td>3.371</td>
</tr>
<tr>
<td>SVA with reduction</td>
<td>3.566</td>
</tr>
<tr>
<td>Back Pressure with MA</td>
<td>3.576</td>
</tr>
<tr>
<td>Back Pressure with ES</td>
<td>3.328</td>
</tr>
</tbody>
</table>

Table 5.3: Runtimes of traffic light algorithms, per algorithm

4. Synchronized VA traffic light algorithm with reduction In this case, we use the earlier mentioned main phases, and we make the following subgroups which are kept synchronized:

- Each of the intersection groups N,W,S and E form one synchronized subgroup.
- The other groups are each divided in four $2 \times 2$ subgroups
- We have a total of 20 subgroups which are synchronized.

5. Back pressure with moving average estimation

6. Back pressure with exponential smoothing estimation

We run each algorithm 100 times, for 100000 seconds. Note that we will only compare their runtime efficiency, not their effects on the queue lengths. We will not measure the runtime of the complete simulation, since it is dependent on the amount of traffic in the system, which depends on the chosen traffic light algorithm. Instead of that, we measure the time spent on the traffic light algorithm itself. The results are given in Table 5.3.

The fixed algorithm is the fastest of all, which is no surprise: all other algorithms have to gain information of the traffic situation, which takes time. We see that the more the system has to check the presence of traffic, the slower the algorithm gets. The isolated VA algorithm has to check the presence of traffic at almost each time step (except for time steps between the start and the minimum length of a phase), and this makes that algorithm the slowest of all. The difference between the fastest and the slowest algorithm is, however, not very big.

We are also able to compare the runtime of the traffic light algorithm to the total runtime, when looking at the runtime results of previous section. We see that the traffic light algorithm takes around 10% of the total run time.

5.3 Stability tests on the expected waiting time

Since we want the simulation model to provide useful information on various performance measures, we need to know how stable these results are. In order to do this, we will run the simulation multiple times on the same network, while measuring the average waiting time of every intersection in the system. After this, we can produce confidence intervals of these expected waiting times.

We will use the earlier mentioned network of 80 intersection (Section 5.2.1), with arrival intensity $\lambda = 0.6$ on the main entrances and $0.5 \cdot \lambda = 0.3$ on the other entrances of the network.
We will do 100 simulation runs, each covering 100000 seconds after a warm up time of 10000 seconds. We will test this strategy for multiple traffic light algorithms:

- Fixed traffic light algorithm
- Isolated VA algorithm
- Synchronized VA algorithm
- Back pressure algorithm

The phases and their lengths are the same as in Section 5.2.2.

For the following part of this chapter we will calculate 95% confidence intervals for the expected waiting times at intersections. This leads to 80 confidence intervals per strategy: one for each intersection. Since this leads to a big amount of information, we will not show all these intervals. For the fixed algorithm, we will show all 80 confidence intervals. For the other algorithms, we will only shortly mention the obtained results.

### Fixed algorithm

For fixed settings, we get the results shown in Table 5.4. The locations of the intersections can be found in 5.11. The expected waiting times are relatively long, since the system is heavily loaded.

We see that the widest confidence intervals occur at the intersections 0, 8, 71 and 79, which are the intersections at the corners of the network. These confidence intervals are around 1.3

<table>
<thead>
<tr>
<th>Intersection</th>
<th>95% Confidence Interval</th>
<th>Intersection</th>
<th>95% Confidence Interval</th>
<th>Intersection</th>
<th>95% Confidence Interval</th>
<th>Intersection</th>
<th>95% Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (68.59, 69.81)</td>
<td>1 (67.28, 68.32)</td>
<td>2 (65.5, 66.48)</td>
<td>3 (64.1, 65.04)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 (67.21, 67.86)</td>
<td>5 (61.25, 62.24)</td>
<td>6 (63.56, 64.4)</td>
<td>7 (66.35, 67.21)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 (64.48, 65.96)</td>
<td>17 (67.49, 68.44)</td>
<td>18 (63.99, 65.01)</td>
<td>19 (62.04, 62.51)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 (61.05, 61.49)</td>
<td>21 (61.15, 61.57)</td>
<td>22 (83.9, 84.42)</td>
<td>23 (58.31, 58.7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24 (61.05, 61.43)</td>
<td>25 (63.6, 64.05)</td>
<td>26 (65.67, 66.76)</td>
<td>27 (61.38, 62.32)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28 (58.64, 59.01)</td>
<td>29 (58.48, 58.86)</td>
<td>30 (60.2, 60.64)</td>
<td>31 (83.28, 84.02)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32 (60.34, 60.75)</td>
<td>33 (61.01, 61.38)</td>
<td>34 (62.56, 63.01)</td>
<td>35 (64.27, 65.32)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36 (67.25, 67.84)</td>
<td>37 (79.99, 80.45)</td>
<td>38 (84.1, 84.59)</td>
<td>39 (83.24, 84.04)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40 (83.07, 83.89)</td>
<td>41 (83.85, 84.37)</td>
<td>42 (79.86, 80.33)</td>
<td>43 (67.32, 68.13)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44 (64.37, 65.34)</td>
<td>45 (62.54, 62.97)</td>
<td>46 (61.61, 63.5)</td>
<td>47 (60.42, 60.87)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48 (83.13, 83.89)</td>
<td>49 (60.27, 60.7)</td>
<td>50 (58.32, 58.74)</td>
<td>51 (58.54, 58.92)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52 (61.38, 62.36)</td>
<td>53 (65.24, 66.16)</td>
<td>54 (63.64, 64.14)</td>
<td>55 (61.04, 61.43)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56 (58.46, 58.84)</td>
<td>57 (84.23, 84.75)</td>
<td>58 (60.97, 61.32)</td>
<td>59 (60.87, 61.35)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60 (62.1, 62.54)</td>
<td>61 (63.96, 64.95)</td>
<td>62 (66.95, 67.98)</td>
<td>63 (65.07, 65.62)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64 (61.93, 62.38)</td>
<td>65 (58.52, 58.87)</td>
<td>66 (79.69, 80.14)</td>
<td>67 (62.5, 62.9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68 (63.24, 63.73)</td>
<td>69 (65.37, 65.9)</td>
<td>70 (66.57, 67.68)</td>
<td>71 (68.34, 69.67)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72 (66.43, 67.49)</td>
<td>73 (63.67, 64.64)</td>
<td>74 (61.38, 62.4)</td>
<td>75 (67.03, 67.69)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>76 (64.72, 65.96)</td>
<td>77 (65.26, 66.16)</td>
<td>78 (66.8, 67.63)</td>
<td>79 (68.24, 69.45)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4: Confidence intervals (95%) of average waiting time per intersection, fixed algorithm
seconds wide. More general, we see that intersections with no external arrivals show lower
variances than those with one or more external arrivals. Intersection with no external arrivals
show us intervals of around 0.5 seconds wide.

We want to conclude whether the system is stable or not. In a big network, we expect that
the last intersections that become stable are the intersections in the middle of the network,
since traffic needs more time to reach them. Since these intersections have the lowest variance
of the system, we assume that the system is stable.

**Isolated VA algorithm**

The ranges of the confidence intervals for the Isolated VA algorithm are slightly wider than in
the fixed algorithm case, but this is a result of the fact that vehicle actuated algorithms are
influenced by traffic demand, making them less predictable. The intersection in the corners
of the network show the widest intervals: the maximum width is obtained at intersection
71 (1.57 seconds). The intervals of intersections without external arrivals are mostly below
1 seconds. Since the widths of these intervals are not excessive, we conclude the system is
stable.

**Synchronized VA algorithm**

The results of the synchronized VA algorithm are very similar to those of the fixed algorithm.
The widest intervals can, again, be found at the edges of the system, where 1.4 seconds
(attained at intersection 0) is the maximum value attained in this test. Since we still do not
see excessively wide intervals, we conclude the system is stable.

**Back pressure algorithm**

We ran the back pressure algorithm, using exponential smoothing for the queue length es-
timations. We chose $\alpha = 0.2$ and $\eta = 0.05$. Since we encountered that the system was
oversaturated at $\lambda = 0.6$, we run this simulation at $\lambda = 0.5$.

The widest interval can be found in the center of the system (0.77 seconds at intersection
31). Since the widths of the confidence intervals are small (smaller than the intervals of the
other strategies), we conclude that the system is also stable when using back pressure, as long
as we make sure the system is not oversaturated.

**Conclusions about stability**

All the tests we have done in this system lead to a stable system. Since the algorithms we did
not have tested are similar to at least one of these tested algorithm, we assume the system is
stable enough to use one single run in the following tests. We have ran these simulations for
100000 seconds. In the remaining part of this chapter, we will run the systems for 1000000
seconds to make the results more reliable.

Obviously, the system is only stable when it is not oversaturated. Our simulation model
will also return the capacity use of every direction lane, which is the ratio between the actual
amount of traffic that crossed the stop line and the maximum amount that could have crossed
the stop line. If this value is close to one, and the waiting times are excessively high, we can
assume the system is oversaturated.
5.4 Impact of the offset on queue dynamics

The average waiting times we have calculated in the previous section are important simulation results, but waiting time and queue length expectations are heavily time-dependent: we expect a significantly shorter queue at the end of a green phase than at the end of a long red period. To visualize this, we can have a look at time-dependent queue volumes.

For each strategy in which the cycle time $c$ is a fixed value, we can obtain the average queue lengths and queue volumes per time step in the cycle ($t = 0, 1, ..., c - 1$). With this information, we can get good insight in how the queue develops over a cycle.

Until now, we have always controlled the network of Figure 5.11 with the same offset for all intersection (this offset was zero). Since the travel time at full speed between two intersections is around 42 seconds and the cycle time is 90 seconds, it is not difficult to see that this is not the ideal offset. In this section, we will compare the queue length development of two sets of offsets with each other:

- The offset between each two connected intersections is 0
- The offset between each two connected intersections is 45

We will run all arterial intersection on the Synchronized VA algorithm, so their cycle time is maintained. The other intersections will run on Isolated VA. The phases and settings are the same in Section 5.2.2.

We will check the influence on the queue volumes on the direction lane of Intersection 31 which is entering from the west, and heading eastwards (straight ahead). We will call this direction lane $l$. Intersection 31 is at the end of the West arterial, so it is one of the busiest intersections of the system, especially from the West side. The green period ends at $t = 26$, and starts at $t = 0$ or earlier (since saved time from other phases can be added).

We edited one small network parameter: the turning fraction of the incoming side roads of the arterial. We now have that 50% goes straight ahead, 30% turns towards the city center and 20% turns away from the city center.

In Figure 5.18 and 5.21, we see the results for $\lambda = 0.6$ and $\lambda = 0.5$. The red lines show the total amount of traffic in the queue. Since, in our model, being part of the queue does not automatically imply standing still, we let the blue lines show the amount of traffic in the queue that is standing still.

It is clear that the offset heavily influences the queue lengths and volumes in the system. We see that the arrival intensities vary heavily over the cycle. Since all traffic is coming from upstream intersection 22, we see that the arrival peaks are closely related to the green phases for traffic at intersection 22 heading for intersection 31.

The biggest group of this traffic is the straight ahead traffic coming from Intersection 13, further upstream on the West arterial. The offset between intersections 22 and 31 is best chosen when the traffic coming from Intersection 13 arrives in, or just before, the green period of direction lane $l$. In the Figures 5.18 and 5.21, we see that an offset of 45 is a far better choice than an offset of 0.

Choosing an offset is, however, not an easy task. Since the arterial road between intersection 22 and 31 is a bidirectional road, we also have to pay attention to the traffic from intersection 31 to 22. In a big network, it is (mostly) impossible to provide an ideal offset to
Each link. Using fast simulation methods, like our model, are very useful to find satisfying offsets when mathematical approaches of offset optimization become too complicated.

### 5.5 Impact of different signal strategies on the network

In Section 5.3 we already measured expected waiting times per intersection. We did that to check the stability of the system with the use of confidence intervals. Now, we will measure these values again, but now we will check the performance of different signal settings on the
We will do this with the use of heat maps, containing the average waiting time for each intersection. These heat maps are tables in which the intersections are ordered as in the network (Figure 5.11). Each cell contains the average waiting time per intersection, and the redness of the cell depends on this waiting time, which makes them very easy to analyze.

In the first example, we will use them with fixed signal settings. We set all offsets to zero, and take the same phases and settings as in Section 5.2.2. The arrival intensity parameter $\lambda$ is set to 0.55. This leads to the results shown in Table 5.5.

<table>
<thead>
<tr>
<th>Int0</th>
<th>Int9</th>
<th>Int18</th>
<th>Int27</th>
<th>Int36</th>
<th>Int44</th>
<th>Int53</th>
<th>Int62</th>
<th>Int71</th>
</tr>
</thead>
<tbody>
<tr>
<td>47.66</td>
<td>51.09</td>
<td>49.96</td>
<td>49.37</td>
<td>60.1</td>
<td>50.59</td>
<td>50.98</td>
<td>51.05</td>
<td>47.71</td>
</tr>
<tr>
<td>50.9</td>
<td>54.58</td>
<td>53.7</td>
<td>52.49</td>
<td>68.76</td>
<td>54.36</td>
<td>54.52</td>
<td>54.44</td>
<td>51.09</td>
</tr>
<tr>
<td>50.47</td>
<td>54.47</td>
<td>54.14</td>
<td>53.12</td>
<td>72.29</td>
<td>54.45</td>
<td>54.08</td>
<td>53.86</td>
<td>50.24</td>
</tr>
<tr>
<td>50.47</td>
<td>54.57</td>
<td>54.66</td>
<td>54.12</td>
<td>70.43</td>
<td>54.09</td>
<td>52.98</td>
<td>52.58</td>
<td>49.23</td>
</tr>
<tr>
<td>60.23</td>
<td>69.0</td>
<td>72.0</td>
<td>70.85</td>
<td>0</td>
<td>71.4</td>
<td>72.51</td>
<td>68.94</td>
<td>60.19</td>
</tr>
<tr>
<td>49.12</td>
<td>52.5</td>
<td>53.19</td>
<td>53.8</td>
<td>70.3</td>
<td>53.88</td>
<td>54.32</td>
<td>54.48</td>
<td>50.47</td>
</tr>
<tr>
<td>50.14</td>
<td>53.89</td>
<td>54.24</td>
<td>54.55</td>
<td>72.04</td>
<td>52.94</td>
<td>53.93</td>
<td>54.26</td>
<td>50.84</td>
</tr>
<tr>
<td>50.67</td>
<td>54.75</td>
<td>54.4</td>
<td>54.33</td>
<td>68.64</td>
<td>52.84</td>
<td>54.08</td>
<td>54.58</td>
<td>51.41</td>
</tr>
<tr>
<td>47.7</td>
<td>50.92</td>
<td>50.72</td>
<td>50.64</td>
<td>60.16</td>
<td>48.97</td>
<td>50.08</td>
<td>50.92</td>
<td>47.96</td>
</tr>
</tbody>
</table>

Table 5.5: Average waiting times per intersection, fixed settings, offset 0

We can clearly see that the arterials of the system are the busiest parts of the system. Since incoming traffic from the side of the arterial prefers to travel towards the city center, we see that traffic gets more busy when it gets closer to the city center. The intersections at the edge of the network have a significantly lower average waiting time, which is caused by the fact that traffic on approaches from outside the system has significantly lower average waiting times than other traffic.

We know that traffic from outside the system arrives according to a Poisson process, while traffic inside the system travels in bigger platoons. The fact that these platoons have longer average waiting times than the Poisson arriving traffic from outside, may be caused by bad synchronization between the intersections. If big platoons arrive when a green period is just finished, they will all face huge delay. This can be fixed by changing the offset between the intersections, like we did earlier in Section 5.4.

We will change the offsets:

- For the intersections 0 to 39, we set the offsets of all even numbered intersections (0,2, etc.) to 0. The offsets of all odd numbered intersections (1,3, etc.) is set to 45.
- For the intersections 40 to 79, we set the offsets of all even numbered intersections to 45, and the offsets of all odd numbered intersections to 0.

With these settings, every intersection has an offset of 45 with all its neighbors. The results can be found in Table 5.6.

We see a sharp drop in average waiting times for every intersection in the section, which clearly shows the importance of a well chosen offset. The average waiting times at the edge of the network are not lower than in the rest of the network (although they also dropped
The waiting times of the other intersection grew slightly, but the difference is small (less than one second on average).

Now, we will try some of our other traffic light algorithms. First, we will run all traffic signals on the Isolated VA algorithm (Section 4.2). Offset does not play any role in this process, since adjacent intersections are not synchronized anymore in this algorithm. The results are given in Table 5.7.

Compared to the fixed settings with offset 45, the waiting times on the arterials grew significantly, which leads to the conclusion that the arterial profit from being synchronized. The waiting times of the other intersection grew slightly, but the difference is small (less than one second on average).

If we synchronize the arterials with using the Synchronized VA algorithm (Section 4.3) with offset 45 and keep the other intersections running on Isolated VA settings, we get the results shown in Table 5.8.

These results are similar to the results in Table 5.6 where all intersections use fixed settings, with the following differences:

- The busiest arterial intersections, next to the city center, seem to profit from the Synchronized VA settings.
- The less busy arterial intersection at the edge of the network face little higher average waiting times.

<table>
<thead>
<tr>
<th>Int0</th>
<th>Int9</th>
<th>Int18</th>
<th>Int27</th>
<th>Int36</th>
<th>Int44</th>
<th>Int53</th>
<th>Int62</th>
<th>Int71</th>
</tr>
</thead>
<tbody>
<tr>
<td>40.44</td>
<td>39.68</td>
<td>38.98</td>
<td>37.94</td>
<td>43.4</td>
<td>38.4</td>
<td>39.56</td>
<td>39.87</td>
<td>40.01</td>
</tr>
<tr>
<td>40.09</td>
<td>39.21</td>
<td>38.56</td>
<td>37.81</td>
<td>41.77</td>
<td>38.1</td>
<td>38.97</td>
<td>39.01</td>
<td>39.52</td>
</tr>
<tr>
<td>39.56</td>
<td>38.74</td>
<td>38.6</td>
<td>38.52</td>
<td>44.6</td>
<td>38.63</td>
<td>38.67</td>
<td>38.36</td>
<td>39</td>
</tr>
<tr>
<td>38.34</td>
<td>37.98</td>
<td>38.54</td>
<td>40</td>
<td>54.72</td>
<td>39.78</td>
<td>38.64</td>
<td>37.52</td>
<td>38.39</td>
</tr>
<tr>
<td>43.63</td>
<td>41.4</td>
<td>44.57</td>
<td>54.85</td>
<td>0</td>
<td>54.43</td>
<td>44.56</td>
<td>41.91</td>
<td>43.49</td>
</tr>
<tr>
<td>38.07</td>
<td>37.73</td>
<td>38.18</td>
<td>39.88</td>
<td>54.89</td>
<td>39.84</td>
<td>38.58</td>
<td>37.93</td>
<td>38.47</td>
</tr>
<tr>
<td>38.88</td>
<td>38.13</td>
<td>38.45</td>
<td>38.52</td>
<td>44.55</td>
<td>38.64</td>
<td>38.69</td>
<td>38.9</td>
<td>39.27</td>
</tr>
<tr>
<td>39.48</td>
<td>38.93</td>
<td>38.81</td>
<td>38.08</td>
<td>41.02</td>
<td>37.55</td>
<td>38.51</td>
<td>39.22</td>
<td>39.6</td>
</tr>
<tr>
<td>40.47</td>
<td>39.85</td>
<td>39.61</td>
<td>38.31</td>
<td>43.65</td>
<td>37.91</td>
<td>38.82</td>
<td>39.39</td>
<td>40.56</td>
</tr>
</tbody>
</table>

Table 5.6: Average waiting times per intersection, fixed settings, offset 45

<table>
<thead>
<tr>
<th>Int0</th>
<th>Int9</th>
<th>Int18</th>
<th>Int27</th>
<th>Int36</th>
<th>Int44</th>
<th>Int53</th>
<th>Int62</th>
<th>Int71</th>
</tr>
</thead>
<tbody>
<tr>
<td>40.96</td>
<td>40.28</td>
<td>40.29</td>
<td>38.81</td>
<td>50.8</td>
<td>39.73</td>
<td>41.14</td>
<td>40.83</td>
<td>41.02</td>
</tr>
<tr>
<td>40.82</td>
<td>39.9</td>
<td>39.51</td>
<td>39.2</td>
<td>52.19</td>
<td>39.87</td>
<td>40.4</td>
<td>40.08</td>
<td>40.41</td>
</tr>
<tr>
<td>40.32</td>
<td>39.46</td>
<td>39.85</td>
<td>39.34</td>
<td>54.67</td>
<td>40.21</td>
<td>39.56</td>
<td>39.5</td>
<td>39.26</td>
</tr>
<tr>
<td>39.9</td>
<td>40.07</td>
<td>40.96</td>
<td>40.39</td>
<td>60.15</td>
<td>40.49</td>
<td>39.25</td>
<td>38.86</td>
<td>39.08</td>
</tr>
<tr>
<td>51.14</td>
<td>53.24</td>
<td>55.28</td>
<td>59.94</td>
<td>0</td>
<td>60.03</td>
<td>56</td>
<td>51.48</td>
<td>49.72</td>
</tr>
<tr>
<td>38.91</td>
<td>39.22</td>
<td>39.16</td>
<td>40.13</td>
<td>59.89</td>
<td>40.69</td>
<td>40.56</td>
<td>40.01</td>
<td>39.99</td>
</tr>
<tr>
<td>39.5</td>
<td>40.24</td>
<td>39.96</td>
<td>40.35</td>
<td>55.37</td>
<td>38.98</td>
<td>39.77</td>
<td>40.13</td>
<td>40.4</td>
</tr>
<tr>
<td>40.06</td>
<td>40.28</td>
<td>39.93</td>
<td>39.96</td>
<td>52.96</td>
<td>38.86</td>
<td>39.75</td>
<td>40.2</td>
<td>40.81</td>
</tr>
<tr>
<td>41.13</td>
<td>41.13</td>
<td>40.81</td>
<td>39.51</td>
<td>50.71</td>
<td>39.14</td>
<td>39.18</td>
<td>40.08</td>
<td>40.47</td>
</tr>
</tbody>
</table>

Table 5.7: Average waiting times per intersection, Isolated VA settings
We will also test the same settings, but then with applying cycle time reduction (Section 4.4). We put all intersections on the same arterial in one group, so that we have four groups on which reduction is applied. The non-arterial intersections are still running on Isolated VA. This leads to the results shown in Table 5.9.

<table>
<thead>
<tr>
<th>Int0</th>
<th>Int9</th>
<th>Int18</th>
<th>Int27</th>
<th>Int36</th>
<th>Int44</th>
<th>Int53</th>
<th>Int62</th>
<th>Int71</th>
</tr>
</thead>
<tbody>
<tr>
<td>40.87</td>
<td>39.67</td>
<td>39.39</td>
<td>38.45</td>
<td>45.5</td>
<td>39.6</td>
<td>39.8</td>
<td>41.25</td>
<td>40.57</td>
</tr>
<tr>
<td>41.03</td>
<td>40.14</td>
<td>39.22</td>
<td>38.13</td>
<td>42.98</td>
<td>39.13</td>
<td>39.99</td>
<td>40.36</td>
<td>39.99</td>
</tr>
<tr>
<td>40.6</td>
<td>40.07</td>
<td>39.27</td>
<td>39.56</td>
<td>44.11</td>
<td>39.68</td>
<td>39.22</td>
<td>39.72</td>
<td>39.61</td>
</tr>
<tr>
<td>39.16</td>
<td>39.55</td>
<td>39.76</td>
<td>39.99</td>
<td>53.59</td>
<td>40.11</td>
<td>38.79</td>
<td>38.83</td>
<td>38.8</td>
</tr>
<tr>
<td>45.51</td>
<td>42.85</td>
<td>44.46</td>
<td>52.96</td>
<td>0</td>
<td>53.38</td>
<td>44.73</td>
<td>42.67</td>
<td>45.65</td>
</tr>
<tr>
<td>38.78</td>
<td>39.19</td>
<td>38.97</td>
<td>40.32</td>
<td>53.06</td>
<td>39.67</td>
<td>39.41</td>
<td>39.76</td>
<td>39.75</td>
</tr>
<tr>
<td>39.61</td>
<td>39.52</td>
<td>39.7</td>
<td>38.67</td>
<td>44.49</td>
<td>38.98</td>
<td>40.3</td>
<td>40.38</td>
<td>40.21</td>
</tr>
<tr>
<td>40.21</td>
<td>40.22</td>
<td>39.95</td>
<td>39.93</td>
<td>42.77</td>
<td>38.77</td>
<td>39.39</td>
<td>40.49</td>
<td>40.5</td>
</tr>
<tr>
<td>40.21</td>
<td>40.8</td>
<td>40.37</td>
<td>39.6</td>
<td>45.56</td>
<td>38.52</td>
<td>39.33</td>
<td>40.25</td>
<td>40.48</td>
</tr>
</tbody>
</table>

Table 5.9: Average waiting times per intersection, Synchronized VA settings with reduction on arterials, Isolated VA elsewhere

We see another improvement for the waiting times at the intersections right next to the center. For the other intersections, the improvements are not significant.

The last shown signal setting leads to the best results in this specific case. Of course, this does not state that it is the best overall setting: when traffic intensities in the system are much lower, the Isolated VA algorithm will be the best strategy to use (research to this has been done in [4]. We see that the use of heat maps gives a very clear view on the performance of signal settings on a network.
Chapter 6

Conclusions

We have developed and implemented a new simulation model for traffic in urban environments. Its performance depends on multiple criteria, which we will discuss one by one.

Efficiency

Since building an efficient simulation model was our main task, we have simplified the representation of traffic, roads and intersections heavily. We did not exactly compare simulation speeds between SUMO and our model, but the difference in efficiency is remarkable. In SUMO, we needed around 10 minutes to simulate 100,000 seconds of traffic on a single three lane road (with a length of four kilometers) with one traffic light. Our model needed 5 minutes to simulate 1,000,000 seconds on a grid network of 80 intersections (with 180 bidirectional roads of 500 meters). We can conclude that this is a huge difference, this model makes simulating urban network much more efficient and even makes multiple runs on a big network possible.

Some modelling choices, like implementing traffic waves and stochastic distribution of traffic over direction lanes, slow down the algorithm a bit, but not significantly. Since it heavily improves realism of our queue representation and our model is still very fast while using this model, we do not think further step in improving efficiency are needed.

Accordance with reality

The representation of traffic and networks in our model is very simplified. Our model can not take into account traffic with different speeds on the same link, and it also neglects differences in driving behaviour (i.e. agressive or patient drivers). Our model is also unable to simulate lateral movements on the road, which occur frequently in real life due to lane changing and prefiltering.

These drawbacks make our system unable to simulate multi-lane highways, where this behaviour plays a big role. In that setting, it is impossible to neglect these effects. However, we developed this simulation model for simulating inner city networks, with heavy and relatively slow traffic and relatively short roads between intersections. In these situations, neglecting effects like overtaking traffic is less problematic.

Of course, we have to be sure that variables like the traffic densities and the departure intensities are realistic, otherwise we can not use this model to simulate reality. If these
variables are not correct, the queue lengths and capacities of roads and intersections are estimated in a wrong way, which will make a simulation study useless. To do this in a right way, real life measurements in the network are essential.

Unfortunately, we did not have tools and data to compare our model to real traffic data. This would have been an interesting extension on this project.

Usefulness of results

Our model can provide good information about bottlenecks in the system. It provides information about queue lengths and blocking probabilities that give a clear view about whether a road is oversaturated or not. We can also calculate average waiting times for traffic on approaches, direction lanes and intersections.

We are not able to calculate average waiting times for specific trips through the system, affecting multiple intersections. Since we do not follow vehicles through the system, we can not count waiting times for them. Simply adding up the average waiting times on each approach and direction lane on a route to get its average waiting time does not give realistic results, since waiting times heavily depend on offsets and the moment traffic arrives at a queue.

Since we developed our model for busy city centers, we think queue lengths and blocking probabilities are more important, since when a queue blocks traffic with other destinations, this can have a catastrophic results and may cause a gridlock. Our simulation only models blocking by direction lanes and not blocking by approaches for efficiency reasons, so it is important to keep an eye on the blocking probabilities of approaches (which is the probability that the queue is longer than the link itself).

Overall conclusion

The newly developed simulation model is a useful tool to get insight in the functionality of traffic signal settings in city traffic networks. It provides useful information on whether bottlenecks in the system occur, and where they are located. Compared to microscopic simulation model, it is significantly faster, which makes it easier to test multiple signal strategies on a traffic network in reasonable time.

When more detailed traffic effects, such as lateral movements and differences in driving behavior, should be taken into account, SUMO or VISSIM are better options, but these models are significantly slower and unpractical on big networks.

Recommendations

For future research, we can recommend the following subjects:

- Comparing this model with real life data, to calibrate its parameters extensively and make it more realistic.
- Extending the model such that it can contain traffic with multiple speeds.
- Extending the model with roundabouts and priority intersections.
Bibliography


