Democratic Traffic Lights

A voting-based approach to decentralized agent-based real-time urban traffic control

A dissertation submitted to the University of Dublin, in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Ronny Hendrych

2009

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Dublin, September 11, 2009

Ronny Hendrych

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Dublin, September 11, 2009

Ronny Hendrych

Acknowledgements

First of all, I want to thank my supervisor Vinny Cahill, for his ideas, guidance and help during this stressful time. Specials thanks go to Ivana Dusparic, who guided me through the shallows of the DTS and its trace files. Niall O'Hara deserves a special thank, too. Without your plug-in DLL for VISSIM I would have had problems finishing this thesis. Also thanks for the many afternoons we spent together, bouncing back and forth ideas for our two projects and the laughs that only programmers can share on more than one occasion.

I wish to thank Clay Stevens and my much better half for their attempts to get this thesis in an English readable format.

I would also like to thank the NDS class of 2008/2009 for their friendship and help. You all made this an incredible year. We had lots of fun, moaned together during the assignments and the time of the exams and shared something very special.

I am also very thankful for my family and friends. I could not have done this without your moral and financial support.

Finally, I like to thank my good friend $C_8H_{10}N_4O_2$. Without you, this would have never been possible!

Abstract

This dissertation describes the application and evaluation of a voting-based decision-making algorithm for decentralized traffic light controllers in an urban traffic control (UTC) system.

Nowadays, UTC systems are very limited in how they can observe the traffic flow on the roads that they manage. Typically, these systems use inductive loops, which are embedded in the streets close to junctions, to count how many vehicles are approaching or passing them in a fixed amount of time. However, these sensors can not definite distinguish between what kind of vehicle is passing, nor the destination of the vehicle. In fact, they can differentiate between a car and a bus, but they can not tell if the bus is a member of the public transport system.

With the emergence of wireless vehicular ad-hoc networks (VANET) vehicles could communicate to the system controllers from a much further distance, to tell them exactly what kind of vehicle they are and where they want to go. Based on this much more detailed input of sensor data, a traffic light agent could base its decision-making process for the next phase on a new set of algorithms. These decisions of the autonomous intelligent parts of an UTC system could be much fairer and increase the overall throughput in an urban area.

One way to use this kind of information could be that cars approaching junctions with traffic lights could vote for the green phase to correspond to an origin-destination pair in the next cycle. Different kinds of vehicles could have a different number of basic votes (e.g. cars 5 votes and buses 80 votes) and vehicles which have to wait during a cycle could temporary have a higher number of votes in the next decision-making step to counteract starvation.

Therefore, two voting-based agents were developed and tested with VISSIM, a microscopic traffic flow simulation software, against two Round Robin agents. In this setup, an agent was responsible to choose the next green phases for a junction with traffic lights.

The results of the evaluation prooved that the used voting-based algorithms could increase the throughput of the system and reduce the average waiting and travel time of vehicles in the urban area.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Motivation

In 1996 approximately 1,406,000 people lived in the Greater Dublin Area (GDA). Ten years later the population was increased to 1,662,000 people and the forecast estimates around 2,413,000 people for the year 2026 [24].

With the increase of the people living in the Dublin area the roads had to handle a higher amount of traffic. This will continue to be an issue in the future. More vehicles mean a lower average speed, more stops during the travel and therefore more environmental pollution for the city area and more frustration for the drivers [21]. This applies especially during so called peak-hours (e.g. morning-peak and evening-peak). Therefore, scientists are searching for ways to increase the traffic flow in urban areas to reduce the waiting time for the overall road users.

So called Urban traffic control (UTC) systems, like the Split Cycle Offset Optimisation Technique (SCOOT)[6] or the Sydney Coordinated Adaptive Traffic System (SCATS) [32] were developed to manage the traffic flow more individually. Therefore these systems uses sensors, like induction loops embedded in the streets closely before the stop lines of a junction, to gather the information on which they base their decision for the next traffic light phases on.

However, these sensors are limited. So is it not possible to determine the target destination of the vehicle with these kind of passive sensors, nor what kind of vehicle (e.g. truck or public bus) is passing through.

Due to the development of wireless communication devices in the last years the idea

came up that cars could communicate to traffic light controllers of the UTC directly. Road users could send their position, the number of people sitting in the car and the desired direction via so called Vehicles-to-Infrastructure communication to a traffic light agent, which is responsible for setting the green phases of the traffic lights.

With that kind of information it should be possible to develop better agents which can increase the overall traffic flow inside urban areas. One approach for a better traffic flow management is a voting-based system, where every vehicle can vote on how the phase should change so as to improve global traffic flow.

In this approach, each kind of vehicles could have a different amount of basic votes (e.g. cars 5 votes and public buses 80 votes) and vehicles which have to wait during a cycle, could temporarily have a higher amount of votes in the next decision-making step to counteract starvation.

This dissertation describes the application and evaluation of such a voting-based decision-making algorithm for decentralized traffic light agents of a virtual Urban Traffic Control (UTC) system.

1.2 Structure of the thesis

The thesis is structured as follows: After a short introduction (chapter 2) into the technical fundamentals which are necessary to understand the rest of this dissertation, the materials used and methods developed are presented (chapter 3). Afterwards, chapter 4 describes the implementation of the necessary tools which were needed to run and evaluate the voting-based agents. Chapter 5 evaluates the developed traffic lights agent against other existing UTC-agents. As an conclusion, chapter 6 discusses the results of the evaluation and sums up this dissertation.

CHAPTER 2

Fundamentals

This chapter provides a brief introduction of the necessary background knowledge for this dissertation. These fundamentals are represented as abstractly as possible. For a full description, the author refers to relevant materials, for example [15] or [32].

2.1 Urban traffic control

Urban traffic control (UTC) is a specialized part of an intelligent transportation system (ITS). UTC systems are responsible for the co-ordination, integration and control of traffic signals over a wide urban area, with the goal to manage the traffic flow [34].

Basically, modern UTC systems collect data from different kinds of sensors to get a picture of the actual volume of traffic on the roads and then try to set the behavior of the collections of traffic lights around the area they control so, that the traffic flow is the best possible for the situation.

For a better understanding, the two main parts of an UTC will be explained.

2.1.1 Traffic lights

Traffic lights came a long way from the first installment of a simple electrical version in the early nineteen-twenties, to being a part of the first (analog) computer based traffic control system in 1952, towards the energy efficient LED-based lamps, which are used nowadays [32]. Shape and technology may have changed, but the basic strategies are still the same.

Use of traffic lights

Traffic lights can be a cost-intensive factor in the budget of a city. However, there are conditions, which require the installment of a new system on a junction [32]:

- A growth in traffic demand produces congestion
- Failure of older equipment
- Traffic flow improvement is only possible with new hardware

Installing a new group of traffic lights on a junction is not enough. These lights have to be configured to improve the traffic flow.

Basic concept

Basically, a traffic light has two important attributes to manage:

- **cycle time** It defines the amount of time the particular traffic light displays something else than the red light in a normal mode.
- **(green) split** It defines the amount of time the particular traffic light displays the green light before changing to the next part of the sequence.

Figure 2.1 displays the standard cycle of a traffic light in Ireland.

Figure 2.1: Full Irish traffic light cycle: From red directly to green and then, after the split time is over, it changes to amber.

Phases

Phases are more like the bigger picture of the behavior of the traffic lights from a junction. Usually, one control unit per junction handles which symbol a traffic light displays at a given time. After the cycle time is over it changes to the next set of symbols for the lights. Each of these sets of symbols is called a phase.

Organizations like the National Electrical Manufacturers Association (NEMA) or the partners behind the Traffic Management Unit Auckland (TMU) have published Guidelines for street designers, to generate well fitting phase-sets for signal controllers of a junction [37][4].

Figure 2.2 displays a common used phase-set. In this example of a two-phase signal sequence a very simple Round Robin approach [33] is used to determine the next phase for the junction. However, there exists a conflict in phase “a)”. Whenever a vehicle from the left side of the picture wants to make a right turn, it could collide with a vehicle approaching the junction from the right side. Therefore, priority rules have to be defined. In Ireland, the car from the right side is allowed to proceed first [17].

A list of examples of phase-sets for different kinds of junctions can be found in[37]. All these phases were generated with the focus of usability with the commonly used sensors of today UTC systems (see section 2.1.2).

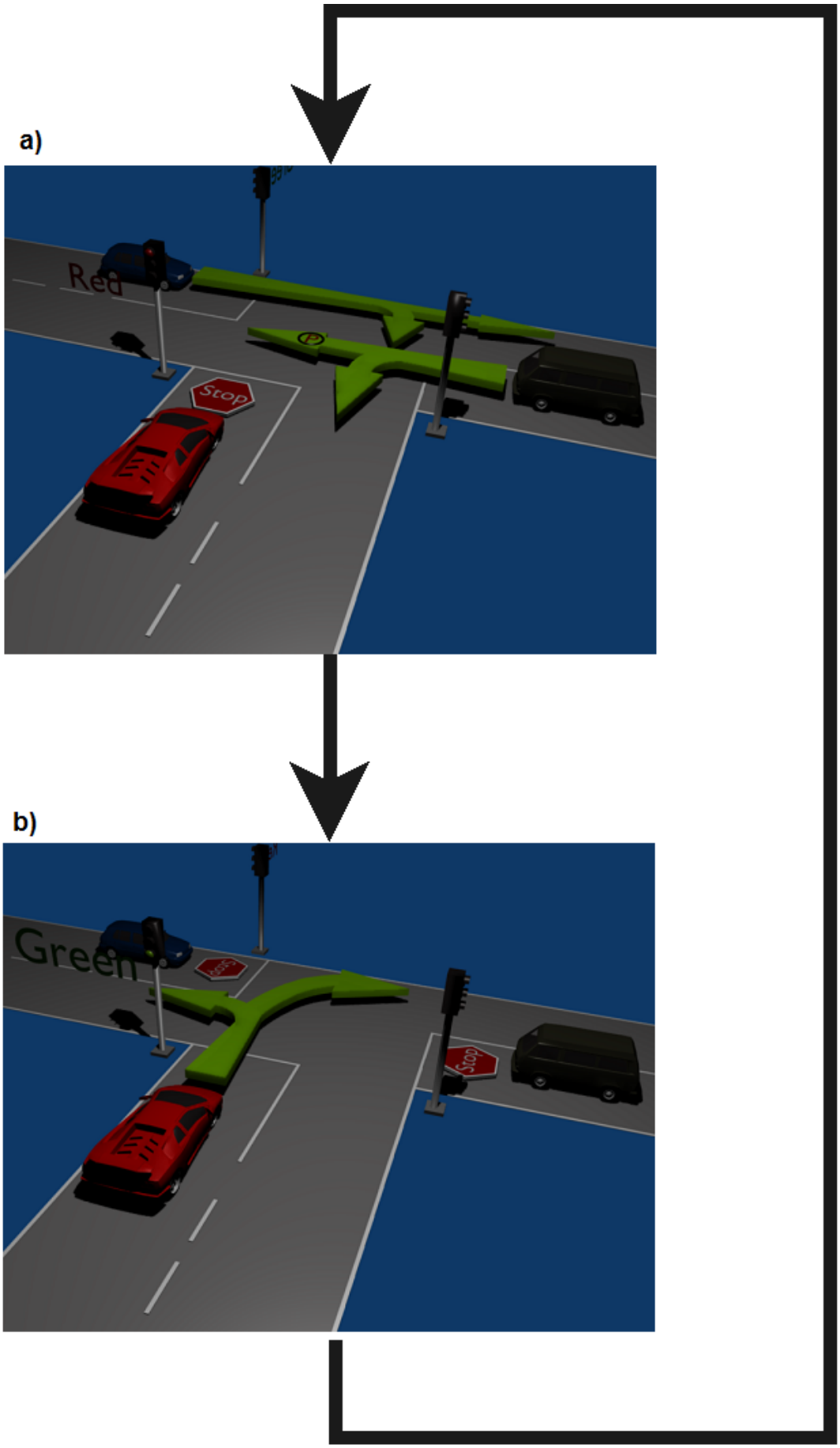


Figure 2.2: Example of a commonly used phase-set for T-shaped junctions: Sequence starts with phase a). After cycle time controller switches to phase b).

2.1.2 Sensors

To get the best possible picture of the traffic density UTC systems use different kinds of sensors. Since their introduction in the early 1960s Inductive Loops have become the most used in-roadway sensor [5]. The basic concept behind this well-understood technology is simple. A conductive element is incorporated in loops in the pavement of a road. These loops usually are embedded in front of the stop line of a junction and are energized with a continuous current electric flow.

When a vehicle enters the area of the loop the metal reduces the loop´s inductance. This reduction can be measured. Bigger vehicles reduce the inductance more and can be recognized, the speed of a vehicles can be determinate. Furthermore, more than one inductive loop in a row can increase the accuracy of measurement and can also be used as a queue counter.

This technique has a lot of advantages [32]:

- A large experience base
- A flexible design
- Insensitive to the climate and the weather
- Provides very accurate count data for all the basic data

However, there also are a lot of disadvantages for this kind of sensor. The most serious is that the maintenance and installation requires construction works on the street, which is expensive and can lead to lane closure [5].

Along with, the in-roadway sensors like inductive loops, magnetometers and magnetic detectors, there is a group of above-roadway sensors which also are used by modern UTC systems.

One well known type is the Video Image Processor sensor. The benefit of video cameras is that multiple lanes can be monitored at once. With the evolving of pattern matching techniques a rich array of data is available to these kind of sensors. However, maintenance is also tricky. So, depending where the camera is mounted, lanes have to be closed for lens cleaning or other repairs. Furthermore, fog or other kinds of bad weather can reduce the quality of the data.

There is one fact that all used sensors have in common – they are only passive. Therefore,

none of them can determine where a vehicle wants to travel, except when there is a dedicated lane for vehicles which want to turn left or right at a junction.

2.1.3 Types of UTC system

Over the years many different UTC systems where created. All these systems can be classified in two major types - fixed time and traffic responsive systems. In the following some of these systems are presented in an overview [23]:

Type	Description
Fixed time systems	The basis of many UTC systems. These systems can not respond in a dynamic way to changes of traffic density due to their use of precalculated time plans for each junction. These fixed time plans were the first step to change the used phase-sets depending on the day and time (peak hour management).
Plan selection systems	These systems also use fixed time plans, but also utilize sensors data to decide which time plan is used at the moment. This first basic attempt of a more dynamic approach was not sophisticated enough and showed not better results than the fixed time systems.
Plan generation systems	This kind of UTC systems generate their own time plans based on detector data. These systems are more adaptive and less under control of traffic engineers. Theoretically, these system can even adapt to traffic accidents on the roads, but in practice restrictions forbid large changes in the plans.
Local adaptation systems	Local adaptation systems take sensor data into consideration and modify the time plans they received from a central entity. In this way, they can respond more dynamically to changes of the local traffic flow.
Traffic responsive systems	These systems respond completely dynamically. One centralized entity collects all the data from the distributed traffic controllers and bases its decision on these facts. These systems have the advantage, that all the important data is collected at one central place. However, there exists also an distributed processing version, which connected to their neighbor controllers, which could have a better performance, but also has a higher management cost.

Table 2.1: Types of UTC systems

Furthermore, UTC systems can also be classified in 4 main categories [8]:

- **Fully centralized** All sensor data is delivered to one central management entity. This has the benefit, that all data is available at one central point, which can help to plan the overall strategies throughout an urban area. This centralized system has to cope with a lot of data and many variables. Therefore, these system tend to plan more for long-term solutions and are not so reactivate. The most popular fully centralized system is SCOOT [6].
- **Fully distributed** In this type of system the traffic light controller (agent) on each junction calculates the next phases on its own. A central entity collects data only for monitoring purposes. The agents of this thesis are based in this approach. PRODYN [9], for example, uses this approach.
- **Centralized hierarchical** These systems try to combine centralized and distributed features. Local adaptation systems (see table 2.1) are based on this philosophy. SCATS (see section 2.1.4) is one of the most well know systems in this category.
- **Distributed hierarchical** Usually, a central manager sends time tables to the controller on the junctions, but they can completely withdraw these orders and use tables based on their sensor data (two level structure). One well known system in this category is UTOPIA [22].

2.1.4 SCATS

The Sydney Coordinated Adaptive Traffic System (SCATS) is a local adaptation system which was first used in Sydney in 1982. Since than it was deployed in many different cities around the world, like New Zealand, Hong Kong and also in Dublin. In 1997 SCATS was responsible for 170 traffic light controllers in Dublin, with expanding tendencies [10].

This centralized hierarchical system works with a fixed phase scheme for the traffic lights for each junction in the system. In its normal mode the local controller adjusts split, offset and cycle time for every junction once every calculation step (usually one minute), which represents its two-level structure.

For deciding the new parameters for the next calculation step SCATS typically uses inductive loop sensors closely before the stop lines of junctions with traffic lights [32].

2.1.5 Conclusion

Today, used UTC systems help to greatly improve the traffic flow in urban areas, but the most spread systems still have problems when it comes to saturated traffic conditions [30]. One of the main problems is that systems like SCATS and SCOOT due to the use of their algorithms are not able to adapt their phase set for the traffic lights in real-time.

Therefore, other approaches were explored over the years (e.g. algorithms which try to calculate the split and offset via dynamic optimization problems in real-time [16], or store-and-forward modeling approaches [7]), but all these attempts have to handle the poor resolution of the used sensor data.

2.2 Vehicular-to-X communication

One approach to improve the quality of the data about the volume and kind of traffic on the roads is, that vehicles actively tell the UTC systems this information. This could be done via wireless communication channels. This chapter provides an insight in this topic.

2.2.1 Vehicular ad-hoc networks

Wireless Vehicular ad-hoc networks (VANET) are a special form of mobile ad-hoc networks (MANET). In contrast to a MANET, which consists mostly of nodes which usually have more a static character (like deployed field sensors from the military), a VANET consists of highly mobile and dynamic vehicles. These mobile nodes form networks via vehicle to vehicle (V2V) and/or vehicles to infrastructure communication [31].

These VANET´s still exist only as an academic area of research, but will have a major impact in the near future [36]. The vehicular communications (VC) between nodes of of VANET can be divided in three major systems:

Communications Type	Description
Inter-vehicle (IVC)	Vehicles only communicate which each other
Hybrid-vehicle (HVC)	Vehicles talk to each other and to roadside units
Roadside-vehicle (RVC)	Vehicles only communicate with roadside units

Table 2.2: Types of VC systems

2.2.2 Wireless communication

One major topic in the research area of VANET´s is the aim, to find the best possible solution for the communication between the involved entities. A lot of different communication standards, such as IEEE 802.11, Bluetooth and cellular mobile networks (for example GSM, and 3G) could be used to connect vehicles and road site units to create a network [3, 31].

However, due to the highly mobile characteristic of VANET´s it is very challenging to find a suitable solution. The IEEE 802.11p draft tries to cope with exactly this problem. 802.11p will use for medium access control (MAC) dedicated short-range communications (DSRC) services [39] to increase the throughput in a short time.

In this thesis cars are approaching junctions in an urban area and talk directly to the traffic controller (the agent) for the junction. Therefore, the communication is not so problematic, like when cars would drive with high speed on a motorway and try to generate a VANET. Furthermore, the experiments in this thesis are completely simulator-based and for the first approach the focus lies on the basic concept, not on the simulation of sophisticated use of different communication standards.

2.2.3 Application of Vehicular communication systems

In the last year a lot of experiments for applications where done to test the different possibilities for vehicular communication systems (VCS). This section will provide some examples and results of these.

At the moment there are two addressing schema commonly considered [31]:

- **Fixed addressing** Every node is given a fixed address when it joins the network. Most ad hoc networking applications assume this method as primary addressing schema.
- **Geographical addressing** In this schema the nodes change addresses, depending on their geographical position.

Table 2.3 provides an overview of the different types of of VCS Applications [31]:

Applications Type	Description
Public safety	The most important type of usage for this new field of applications. These applications could save lives. Drivers could be informed of collisions on motorways in front of them, or cars could brake if a driver misses a red traffic light on a junction. The difficulty in this applications lies in the hard real time constrains.
Traffic management	This kind of application focuses on improving the traffic flow by reducing congestion. This is the type of application which could benefit from the vote-based approach in this thesis. A traffic management application usually uses a sparse roadside-vehicle communications (SRVC) system.
Traffic coordination and assistance	These type of applications try to increase the capacity of existing motorway roads. One main focus lies in Platooning, where vehicle form a group and travel together for a part of their journey. The other focus is passing and lane change assistance, with the aim of reducing risk during these maneuvers. Both techniques require close-range IVC with real-time constraints, which are very hard to achieve.
Traveler information support	This technique is not really new. Nowadays, navigation systems can provide information like gas stations and parking areas. The idea is to increase the amount of information via selected infrastructure places. Also road warnings could be deployed by local authorities.
Comfort	Comfort applications try to improve the comfort for the traveler. Targeted vehicular communications aims at communication with surrounding vehicles for playing games or exchange massages. Vehicle to land-based destination communications try to connect to road-side stations to enable more traditional applications like web or email services.

Table 2.3: Types of VCS Applications

2.2.4 Conclusion

Vehicular communication systems are very popular area of science in the moment. Although research results in this area are very promising it remains to be seen how these can be used in practice. Large scale projects, like the new test track in Germany, with 400 test cars and 150 roadside stations [1] will show how well reality matches the simulations.

CHAPTER 3

Materials and Methods

After illustrating the fundamentals which are necessary for understanding this thesis the following chapter will discuss the necessary materials and methods for applying a voting-based decision-making algorithm for decentralized traffic light agents.

The first part of this chapter deals with the theoretical parts for the task. Afterwards, the used simulator is explained and a work flow for running a simulation for the agents is developed.

Finally, the materials which are essential for the development of the application are presented.

3.1 Specialized Phases

Section 2.1.1 describes the basic concept of phases. Nowadays, all commonly used phases are a result of the limited sensor data. In this approach traffic light agents are aware where a vehicle is at the moment and where it wants to travel. This could be accomplished via RVC and GPS or WLAN tracking [29].

This “From to”-reference could be achieved with the following solutions:

- Navigation systems send their position and the next target road to the traffic light agent. For this approach the driver has to use the navigation system actively.
- A navigation could run passively. Therefore, it always knows its position. When the car is approaching a junction and uses the blinker to signal the next direction, this information could be used to calculate the next road and send that information to the traffic light agents.

- It is also possible to use the global position and the lane the car is queuing. This only works when there are dedicated lanes for right and left turns on the street.

With this focus we can define three designated directions when a vehicle is approaching a junction:

- Left
- Straight
- Right

With this better information it is possible to generate more specialized conflict free phases (see section 2.1.1) which could better suit the actual traffic situation.

3.1.1 Phase-to-Road mapping

Before we can define these phases it is important, that there is one unique view on every type of junction in an urban area. Figure 3.1 presents the mapping of the incoming directions to every junction type. For the work flow it is important, that for example the continuous road for every T-shaped junction is defined as “left” and “right” and the road which is discharged into the ongoing road is always defined as “down”.

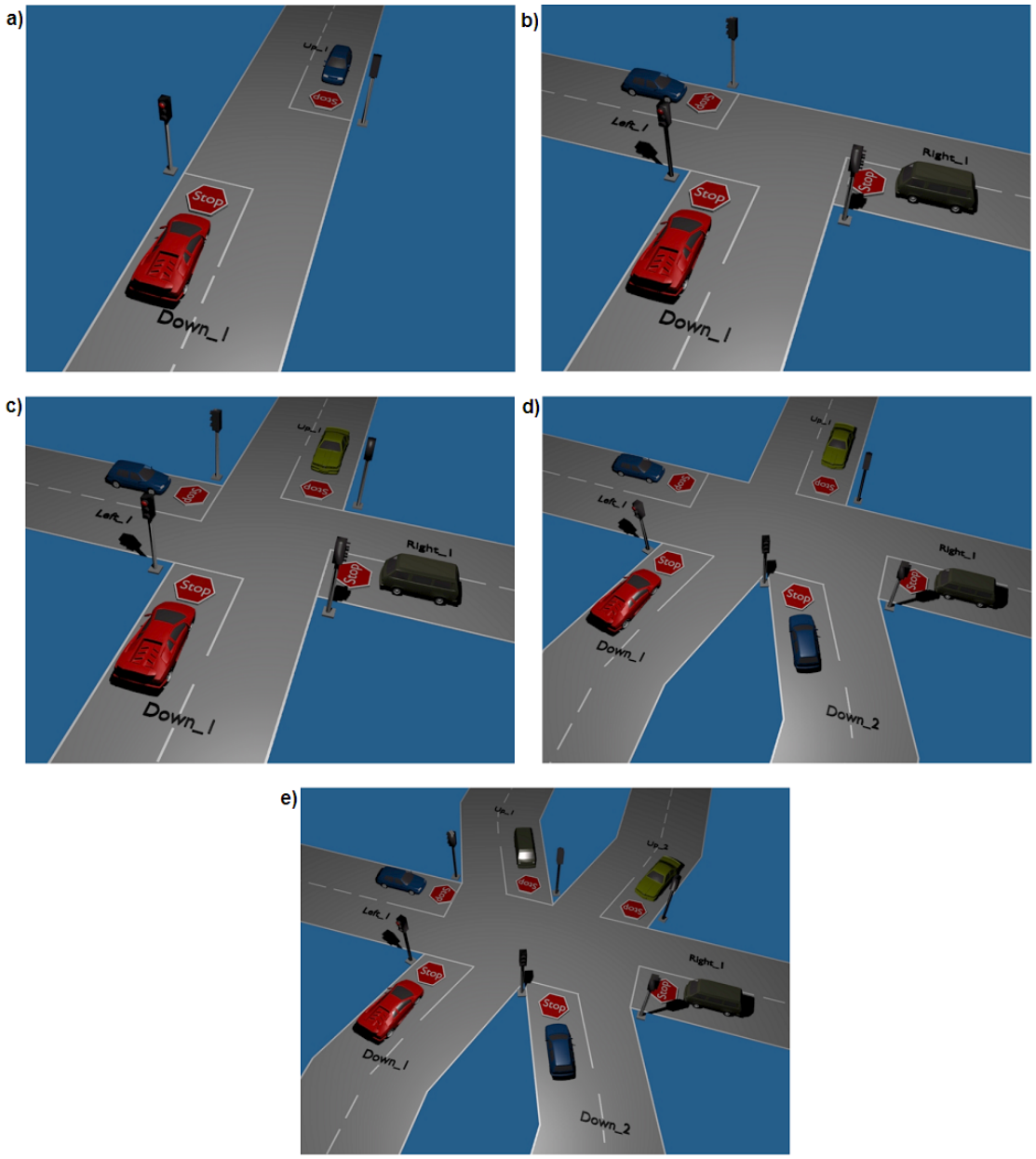


Figure 3.1: Definition of Phase-to-Road mapping: Every junction on the virtual map can be classified on the number of in and outgoing directions. Used types: a) Simple Road, b) T-shaped junctions, c) cross-shaped junctions, d) 5-sided junctions and 6-sided junctions.

3.1.2 Phases for the different junction types

Phases for Simple roads

The setup for simple roads is really basic. There are only two possible phases. In this first approach we do not use any pedestrians. Therefore, we do not need to create phases which would involve pedestrian crossing.

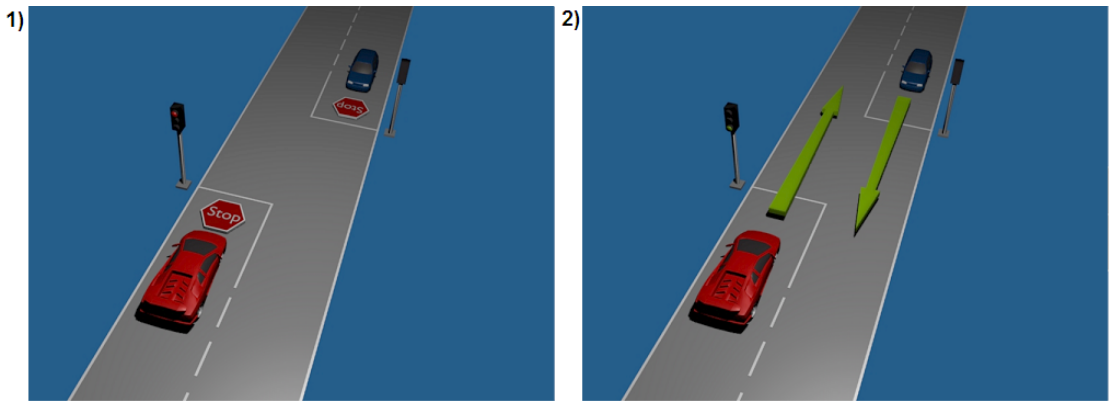


Figure 3.2: Definition of phases for simple roads: only 2) will be used in this simulation, because we have no pedestrian.

Phases for T-shaped junctions

T-shaped junctions need four phases to cover all possibilities for this kind of junction. Because of the better quality of the sensor data no phases which would produce a conflict where chosen.

Furthermore, only phases where chosen, which enable three directions at a time. Figure 3.3 presents the phases for T-shaped junctions.

Phases for cross and special junctions

Figure 3.3 represent the phases for cross-shaped and special junctions. This set is special in that 5-sided and 6-sided junctions also use the phase-set of cross-shaped junctions.

To simplify the phase-sets, the 5-sided junctions mapping (see figure 3.1) “Down_1” and “Down_2” are combined to an overall “Down”. In addition, the 6-sided junction also transform “Up_1” and “Up_2” in “Up”.

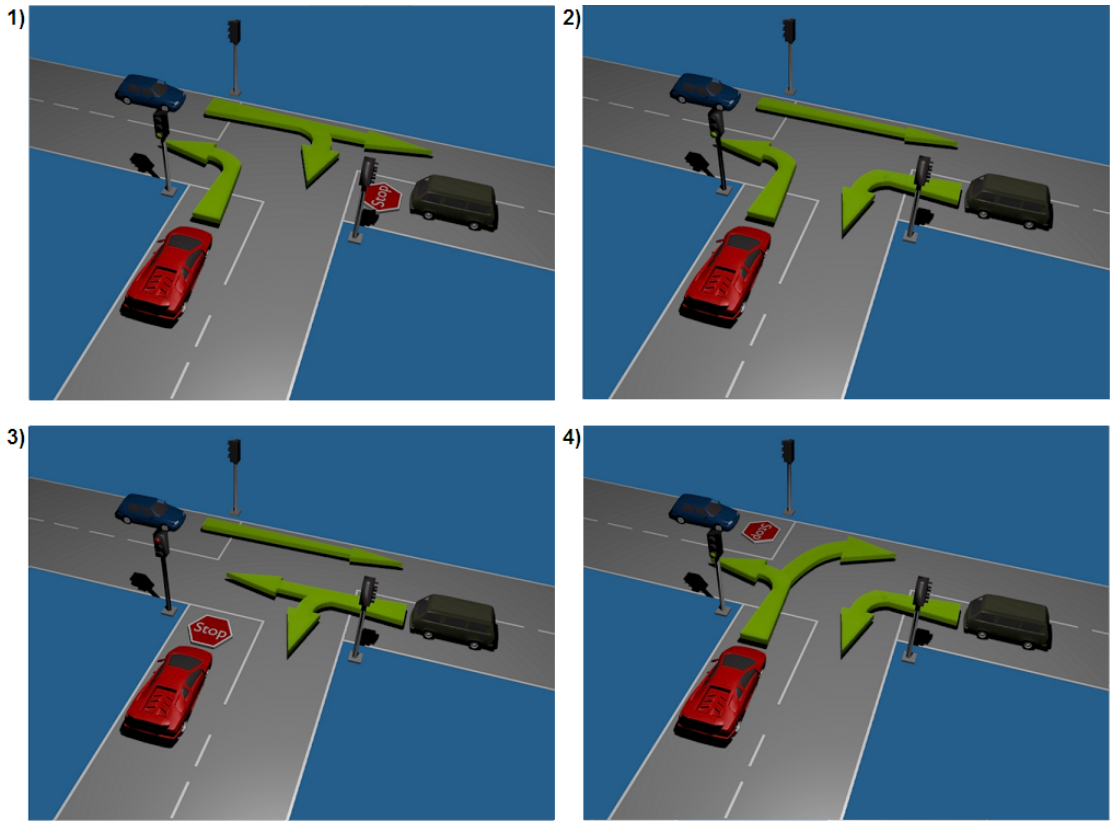


Figure 3.3: Definition of phases for T-shaped junctions.

Similar to the T-shade junctions, only phases where chosen, which enable four directions at a time.

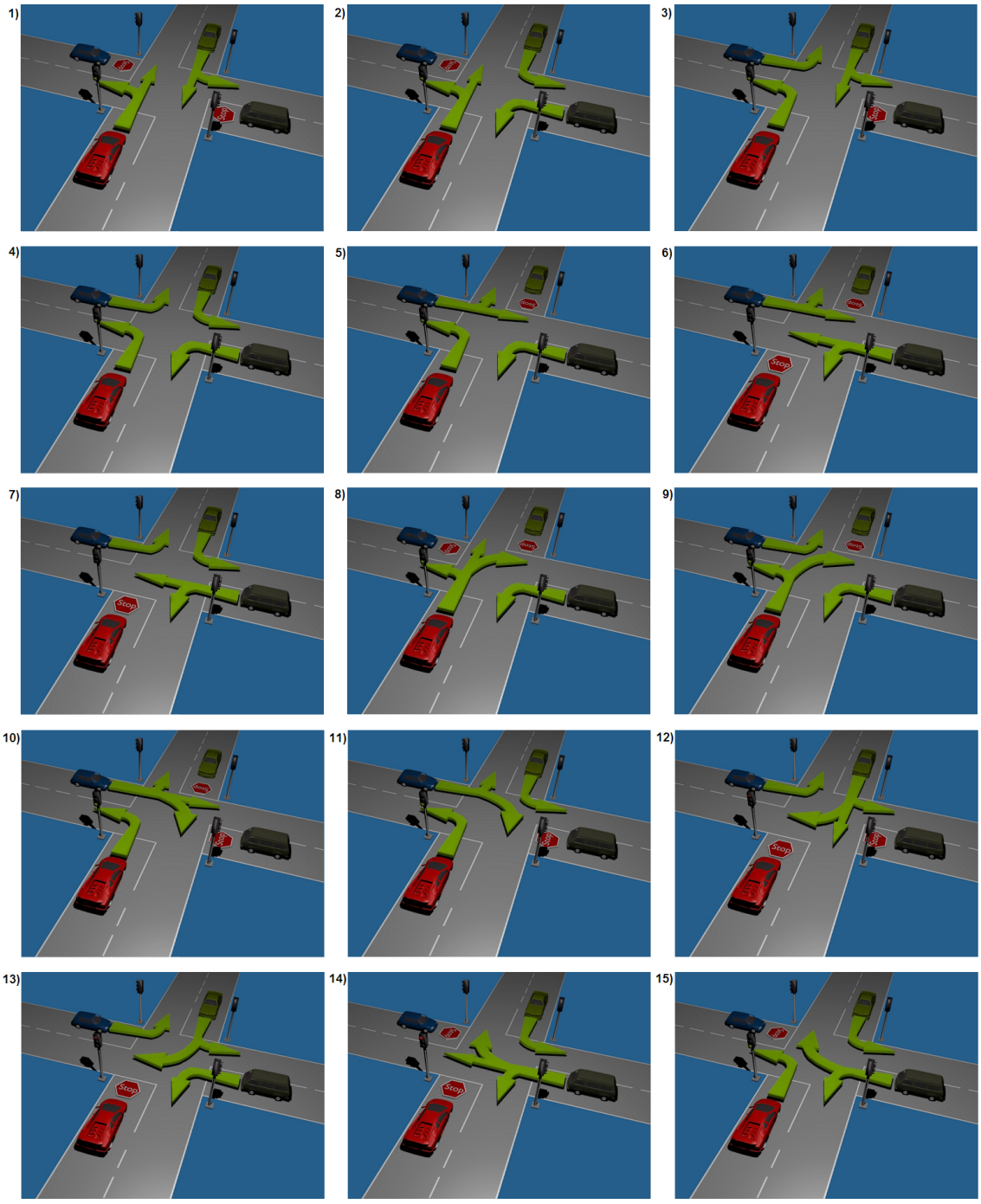


Figure 3.4: Definition of phases for cross-shaped and special junctions.

3.2 Voting-based decision-making algorithms

After defining a set of phases for the different junction types a new approach of decision making has to be created. The goal was, to find a fairer and better way of phase-selection

than system used with the limited sensor data, nowadays. This new method should increase the traffic flow and therefore the overall capacity of urban area roads.

3.2.1 The idea of voting

The idea of voting is very old. Even in old Sparta 400 B. C. some decisions were made by vote [2]. In this approach a car arrives at a junction and can vote which direction should receive a green phase in the next cycle.

3.2.2 Possible vote strategies

There are a lot of different ways to come to a consensus via votes. Table 3.1 provides an overview of commonly used vote system types.

Type of system	Description
Majority vote system	A very basic voting system. The elector votes on his preferred candidate. The candidate with the most votes wins. This kind of system has the benefit, that it only needs one collection step to find a consensus.
Multiple vote system	The voter can vote for more than one candidate. It is possible for a vehicle to vote for a green phase for the right and straight directions, but not for left. A driver could for example arrive at the junction from the south and want to travel to the north west. Then he would maybe not care if he turns right on this junction or just crosses it.
Ranked vote system	Basically, the ranked vote system is similar to the multiple vote system. They only difference is the fact, that in this system the candidates are given a descending order. So the driver in the example above could prefer the green phase for turning right, but in the second step a green phase for straight would be fine as well.
Range vote system	The elector has to vote for all candidates by allocating them a value in the set of the given range (e.g. rating form 1 to 10). The candidate with the highest value wins.
Weighted vote system	The system is based on the idea that there are different kinds of voters. A public bus could have more votes than a simple car, because a bus can transport more passengers. Voting itself is identical to the majority vote system.

Table 3.1: Types of voting systems

3.2.3 Chosen vote strategies

It is important that the used vote strategy does not require complex calculations and can be handled by a low amount of communication effort. Therefore, the simple majority vote and the weighted vote system were chosen. For the majority system every vehicle has one basic vote for each election. In the weighted system buses receive 58.7 basic votes and vehicles 1.4 to test the impact on the phase elections on the roads (see section 5.1.1 for more details).

Furthermore, a process to counteract starvation of vehicle waiting on less frequently used roads and junctions was defined. Each vehicle on a road which was not elected for a green phase was given the double amount of votes in the next election cycle. This bonus is accumulative, so that:

$$n_{+1} = n^2 \tag{3.1}$$

represents the next amount of votes.

In this approach the vote counters are reset, after vehicles are allowed to proceed, but other strategies are imaginable, too. For example, a vehicle, which travels a long journey through an urban area, could collect more and more votes and therefore gain a higher weight in the system.

3.3 VISSIM

VISSIM is a microscopic traffic flow simulation software, developed by the company PTV AG from Germany. PTV AG claims that VISSIM in its fifth iteration counts as the market leader when it comes to traffic simulation [28]. This is up to debate, but it is a fact, that it is well used. Car manufactures like VW use it [38] and it is also used in an scientific environment [14]. The German name is derived from “Verkehr In Städten - SIMulationsmodell” which can be translated as “Traffic in cities - simulation model”.

The Distributed Systems Group (DSG) decided to use this simulator for their test and evaluation purposes in the future. Therefore, this was the simulator of choice for this thesis as well.

3.3.1 General description

Microscopic simulators, like VISSIM, simulate each entity individually. This has the benefit, in contrast to macroscopic simulators, that interactions can take place between every entity in the system. Macroscopic versions work with average representations in form of traffic density or flow instead.

VISSIM has also multi-modal properties, which means that different kinds of traffic can be simulated at the same time. This reaches from standard types like vehicles (e.g. cars and buses) to cyclist or pedestrians (even in buildings).

Also, thanks to its plug-in capabilities, it is possible to use a variety of different signal controllers. Via this add-on functionality, well known controller like SCATS or SCOOT can be used [28]. This feature was also utilized as we generated our traffic light agents. Figure 3.5 displays the VISSIM GUI from version 5.10.-05 during a simulator run with an INP-file.

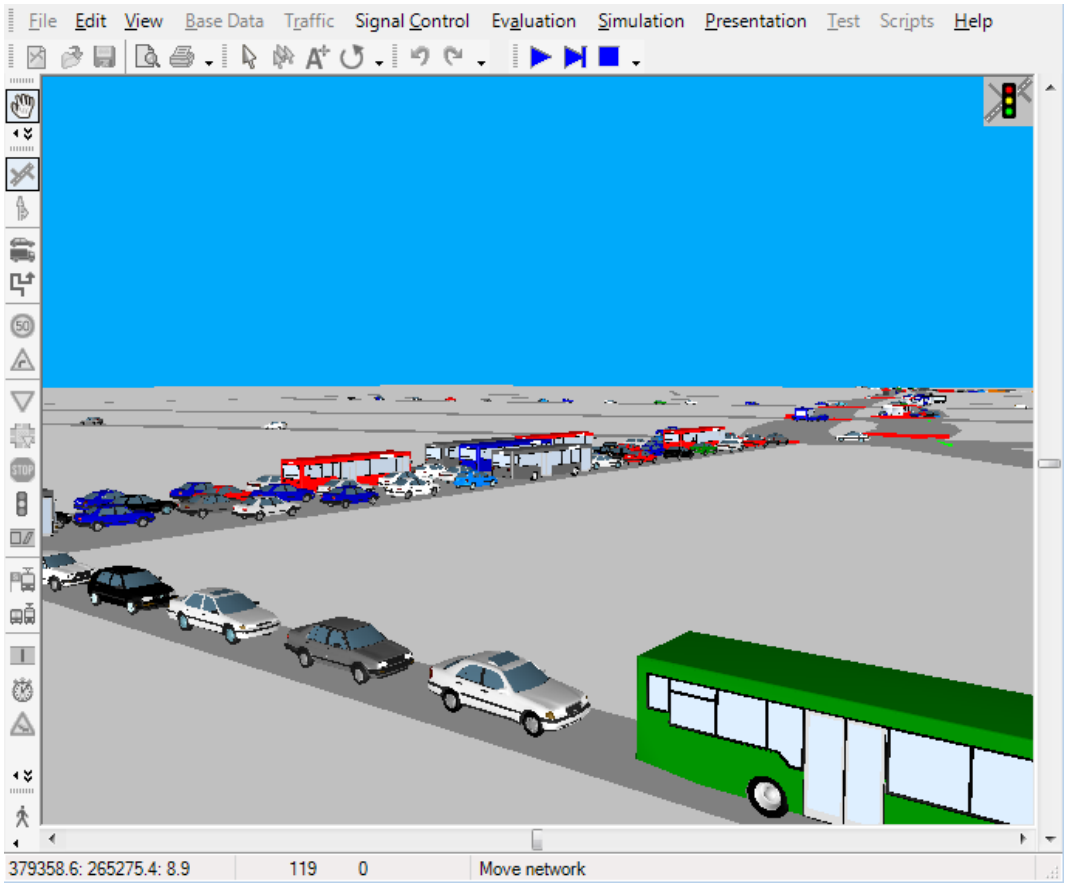


Figure 3.5: The VISSIM GUI.

3.3.2 The INP-file format

The heart of every simulation in VISSIM is the INP-file. In this document all basic information for the map is stored. The file itself is completely text based. This aged style of representing data saves resources, but an object oriented representation would make it much easier to use the data in a different context. Table 3.2 presents the important parts of an INP-file for this thesis. A full description can be found in [28].

INP-file part	Description
Links and connectors	Links are the representation of roads in VISSIM. Every link has only one direction and can have multiple lanes. Therefore, for a bidirectional road 2 links are needed. Connectors join two links together to a road structure and can be seen as a part of junction.
Desired speed decisions	Every type of traffic can have its own set desired speed. However, this part represents the actual maximum allowed speed on the road.
Conflict areas	This part is important for vehicle to recognize each other on junctions without traffic lights. Therefore, each pair of overlapping connectors should have a set of conflict areas, which determinate that vehicle, that is allowed to pass the junction first.
Signal heads	This property defines the actual signal heads on the map. Every signal head is part of a signal group and every signal group has one signal agent, which is responsible for the whole junction.
Parking lots	To generate dynamic traffic a special type of parking lot is essential. See section 3.3.3 for more details.
Nodes	Every junction has an abstract representation as a node. These nodes are also important for dynamic traffic assignment.
Traffic Compositions	Traffic compositions are used to define what kind of traffic will be spawned at a vehicle input. With these compositions it is possible to declare which type of vehicle will be used in which percentage of the overall traffic.

Table 3.2: Parts of the INP-file.

3.3.3 Dynamic traffic assignment

There are two ways to generate traffic in VISSIM. The common way in former version was to create so called “Vehicle inputs” at the start of a link. Every input chose one of the predefined traffic compositions and decided how many vehicles should be spawned from this location in a particular amount of time.

Afterwards, routes had to be defined manually from every vehicle input to a wanted destination. This can become very difficult and error-prone for large maps.

The newer and recommended way for traffic injection is called dynamic traffic assignment. To use this method the map has to have nodes on each junction. These nodes are later used to generate an abstract node-edge-graph of the virtual map. Based on this weighted graph [33] VISSIM uses a shortest path algorithm to find the best possible route from an origin to a destination. Parking lots of the type “Zone connector”, with a relative flow greater 0, are used as origins and parking lots with a relative flow of 0 at the end of the map are used as destinations (see figure 3.6)[28].

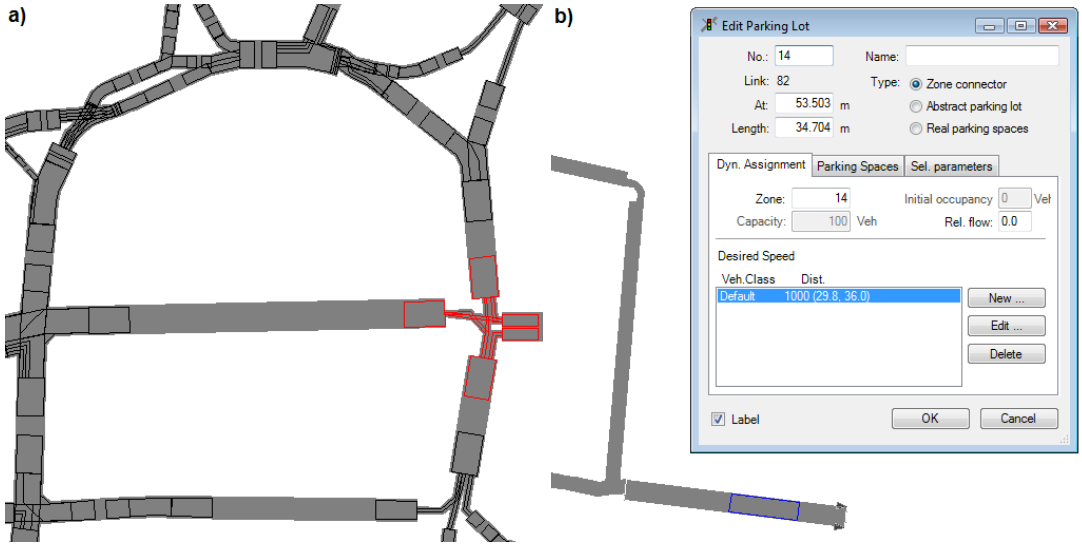
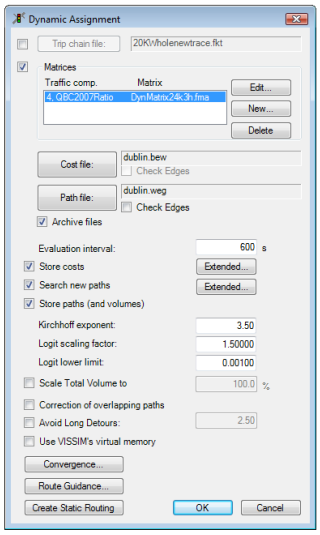


Figure 3.6: Basic Components for dynamic traffic in VISSIM: Figure a) shows the representation of nodes in VISSIM. Picture b) displays a destination parking lot.

Nodes and parking lots represent the basic components for dynamic traffic assignment. To populate the roads with vehicles a dynamic matrix is necessary (see part b) of figure 3.7). This matrix contains for every parking lot a mapping of how many vehicles are starting from it to the destination parking lots in a fixed amount of time.

Which matrix file is used is decided in the “Dynamic Assignment” window (see part a) of figure 3.7). is is also possible to set the logit scaling factor in this menu. VISSIM is using the Logit function to model a discrete choice behavior for the drivers. A high factor means that most (or all) drivers will decide the fastest way to the destination and only some vehicles uses a longer route, to simulate drivers which are not familiar with the part of the city [28].

a)



b)

```
* time interval [hh.mm]
 0.00  3.00
* scaling factor
 1.0
* number of zones:
 17
* zones:
1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17
* number of trips between zones
0   0   0   0   0   0   0   0   833 833 833 833 833 833 833 833 833
0   0   0   0   0   0   0   0   833 833 833 833 833 833 833 833 833
0   0   0   0   0   0   0   0   833 833 833 833 833 833 833 833 833
0   0   0   0   0   0   0   0   833 833 833 833 833 833 833 833 833
0   0   0   0   0   0   0   0   833 833 833 833 833 833 833 833 833
0   0   0   0   0   0   0   0   833 833 833 833 833 833 833 833 833
0   0   0   0   0   0   0   0   833 833 833 833 833 833 833 833 833
0   0   0   0   0   0   0   0   833 833 833 833 833 833 833 833 833
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Figure 3.7: a) VISSIM´s Dynamic Assignment menu. b) a dynamic matrix with 60,000 vehicles which will be injected in the system during a timespan of 3 hours.

A second option to dynamic matrices, so called trip chain files, can be used. These files contain on a line by line basis driving instructions for vehicles. Each line starts with the identifier of the vehicle, followed by its type and its zone of origin. the next 4 semicolon separated numbers represent a trip of the vehicle. Therefore, a vehicle can drive to more than one zone before it will be deleted by VISSIM. Listing 3.1 provides an example for a trip chain file.

Listing 3.1: Example of a trip chain file

```
1.1
1; 100; 44;  0; 171;  0; 0;  0; 106;  0; 0;  0; 66;  0; 0;  0; 145;  0; 0;
2; 100; 151;  0; 150;  0; 0;  0; 157;  0; 0;  0; 209;  0; 0;  0; 83;  0; 0;  0; 225;  0; 0;
3; 300; 44;  0; 10;  0; 0;  0; 171;  0; 0;  0; 106;  0; 0;  0; 66;  0; 0;  0; 62;  0; 0;
4; 100; 184;  0; 181;  0; 0;
5; 100; 243;  0; 239;  0; 0;  0; 180;  0; 0;  0; 25;  0; 0;  0; 20;  0; 0;  0; 182;  0; 0;
6; 300; 247; 0; 177;  0; 0;  0; 232;  0; 0;
7; 100; 184;  0; 181;  0; 0;  0; 187;  0; 0;  0; 14;  0; 0;  0; 235;  0; 0;  0; 55;  0; 0;
8; 100; 128;  0; 80;  0; 0;  0; 68;  0; 0;  0; 76;  0; 0;  0; 74;  0; 0;
9; 300; 197;  0; 143;  0; 0;  0; 134;  0; 0;  0; 115;  0; 0;  0; 218;  0; 0;  0; 114;  0; 0;
10; 100; 110;  0; 94;  0; 0;  0; 221;  0; 0;  0; 229;  0; 0;  0; 6;  0; 0;  0; 149;  0; 0;
```

```
11; 100; 243;  1; 239;  0; 0;  0; 180;  0; 0;  0; 183;  0; 0;  0; 22;  0; 0;  0; 24;  0; 0;
12; 300; 247; 1; 177;  0; 0;  0; 232;  0; 0;
13; 100; 244;  1; 239;  0; 0;  0; 180;  0; 0;  0; 183;  0; 0;  0; 22;  0; 0;  0; 24;  0; 0;
14; 100; 245;  1; 120;  0; 0;  0; 119;  0; 0;  0; 250;  0; 0;  0; 248;  0; 0;  0; 177;  0; 0;
14; 100; 247;  1; 120;  0; 0;  0; 119;  0; 0;  0; 250;  0; 0;  0; 248;  0; 0;  0; 177;  0; 0;
16; 100; 197;  1; 143;  0; 0;  0; 134;  0; 0;  0; 115;  0; 0;  0; 218;  0; 0;  0; 114;  0; 0;
17; 100; 128;  1; 80;  0; 0;  0; 68;  0; 0;  0; 95;  0; 0;  0; 67;  0; 0;  0; 145;  0; 0;  0;
18; 300; 244;  1; 120;  0; 0;  0; 119;  0; 0;  0; 250;  0; 0;  0; 245;  0; 0;  0; 120;  0; 0;
19; 100; 128;  1; 80;  0; 0;  0; 68;  0; 0;  0; 76;  0; 0;  0; 74;  0; 0;
20; 100; 197;  1; 143;  0; 0;  0; 138;  0; 0;  0; 142;  0; 0;  0; 249;  0; 0;  0; 248;  0; 0;
```

3.3.4 COM interface and external plug-ins

The COM interface provides objects and methods to read and (sometimes) write data back to VISSIM [27]. Thanks to this interface it is able to get the necessary vehicle information out of VISSIM. It is also possible to use DLL-files to replace the internal traffic light controller with an external version. Without this functionality it would not be possible to expand VISSIM in a fashion this project needed.

3.4 Work flow for the extended VISSIM simulation

Figure 3.8 illustrates the work flow of a Democratic Traffic Light (DTL) simulation with VISSIM.

There are two ways to execute a DTL simulation. Path 1 was the initial designed work flow for the simulation, but due to complications during the evaluation of the system a new work flow (2) had to be created. For completeness reasons both ways are presented. Section 4.6 will explain what exactly forced the design of a second work flow.

Start This part of the work flow only has to be executed when a new part of the DSG-map has to be converted into the VISSIM-format. Only the map generator for the in house simulator has the capabilities to choose a map section from the overall map of Dublin.

3.4.1 Work flow path 1

1a This step can be used to generate trace files of vehicles to innervate the streets of the virtual map.

1b + c In this step of the work flow the in house map and the trace files are converted into a VISSIM readable format (see section 3.3.2 and 3.3.3). Therefore, appropriate converters had to be developed (see section 4.3).

1d Before a simulation with a new map and the DSG VISSIM extension is full functional the map and the trace files have to run in VISSIM without external interference via the COM interface. In this step VISSIM calculates the new pathways for the map and the vehicles on it.

1e Before running a simulation via DSG VISSIM extension the configuration file has to be edited, so the new map and the parameters for the traffic light agents can suit the test run.

1f In the final step, the simulation runs with all the parameters from the configuration file.

3.4.2 Work flow path 2

2a Basically, the same step as in 1b). The main difference is, that no zone connectors are generated automatically.

2b Before a simulation with a new map and the DSG VISSIM extension is full functional, origin and destination zones have to be defined and also traffic has to be created via a dynamic matrix file.

2c VISSIM needs a short test run to calculate, based on the defined zones, the possible routes throughout the map.

2d As in work flow part 1e) the configuration file of the DSG VISSIM extension has to be edited, so the new map and the parameters for the traffic light agents can suit the test run.

2e In this final step, the simulation runs with all the parameters from the configuration file.

2f VISSIM saves all vehicle records during the simulation in a text file. Based on this information we can evaluate our agents.

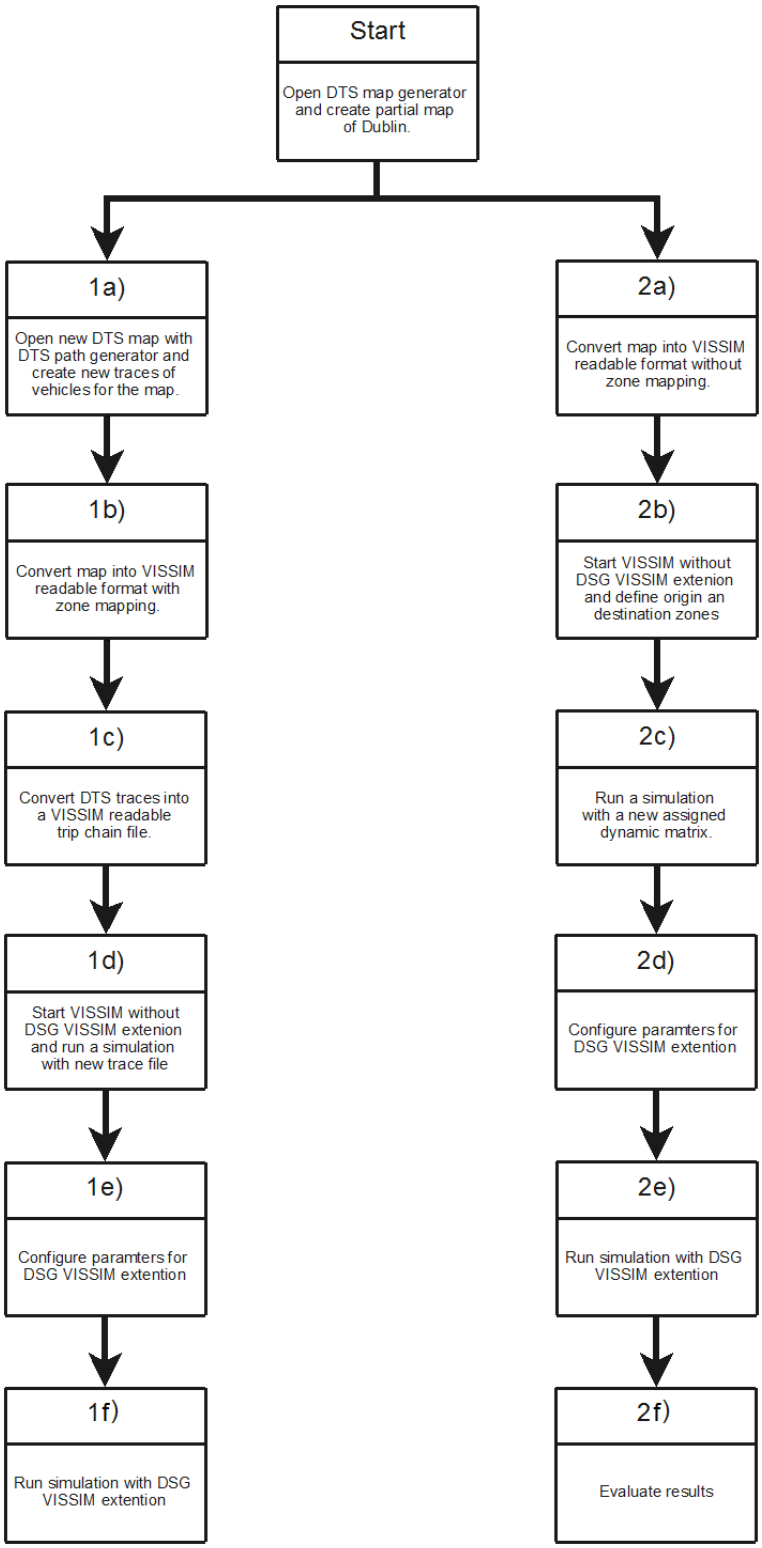


Figure 3.8: Overview of the work flow of the DTL simulation.

CHAPTER 4

Design and Implementation

After illustrating the used materials and developed methods this chapter will provide an overview of the implementation of the necessary tool to execute VISSIM with voting-based traffic light agents.

4.1 Development environment

The development of the software was done with with IBM-PC. The PC was equipped with a Intel®Core™2 Duo CPU with 4 GB DDR3 Ram. Windows Vista Professional 32-Bit was used as OS. For the implementation of the converter (see section 4.3) Visual C++, with the Xerces-C++ XML parser[11] in addition, was used. For the agent framework (see section 4.5) the .Net language C# was chosen. As planing aid the UML-Design program DIA in version 0.96.1 was utilized.

4.2 The DSG in house simulator

The DSG uses an in house simulator to observe different traffic behavior in the city off Dublin called Dublin Traffic Simulator (DTS). However, stated in section 3.3 the DSG decided to change to a external simulator for their studies, which has the benefit that an external company has more manpower to develop such a sophisticated tool.

To provide a smooth transition one part of the work of this thesis was to evaluate and test possible solutions to reuse as much functionality from the in house simulator in VISSIM.

This section will provide an overview over the DTS and its data formats to allow a better understanding of the challenges involved in converting the old format to VISSIM readable design.

4.2.1 The DTS map format

The DTS uses an XML-based representation of the inner area of Dublin (see figure 4.1). This document represents the necessary data, like the coordinates of the junctions or which junction is connected to another one, in an object oriented format [13].

Figure 4.1: DTS map represented with in house map generator.

Junctions are also the main objects in this format. Table 4.1 provides an overview of the most important elements of a junction in the XML-based map.

Element	Description
id	The global unique identifier of the junction in the system.
type	What kind of junction the object represents. The type defines if the junction has traffic lights or not.
location	The position of the junction on the world map.
incomingJunction	Every junction which is a direct neighbor of this junction and also has a street, where vehicles are allowed to travel towards this junction has one incomingJunction-element. A very important sub element is the outgoingJunctionRef-element. It describes which choices a driver has when approaching the junction from the other junction (left, right or straight).

Table 4.1: Important elements of a junction record in the DTS map format.

For a detailed explanation of the structure of that map see Appendix A.

4.2.2 The DTS phase generator

The DTS uses its own set of phases for traffic lights, which are generated by a python based parser. The parser takes a DTS map as input and decides on a junction type basis which phases could be possible. The phase generator uses the incomingJunction-elements, especially the outgoingJunctionRef-sub elements to define a set of phases for this particular junction [13].

This phase information is also stored in a XML-based format.

4.2.3 The DTS vehicle path

The DTS path generator offers a variety of possibilities to generate traces for cars through a simulated map. Basically, you choose a start and a destination junction. Afterwards, a shortest path algorithm [20] is applied to find the best route between these two junctions. This approach has the drawback, that no vehicle is choosing a route which uses side roads [13]. Listing 4.3 provides an example for an input file for the path generator.

Listing 4.1: Example of a DSG input file for path creation

```
//Time of simulation
6000000
//IdSourceJunction,IdDestination#NbJourneys
3073,1702#135,C
3073,1690#111,C
3073,1545#10,C
3073,1541#1,B
3073,1489#20,B
3073,1444#5,C
```

Listing 4.2 shows a section of a trace file after the path generator applied its shortest path algorithm.

The output file is formated on the following terms [13]:

departure time 1 1 source junction id ... following junction ... destination junction

Listing 4.2: Example of a DSG output trace file from the Path Generator

```
103,1,1,C,14,1517,744,1521,760,758,1273,7657,766,1521,1531,577,1532,1556,104
135,1,1C,13,1517,1518,1519,802,1522,761,757,1521,1531,866,1530,801,810
150,1,1,C,11,1517,744,1521,760,758,1273,1272,1659,583,837,8367
510,1,1,C,4,1517,744,1521,760,758,1273,1272,1659,583,837,8367
648,1,1,C,8,1560,795,1561,1551,1536,1537,1550,1549
666,1,1,C,2,1560,795,1561,1551,1536,1537,1550,1549
814,1,1,C,14,1549,1550,1551,1552,713,1560,741,1559,1558,1557,549,550,1556,104
882,1,1,B,7,8367,837,1656,1275,827,828,829
904,1,1,B,16,104,1556,1356,578,1268,5767,1269,1273,1272,1271,1266,1361,1360,1670,1359,1225
954,1,1,B,53,774,773,1506,1630,607,1520,1519,802,1522,865,1530,866,1531,577,1532,1556,1356,578,1268,1532
1020,1,1,B,68,1560,795,1561,1551,1550,1537,1536,1535,1529,1635,607,1520,1519,802,1522,1608,804,769,1635
1050,1,1,C,68,1560,795,1561,1551,1550,1537,1536,1535,1529,1635,607,1520,1519,802,1522,1608,804,769,1635
1064,1,1,C,68,1560,795,1561,1551,1550,1537,1536,1535,1529,1635,607,1520,1519,802,1522,1608,804,769,1635
```

4.3 DSG Converter tools

To reuse the old functionality from the DTS in VISSIM two converter tools had to be created. This section provides a brief description of them.

4.3.1 Map converter

The DSG had started to create a basic map converter, to transfer the DTS XML-format into the INP-file format [36]. In the given version the converter was not able to provide all the necessary features. Therefore, the converter, which was written in C++, was expanded.

The converter is a Windows console based application which takes a DTS map and its XML-based phase file as input and generates an INP-file and an extended version of the DTS map as output. The extended DTS map provides the identifier of the VISSIM representations of the junction data from the DTS map. With these additional information external programs like the DSG VISSIM extension can generate additional functionality like the WLAN module (see section 4.5.1).

The basic design concept of the converter makes it hard to expand it with additional functionality, so maybe for future projects it would be better to redesign it and implement it in a more modern language.

4.3.2 Trace file converter

The trace file converter takes an DTS trace file as input and generates a VISSIM readable trip chain file as output (see section 3.3.3). The DTS is focused on junctions. As explained in section 3.3.2 VISSIM in contrast operates on bases of links (street) and its connectors and does not really have an object for a junction like the DTS. The biggest challenge was to find a suitable counterpart of junctions in VISSIM to convert the old junction-based traces in VISSIM´s trip chains.

Trip chains work with a source and a destination zone. To map the DTS junction with this system every link in VISSIM was given its own zone connector parking lot. Figure 4.2 provides an example of the idea behind the conversion. A trip in DTS from junction a) to c) would create a trace with all three junctions in it. The converted file would set the origin zone of the train chip file between junction a and b) and the destination zone would be the zone between b) and c).

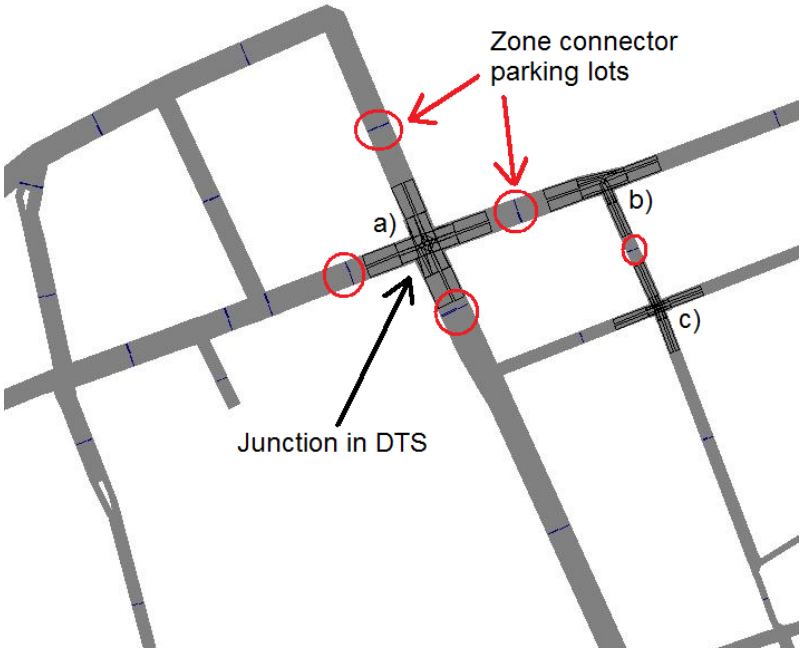


Figure 4.2: Converting DTS trace files to VISSIM trip chain files.

4.4 DSGVissimEvaluationFileReader

For evaluation purposes a simple text reader was developed. VISSIM is able to dump so called vehicle record data in a text file with the format "[INP-file name].fzp". What exactly and in which order is saved in this file is defined by the associated "[INP-file name].fzk" file. To use the evaluation file reader the fzp-file has to have the following format:

Listing 4.3: Format for the fzp-file

```
FZNR
TYPNR
SIMZEIT
VEH_INPSEC
V
TGES
TACHOX
```

The file has to be stored in the same directory as the used INP-file. For a detailed explanation see appendix B.

4.5 DSG VISSIM extension

This section describes the design and implementation of the DSG VISSIM extension (DSGVE).

4.5.1 Design

Figure 4.3 illustrates the objects of the DSGVE. The central object is called **DSG-Simulation**. Any simulation which uses the COM interface of VISSIM (see section 3.3.4) has to be started via an external program. The **DSG-Simulation**-object represents the main class which serves this purpose.

The **Global clock**-object is the pacemaker for the whole simulation. It controls when the next internal simulation-second is calculated. There can be problems with interleaving processes when using the VISSIM COM interface. Therefore, it was necessary to to take the clock control away from VISSIM and put it in the hand of the **DSG-Simulation**-object.

VISSIM wrapper and the **Vissim COM interface** provides all the methods and interfaces which are important for collecting and writing data from and to VISSIM. For example, the wrapper-object is responsible to remotely start the VISSIM program and loading the simulation map afterwards. The COM Interface provides current data of the object inside of VISSIM during every ongoing simulation step.

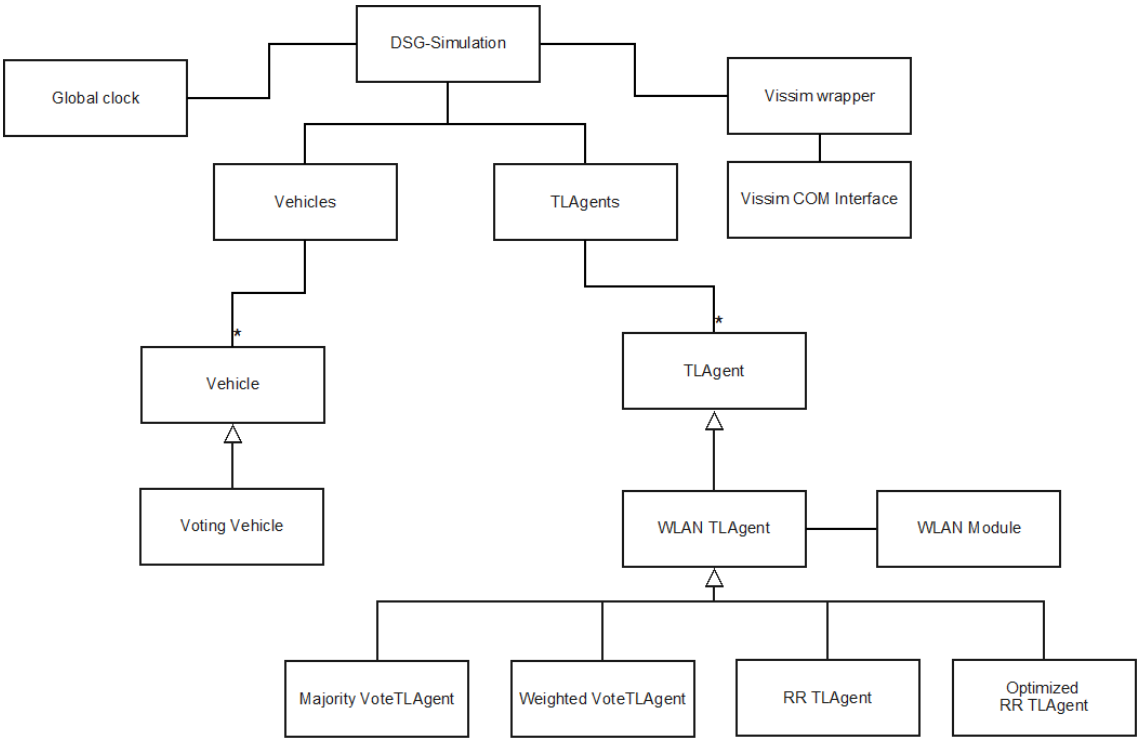


Figure 4.3: UML-Diagram of the DSGVE.

The **Vehicles** and **TLAgents** objects are containers for the vehicles and traffic light agents in the system.

The **Vehicle**-object gathers all the basic data of a vehicle instance:

- ***VissimType*** There are different kinds of types defined in the INP-file (see section 3.3.2). These data can be important for different reasons (e.g. maximum passengers or basic votes of the entity).
- ***VissimVehicleID*** Every vehicle in a VISSIM simulation has a unique identifier. This identifier is stored in the corresponding instance of the **Vehicle**-object.
- ***VissimNextZone*** One benefit of wireless communication between vehicles and infrastructure is a more detailed knowledge base of the traffic on the roads (see chapter 2.2). This information represents the next destination of the car (e.g. the next street on a junction a car wants to reach).
- ***VissimCurrenLinkID*** As the VissimNextZone element represents a destination on a trip, the VissimCurrenLinkID element symbolizes the actual position of the

vehicle in the simulation (e.g. the actual street).

- ***VissimCarCoordinates*** This is a more detailed position of the vehicle and can be seen as the representation of GPS-coordinates of the vehicle.

These information can be inherited from more specialized vehicle-objects. For example, the **Voting Vehicle** class has, besides the basic methods and data, additional data like the basic votes this instance of vehicle has and the actual vote counter (see 3.2).

The **TLAgent**-object gathers the following basic data of a traffic light agent instance:

- ***TlAgentId*** Every object in VISSIM has a unique identifier. This is the identifier for the traffic light agent.
- ***JunctionAncientId*** The identifier of the junction from the original map (see section 4.2.2).
- ***TLJucntionCoordinates*** The coordinates of the junction on the virtual map. This data can be important for location purpose.For example, the more sophisticated **WLAN TLAgent** eses this information for checking if a vehicle is in WLAN range.
- ***DSGSignalGroups*** The Signal groups which are used from the agent.
- ***DSGCarIds*** a container for cars or other vehicles which are interacting with this traffic light agent.
- ***CycleTime*** Every set of traffic lights from one junction can have its own set of phase time (in seconds).
- ***AmberTime*** The amount of time for the amber phase. In many countries around the worlds this would be usually 3 or 5 seconds.
- ***DSGLanePosition*** This class is responsible for the mapping of the actual junction layout to the standardized set of junction used for the phase decision algorithms (see section 3.1.1).

Following the same principles as the Voting Vehicle-class, the **WLAN TLAgent**-object inherits all these basic data from the TLAgent-object. Furthermore, this object also owns an instance of the **WLAN Module**-object, which is responsible for the management of all WLAN functionality like communication and scope examination.

However, due to the time constrains of this thesis, the module checks at the moment only if a vehicle is in the defined WLAN area and saves the information for the traffic light agent. During the phase decision process, the WLAN agent collects all votes from the vehicles in the area. Thereby, no real communication is done via virtual communication channels, so no packet drops or other interesting aspects of WLAN communication, like effects of a high amount of vehicles in an area can be evaluated at the moment.

Finally, the **Vote TLAgent** inherits all objects from the WLAN TLAgent and extends it with the necessary variables and methods for the voting-based algorithms.

4.5.2 Calculating the next phase of an agent

Voting-based agents

After all votes are collected from the vehicles and the voting-based agent decided, based on the defined voting strategy (see section 3.2.3), which phase will be next, this decision has to be converted in a VISSIM readable expression. Therefore, every traffic light agent has a container of its VISSIM internal traffic light group identifier for the mapping of the junction to the standardized junctions (see section 3.1). Based on this mapping a string is created which has the following format:

[Cycle time] [Amber time] : [Traffic light group ID] ... [Traffic light group ID]
This string is copied into the corresponding traffic light agent file.

After the calculation for every traffic light junction is finished VISSIM is allowed to run for the next cycle time via the **Global clock**-object. The plug-in DLL, which is responsible for setting the traffic lights on the map, will read the new data from the files and act accordingly.

Round Robin agents

This procedure is similar for the two Round Robin agents, except, that tehy do not have to calculate the votes of the vehicles. The RR agents just store which phase was used in the last round and choose the following phase number from the phase-set of the junction. Afterwards this phase is translated and copied in the proper file just id is done by the voting-based agent.

4.5.3 Security Considerations

The study in this master thesis is completely simulator based, but when this approach of voting-based traffic light agents would be implemented, then there would be different areas of security relevant aspects to take in considerations. This section tries to identify the most important ones and offers a solution for them.

General functionality

As explained before. the basic idea behind the thesis is, that every car approaching a traffic lights votes for a green phase and that the phase set with the most votes is chosen for a junction for one cycle. Therefore, the basic areas for security risk are:

- The communication and authentication between the entities
- Fraud detection
- Privacy issues
- Denial of Service

Communication and authentication issues

There are two possible communication pathways:

Traffic light agent to Management-station or other agents For statistic reasons the local traffic light agents could send the phases they decided for green time back to a centralized management station. We could also think of a voting strategy, which involves surrounding neighbor traffic light junctions. In both ways, the communication channel should be secured.

The advantage of this communication pathway is, that these connections are usually the same, stable and with the same entities. To prevent a tampering with these connections we can apply usual methods for a safe channel. So for example IPSEC or SSL-tunnels could be established between the communication partners.

The main challenge would be, to set up a good PKI-structure which is able to provide new keys periodically. SSL would also take care of the integrity of the message. Overall a reliable transport method (like TCP) between the entities of the communication would be recommended. With a common setup between these entities it should be possible to prevent passive (e.g. eavesdropping)and active attacks (e.g. repeat attacks). Overall,

these security issues are more basic and there are good patterns to make such kind of system save enough [18].

Vehicle to traffic light agent communication
Securing these kind of communication is much harder than between well known entities. The location and the identifier of traffic light agents is known to the system and therefore it is more or less easy to find ways to make that part of the system secure, but vehicles which vote for a green phase are unknown to that system and the communication partners change all the time.

It must be ensured that the entity which votes for a green phase is really a vehicle on that road and is therefore allowed to vote. The focus in this scenario is not that the information which is sent from the vehicle to the traffic light has to be a secret (it should be clear to everybody that the vehicle always votes for a green light), but that there must be a way to ensure the authentication of it.

One basic concern is, that a security system can only be safe when it is able to update the system (e.g. new private keys for a vehicle). There exist a lot of vehicle manufacturers and all should be able to talk with the same security standard to the traffic light agents. These issues are not really solved today, so we have to see how the manufacturers will implement mechanisms for safe V2C or V2I communication. Maybe a possible solution for the case of the vehicle to traffic lights communication could be that there are 3 different public keys for each manufacturer.

All manufacturers issue 3 keys from a CA where the traffic light agents have access to them. There should also be a black list for the bad keys. So periodically (e.g. once a day) the a management stations could collect these public keys and distribute them to the traffic light agents. So when a vehicle wants to vote the traffic light it sends its manufacturer identifier to the traffic light. The light sends back a challenge encrypted with the public key of that manufacturer. The vehicle has the private key of that manufacturer and can respond to that challenge and can now vote for a green phase. The private keys have to be changed from time to time, so there must be a secure way to send vehicles the new manufacturer keys.

Fraud detection

It must be sure, that only vehicles can vote (see authentication of the cars) and each car can only vote as often as it is allowed for the entity. It could be a way to solve that

problem to have a device in the vehicle which stores the keys and also have a global unique identifier for the vehicle in it (like MAC addresses of network cards).

After the vehicle has authenticated itself by the traffic light agent it could also send a public key to the vehicle. The vehicle could encrypt its answer with the public key of the traffic light agent and encapsulate in its Globally Unique Identifier (GUID). So the traffic light has its unique ID for every vote of a vehicle. This can only work when every GUID is safe in the device and nobody can tamper with it.

Privacy issues

With this system it could be easy to generate a pattern for every vehicle and therefore for every owner of it, so there should be privacy policies in place which clearly state how these data are used. A good way could be to delete all logs periodically and that there would be a third party which ensures, that these data are deleted or made anonymous.

Denial of Service

An attacker could use a mobile device to send vote requests to the traffic light agents. In a scenario where we use PKI-systems it could be easy to generate a lot of requests which have to be checked.

To prevent such attacks could be very difficult, but maybe when such an attack happens a management station could use patterns from past days to set the behavior of the traffic lights until the attack is over. Furthermore the management-stations could send an alarm that such an attack happened to an administrator which could ask the Garda to use CCTV to search the area for suspicious persons.

Conclusion of the Security Considerations

There are a lot of open issues when it comes to safe V2V and V2I communication. Manufacturers have to define a standard way for these communications before a system like the democratic traffic lights could be implemented.

Furthermore, the performance of these secure protocols must be good enough to handle a lot of vehicles at the same time. Maybe such systems can use elliptic curve cryptography to reduce the length of the secret keys to improve the overall performance of the system. Also privacy must have a high priority or such system would not have the consent of the community.

4.6 Complications during implementation and test runs

As stated before in section 3.4 there were complications during the development of the DSG VISSIM extension and the converted elements from the DTS data. This section will provide an overview of the problems that occurred and in which way they where handled.

4.6.1 Problems with converted DTS maps

The old DTS map format is only capable to draw direct streets between two junctions. Therefore, if the road has curves it is simulated by small "junctions" between the two real junctions. This causes major trouble after the conversation in VISSIM.

VISSIM simulated sensors like inductive loops with queue counters at the end of each link. These counters are only able to count vehicles on their links. Picture a) from Figure 4.4 presents a problematic case in the map used for simulations. After the conversion of this part of O'Connell Street there are two unnecessary junctions and thereby, two additional queue counters as well. In VISSIM it would be possible to bend links with intermediate points. Picture b) presents a good solution for this part of the street. Only one link and queue counter could count the complete amount of vehicles.

Due to the time constrains of the thesis there is no solution for this problem at the moment. Maybe an algorithm could be specified which collects all the intermediate queue counters between two junctions with traffic lights. This algorithm would have to visit every neighbor junctions and its followers until a next junctions with traffic lights is found. In the eyes of the author this seems like an NP-Complete problem [12]. Therefore, it could be a better solution to break with the old DST map and generate a completely new map of Dublin in VISSIM. Afterwards the INP-file could be parsed in an object oriented format to use with further extension.

The agents used in this thesis do not use queue counters, but the framework is supposed to be used with other projects in the DSG. Therefore, this issues should be solved for a better overall quality of the map.

4.6.2 Use of DTS trace files

As mentioned in section 4.6 there are problems with using the the converted trace files (see section 4.3.2) from the DTS. To use the converter DTS trace files every link needed a zone connector parking lot. These parking lots are virtual and have a unlimited capacity.

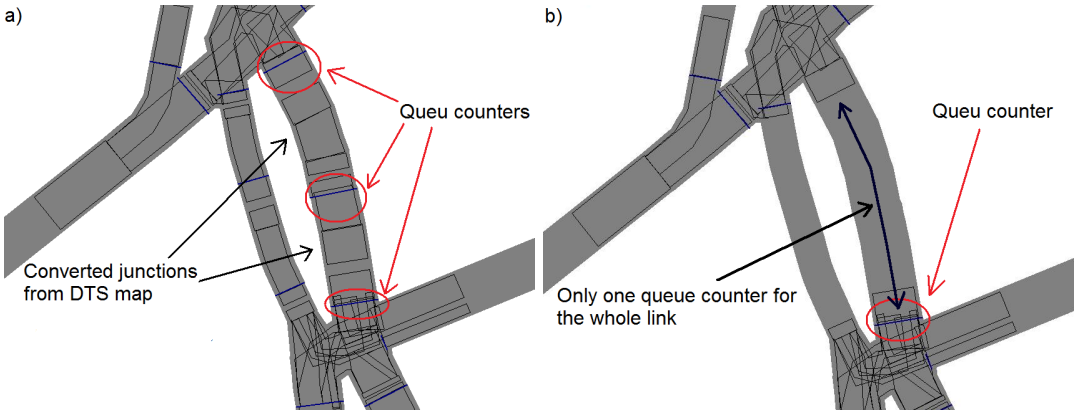


Figure 4.4: Problems with converted DTS maps: a) Every micro link has its own queue counter - b) one queue counter could save resources and would be enough for the link.

During the evaluation process it stand out that vehicles approaching the next destination parking lot on their trip disappear in these parking lots when the link after the parking lot is full with vehicles (see figure 4.5). The vehicle would reappear when after the parking lot is a free space, but only as many vehicles as there are spaces. The rest of the vehicles would still be in this virtual space outside the map and would not be counted for evaluation purposes. Additionally, because these vehicles disappear a tailback is avoided which also distorts the evaluation.

For these reasons the DTS trace files are not used in this thesis and should be avoided for further project with this framework. Instead, dynamic matrices are used with manually placed zone connector parking lots, which is also the recommended best practice for traffic generation in VISSIM [28].

4.6.3 Problems with the VISSIM COM interface

There evolved two major problems during the work with VISSIM´s COM interface.

Bug with queue counters during simulation

After 7200 seconds in a simulation every queue counter only returns “0” as the number of vehicles on its link, when asked via the COM interface. PTV AG, the developer of VISSIM is informed, but did not respond at the moment.

Another member of DGS (Niall O’Hara - niohara@tcd.ie) is searching for a work around

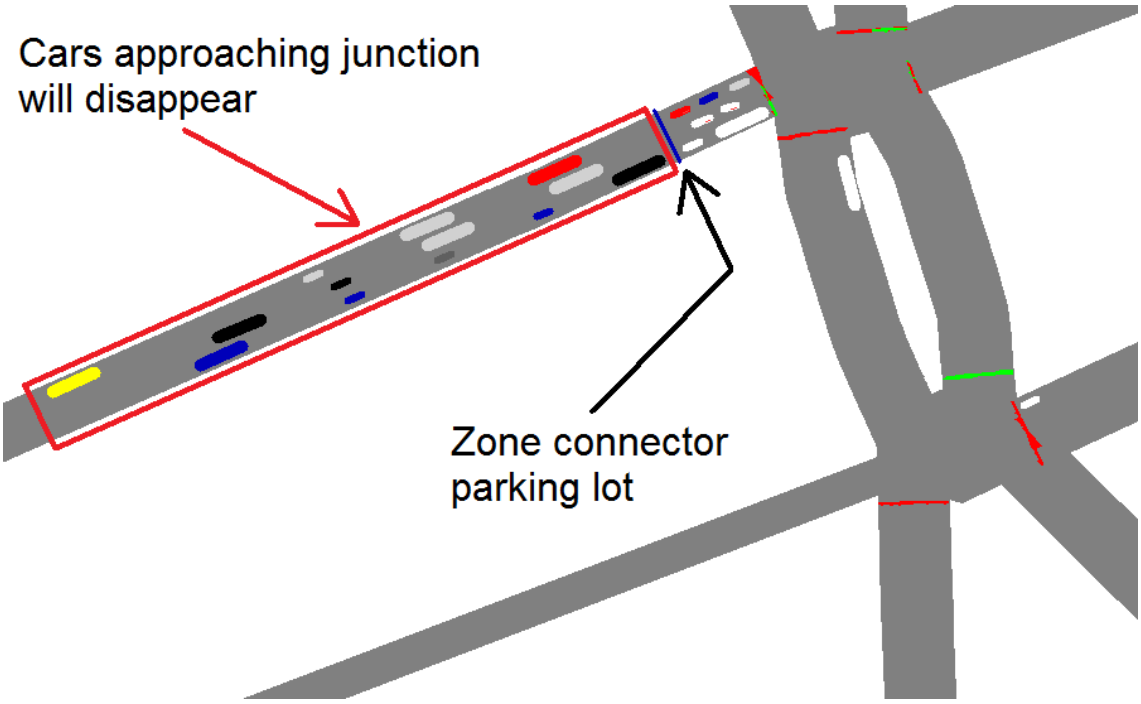


Figure 4.5: Problems with DTS trace files in VISSIM.

at the moment. Again, this thesis does not use the queue counters, but there is also a Collaborative Reinforcement Learning (CRL) [30] agent in development which uses this information.

Missing information about vehicles desired direction

The map used does not have a zone connector on each link anymore (see section 4.6.2). Therefore, it is not possible to ask vehicles for the next destination zone, to figure out the next link on their journey. This was used to calculate the "From-to"-reference (see section 3.1). Another solution would be to ask the vehicle what the next desired location (link) is on his or her trip. This attribute exists and can be displayed in VISSIM during a simulation, but the COM interface is not able to provide this information, yet. The developers are informed about this problem and they will fix this issue with one of the upcoming service packs.

Due to these complications it is not possible to use a "From-to" relationship. Only a "From"-reference is used instead, which is more like the commonly used inductive loop sensors, with the benefit that the defined WLAN range with 75 meters is much higher, than the usual sensor range of inductive loops, which usually are only 21 meters long [5], but this benefit can only be used when the link which is approaching the junction is large

enough.

The phases for the implementation (see section 3.1) were designed for the “From-to”-reference and the new restriction caused a major problem in the decision making algorithm. For example, when a vehicle is approaching from the “Up1” direction it can not be determined if the vehicle wants to turn left, right or travel straight. Therefore, the votes of the vehicle were added to all three internal counter for “’Up1’. This procedure is used for every vehicles from the different directions.

Afterwards, the new phase was chosen by the agent. Phase number 8 from the defined phases for cross-shaped junction (see figure 3.4) enables all directions from the “Down”-position and also the “From-right-to-left”-direction. Usually, all votes from vehicles which wanted to travel on these four directions would be reset and the votes of the waiting vehicles would be doubled, but the system can only distinguish between “Up” and “RIGHT” at the moment so every vehicles on these positions have to be reset.

This would contort the system, because the votes of vehicles which want to travel from “Right”to straight or right directions are reset as well. To solve the issues, the traffic light agents do not reset all their internal vote counters per decision cycle. Instead, only the vote counter of the allowed direction from the phase are reset. So the storage location for the votes was shifted from the vehicles towards the agent, which have now a little bit more complexity, but this was the only way to rescue the votes from vehicles which are still waiting on junctions without changing the complete concept of the framework.

These adaption should be disabled, when the VISSIM developers fix the issue of the missing vehicle attribute in the COM interface with the upcoming service pack.

CHAPTER 5

Evaluation

This chapter provides an overview of the simulations which are run with the different traffic lights agents and their results. Furthermore, the performance analysis of a simulation with the different agents is provided.

5.1 Experiments

This section explains the setup, the evaluation metrics and the results of the executed simulation to evaluate the DSGVE and the voting-based agents.

5.1.1 Experiment setup

Figure 5.1 shows the test map used for the test runs. As a test environment a part of the city center of Dublin was chosen. The lower boundary is around the Trinity College and the upper border around Parnell Street. Therefore, all of O'Connell Street with its side streets are part of the map. This area contains a high number of Bus services and was thereby very interesting for testing the weighted voting strategy. On the right border the map ends with the roundabout around the Custom House and a part of the R105 which also could provide interesting traffic patterns.

The black dots on the map symbolize the positions of the zone connector parking lots (see section 3.3.3). Most of nine origin zones around the map were chosen with the focus on the main roads where usually more vehicles approach the city center. Two of them where placed on side street to simulate chars which where parked in that area.

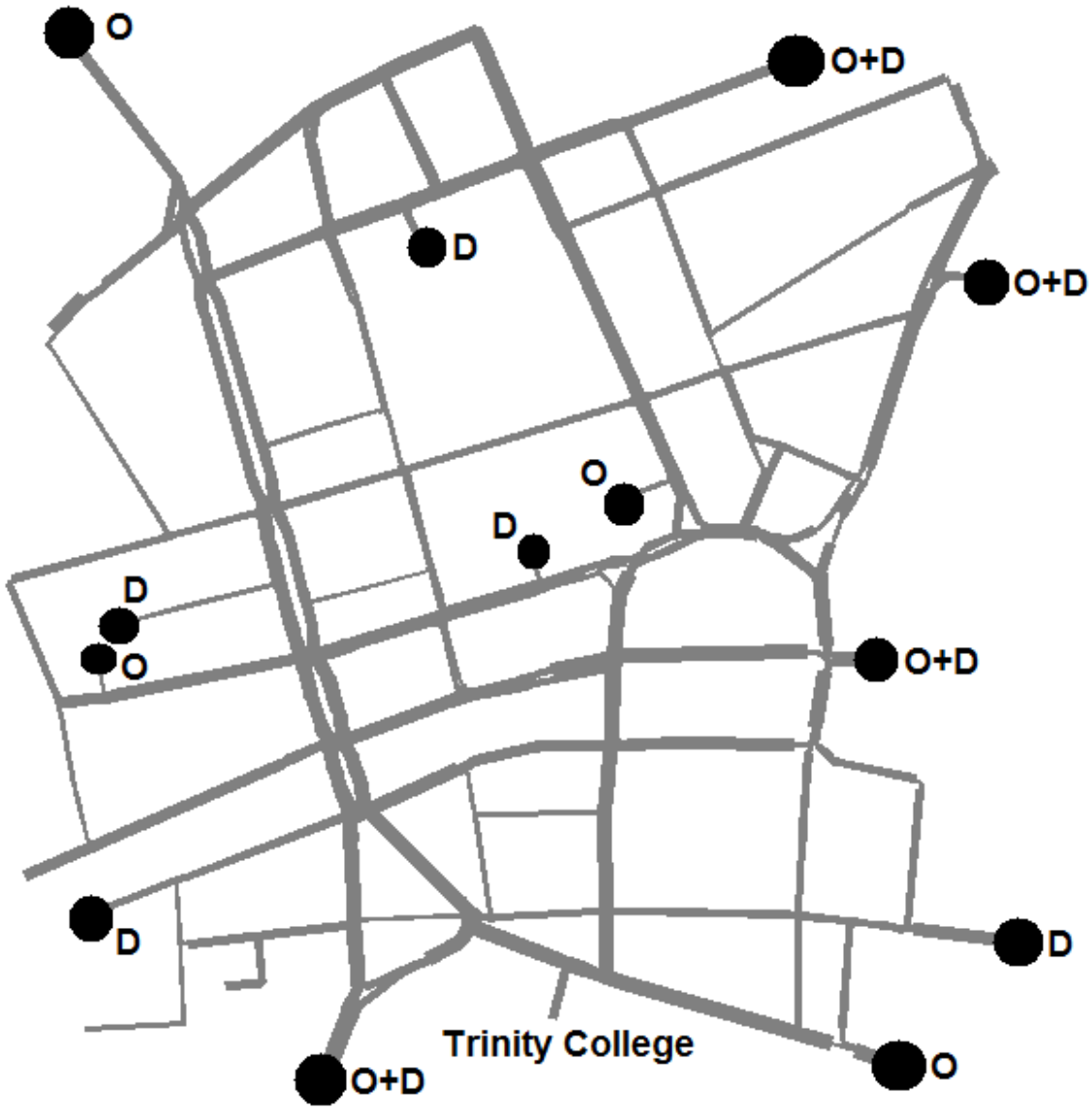


Figure 5.1: Used test map for the simulations: O = Origin zone; D = Destination zone.

The eigth destination zones are distributed around the map. From every origin the same ratio of vehicles is send to every destination, with the aim of a good saturation of vehicles on the roads around the map.

The aim of the experiment was to simulate the inbound traffic at the peek hours from 7am to 10am on a workday. Therefore simulation time was set to 10800 seconds (3 hours). As an estimation base for the traffic amount and the traffic pattern the Road User Monitoring report (RUM) 2008 and the Quality Bus Corridor (QBC) monitoring report 2007 from

the Dublin Transportation Office (DTO) were used [25, 26].

In this given time, 24,000 cars and 851 buses are approaching the Dublin city center via the QBC routes [25]. Overall, around 60,000 vehicles are counted as inbound traffic for the complete city center in the morning peak hours [26]. However, the test map is only a small part (around a 5th) of the complete city center of Dublin, so an estimation had to be found for the number of vehicles.

93% of all QBC routes travel through the test map, so with this in mind a test matrix with 24,000 vehicles with a split of 93% cars and 7% buses was chosen.

A second matrix with 60,000 vehicles and the same car bus split was also created and tested, to figure out which impact a strong increase in the numbers has on the tested agents.

In the scenarios with the simple majority voting agent each type of vehicle has one basic vote. In the weighted voting agents scenarios cars have 1.4 and buses 58.7 votes. These represent the average number of people in the current type of vehicle in the morning peak hours [26].

The simulations were also run with a simple Round Robin agent which took every defined phase in consideration and chose every 30 seconds the next phase from the set. Also, an optimized version of the Round Robin agent was used. This version took all phases from simple road and T-shaped junctions, but only phase number 8 10 12 and 14 from the cross-shaped and special junctions (see figure 3.4).

Table 5.1 and table 5.2 outlines the configuration parameters again.

Parameter	Value
Simulation run time	10,800 seconds
Phases cycle time	30 seconds
Phases amber time	3 seconds

Table 5.1: Basic setup of the simulator

Agent	votes for cars	votes for buses
Majority voting-based agent	1	1
Weighted voting-based agent	1.4	58.7

Table 5.2: Votes per type of vehicle

5.1.2 Evaluation metrics

Table 5.3 provides a description of the used evaluation metric.

Metric	Description
Average Travel Time (**ATT**)	The ATT is defined as the time the vehicles spen1 with their speed greater than 0.
Average Waiting Time (**AWT**)	The AWT is defined as the time the vehicles spen1 with their speed equals 0.
Throughput	The throughput is defined as the number of vehicles which arrived at their destination at the end of the simulation.

Table 5.3: Used evaluation metrics

Furthermore, for the majority voting agent and the weighted voting the results for cars and buses are examined separately to figure out if the weighted agent had an impact on the overall transported person in the system.

5.1.3 Results

Experiment with 24,000 vehicle in 3 hours

Table 5.4 provides an overview of the results of the simulation with 24,000 vehicles after 3 hours.

Agents	ATT	AWT	Throughput
Majority voting agent	135	49	21113
Weighted voting agent	138	51	21112
Round Robin agent	843	722	10961
Optimized Round Robin agent	316	209	18964

Table 5.4: 24.000 vehicles in 3 hours: Average waiting and travel time per vehicle in seconds and the total number of arrived vehicles to their destinations

As we can see, both voting-based agents significantly outperform the Round Robin agents. The majority voting agent reduces the ATT compared to the optimized Round Robin agent by 57%. The normal Round Robin agent is no match, but the agent had to use phases which are not benificial for this kind of agent, especially for the cross-shaped and special junctions.

Table 5.5 compares the throughput of the voting-based agents in detail.

Metric	Majority voting agent	Weighted voting agent
Total arrived vehicles	21113	21112
Arrived cars	20474	20460
Arrived buses	639	642
Total arrived Persons	66173	66329

Table 5.5: Detailed analysis of throughput for 24,000 vehicles

Due to the higher basic votes of the buses the weighted voting agent was able to guide 3 more buses and fewer cars to their destination, which resolved in 156 more persons who arrived their destination. Both agents performed very well and the differences are not really significant.

Figure 5.2 display the distribution of the ATT and AWT over time. As mentioned above, both voting agents behave more or less similar. It is hard to see the graph of the weighted voting agent behind the majority voting agent. Over time both Round Robin agents perform worse. This happens because more and more vehicles were injected in the system and these agents are not fast enough to allow them to proceed to their destination.

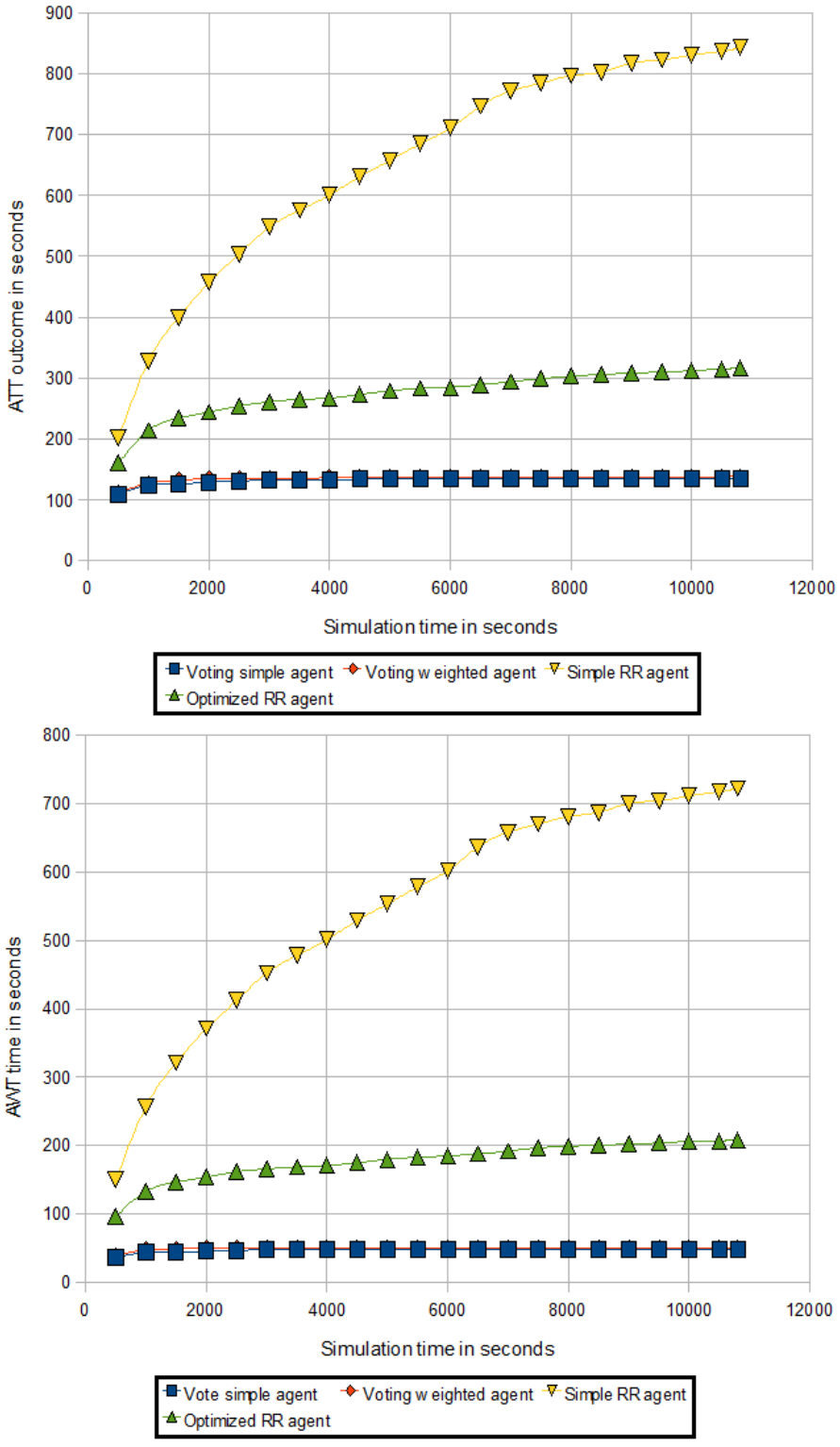


Figure 5.2: Input of 24.000 vehicles over 3 hours: Graph a) displays the development of the ATT over 10800 seconds with a measuring accuracy of 500 seconds. Graph b) shows the same for the AWT.

Experiment with 60,000 vehicle in 3 hours

Following the same schema, table 5.6 provides an overview of the results of the simulation with 60,000 vehicles after 3 hours.

Agents	ATT	AWT	Throughput
Majority voting agent	335	192	31148
Weighted voting agent	375	230	29467
Round Robin agent	908	781	11103
Optimized Round Robin agent	598	443	20579

Table 5.6: 60,000 vehicles in 3 hours: Average waiting and travel time per vehicle in seconds and the total number of arrived vehicles to their destinations

WWith regards to the Round Robin agents, the experiment with 60,000 vehicles does not provide any surprises. Both voting agents out-perform the Round robin agents significantly, too. The Majority voting agent provides the best performance and reduced the ATT compared to the optimized Round Robin agent by 44%. The weighted voting agent performed worse than the majority voting agent which will be discussed after the presentation of table 5.7.

Following the schema of the experiment with 24,000 again, table 5.7 compares the throughput of the voting-based agents in detail.

Metric	Majority voting agent	Weighted voting agent
Total arrived vehicles	31148	29467
Arrived cars	30238	28596
Arrived buses	910	871
Total arrived Persons	95750	91162

Table 5.7: Detailed analysis of throughput for 60,000 vehicles.

It seems that the majority voting agent can handle more vehicles better than the weighted voting agent. The simpler agent could not only guide more cars to their destination, but also more buses. This seems odd, because a higher throughput of the buses should be expected with the weighted agent. However, with the QBC ratio of only 3% buses it seems that the weighted agent does not have a beneficial impact.

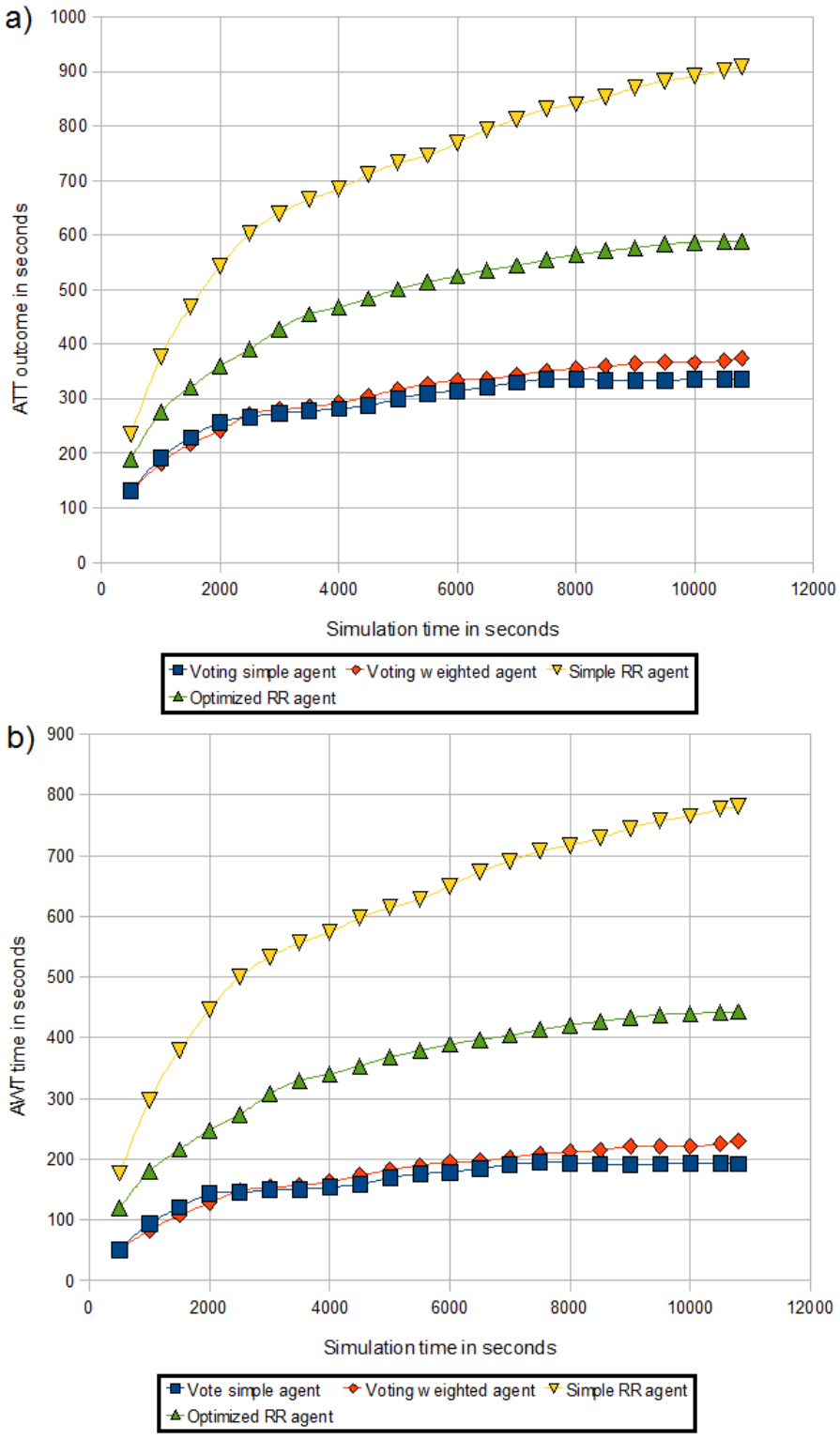


Figure 5.3: Input of 60.000 vehicles over 3 hours: Graph a) displays the development of the ATT over 10800 seconds with a measuring accuracy of 500 seconds. Graph b) shows the same for the AWT.

Furthermore, buses which are approaching high frequency junctions from side street force the weighted agent to prefer these minor roads which also could results in lesser throughput, especial when a bus on the major road is just out of the sensor range. In this case, a few seconds later it would have been a better to allocate the major road the green time.

Figure 5.2 displays the distribution of the ATT and AWT over time.

Both graphs show, that the weighted agent handles the traffic slightly worse than the simpler version when more and more vehicles where injected in the system. The Round Robin agent performed in both cases very poorly. Its optimized version is still much better, but compared to the voting-based agents, it is still not an alternative.

5.2 Performance of the simulation

The simulations were executed on the same system, which also was used for the development of the DSG VISSIM extension (see section 4.1) and the converter tools. This section will present an evaluation of the simulation performance in respect of the system usage and the overall speed of the simulation.

5.2.1 System usage

VISSIM itself supports the use of multiple cores, but during the test runs it became clear, that the implementation of this mode is not well-balanced, yet. Usually VISSIM used up to 100% of one core, but only maximal 10% of the second core of the system.

Furthermore, VISSIM needs to run the simulation with the test map, which contains 245 junctions, 36 of them with traffic lights, on average 1.1 GB (dynamic matrix with 8,000 vehicles) up to 1.5 GB (120,000 vehicles)of ram. The DSGVE needed around 8% of the CPU and 8 MB ram during the test runs.

5.2.2 Average performance factor of the simulations

Table 5.8 provides an overview of the average performance factor of the simulations. For example, a factor of 3.3 means that the simulation ran 3.3 times faster compared to a second in reality.

With only 8,000 vehicles over 3 hours the simulation performance is almost as good as with the Round Robin agents, but with the increasing number of vehicles the efficiency

Agents \ Vehicles in matrix	8,000	24,000	60,000	120,000
Majority voting agent	17.8	9.2	5.1	3.1
Weighted voting agent	18	9.1	4.5	3.2
Round Robin agent	20.0	18.4	17.9	16.7
Optimized Round Robin agent	20.0	18.5	17.6	16.7

Table 5.8: Overview of the average performance factor of the simulation

drops significantly. The main reason can be found in the WLAN module of the DSGVE. After a complete cycle time (in the test runs after 30 seconds) the WLAN module has to check if there are vehicles in range of each traffic light junction to figure out which vehicles have to talk to the respective voting-based agents. Therefore, it has to collect the position of each vehicle in the system. This causes a visible delay up to 4 seconds for every calculation step.

The Round Robin agents just have to choose one phase after another from their given phase sets, so the do not have to talk to vehicles at all. These agents only have to write the next phase in the corresponding file for the traffic light DLL and these 36 write operations do not cause as delay in the VISSIM simulation. The slight drop in the performance is caused by VISSIM itself which has to handle more vehicles on the road.

CHAPTER 6

Conclusion

This chapter summarizes and discusses the findings of this thesis and also describes possible extensions and further work in the future. Finally, some conclusions are drawn.

6.1 Achievements

To satisfy the goals of this thesis a framework was created which enables external extension to the functionality of VISSIM. With this framework and a DLL based plug-in to set traffic light phases via an external channel it was possible to implement a new kind of WLAN based voting agents. Furthermore, it was able to equip vehicles with more functionality and it was shown, that it is possible to do this with different objects in VISSIM.

The resulting voting agents were, as a proof of concept, evaluated against two additionally created Round Robin agents. The evaluation could show, that the major objectives of the thesis could be achieved. The major achievements were as follows:

- Creation of a new framework for VISSM simulations with extension.
- Creating tools for converting DTS maps and trace files towards a VISSIM readable format.
- Development of voting-based agents which can improve the ATT, AWT and throughput of vehicles in the simulated environments.

The most important achievement is the new framework. Thanks to this fundamental step it is possible to generate a huge variety of different agents in the future which can be tested against each other on an approximation of the city center or other parts of Dublin. These evaluations have the benefit that they use VISSIM as a basis, which is better known than the formerly used in house simulator of the DSG.

It was also possible to convert the used DTS map, but given to the problems with the unnecessary junctions (see section 4.6.1) it could be better to redo the map once and for all in VISSIM.

The voting-based agents were able to reduce the ATT up to 57% compared to the Round Robin agents which definitely can be counted as a success for the method. These results could even be improved when the "From-to"-reference would work as it is supposed to (see section 4.6.3). Visual examinations with the problematic map showed a promising performance, but could not be evaluated due to the these problems.

6.2 Future work

Due to the time constrains of the thesis there were some aspects which could be improved in the future. The WLAN module of the DSGVE provides only a basic functionality and does not reflect a proper behavior when it comes to packet drops or other properties. To test how WLAN interruptions affect the system this module should be extended.

Furthermore, when in one of the upcoming service packs for VISSIM the issue with the missing information about vehicles desired direction (see section 4.6.3) is fixed, the voting-based agents should be re-evaluated to see how much the better sensor data affect the overall performance of the agents. Also, more complex voting agents could be used. For example, a system where the votes of vehicles were not, or only partially, reset after each green phase. These vehicles would be treated as more important over time they spent in the system. Agents could also start to communicate with their neighbors with the aim to generate green waves.

In addition, it would be interesting to test the voting-based agents against SCATS, which is used in Dublin for urban traffic control at the moment. SCATS can be simulated with SCATSIM [19] which is compatible with VISSIM, but additional licenses are needed.

Finally, it would be interesting to combine a CRL based agent with the voting-based strategies. DSG is working on a CRL agent which is working within the same framework at the moment. First test runs implied that the voting-based agents perform better than the CRL agent, at least in the first simulation hours, because the CRL agents have to learn the best strategies first. However, because the agents are not well tested and implemented at the moment, the exact test results are not included in the evaluation. it could be a good solution when voting-based agents decide which phase is chosen and the CRL part

of the agent decides how long the green time will be. At the moment, the time for a phase was fixed to 30 seconds, which could be too long for a phase involving a minor road.

6.3 Discussion

Over the years many different approaches for better traffic management were developed and evaluated. Commonly used adaptive systems like SCATS are able to improve the throughput of vehicles compared to the early fixed time table systems (see section 2.1.3) and can thereby increase the capacity of roads in urban areas. However, roads are still and will be a limited good and increasing the capacity will only work to a certain point.

Besides the improvement of UTC systems there are also different approaches to that problem. Some people say that instead of researching, developing and applying more and more sophisticated systems to achieve a higher capacity of the road, this money and manpower should better be spent in traffic calming techniques and the improvement of public transport which could increase the overall throughput significantly [35]. In my opinion, they definitely have a point. Maybe a combination of improved UTC systems and more park and ride, bus priority and an overall better public transport system could have the best effect on improving the throughput in urban areas.

6.4 Conclusion

This dissertation described the application and evaluation of a voting-based decision-making algorithm for decentralized traffic light controllers in an urban traffic control system. With use of better sensor data and different voting strategies it was possible to outperform the other tested agents. Finally, the developed modular framework can be used to create and test new agents combined with VISSIM in the future.

APPENDIX A

Schema definition of the internal map

This appendix presents the XMLv2-format for the DSG internal map structure. This format provides an symbiosis of data which is necessary for the in house simulator and the additional information which is needed for the VISSIM converter.

A.1 Schema file

Listing A.1: junctionDataV2.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'>
<!--
History:
Date        -    Version     - Author             -     commend
unknown     -      V1        - unknown            -     original version
06/05/09    -      V2        - hendrycr@tcd.ie    -     added element "Link" to "incomingJunction" for VISSIM
                                                        support of mapping links to lanes via connectors
                                                        fixed mistakes in schema
23/06/09    -      V2        - hendrycr@tcd.ie    -     refactored names so all fit in the same name convention
                                                        deleted element "RefLanActionType" -
                                                        use "actionType" instead
                                                        added element "linkPriority" which is
                                                        necessary for VISSIM routing
24/06/09    -      V2        - hendrycr@tcd.ie    -     changed position of the linkPriority element in
                                                        incomingJunction and also added it to the
                                                        outgoingJunction element
29/06/09    -      V2.01     - hendrycr@tcd.ie    -     Added "vissimLinkId" "vissimQueueCounterId"
                                                        "vissimParkingLot" "vissimNodeId" "vissimConnectorId"
                                                        "vissimDelayTravelTimeZoneId"
01/07/09    -      V2.02     - hendrycr@tcd.ie    -     Added elements "vissimTlAgent" "vissimTlGroup"
                                                        "vissimTlHead" which are needed for the traffic
                                                        lights in VISSIM
03/07/09    -      V2.03     - hendrycr@tcd.ie    -     Deleted "linkPriority" element in the
                                                        "outgoingJunction" element, because it was unnecessary
-->
  <xs:element name="junctionData">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="details" type="xs:string" />
     <xs:element name="junctionRec" maxOccurs="unbounded">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="id" type="xs:integer" />
         <xs:element name="type">
         <xs:simpleType>
```

```
37          <xs:restriction base="xs:string">
38                  <xs:enumeration value="TL"/>
39                  <xs:enumeration value="Non-TL"/>
40                  <xs:enumeration value="SourcecSink"/>
41                  <xs:enumeration value="NOTO"/>
42          </xs:restriction>
43         </xs:simpleType>
44         </xs:element>
45         <xs:element name="location">
46         <xs:complexType>
47          <xs:sequence>
48                  <xs:element name="xCoordinate" type="xs:decimal" />
49                  <xs:element name="yCoordinate" type="xs:decimal" />
50          </xs:sequence>
51         </xs:complexType>
52         </xs:element>
53         <xs:element name="incomingJunction" maxOccurs="unbounded">
54           <xs:complexType>
55            <xs:sequence>
56                    <xs:element name="id" type="xs:integer" />
57                    <xs:element name="numLanes" type="xs:integer" />
58                    <xs:element name="linkDistance" type="xs:float" />
59                    <xs:element name="maxVelocity" type="xs:integer" />
60                    <xs:element name="linkPriority">
61                      <xs:simpleType>
62                        <xs:restriction base="xs:string">
63                          <xs:enumeration value="major"/>
64                          <xs:enumeration value="minor"/>
65                        </xs:restriction>
66                      </xs:simpleType>
67                    </xs:element>
68                    <xs:element name="outgoingJunctionRef" maxOccurs="unbounded">
69             <xs:complexType>
70              <xs:sequence>
71               <xs:element name="id" type="xs:integer"/>
72               <xs:element name="actionType">
73                 <xs:simpleType>
74                  <xs:restriction base="xs:string">
75                   <xs:enumeration value="L"/>
76                   <xs:enumeration value="R"/>
77                   <xs:enumeration value="S"/>
78                  </xs:restriction>
79                 </xs:simpleType>
80               </xs:element>
81                 </xs:sequence>
82             </xs:complexType>
83             </xs:element>
84                    <xs:element name="link">
85                      <xs:complexType>
86                        <xs:sequence>
87                          <xs:element name="vissimLinkId" type="xs:integer" />
88                          <xs:element name="vissimQueueCounterId" type="xs:integer" />
89                          <xs:element name="vissimParkingLot" maxOccurs="unbounded">
90                            <xs:complexType>
91                              <xs:sequence>
92                                <xs:element name="vissimParkingLotId" type="xs:integer" />
93                                <xs:element name="vissimParkingLotType" >
94                                  <xs:simpleType>
95                                    <xs:restriction base="xs:string">
96                                      <xs:enumeration value="zone"/>
97                                      <xs:enumeration value="abstract"/>
98                                      <xs:enumeration value="real"/>
99                                    </xs:restriction>
100                                 </xs:simpleType>
101                               </xs:element>
102                             </xs:sequence>
103                           </xs:complexType>
104                         </xs:element>
105                         <xs:element name="lane" maxOccurs="unbounded" minOccurs="0">
106                           <xs:complexType>
107                             <xs:sequence>
108                               <xs:element name="laneId" type="xs:integer" />
```

```
                              <xs:element name="toJunctionRef" maxOccurs="unbounded">
                                <xs:complexType>
                                  <xs:sequence>
                                    <xs:element name="refJunctionId" type="xs:integer" />
                                    <xs:element name="refLaneId" type="xs:integer" />
                                    <xs:element name="vissimConnectorId" type="xs:integer" />
                                  </xs:sequence>
                                </xs:complexType>
                              </xs:element>
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
            <xs:element name="outgoingJunction" maxOccurs="unbounded">
              <xs:complexType>
               <xs:sequence>
                <xs:element name="id" type="xs:integer" />
                <xs:element name="numLanes" type="xs:integer" />
                <xs:element name="linkDistance" type="xs:float" />
                <xs:element name="maxVelocity" type="xs:integer" />
                       <xs:element name="vissimDelayTravelTimeZoneId" type="xs:integer" />
               </xs:sequence>
              </xs:complexType>
            </xs:element>
                 <xs:element name="vissimNodeId" type="xs:integer" />
                 <xs:element name="vissimTlAgent" maxOccurs="1">
                   <xs:complexType>
                     <xs:sequence>
                       <xs:element name="TlAgendId" type="xs:integer" />
                       <xs:element name="vissimTlGroup" maxOccurs="unbounded">
                         <xs:complexType>
                           <xs:sequence>
                             <xs:element name="vissimTlGroupId" type="xs:integer" />
                             <xs:element name="vissimTlHead" maxOccurs="unbounded">
                               <xs:complexType>
                                 <xs:sequence>
                                   <xs:element name="vissimTlHeadId" type="xs:integer" />
                                   <xs:element name="vissimTlHeadLinkId" type="xs:integer" />
                                   <xs:element name="vissimTlHeadLaneId" type="xs:integer" />
                                 </xs:sequence>
                               </xs:complexType>
                             </xs:element>
                           </xs:sequence>
                         </xs:complexType>
                       </xs:element>
                     </xs:sequence>
                   </xs:complexType>
                 </xs:element>
               </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

A.2 Schema definitions

Following every possible element of the XMLv2 data which is defined in the schema above is explained.

Since version 2.01 there are elements in the schema which are added from the "‘XMLv2 to

.INP"'-converter tool. Every element which is added by the converter begins with "'vissim"'. These elements provide the necessary data for an mapping between the old simulator and the new VISSIM simulator.

A.2.1 junctionData element

The root element of the XML document.

Possible children

- *details*
  this element is used for a short description of the part of the city - e.g. when only a part of the whole map of Dublin is used then the coordinates of the cut-out could stay in this element.
- *junctionRec*
  Every junction on the map has one record where all necessary data for the simulator is gathered.

A.2.2 junctionRec element

Possible children

- *id*
  The unique identifier for the junction in the map.
- *vissimNodeId*
  After transforming the XMLv2 format into the VISSIM readable "'.INP"' format every junction from the old simulator can be mapped to one unique node inside the VISSIM system.
- *type*
  The type of the junction - e.g a junction which is controlled by traffic lights would have the value "TL".
  Possible values:

TL	=	with traffic lights
Non-TL	=	without traffic lights
SourcecSink	=	deprecated
NOTO	=	?

- *location*
  The X and Y coordinates of the junction. At the moment there is no plan to create an 3-Dimensional environment, but VISSIM would support that feature.

- *incomignJunction*
  Every junction from which you can arrive at the current junction has to have an entry in a junctionRec element.
- *outgoingJunction*
  Every junction from which you can reach from the current junction.
- *vissimTlAgent*
  When a junction has traffic lights the converter adds the information about these in this element.

A.2.3 location element

Possible children

- *xCoordinate*
  The X-coordinate of the junction.
- *yCoordinate*
  The Y-coordinate of the junction.

A.2.4 incomingJunction element

Every incoming junction that is to say a junction from which you can arrive at the current junction.

Notice that for each incoming junction, there are elements that represent the outgoing junctions that you can reach if you arrive from this incoming junction.

Possible children

- *id*
  The unique identifier of the incoming junction.
- *numLanes*
  The number of lanes which arrive from the incoming junction to the current junction. This field is more or less deprecated, because you could calculate the overall number of lanes from the link element (see below), but in cases where the specific data does not exists it still is useful. Furthermore, the current converter does use it in some cases.
- *linkDistance*
  The distance between the two junctions.

- *maxVelocity*
  The maximum allowed speed on the link between the two junctions.

- *linkPriority*
  This is an important feature for the VISSIM priority rule. It manages which road user has priority on the junctions. Possible values:
  major = the incoming link of the junction is a major road
  minor = the incoming link of the junction is a minor road

- *outgoingJunctionRef*
  For each incoming junction, these elements represent the outgoing junction that you can reach if you arrive from this incoming junction.

- *link*
  Every incoming junction has exact one link. A link contains a set of separate lane elements.

A.2.5 outgoingJunctionRef element

For each incoming junction, these elements represent the outgoing junction that you can reach if you arrive from this incoming junction.

Possible children

- *id*
  The unique identifier of the junction you can reach.

- *actionType*
  There exist 3 different actions from the old phase generator.
  L = Left
  R = Right
  S = Straight
  These actions represent the action a road user has to take when he comes from the incoming junction to move to the outgoing junction.

A.2.6 link element

To rebuilt maps as precisely as possible in VISSIM you need to know which lanes of the links are connected to other lanes from other links.

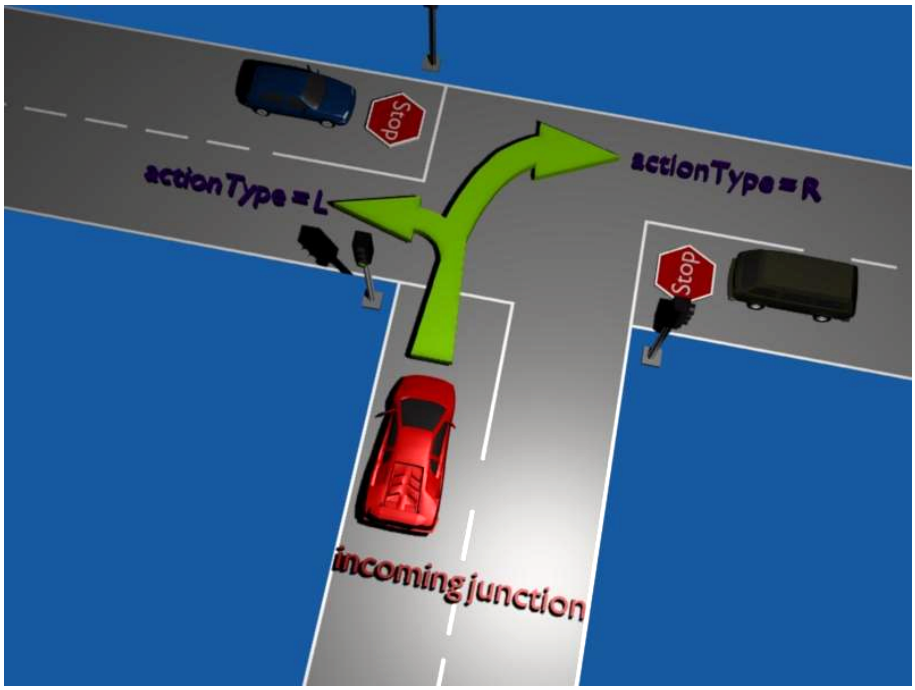


Figure A.1: Example for action types.
The Lamborghini arrives at the junction on a incomingJunction element. This incomingJunction element has two outgoingJunctionRef elements.
One reference has the action type entry "'L"' for left and the other has a "'R"' for right.

Possible children

- *vissimLinkId*
  After transforming the XMLv2 format into the VISSIM readable "'.INP"' format every link element can be mapped to one unique link inside the VISSIM system.

- *vissimQueueCounterId*
  This element represents the identifier of the queue counter which is responsible for metering the amount of waiting cars on that link.

- *vissimParkingLot*
  Every link in VISSIM needs one parking lot with the type "'zone collector"' to enable a trace routing mechanism which behaves like the trace files system of the old DSG simulator.

- *lane*
  Every lane which arrives from the incoming junction has to have a separate lane element.

A.2.7 vissimParkingLot element

Possible children

- *vissimParkingLotId*
  The identifier of of the parking lot in VISSIM.
- *vissimParkingLotType*
  There are 3 different kinds of parking lots in VISSIM. "'Zone connector"' and "'Abstract parking"' are both types which are used for dynamic assignment of cars. "'Real parking spaces"' are modeled parking capacity which present real parking spaces in the environment.

A.2.8 lane element

Every incoming lane of the link has to have an outgoing reference.

Possible children

- *laneId*
  The identifier of of the lane.
- *toJunctionRef*
  For every junction to which this lane is connected there has to be a reference element

A.2.9 toJunctionRef element

Possible children

- *refJucntionId*
  The identifier of the outgoing junction.
- *refLaneId*
  This lane of the link from an incoming junction maps to a lane of a link to an outgoing junction.

A.2.10 outgoingJunction element

Every junction from which you can reach from the current junction.

Possible children

- *id*
  The unique identifier of the junction.

- *numLanes*
  The number of lanes which can be used to travel from this junction to the referenced junction.

- *linkDistance*
  The distance between the two junctions.

- *maxVelocity*
  The maximum allowed speed on the link between the two junctions.

- *vissimDelayTravelTimeZoneId*
  These zones are used for collecting the amount of cars which are leaving a junction from one link in VISSIM.

A.2.11 vissimTlAgent element

When a junction has traffic lights the converter adds the information about these in this element.

Possible children

- *TlAgendId*
  The unique identifier of the traffic light agent.

- *vissimTlGroup*
  In VISSIM every traffic light head is part of a group, which is part of one traffic light agent.

A.2.12 vissimTlGroup element

Possible children

- *vissimTlGroupId*
  The unique identifier of the traffic light group.

- *vissimTlHead*
  A head is the actual sign on the road.

A.2.13 vissimTlHead element

Possible children

- *vissimTlHeadId*
  The unique identifier of the traffic light head.
- *vissimTlHeadLinkId*
  The identifier of the VISSIM link.
- *vissimTlHeadLaneId*
  The identifier of the lane which is a part of the VISSIM link.

APPENDIX B

Instructions to run a simulation with the DSGVE

This chapter will provide a step by step instruction for using VISSIM with DSGVE by using parts of the DTS map.

B.1 Basic VISSM Setup

- Install VISSIM (Version 5.10) with CodeMeter extension
- Start WebAdmin tool from CodeMeter and switch to the configuration tab
- Add server address (last known IP: 134.224.36.119) using port 22350
- Press apply and close CodeMeter
- Got to folder $VISSIM510/API/SignalControl_DLLs/$SignalGUI_DLL
- Run SignalGUI.vcproj with Visual Studio and compile Release version.
- Copy the fresh compiled "‘SignalGUI.dll"’ int the EXE folder of VISSIM
- Go to folder $VISSIM510/API/SignalControl\_DLLs/Examples/STD\_GUI$
- Run STDSC_GUI.sln with Visual Studio and compile Release version.
- Copy the fresh compiled $STDSC\_GUI.dll$ int the EXE folder of VISSIM
- Create an empty file named "‘test.wtt"’ in the EXE folder of VISSIM
- Start VISSIM
- Go to menu "‘Simulation -> Parameters” and change the number of cores to maximum

B.2 Converting a DTS map to use it with VISSIM

This section will explain the converting process after the user has generate a map with the DTS map and path generator. For an explanation to use this tools see **??**.

For the conversion the following files are necessary:

- Start ConvertXMLtoINP.exe
- Type in the path of the DTS map which should be converted
- Type in the path of the prefix file - just press enter
- Type in the path of the file that has to be written between the converted data - just press enter
- Type in the path of the suffile file - just press enter
- Type in the path of phase file
- Type in the path of the data phases files directory - just press enter
- Type in the path of the data numeration file - just press enter
- Type in the path where there INP-file should be generated
- Type in the name of the DLL which is responsible for the external signal controller - Should be “mmf_sc.dll”

The INP-File was generated. Also, a file named “[DTS map name]-converted.xml” is generated which is important for the DSGVE. It contains a mapping of the old junctions to the internal VISSIM identifiers for the map.

B.3 Starting a simulation with the DSGVE

After the conversation the following files are needed for the VISSIM simulation:

- The INP-file
- The “[DTS map name]-converted.xml”-file
- dublin1.scprops
- The external DLL - should be “mmf_sc.dll”

The best solution it to copy them all in on folder. Afterwards it is recommended to generate a folder called “junction_input” in that folder.

In these folder tehr als have to be file called “config.xml”. Listing B.1 provides an example of the values of the file.

Listing B.1: Example of the config.xml file

```
<?xml version="1.0" encoding="utf-8" ?>
<config>

 <!-- ##########################################################################################
 #   General config                                    #
 ##########################################################################################-->

 <path>C:\\Users\\Rascil\\Documents\\My Dropbox\\Ronny\\Sourcecode\\DSGVissimExtention\\RHDissertationMap
      \\</path>
 <outputdir>junction_output\\</outputdir>
 <inpfile>Dublin.inp</inpfile>
 <xmlfile>rtmap09-modified-converted.xml</xmlfile>
  <xmlspecialtlfile>RTMap09-SpecialTLJunctions.xml</xmlspecialtlfile>


 <!-- These 2 are only used sometimes in the wrapper but not in the DSGVissimExtenstion version!-->
 <speed>10</speed>
 <refresh>200</refresh>

 <!-- ##########################################################################################
 #   Basic config for the simulation                          #
 ##########################################################################################-->

 <simulationtime>10800</simulationtime>

 <!-- ******************************************************************************************
 # <VissimVisualization>: Should teh simulation be visualized in the VISSIM window?  #
 #           #
 # 0 = No          #
 # 1 = Yes          #
  ****************************************************************************************** -->

 <VissimVisualization>1</VissimVisualization>


 <!-- ##########################################################################################
 #                                          #
 #   Traffig light agent config                          #
 #                                          #
 ##########################################################################################-->

 <!-- ******************************************************************************************
 # <TLAgentMode>:         Which version of the agent?                #
 #                                          #
 # SimpleVote  = Simple vertsion every type of vehicle has 1 vote           #
 # BusVote     = All vehcilse have 1 vote, but busses have 10 votes           #
 # SimpleRR    = Very basic RoundRobin agent                      #
 # OptimizedRR = Opimzed RR uses only "good" RR phases                #
  ****************************************************************************************** -->
 <TLAgentMode>BusVote</TLAgentMode>

  <TLCycleTime>30</TLCycleTime>

 <TLAmberTime>3</TLAmberTime>

 <TLAgentFiles>junction_input\\</TLAgentFiles>
</config>
```

The text of the elements:

- path
- outputdir
- inpfile
- xmlfile
- xmlspecialtlfile
- TLAgentFiles

have to be changed according to the new files for the simulation,

“xmlspecialtlfile” is only needed when the simulated area has a 5-sided or 6-sided traffic light junction. Listing B.2 provides an example for an entry of a 6-sided traffic light junction.

Listing B.2: Example of the config.xml file

```
<?xml version="1.0" encoding="utf-8" ?>
<specialJucntions>
  <tlJunction>
    <junctionId>1521</junctionId>
    <type>5</type>
    <postions>
      <Up1>
        <refJunctionId>767</refJunctionId>
        <FromZoneID>0</FromZoneID>
        <ToZoneRefID>173</ToZoneRefID>
      </Up1>
      <Up2>
        <refJunctionId>757</refJunctionId>
        <FromZoneID>8</FromZoneID>
        <ToZoneRefID>0</ToZoneRefID>
      </Up2>
      <Left1>
        <refJunctionId>744</refJunctionId>
        <FromZoneID>10</FromZoneID>
        <ToZoneRefID>0</ToZoneRefID>
      </Left1>
      <Right1>
        <refJunctionId>1531</refJunctionId>
        <FromZoneID>11</FromZoneID>
        <ToZoneRefID>36</ToZoneRefID>
      </Right1>
      <Down1>
        <refJunctionId>766</refJunctionId>
        <FromZoneID>9</FromZoneID>
        <ToZoneRefID>0</ToZoneRefID>
      </Down1>
      <Down2>
        <refJunctionId>760</refJunctionId>
        <FromZoneID>0</FromZoneID>
        <ToZoneRefID>171</ToZoneRefID>
      </Down2>
  </postions>
  </tlJunction>
</specialJucntions>
```

Afterwards, a simulation with the DSGVE can be executed with the “DSGVissimExtention.exe”.

After the simulation the fzp-file can be evaluated (see section 4.4).

APPENDIX C

Content of the DVD

- DSG VISSIM extension
- DSG map converter
- DSG trace file converter
- DSG evaluation file reader
- Used map for evaluation
- Evaluation files (results)
- Dissertation
- Presentation slides
- Presentation poster

APPENDIX D

List of abbreviations

ATT	.....	**A**verage **T**ravel **T**ime
AWT	.....	**A**verage **W**aiting **T**ime
DSG	.....	**D**istributed **S**ystems **G**roup
DSGVE	.....	**DSGV**issim**E**xtension
DSRC	.....	**d**edicated **s**hort-range **c**ommunications
DTL	.....	**D**emocratic **T**raffic **L**ight
DTS	.....	**D**ublin **T**raffic **S**imulator
GDA	.....	**G**reater **D**ublin **A**rea
ITS	.....	**I**ntelligent **t**ransportation **s**ystem
MANET	.....	**M**obile **a**d-hoc **net**works
QBC	.....	**Q**uality **B**us **C**orridor
SCAT	.....	**S**ydney **C**oordinated **A**daptive **T**raffic
SCOOT	.....	**S**plit **C**ycle **O**ffset **O**ptimisation **T**echnique
SRVC	.....	**S**parse **r**oadside-**v**ehicle **c**ommunications
UTC	.....	**U**rban **T**raffic **C**ontrol
VANET	.....	**V**ehicular **a**d-hoc **net**work
V2I	.....	**V**ehicle-**t**o-**I**nfrastructure
V2V	.....	**V**ehicle-**t**o-**V**ehicle
VCS	.....	**V**ehicular **C**ommunication **S**ystems
VISSIM	.....	**V**erkehr **I**n **S**tädten - **SI**Mulationsmodell

Bibliography

[1] URL `www.simtd.de`. (cited on page 12)

[2] *The Encyclopaedia Britannica Eleventh Edition*. Encyclopaedia Britannica Inc., 1911. (cited on page 19)

[3] O. Andrisano, R. Verdone, and M. Nakagawa. Intelligent transportation systems: The role of third-generation mobile radio networks. *IEEE Communications Magazine*, 38:144–151, 2000. (cited on page 11)

[4] National Electrical Manufacturers Association. *NEMA Standards Publication TS 2-2003*, 02.06 edition, 2003. (cited on page 5)

[5] Turner-Fairbank Highway Research Center. *Traffic Detector Handbook: Third Edition*. 2006. (cited on pages 7 and 42)

[6] I. Day. Scoot - split, cycle & offset optimization technique. In *Adaptive Traffic Signal Control Workshop*, 1998. (cited on pages 1 and 9)

[7] C. Diakaki, M. Papagerogiou, and K. Aboudolas. A multivariable regulator approach to traffic-responsive networkwide signal control. *Control Engineering Practice*, 10:183 – 195, 2002. (cited on page 10)

[8] F. Dion and S. Yagar. Real-time control of signalised networks - different approaches for different needs. *IEE Conference Publications*, 1996:56–60, 1996. (cited on page 9)

[9] J. Farges, I. Khoudour, and J. Lesort. Prodyn: on site evaluation. *Road Traffic Control, 1990*, pages 62 – 66, 1990. (cited on page 9)

[10] M. Fellendorf. Public transport priority within scats - a simulation case study in dublin. *67^{th} Annual Meeting of the Institute of Transportation Engineers*, 1997. (cited on page 9)

[11] The Apache Software Foundation. *Xerces-C++ Documentation*, 2005. (cited on page 28)

[12] M. Garey and D. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. (cited on page 40)

[13] Distributed Systems Group. *Dublin Traffic Simulator documentation*. Trintiy College Dublin. (cited on pages 29, 30, and 31)

[14] Z. Gu and L. Han. Evaluation of dynamic weight threshold algorithm for wim operations using simulation, 2003. (cited on page 20)

[15] S. Guberinic, G. Senborn, and B. Lazic. *Optimal Traffic Control: Urban Intersections*. CRC, 2007. (cited on page 3)

[16] L. Head and P. Mirchandani. A real-time traffic signal control system: architecture, algorithms, and analysis. *Transportation Research Part C: Emerging Technologies*, 9: 415 – 432, 2001. (cited on page 10)

[17] Road Safety Authority Ireland. *Rules of the road*. 2008. (cited on page 5)

[18] C. Kaufman, R. Perlman, and Speciner M. *Network Security: Private Communication in a Public World*. Prentice Hall, 2002. (cited on page 38)

[19] Simon Kinnear. Vissim scatsim: Beyond fixed time modelling, 2007. (cited on page 55)

[20] D. E. Knuth. *The Art of Computer Programming Vol. 3*. Addison-Wesley Longman, 1999. (cited on page 30)

[21] Booz Allen Hamilton Ltd. Greater dublin area travel demand management study. Technical report, Dublin Transportation Office, 2004. (cited on page 1)

[22] V. Mauro and C. DiTarano. Utopia. *Control, Computers, Communications in Transportation selected papers from the IFAC Symposium*, pages 245–252, 1990. (cited on page 9)

[23] Institute of Transportation Engineers. Transport in the urban environment. *ITE Journal*, 1997. (cited on page 8)

[24] Dublin Central Statistics Office. Regional population projections 2011-2026. Technical report, 2008. (cited on page 1)

[25] Dublin Transportation Office. Quality bus corridor monitoring report. 2007. (cited on page 46)

[26] Dublin Transportation Office. Road user monitoring 2008. 2009. (cited on page 46)

[27] *VISSIM 5.10-03 COM Interface Manual*. Planung Transport Verkehr AG, 2008. (cited on page 25)

[28] *VISSIM 5.10 User Manual*. Planung Transport Verkehr AG, 2008. (cited on pages 20, 21, 22, 23, 24, and 41)

[29] I. Quader, B. Li, W. Peng, and A. Dempster. Use of fingerprinting in wi-fi based outdoor positioning. Technical report, The University of New South Wales, 2007. (cited on page 13)

[30] A. Salkham, R. Cunningham, A. Garg, and V. Cahill. A collaborative reinforcement learning approach to urban traffic control optimization. In *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, 2008. (cited on pages 10 and 42)

[31] M. Sichitiu and M. Kihl. Inter-vehicle communication systems: a survey. *IEEE Communications Surveys*, 10:88 – 105, 2008. (cited on pages 10 and 11)

[32] A. G. SIMS and K. W. DOBINSON. The sydney coordinated adaptive traffic (scat) system philosophy and benefits. *IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY*, VT-29:130–137, 1980. (cited on pages 1, 3, 4, 7, and 9)

[33] A. S.Tanenbaum. *Computer Networks*. rentice Hall International, 2002. (cited on pages 5 and 23)

[34] Siemens Intelligent Transportation Systems. *Traffic control systems handbook*. Federal Highway Administration, 2005. (cited on page 3)

[35] Kenneth Todd. Traffic control: An exercise in self-defeat, 2006. (cited on page 56)

[36] Y. Toor and P. Muehlethale. Vehicle ad hoc networks: Applications and related technical issues. *IEEE Communications surveys*, 10:74 – 88, 2008. (cited on pages 10 and 31)

[37] Auckland Traffic Management Unit. *Traffic Signals Design Guidelines*, 2.0 edition, October 2007. (cited on page 5)

[38] C. Wewetzer. Thevolkswagen approach to simulation of car-to-xcommunication, 2007. (cited on page 20)

[39] J Zhu and S Roy. Mac for dedicated short range communications in intelligent transport system. *IEEE Communications Magazine*, 41:60–67, 2003. (cited on page 11)

Ronny, you are turning into a leprechaun. Stop it!
-Node from one crazy mind to another-
In memory of Douglas Adams