Masters in Informatics Engineering
Dissertation/Internship
Final Report

Cosmo
Optimizing the city with collaborative route planning

João David Marques Gil
jdgil@student.dei.uc.pt

Advisor:
Francisco Câmara Pereira
Date: August 31, 2011

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Abstract

The ever increasing world population added to the always present need of human transportation makes road traffic in the big cities a huge and urgent issue, and, consequently, the target of substantial investigation. The difficulty to solve such a complex problem through analytical methods alone, and the need to test, evaluate and demonstrate a proposed course of action, before real-world implementation, have led to an incessant growth of artificial intelligence and computer simulation in traffic flow analysis and optimization. In this work we present a different approach to traffic flow optimization, by testing a set of new algorithms in an expressly created test bed. The test bed was implemented has an extension of a road traffic simulator and tries to mimic a real-world system where drivers use vehicle GPS technology to find the best route to their destination. The novelty of the tested algorithms, when compared with similar studies, is in the elimination of the need for users to share their private location information. This is done through the distribution of differentiated traffic information, causing users to choose diverse routes and better redistributing the traffic in the city. Early results show that it is hard to match the performance of traffic flow attained by systems that make use of present and intended location of drivers.

Keywords

collaborative route planning, intelligent transport systems, agent-based computing, traffic information, transportation management

Table of Contents

Table of Figures

Chapter 1 Introduction

Time is money! The crucial importance of ever-faster transportation of people and goods clashes with the unstoppable growth of modern societies to create one of the biggest problems big cities face today: a ceaseless increase of vehicles, circulating in road networks physically limited in space.

There have been, over the years, innumerous attempts to tackle this problem, for instance by building bigger and better networks, promoting the use of different kinds of transportation or organizing traffic flow to enhance the utilization of available resources. Focusing on this last method, the utilization of vehicle GPS (Global Positioning System) devices, receiving real time traffic information, is an easy accessible answer for individual drivers to improve their daily trips. If each person, individually, tries to avoid highly congested areas of the network, together, they will manage a better redistributed traffic flow.

One problem arises though: if everyone has the same information, will they not end up making the same decisions? If all the drivers try to avoid an existing traffic jam by choosing the second best route, will they not end up just relocating the jam to a different area? Substantial research has been done to address these questions (e.g. Adler et al., 2005; Yamashita et al., 2005), but they all have one common weakness: the mandatory sharing of private and sensible location information by the users. Their need for anonymity services impairs the systems and will still not guarantee total confidentiality.

This work attempts to face the same questions from a different angle. It focuses on the idea of spreading differentiated network and traffic information among drivers, i.e., with altered weights assigned for the city roads/network links, resulting in unconsciously diverse route choices. The implementation of different algorithms will present varied ways of data customization, by the server to the users.

With all this in mind, in order to try and prove this concept, an available road traffic simulator will be extended and adapted for the test of a selection of algorithms. In the end, there will, hopefully, be a significant improvement, in relation to the simple distribution of the same traffic information, without the need for the exposure to privacy attacks.

Following the latest trends in road traffic simulation, the chosen base tool for the implementation of our test bed will use agent technology to run micro-simulations. That is to say, it will simulate the individual behavior of drivers in the network, representing them as agents. This methodology allows for a suitable and realistic simulation of the autonomy, collaboration and reactivity of drivers.

The following chapter, Chapter 2 - State of the Art, will give a brief review of the literature used as a basis for the construction of this project, mentioning the work done on the area of traffic flow optimization as well as papers directly involved with the study and testing of traffic simulators. Furthermore, in Chapter 3 - Work Process, there will be a description of all the steps taken during this past year. This includes the comparison of micro-simulators, the trial of MATSim and the exploration of SUMO, aiming at the implementation of our test bed, as well as the implementation and testing of the traffic flow optimization algorithms. Chapter 4 - Results will give an overview of the functionalities of the created application and discuss the performance of the tested algorithms, while Chapter 5 – Conclusions will present the final allegations of this project and talk about potential work to be done in the future.

Chapter 2
State of the Art

2.1. Traffic and transportation management systems

The main goal from Cosmo, improving road traffic flow using an agent-based car navigation system, is not new. In fact, the exact same problem is addressed by Yamashita et al. (2005): in a car navigation system that recommends the best route based only on the current state of traffic congestion, several users will simultaneously get the same information, leading to a frustrating concentration of traffic on a new route or area.

However, while they propose a cooperative car navigation system where each vehicle transmits route information, like current position, destination and route to the destination, in an attempt to estimate the future traffic flow, our approach tries to balance it simply by providing each user with slightly different views of the traffic in the network. Thus, we prevent possible privacy issues and distribute route calculation to the agents. The other big difference between our works lies on the testing part: while they rely on a simple traffic flow model, we decided to develop a test bed on top of a road traffic simulator. This will not only give more accurate and reliable results, but also allow for the easier testing and comparison of several routing and traffic flow balancing algorithms.

Apart from these contrasts there are many premises and approaches that can be harnessed to our research. For instance, the assumption of integral knowledge of the network state by the system, through road monitoring, is of great importance. The reductionist premise that all drivers that choose to ignore congestion information follow the shortest distance path is also shared. Two of the networks used in system testing, lattice and radial and ring, are identical as well.

In their effort to prove the advantage of the route information sharing (RIS) mechanism, they compare it to other two common route choice modes: shortest distance (SD) paths for drivers who choose to ignore information on traffic congestion and shortest time (ST) for the ones that rely solely on the current available information. The obtained results proved the superiority of RIS at all levels: the average travel time of RIS users is always shorter than the ones using SD or ST, offering individual incentive, and the travel time of the same RIS drivers shortened as their percentage increased, evidencing social acceptability. Cosmo aims to mimic these results, without the need to ask drivers for their intentions and private information.

Another similar work, Adler et al. (2005), focuses on the development of a conceptual framework of a Cooperative Traffic Management and Route Guidance System (CTMRGS) using the integration of multi-agent systems and principled negotiation. This framework aims to conciliate system with driver objectives, i.e., it "seeks to improve network performance by systematically allocating demand across capacity in a manner that satisfies both driver and system operator interests.

The CTMRGS is comprised of three types of agents:

- Agent-IRANS, representing the drivers, which must be capable of planning the route and communicating with the Agent-ISP. They are conceived as intelligent traveller information systems that use machine learning and artificial intelligence to learn driver route choice preferences and aid with pre-trip route choice;
- Agent-ISP, the information service provider, will act as a mediator between Agent-IRANS and Agent-TMC, negotiating with the first while defending the interests of the second;
- Agent-TMC, or system operator, will be given the task of monitoring and controlling the network, for example with phase control strategies for traffic lights. It passes relevant network information to Agent-ISP, which in turn will be used in the negotiations with Agent-IRANS.

A linear normalized weighed utility maximization model was used to measure system and driver satisfaction. In other words, path performance is assessed through the maximization of a normalized set of goals and its assigned weights. Examples of these goals are the minimization of travel time and distance, number of road changes and occupation of network links.

The negotiations between Agent-ISP and Agent-IRANS are thus made through utility comparison of possible paths, with the help of two additional thresholds: path utility indifference and weight change tolerance. The chosen route is the one that satisfies both parties, maximizing path utility while minimizing tolerance use. If no compromise is reached, Agent-IRANS leaves the decision to the driver.

The CTMRG was implemented in OpenCybele, a platform for control and event-driven execution of agents, and integrated with the Autonomous Agent Simulation (AASIM), much like it was done with Cosmo extension and SUMO. Subsequently, a simple network, consisting of two highways crossing each other and one beltway, together with five different travel scenarios were used for testing and validation purposes. Each scenario was simulated under low, moderate and heavy traffic conditions, with the creation of 1800, 3600 and 4800 vehicles per hour.

Adler et al. (2005) conclude that CTMRGS-based routing manages to improve network traffic flow, without compromising individual driver satisfaction. They also suggest several potential improvements including an interesting congestion/redistribution pricing method, where the ISP charges drivers that take the best routes while paying for the use of less than optimal ones, thus increasing the acceptance of negotiated path choice.

In my opinion, their work is only compromised by its unsatisfactory testing, due to both an excessive simplicity of the used network and the hard to read measurements. They themselves call it a preliminary experiment and propose further testing. Nevertheless, one of the main objectives of Cosmo is to improve network traffic flow without the negotiation between the network manager and users. This limits the choice of the user, decreasing individual incentive. To counteract this, our system will try to present a considerable and immediate improvement of individual trip performance.

Kuriyama et al. (2007), based on the work from Yamashita et al. (2005), proposed an interesting spot scheduling technique to alleviate traffic congestion. The only difference resides in the fact that the drivers share a list of places to visit instead of only one destination. They evaluated their mechanism by comparing it against an iterative version of RIS. The results were quite satisfactory, with good levels of individual incentive, diffusion rate tolerance and feasibility.

Although this new method presents a possible weak point in our work, increasing the exploration of information sharing by predicting routes and traffic congestions further in the future, it can only be used in very specific situations, like sightseeing and a small group of business activities, greatly reducing its interest to the general public.

Yet another multi-agent system is presented by Paruchuri et al. (2002). Their work concentrates on the simulation of unorganized traffic, that is to say, without the commonly used unrealistic strict simulator rules.

Basically, the only interesting feature of their system, when compared to the SUMO simulator, used in Cosmo, is the existence of overtaking. This is made possible thanks to a small set of implemented driver characteristics, like free will speed, free will braking power, maximum braking power, free will acceleration, maximum acceleration and minimum gap maintained with other drivers, as well as to a driver perception model, where the roads themselves are agents, responsible for sensing the vehicles that pass over them. In this model, the sectors and junctions of the network can be seen as a blackboard, through which the agents communicate, updating their positions and monitoring their surroundings.

Driver parameters mentioned above are tuned in accordance to a defined set of psychological traits: aggressive, normal and cautious. Moreover, overtaking is influenced by confidence and rush factors. Both play an important role on the necessary conditions to overtake, however, while the first is determined by driver personality, the second is only a matter of circumstance. There is still yet another feature called the dominance effect, which consists in the switching of traffic flow in junctions, only accomplished by dominating vehicles.

Testing was done using only a simple 4-way intersection, varying driver aggressiveness. Although the truly remarkable emergent behavior of realistic agent interactions, the computational costs of such traffic simulation, in a considerably more complex network, with a decent number of vehicles, would probably be too high. In the sole interest of creating a test bed for traffic management algorithms, SUMO provides realistic enough results. Still, this work provides valuable information on how to best define individual driver behavior.

2.2. Data privacy and user anonymity

As previously mentioned, Cosmo intends to approach the problematic of traffic flow optimization excluding the need for user location information sharing, thus avoiding privacy issues inherent to similar projects.

The most common method to deal with privacy concerns of location-based services consists in the concealment of spatial and temporal position of users. The principle of minimal collection, presented by Gruteser and Grunwald (2003), uses it to obtain practically anonymity data, i.e., altered information that makes re-identification of the subject, with reasonable efforts, impossible. A combination of two algorithms is used to meet a specified anonymity constraint:

- an adaptive quadtree-based algorithm that decreases the spatial resolution of location information;
- an algorithm that yields higher spatial resolution through decreasing temporal resolution.

The required data accuracy is given by the k-anonymity technique, applied to location information. A subject is considered as k-anonymous if, and only if, the location information presented is indistinguishable from the location information of at least k - 1 other subjects.

Another paper goes even further. Damiani et al. (2008) claim that geometry-based obfuscation techniques fail to protect against some particular kinds of privacy attacks. They suggest a way to adjust data granularity, taking space semantics in consideration. For this purpose, two core components of an obfuscation system are defined:

- a privacy model supporting the obfuscation of sensitive locations based on user preferences;
- an algorithm, called sensitive flow (SensFlow), implementing the obfuscation strategy.

Nevertheless, both of the aforementioned works involve the use of a dedicated anonymity server, creating a possible performance bottleneck and further trust issues. Moreover, the increase of data coarseness can, in our specific problem, impair the well-functioning of the system. For example, if the starting, current or ending, position of a user cannot be directly related to a unique link in the network, simple routing algorithms will not work.

To ensure the anonymity of drivers, Yamashita et al. propose the use of a system designed for anonymous auctions, for instance the one demonstrated by Yokoo and Suzuki (2002). Their work discusses an ingenious secure dynamic programming protocol that utilizes homomorphic encryption, allowing multiple agents to solve a combinatorial optimization problem among them without leaking their private information. More specifically, multiple servers cooperatively perform dynamic programming procedures for solving a combinatorial optimization problem by using the private information sent from agents as inputs. Although the serves can compute the optimal solution correctly, the inputs are kept secret even from the servers.

The indistinguishable, homomorphic, and randomizable public key encryption scheme used in the implementation of their protocol is thoroughly explained. In this case, the most interesting and useful encryption property is the homomorphism, a structure-preserving map between two algebraic structures. To be more precise, an encryption is homomorphic if algebraic operations of decrypted text have an exact correspondence to the same operations applied to the encrypted text ($E(x)E(y)=E(xy)$). Hence, it is possible for servers to operate on the inputs sent by the users, without ever knowing their true meaning.

Two problems arise from the brief, unexplained, reference of Yamashita et al. [] to this work. First, one of its conclusions is that applying dynamic programming techniques to general combinatorial auctions is not feasible for large-scale problems, since it requires an exponential number of nodes. Testing for a large number of drivers in a complex city network remains to be done. Second, although, in anonymous auction, the general bids are kept hidden from the servers, the winning one is not. Assuming drivers are going to bid for links or routes, does that not mean that the privacy issues persist?

2.3. Road traffic simulation

Traffic system analysis through computer simulation dates from over fifty years ago (Pursula, 1999), growing since then to become an area of great importance in the field of transportation engineering. It allows for a detailed study of transportation models, too complex to be solved though analytical methods, satisfying the need for testing, evaluation and demonstration of possible scenarios.

Traffic simulation models can be categorized through a set of three fundamental properties: time, state and space. Different methods and theories gave birth to models with distinct combinations of these properties, where each one can be discrete or continuous. Another used classification distinguishes models according to the level of detail of the simulation. For example, a model can be macroscopic, if the traffic flow is treated as the basic entity and microscopic, if the behavior of every single vehicle is simulated independently.

Considering the kind of algorithms that Cosmo proposes, an individual simulation of each vehicle is mandatory. Therefore, the use of a microscopic simulator is required and, consequently, this research will now focus on documentation about that kind of software.

Chen and Cheng (2010) wrote a nice introductory work to the world of traffic and transportation systems, giving a thorough review of a wide range of agent-based approaches applied to a set of its aspects, such as modeling and simulation, dynamic routing and congestion management, and intelligent traffic control. The autonomy, collaboration and reactivity of agents are the main characteristics that render them such a suitable solution to these problems.

The reviews are divided into five parts:

- agent-based traffic control and management system architecture and platforms;
- agent-based systems for roadway transportation;
- agent-based systems for air-traffic control and management;
- agent-based systems for railway transportation;
- multi-agent traffic modeling and simulation.

The most interesting part for Cosmo is the last one, since it focuses on several projects which deal with similar problems, such as Adler et al. (2005), and mentions two open source agent-based traffic simulators:

- Multi-Agent Transport Simulation Toolkit - a toolbox for the implementation of large-scale agent-based transport simulations. It can be used for agent-based traffic flow simulation and other applications;
- Simulation of Urban Mobility - a portable microscopic road-traffic-simulation package designed to handle large road networks.

Finally, some visions for future research directions are presented. These stress the need for system interoperability, the ability to handle uncertainty, and system extensibility.

To deal with the first issue, the use of IEEE FIPA (Foundation for Intelligent Physical Agents) standards is advised. Its goal is to guarantee interoperability between agents by coordinating different aspects of systems, including system architecture, agent communication, agent management, and agent message transportation. To achieve this goal, FIPA provides specifications for an agent communication language to express exchanging messages, ontologies to define semantic contents of these messages, and agent platform architecture to support inter-agent communication.

Overcoming the weakness of uncertainty in dynamic environments can be accomplished through the use of mobile agents for ITS applications. The increased flexibility, achieved through their ability to adapt according to the circumstances and to transport themselves from one system in a network to another, results in strengths such as reduced network load, overcoming network latency, supporting disconnected operation, working in heterogeneous environments, and the ability to deploy new software components dynamically.

MATSIM

A group of top researchers in the field of road traffic simulation, including Kai Nagel and Kay W. Axhausen, are currently involved in a considerable project named MATSim. Their main goal: the development of the agent-based traffic micro simulator MATSim-T (Multi Agent Transport Simulation Toolkit). Its, claimed, key features:

- Fast and dynamic;
- Multi-modal traffic;
- Supports Large Scenarios;
- Versatile Analyses and Simulation Output;
- Modular Approach;
- Interactive Visualizer;
- Open Source;
- Active Development.

Balmer et al. (2009) give a detailed description, in terms of design, implementation issues and computational performance, of the different parts of this simulator. They base themselves on the results of a large-scale transport planning study, firstly presented in the work of Meister et al. (2008). In this, MATSim-T is used to simulate a complete workday of traffic for the whole of Switzerland, with roughly 2.3 million agents, producing around 7.1 million trips, assigned to a network model with about 60.000 links, in about 36 hours computation time.

One of the most important assumptions of MATSim is that it is not the vehicles per se that produce traffic, but the people who drive them. In fact, traffic optimization should be done considering a complete daily schedule, and the decisions behind it, instead of simply trying to find the best route for individual trips. Each person is modeled as an agent, and the sum of all agents should reflect the statistically representative demographics of the region. Activity demand is modeled and optimized individually for each agent, not only for some parts of the demand like departure-time and route choice, but as a complete temporal dynamic description of the daily demand of each agent.

The process of creation and optimization of agent demands, or plans, can be split up into four parts, presented in Figure 1:

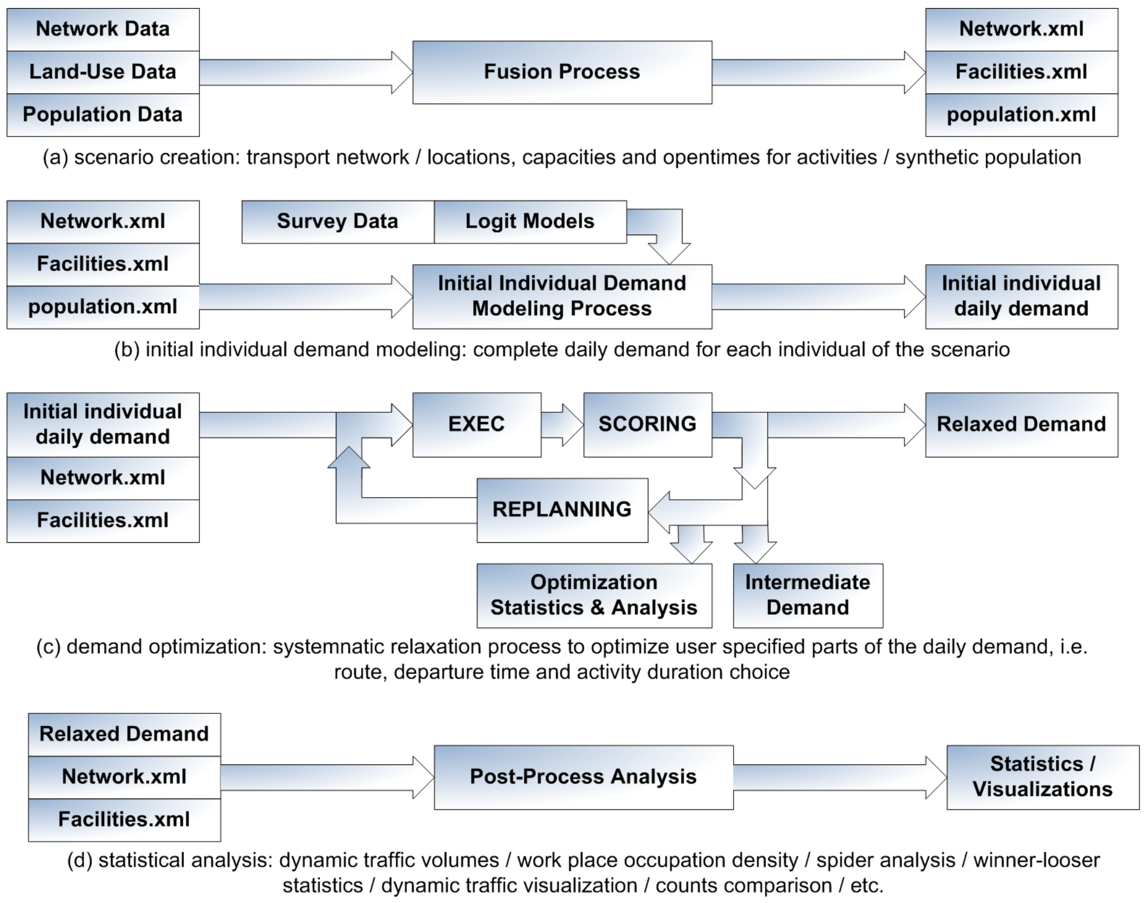

Network Data
Land-Use Data
Population Data
Fusion Process
Network.xml
Facilities.xml
population.xml
(a) scenario creation: transport network / locations, capacities and opentimes for activities / synthetic population
Network.xml
Facilities.xml
population.xml
Survey Data
Logit Models
Initial Individual Demand Modeling Process
Initial individual daily demand
(b) initial individual demand modeling: complete daily demand for each individual of the scenario
Initial individual daily demand
Network.xml
Facilities.xml
EXEC
SCORING
Relaxed Demand
REPLANNING
Optimization Statistics & Analysis
Intermediate Demand
(c) demand optimization: systemnatic relaxation process to optimize user specified parts of the daily demand, i.e. route, departure time and activity duration choice
Relaxed Demand
Network.xml
Facilities.xml
Post-Process Analysis
Statistics / Visualizations
(d) statistical analysis: dynamic traffic volumes / work place occupation density / spider analysis / winner-looser statistics / dynamic traffic visualization / counts comparison / etc.

Figure 1 - Demand creation and optimization process of MATSim

MATSim-T follows a modular approach, dividing the problem in logical parts (FUSION, IIDM, EXEC, SCORING and REPLANNING) and allowing users to rewrite each one separately, plugging them in as interfaces.

The FUSION and IIDM modules are responsible, respectively, for the parsing of all the available information about the scenario being simulated and the conception of the first set of plans for the synthetic population. Different kinds of available data can be dealt with by adjusting these modules to its form and detail.

EXEC, SCORING and REPLANNING are the modules in charge of the iterative demand optimization process, the third step in figure x. This cycle, considered as the main core of MATSim-T, consists of an evolutionary algorithm for the optimization of the complete daily plan of each agent, specifically, its routes, times, locations, sequence of activities, activity types and so on. First, the existing plans are handled by a corresponding traffic flow simulation module, in which the individuals interact with each other. Then, the utility, or fitness, of the execution of each plan is calculated. And finally, plans evolve, i.e., low utility plans are deleted, while others are duplicated and modified.

With regard to transport planning, MATSim considers the queue model (Gawron, 1998) as the most appropriate: every street is modeled as a queue, limited both in flow and storage capacity, in which vehicles have to wait for at least the free speed travel time on that street. The information produced by the simulations consists in a series of temporarily and spatially localized events that describe every single action for the entire population of agents (begin/end of an activity, entering or leaving a link, etc.).

Two different choices of traffic flow simulation, available for MATSim-T, are described by Balmer et al. (2009). The default one is a single CPU Java micro-simulation. It uses seconds as the smallest entity of time, in which all queues get synchronously updated. The alternative, deterministic and event-based queue-simulation (DEQSim), extends the queue model, adding more realistic dynamics of congested links. Furthermore, its implementation is event-based, that is, links are only updated at the exact time when a vehicle enters or leaves them. As such, the updating of links for every time step is exchanged by the computation of events produced by agents. However, as DEQSim is written in C++, the data sharing with MATSim-T has to be done through the hard disk. This is a big disadvantage compared to the default traffic flow simulation, and it is something to improve in the future.

Although MATSim is a really remarkable project, it does not fulfill the needs of Cosmo. Both its daily plan based simulation and its iterative demand optimization process are counterproductive to the extent that add unnecessary complexity. We are looking for a simulator that allow us to improve individual trips, prompting agents to make decisions in-real time, and not through a process of trial and error.

SUMO

SUMO ("Simulation of Urban Mobility"), one of the many available microscopic traffic simulators, is studied by Krajzewicz et al. (2002). Their work concentrates on the different principles, features and components of this software. It also describes the building, calibration and validation of a model for a real world scenario.

They label SUMO as a microscopic, space continuous and time discrete traffic simulator:

- it simulates the movement of every single vehicle on the streets, assuming an independent behavior determined by both driver and vehicle capabilities, using a model developed and described by Stefan Krauß (1998);
- vehicle position is described by a floating point number, opposed to a cellular automata kind of system, where vehicles jump from cell to cell;

- simulation variables, or system state, are updated by steps, with a default 1 second time interval;

It offers features such as a collision free vehicle movement, different vehicle types, multi-lane streets with lane changing, junction-based right-of-way rules, lane-to-lane connections, XML-input/output and routing in dynamic networks. It also allows for static or dynamic traffic light control of junctions.

Furthermore, three different types of possible outputs are identified:

- the most complete output contains all edges and all lanes together with vehicles driving on them for every time step. Each vehicle is described by their name, position and speed. An XML-compliant format facilitates its use as input to post-processing tools for further evaluation. The downside is the insurmountable amount of data produced for large simulation;
- common values used in traffic research, such as flow and average speed, aggregated for specified time intervals which may be changed by the user;
- the same values as before but computed for specific positions in the network, defined by preset detectors or induction loops.

Being an open source software is one of the greatest strengths of SUMO, and its portability is said to be guaranteed by its standard C++ implementation. Additionally, the development in modules greatly simplifies the understanding, changing and extension of SUMO. The ones described in the paper are:

- SUMO, responsible for reading scenario input information, simulating traffic flow and saving the results in different types of output files;
- NETCONVERT, used for the processing of different types and formats of network descriptions, computing streets, lanes, right-of- way rules, lane-based street to street connections and traffic lights for junctions, and writing the results to an XML file that the SUMO module can understand;
- ROUTER receives the departure times, origins and destinations for a set of agents and computes the routes through the network using the well-known Dijkstra algorithm. It is also said to use the Dynamic Traffic Assignment approach, taking into account speed changes due to traffic load, achieving a real world behavior of drivers through the repetition of routing and simulation;
- SUMO-GUI extends the original simulation package by adding a graphical interface.

Validation and calibration were done with the simulation of a real life scenario, namely, a part of the I-880 American highway. This specific part was chosen because of its many detectors, which allow the measurement of mean speed of passing vehicles.

The network was modeled by hand, including a direct mapping of the real detectors to SUMO induction loops. Then, a small program was built for the calibration, comparing simulated travel times to the detector data available from the real world and minimizing the error. This program would create a file, with the vehicle types and parameter definitions used within the simulation, and a metafile containing the description of the calibration process.

The result was a model that reflected reality with an approximately 15% error, down from an error around 40% with no calibration. This work shows that, when using real world data, the calibration process is of vital importance for a reliable outcome. A process that is achievable with small effort with the use of SUMO.

According to Krajzewicz et al. (2006), SUMO is being developed by the Institute of Transportation Research (IVF) at the German Aerospace Centre (DLR) since the year 2000. As of 2003, the IVF has been working on TrafficTower, a virtual traffic management center and a host for several, large-scale, traffic management projects. Thanks to a very fast but still valuable traffic simulation model, able to simulate around 100.000-200.000 vehicles in real time on a common desktop PC, SUMO has the potential to simulate substantial traffic scenarios for these projects.

Two projects are addressed in this paper:

- INVENT, a project in which SUMO was used to simulate the city of Magdeburg and the highway ring around Munich, as well as to develop and test modern traffic management scenarios. New types of dynamic traffic light systems and a traffic rerouting system based on dropping loads from the streets were implemented. The rerouting system required two new extensions, of both the simulation and routing applications, to share information during the simulation runs;
- WJT, Weltjugendtag or world youth day, was an important event held in August 2005, in the area of the city Cologne, in Germany. About 1.000 highway detectors, 100 city detectors and a zeppelin were used, by the IVF, to collect data about the current traffic, created by over 1.000.000 participants. An extended version of SUMO used this information to predict the situation in half an hour in the future and present it to the local authorities;

SUMO is also presented as a possible solution for traffic management and simulation of catastrophe scenarios. It already comprehends the rerouting of vehicles and road closures, and the current development of a multi-modal traffic simulation, with transports such as trains and public busses, will be useful to simulate the rescuing of pedestrians. Further needed structures include an application to simulate the catastrophe itself, changing the situation of the road network, a process through which the solver can enter his actions and internal measuring methods to rate its performance.

In conclusion, SUMO is a very interesting, and easily extendable tool, for simulating large traffic scenarios and testing automatic management algorithms.

Integration of simulators

To better integrate behavioral models of human travelers reacting to traffic patterns with control measures of these traffic patterns, focusing on distributed and decentralized methods, Bazzan et al. (2009) came up with the idea of combining MATSim and ITSUMO systems. Their work follows previous attempts to integrate traffic assignment and control, demand and supply. Thus, the focus of ITSUMO in short time control by means of agents in charge of optimization of signal plans helps strengthening MATSim, which has a much more solid planning side, with its optimization cycle of complex agent strategies. The integration methodology is displayed in the following Figure 2:

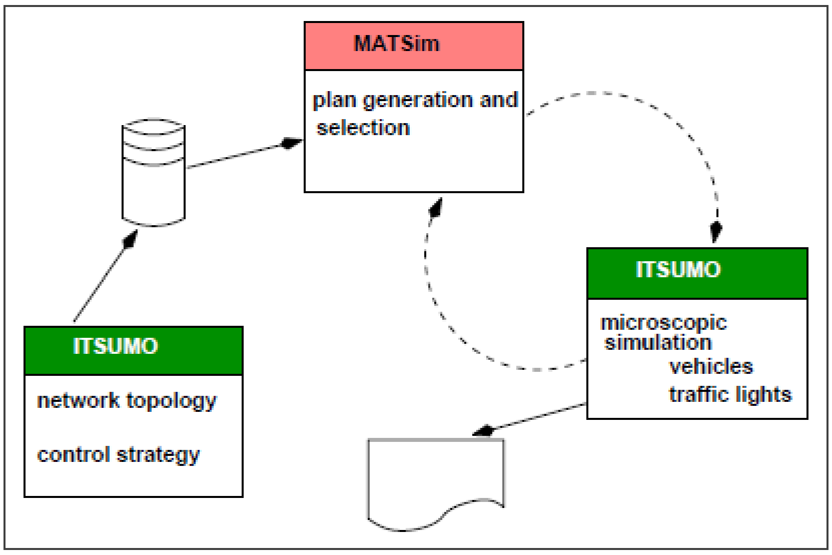


Figure 2 - Integration of MATSim and ITSUMO (Bazzan et al., 2009)

Both simulators are briefly described but, with the exception of one particular feature of ITSUMO, everything else has already been mentioned in the previous reviews. This feature is defined as an independent module for the control of traffic lights. It establishes a socket-based communication with the simulation kernel, receiving the necessary information to respond with the appropriate control actions. In fact, the traffic control interface module of ITSUMO (TraCI) is now much more than that. The vast amount of information that can be retrieved, in real-time, about every single aspect of the system state, can now be used to introduce a wide range of changes to the simulation. These include, for example, the revision of network configuration, the adjustment of agent properties and plans, or even the addition of new vehicles. TraCI is an incredibly useful tool for the implementation of extensions, for ITSUMO, that intend to control a simulation in real-time, as is the case with Cosmo.

The developed tool was tested using a simple case study, which deals with the simulation of 400 and 700 driver agents in a 6x6 grid network, testing for combinations of fixed/adapting drivers and fixed/greedy traffic light approaches. The obtained results led them to conclude that, although the use of adaptive day-to-day drivers together with the greedy plan for traffic lights provides the best performance, the additional gains, in relation to the use of only one dynamic method, are not significant.

In the end, even though presenting a really interesting approach, their system ends up using the very same properties that led us to turn down MATSim.

Chapter 3
Cosmo Platform Development

From the start, this project aimed for the completion of two main objectives, namely, the conception of innovative algorithms to help manage road traffic and optimize its flow, and the implementation of a platform to test them with. Therefore, after clarifying the purposes of Cosmo and planning for the year ahead, the first task would be to research, compare and finally choose a third party simulation program, on top of which it would be possible to implement a structure capable of running information distribution algorithms.

3.1. Simulator comparison

Considering the fact that the program we were looking for would have to simulate individual agent behavior, a mandatory microscopic scope was the first and most clear restriction. A macroscopic simulator would only simulate the traffic flow as the basic entity, making it impossible to plan individual routes, and a submicroscopic model would add immense and useless complexity.

With this in mind, a first general research led to a few noteworthy micro-simulators: Aimsun, CORSIM, DRACULA, MATSim, MITSIM, PAMINA, Paramics, SUMO and VISSIM. One of the greatest inconvenients of most of these simulators is in the fact of them not being open source, making them undesirable, both in terms of access to the code and utilization costs. Also, PAMINA was discarded by its total lack of available documentation, as well as reviews from previous applications. Thus, we agreed on a deeper exploration of the remaining simulators (MATSim, MITSIM and SUMO):

MATSim (Multi-Agent Transport Simulation Toolkit)

Advertised as a fast dynamic and agent-based traffic simulator, it is prepared to conduct event-driven simulations, with the possibility of parallel event handling. Moreover, it has the advantage of being open source and having a modular approach which allows it to be easily extended by new algorithms. It is written in Java and, supposedly, runs on any operating system for which Java and the Eclipse IDE are available, including Windows XP and above, Mac OS X and GNU/Linux. It also has lots of recent testing, for example the simulation of a network model with approximately 1 million links of Switzerland populated by around 6 million agents (Meister et al., 2008; Balmer et al., 2009), and documentation.

As shown in Figure 3, it begins by compiling information regarding the network, population and intended interactions between them, i.e., the set of all daily activities planned by every actor in the simulation. The second phase, in blue, is a genetic algorithm, which iterates an execution, scoring and re-planning cycle, until it reaches a steady state that would best represent the optimum traffic flow in the network.

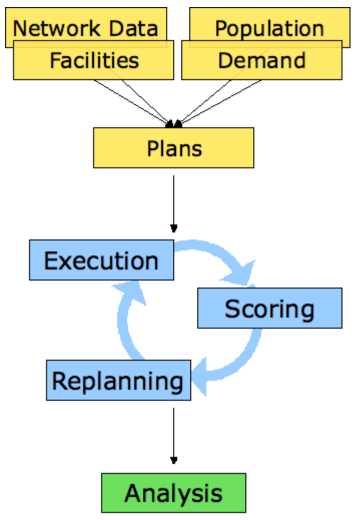


Figure 3 – Schematic of the execution flow of MATSim

MITSIM (Microscopic Traffic Simulator)

MITSIM is, together with the correspondent traffic management simulator and graphical user interface, a part of MITSIMLab: a simulation suite developed by the MIT Intelligent Transportation Systems Program. Its code is in C++ and it uses the Parallel Virtual Machine (PVM), a software package that connects a group of Unix and/or Windows machines to create a single virtual parallel computer, in order to effectively solve large computational problems. Linux is the only operating system it runs and compiles in, depending, specifically, on a Redhat Linux 7.3 distribution to compile the source code. Interactions between MITSIMLab modules are presented in Figure 4.

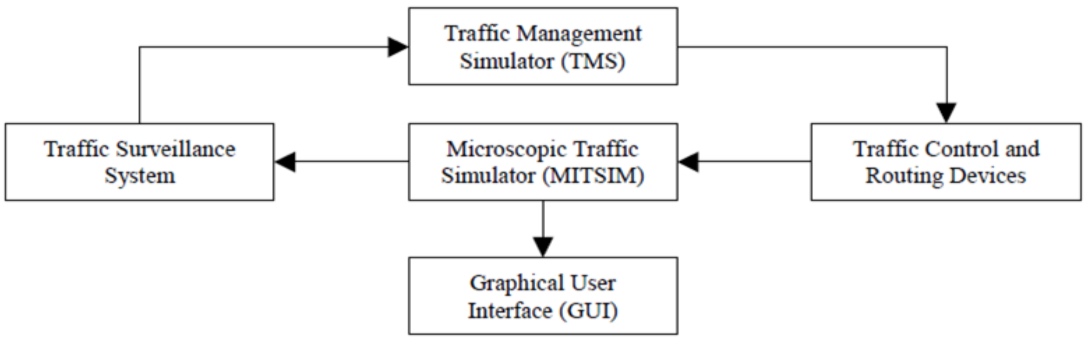

Traffic Management Simulator (TMS)
Traffic Surveillance System
Microscopic Traffic Simulator (MITSIM)
Traffic Control and Routing Devices
Graphical User Interface (GUI)

Figure 4 – Representation of module interdependency in MITSIMLab

One of the known MITSIMLab applications was in Stockholm, Sweden, for the entity responsible for traffic planning and operations within the city (City of Stockholm Real Estate and Traffic Administration or GFK) (Ben-Akiva et al., 2002).

SUMO (Simulation of Urban Mobility)

SUMO is an open source microscopic road traffic simulation package, mainly developed by employees of the Institute of Transportation Systems at the German Aerospace Center. It is implemented in C++ and runs on, at least, Linux and Windows operating systems. The main module, Sumo/Sumo-GUI, receives information, about the network and agent routes, through XML files and performs a space continuous and time discrete simulation (Krajzewicz et al., 2002). Other modules help building the network (Netconvert and Netgen), planning the routes (DFRouter, DUARouter and JTRRouter) and even controlling a running simulation through a TCP connection with SUMO (TraCI). Successful testing has been accomplished for the city regions of Magdeburg and Cologne (Krajzewicz et al., 2006).

The first choice - MATSim

At the end of this investigation, MATSim was chosen for various reasons:

- it was the most recent project, making it possible for its researchers to have a deeper knowledge, working from all the other previous work done in the area;
- the one with more publications, where its workers strive for a better simulation of the reality, providing our future algorithms a more solid base and higher validity;
- and its vast amount of documentation allied to the convenient fact that it is written in Java, made it seem the easiest to alter according to the needs of the project.

Consequently, the first thing to do should be to probe the code of MATSim, aiming at a better understanding of its inner workings, thus enabling its extension. This turned out to be a bad decision since, as later realized, MATSim clearly did not suit our needs. It was a mistake, caused by the insufficient research done in the area, which resulted in a significant waste of precious time and effort. Still, as it was a substantial part of the work employed in the beginning of the project, the initial exploration of MATSim is presented in the following section.

3.2 Exploration of MATSim

After getting the code with the help of the Eclipse IDE, and the respective plugins offering subversion and Maven support, it was possible to ascertain the existence of a controller module responsible for running MATSim itself. The class Controller.java is, among other things, in charge of initializing a logger, getting the required scenario and configuration information from the correspondent files, loading the necessary modules and listeners, calling each of the simulation iterations, done by mobsim (mobility simulator), and handling its results, and finally shutting down the program with all its implicit preparations. Other important modules include the one that contains general information about the simulation (global), the one that defines which re-planning modules will be called or stopped and when (strategy), and calculator modules like planCalcScore and travelTimeCalculator.

When executed, MATSim starts by loading all the information about the network and population, with its planned trips or demands, and initiates a cycle where the main goal is optimizing an adjustable plan scoring function. This cycle, visible in blue on figure 3, comprises 3 stages:

- execution, which consists of an entire simulation given the current plans;
- scoring, or the analysis and value assessment of the results obtained in the previous simulation;
- and re-planning, which tries to create a new and improved set of plans to optimize the scoring function, while being bound by several restrictions.

The first negative thought about the use of MATSim was the impression that its simulation iteration was not serving our purposes. We were only looking for a single continuous run, where the individual routes of the agents could be altered at any time. And that proved very hard to do, considering the structure of program. In fact, one of the people directly involved in the development of MATSim, contacted by my advisor, was really skeptic about the way we intended to modify it. Additionally, it was impossible to determine the exact position, at any given time, of an agent, being that the only information available was its current link (road). On the contrary, SUMO is space continuous, which permits the obtaining of the precise coordinates of each agent.

Figures 5 and 6 show a graphical representation of the simulations of MATsim and SUMO, respectively. Their interpretation can help us better understand the considerable advantage of the amount of information provided by SUMO over the one outputted by MATSim:

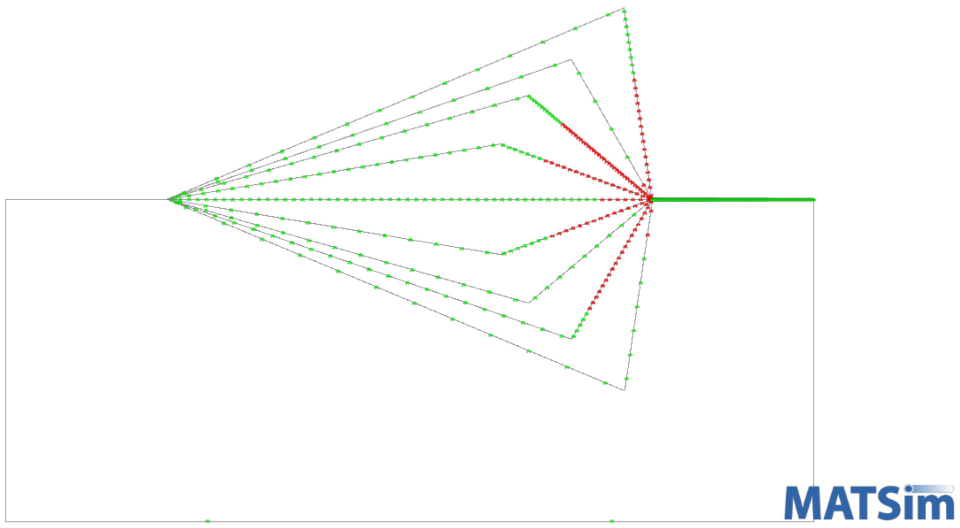


Figure 5 – Snapshot of the graphical simulation of MATSim

while MATSim only displays the occupation of each link, represented by the amount and color of the dots in each line (more dots and the red color for higher occupancy),

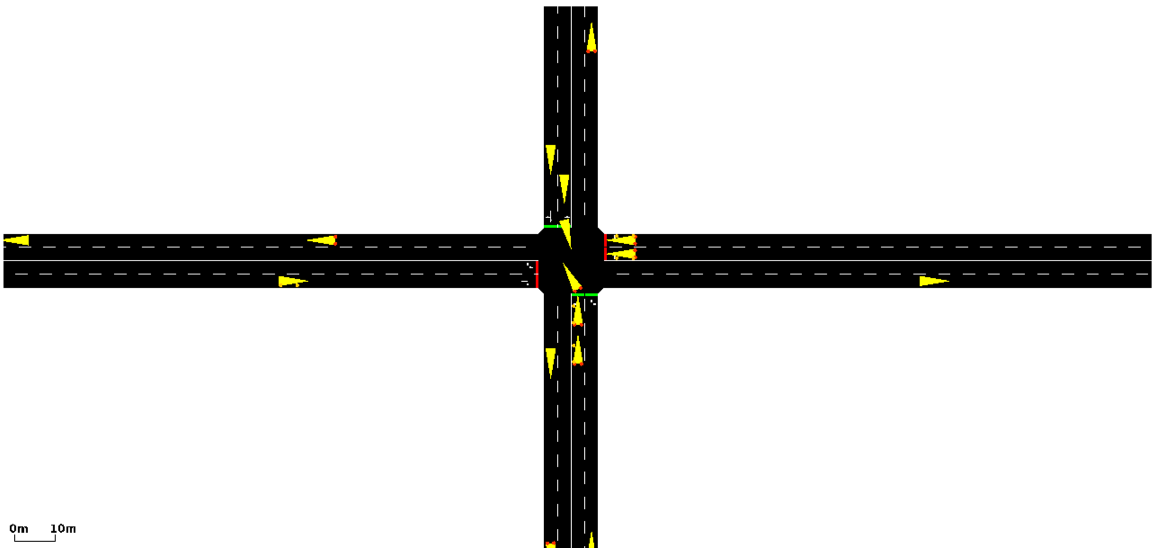


Figure 6 – Snapshot of the graphical simulation of SUMO

SUMO illustrates every vehicle, as yellow triangles in this case, in their current position.

This, along with the fact that the simulation in SUMO can be easily controlled, in real-time, thanks to an interface which provides access through a TCP based client/server architecture (TraCI), proved my preliminary study on the simulators insufficient. Ultimately, even taking into account the lost time of a late simulator change, the decision was made to adopt SUMO.

3.3 Description and usage of SUMO

The XML files required to run simulation in SUMO are presented in Figure 7.

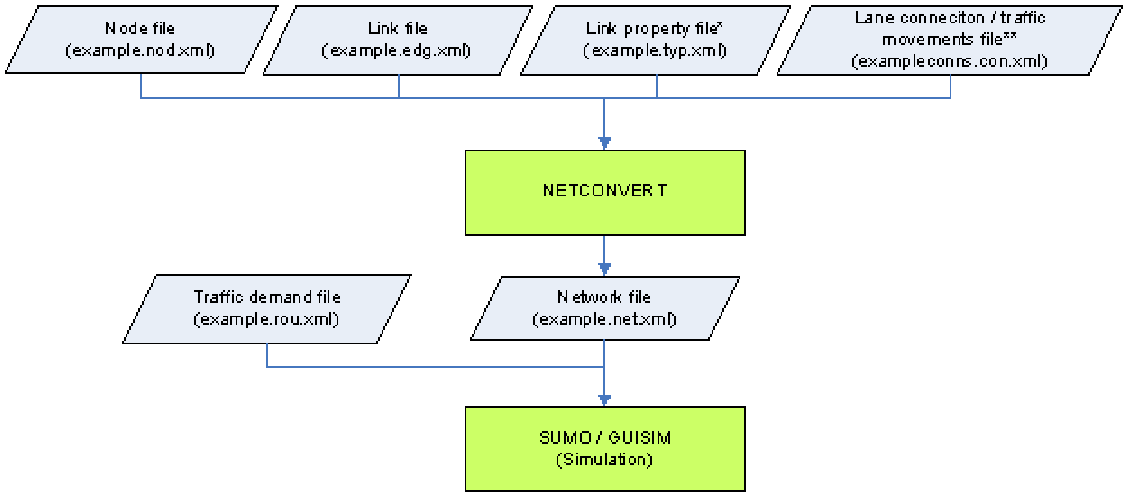


Figure 7 – Required input files for a simulation in SUMO

Of the four files used as input for the initial processing, with NETCONVERT, only the first two are mandatory. While the data stored in the link property file can also be added the link file, the lane connection/traffic movements file is optional. It is only used to change a few, otherwise, default values. Furthermore, the first row is only valid when manually building a network, and can be simply replaced by importing a variety of foreign networks (e.g. VISUM, ArcView and OpenStreetMap networks).

The traffic demand file contains all the agents that will participate in the simulation as well as their individual characteristics and planned routes. This file can also be automatically made through a diversity of methods, one being the use of the file randomTrips.py, available in the official site of SUMO, combined with the module DUARouter.

Lastly, configuration files for the running of SUMO modules may likewise exist. For example, a graphical simulation can be executed using the command: sumo-gui -c sim.sumo.cfg, where the file sim.sumo.cfg contains the instructions otherwise given in the command line. These include the path of network and traffic demand files, possible starting and finishing times and output options.

SUMO networks are constituted of one way edges or links, with its own characteristics like priority, number of lanes and maximum speed allowance. In the same way, the nodes they connect have a mandatory x and y coordinates along with other possibly defined attributes, such as type. This in particular allows the addition of traffic lights to a simple intersection.

The traffic lights can have a static behavior, defined in the network XML file, or be dynamically controlled during the simulation, using TraCI. In fact, dynamic control of traffic lights in a network is another well researched way of optimizing the traffic flow (Chen and Cheng, 2010).

There can still be restrictions as to how the edges are connected, i.e., linking certain lanes or prohibiting certain routes. In turn, an agent, defined as a vehicle in the traffic demand file, also has attributes, like the departing time, maximum acceleration and deceleration, driving skill and vehicle length and color. Apart from this, each agent is associated with a route, consisting of a list of edges, through which it will have to pass.

Once sufficient knowledge of the basic functioning of SUMO was gathered, TraCI (Traffic Control Interface) became a priority. When executed with the --remote-port option, SUMO gets the simulation ready and waits for an external application to take control. This client application connects to SUMO, via TCP, at the specified port, and starts a flow of messages, each containing one command that will influence or ask for information details about the simulation. The existing data retrieval and state changing commands, from getting the state of a traffic light to setting a new route for a specific vehicle, and respective messages, are well documented. There is also an interface implementation using Python, which was the one used as a basis for our extension.

The use of Python was debated regarding its lack of efficiency compared to other programming languages, but it was eventually chosen, bearing in mind that it had the best available interface and that the time was beginning to run low.

Running a SUMO simulation

In practice, the use of the testing platform is divided into two major steps: scenario construction and simulation execution. The first part includes the generation of the three indispensable files.

1. The network file can be obtained thanks to the netconvert module. It imports digital road networks from different sources and converts their information, so it can be readable by SUMO. The following commands are examples of the creation of a network file named lattice.net.xml:

```
netconvert --xml-node-files=lattice.nod.xml --xml-edge-files=lattice.edg.xml
--output-file=lattice.net.xml

netconvert --osm-files= lattice.nod.xml --output-file= lattice.net.xml
```

The first is used to convert node and edge data, stored in XML files, and it is used for hand built networks. More on how to build these custom networks is available in the section 2.1 of the tutorial http://sourceforge.net/apps/mediawiki/sumo/index.php?title=QuickStart Tutorial. The second allows the conversion of OpenStreetMap files. To learn all the options of netconver just call netconvert --help.

2. There are several methods of acquiring the trips file, including setting by hand all of the drivers properties and plans. For this project a python generator of random trips was used. In fact the randomTrips.py file, provided by SUMO in its official site, was upgraded to allow

the generation of several agents per step, a feature not present in the original version. This way, only a simple command is needed for the generation of an entire set of plans:

```
Python randomTrips2.py -n lattice/lattice.net.xml -r lattice/lattice.rou.xml -e 100 -p 1 -B 1
```

In the aforementioned example 5 different inputs are given, respectively, the file paths for the existing network and the desired output, the generation end time, the repetition period and the batch size. This will spawn 100 agents, 1 (-B) every time step (-p) until the 100th (-e).

3. The creation of the configuration file needs to follow an XML structure, as shown in the example below:

```
<configuration>
	<input>
		<net-file value="lattice.net.xml"/>
		<route-files value="lattice.rou.xml"/>
	</input>
	<output>
		<tripinfo value="lattice.out.tripinfo.xml"/>
	</output>
	<traci_server>
		<remote-port value="8814"/>
	</traci_server>
</configuration>
```

Further recognized options can be obtained by calling sumo –help or a file with the default settings can be saved by calling sumo –save-template <file> --save-comented.

When all of the above files are ready for execution, the running of the program itself is achieved with a simple command, e.g.

```
python cosmoController.py ../simulations/lattice/lattice.sumo.cfg,
```

where lattice.sumo.cfg is the name of the previously created configuration file, and ../simualtions/lattice/ its relative path. Later adjustments in the simulation, added by the implemented extension, can be done through the modification of the global constants present in cosmoConstants.py.

Planning of the testing platform

Finally, the extension of SUMO was planned. Its structure tries to mimic the reality of a traffic control system:

- a Controller module that has access to the state of the network at any given time, including a list of active agents and the occupancy for each edge, and that can alter its state in various ways;
- a Population class which is responsible for controlling the behavior of each agent during the simulation and handle the communications between them;

- the Agents themselves, which must re-plan their routes, taking into consideration their individual views of the network;
- and the Network, or how it is perceived by the controller and by each agent, with its graphs of edges and respective weights.

For a better understanding of design, the final version of the class diagram is available in the section Appendix A. Further details on the implementation of the platform will be discussed in the following chapter, Results.

Implementation of routing algorithms for the optimization of traffic flow

Only when the platform was ready for experimentation, was the second phase of the project finally carried out. At that point, four different routing algorithms were created to test our hypothesis, i.e., the possibility of traffic flow optimization without access to private individual location information. All of the implemented algorithms are based on the known Dijkstra algorithm, diverging only in the assignment of weights to the different network links.

The first routing algorithm simply finds the shortest paths, hence, assigning each link the weight of its own distance. It is just the greedy and naïve approach, paying no attention to traffic information whatsoever, and will only be used for comparative purposes.

The second represents the solution of current individual use of GPS devices, with access to traffic information, by each user. It is precisely what we are aiming to enhance, avoiding the creation of unnecessary traffic jams, caused by the instinctive selfishness of drivers. It is presently programmed to add to the weight of each link, an amount equivalent to the square of its own occupation, measured as the space used by existing vehicles.

In the third method, the perception of the network by the agents is drastically changed. The server virtually blocks any road that has an occupancy greater than an adjustable threshold, preventing users of choosing paths that comprise them. Basically, it is a simple distance-based Dijkstra algorithm, applied to a network without the links with already too much traffic, hopefully avoiding future congestions.

The fourth and last algorithm is very similar to the second. The difference is in the fact that the traffic information is altered, i.e., the server changes the individual driver perception of the network, changing the occupancy values for each link. This is done through the addition of a new random weight, proportional to the standard occupancy of each link. The randomness of this method can be adjusted through a variable created, specifically, for the purpose.

Currently every agent in a given simulation will use the same algorithm, chosen through a global variable. However, since the percentage of drivers that use the route re-planning process can be setted, it is possible to have a simulation where a part of the population uses the SUMO default routing, that is, sticking with the initial shortest path, while the rest uses one of the other available options.

Chapter 4
Results

4.1. Obtained testing platform

As exposed in the previous chapter, the resulting program, coded in Python, makes use of the TraciControl interface to communicate with, and influence, a running simulation. It is executed with one, and only one, argument: the name of the simulation configuration file (*.sumo.cfg following the recommended naming conventions). With this information, the main class, Controller, begins parsing the mandatory simulation, network and traffic demand configuration files, saving the relevant data to memory, including the network graphs and the preliminary attributes for each agent in the population. If, any time during the analysis of these files, any problem arises, the program is immediately terminated, which would also happen with a simple SUMO simulation anyway. Additionally, the --remote-port option with a valid value is equally required. The beginning and ending times for the graphic simulation, if present, can also be parsed from its respective configuration files.

Once the said necessary preparations are in order, an optional SUMO graphical user interface process is opened and a TraCI communication is established, preceded by the execution of a cycle that pretty much encloses all the real simulation. As soon as this cycle is completed, the former connection is finalized and a few statistical values are printed to an output file. Presently, these consist in the average duration and length of all trips, as well as the average and standard deviation of network occupancy. While the conclusion time is not reached (3600 seconds/steps by default) or there is still active agents in the network, the cycle will repeat the following instructions:

1. order SUMO to advance 1 second in the simulation and update the respective variable;
2. count down the time for any potential stopped agents, due to previous accidents;
3. update the perception of the network by the controller, keeping its edge weights/occupancies up to date, and ask the Population to compute any actions taken by agents, providing, in return, information about the ones that just departed or arrived at their destinations;
4. cause accidents, taking into account a global and constant probability.

Following the third point of the aforementioned cycle, and based on the received arguments, the Population initiates its role by updating the state of any referenced Agent. Subsequently, it cycles through any Cosmo user, checking its state and performing actions accordingly:

1. if the Agent is active, update its perception and positioning information and, taking into account the check route probability, re-plan the path it is currently following;
2. if the Agent is stopped due to an accident, decrease its waiting steps by one, resuming its simulation in case they are already over;
3. if the Agent has pending communications, dispatch them to the respective recipients.

The attribute that determines if an Agent uses the Cosmo functionalities is set, while parsing the corresponding configuration file, with an adjustable, global and constant probability.

Apart from the first active step of an Agent in the simulation, when it has a 100% probability of re-planning its path, the check route probability is given by the combination of two functions. One that is proportional to the number of steps since that Agent last checked its route, in other words, which grows with time, together with another that varies with the current speed of the vehicle, reaching maximum percentage values for both minimum and maximum speed values. With this, we hope to better simulate the behavior of a real driver, with a higher tendency to re-use its GPS with time, when it is stopped or even when it reaches its maximum velocity, meaning it has no obstacles ahead.

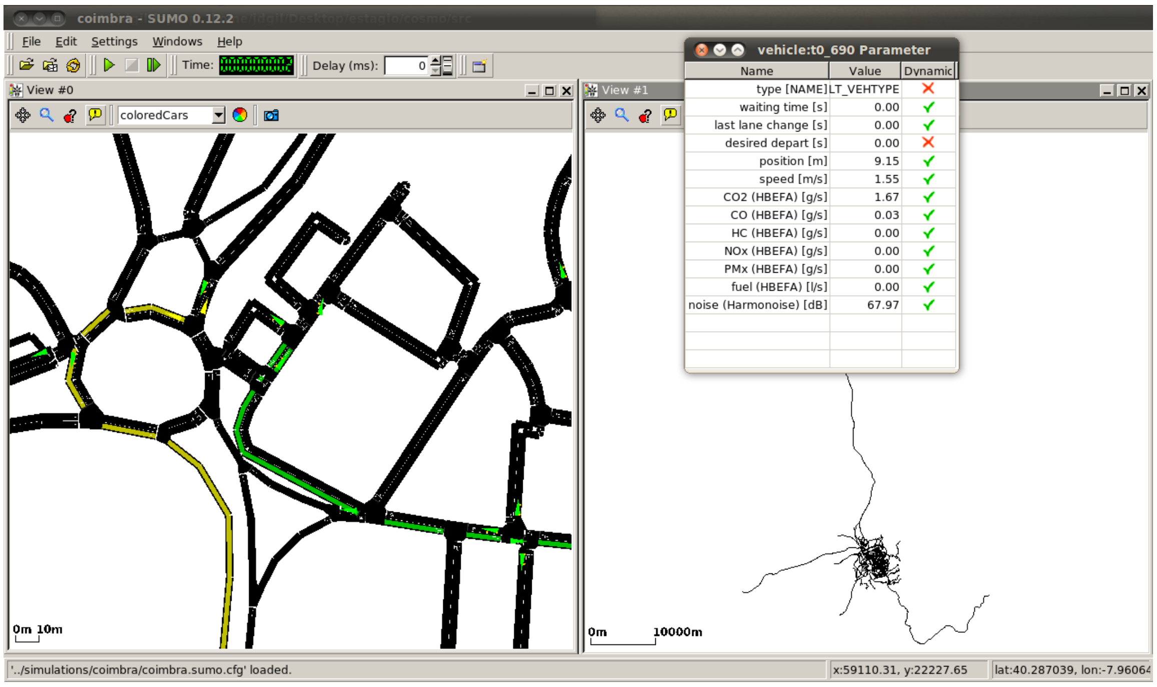
Each Agent is individually responsible for the re-planning of its own route, ultimately distributing, in practice, the majority of the required computation between the devices in each vehicle. On the communications side, each one of them is currently being sent to any Cosmo user less than 100 meters (default value) away from the transmitter. If an Agent has already received that message it will simply be ignored, else it will be added to its own pending communications and sent in the next simulation step, thus implementing a broadcast system.

The most important program constants can be easily modified in a separate file created for that purpose: cosmoConstants.py. These currently consist in:

- distinct vehicle colors for each of its different states;
- accident probability and stop time;
- Cosmo usage percentage;
- variation of the route check frequency;
- broadcast distance;
- the acceptance probability of a newly computed route;
- the used routing algorithm;
- link occupancy limit and the percentage of weight variation, for the third and fourth routing algorithms, respectively.

Figures 8 through 11 contain examples of a traffic simulation in the city of Coimbra, starting with a general view of the SUMO graphical interface. In this it is possible to observe several interesting features:

- current simulation time step;
- play, stop and next step buttons;
- adjustable display frequency, measured in steps for second;
- distinct simultaneous views of the network;
- the highlight of tracked individual routes;
- and some current parameters for a chosen entity.



vehicle:t0_690 Parameter

Name	Value	Dynamic
type [NAME]	LT_VEHTYPE	✗
waiting time [s]	0.00	✓
last lane change [s]	0.00	✓
desired depart [s]	0.00	✗
position [m]	9.15	✓
speed [m/s]	1.55	✓
CO2 (HBEFA) [g/s]	1.67	✓
CO (HBEFA) [g/s]	0.03	✓
HC (HBEFA) [g/s]	0.00	✓
NOx (HBEFA) [g/s]	0.00	✓
PMx (HBEFA) [g/s]	0.00	✓
fuel (HBEFA) [l/s]	0.00	✓
noise (Harmonoise) [dB]	67.97	✓

Figure 8 - General view of the SUMO graphical interface

Figure 9 gives us, with a little more detail, a global view of the virtual network of Coimbra in a SUMO graphical interface. This specific network was generated using OpenStreetMap information.

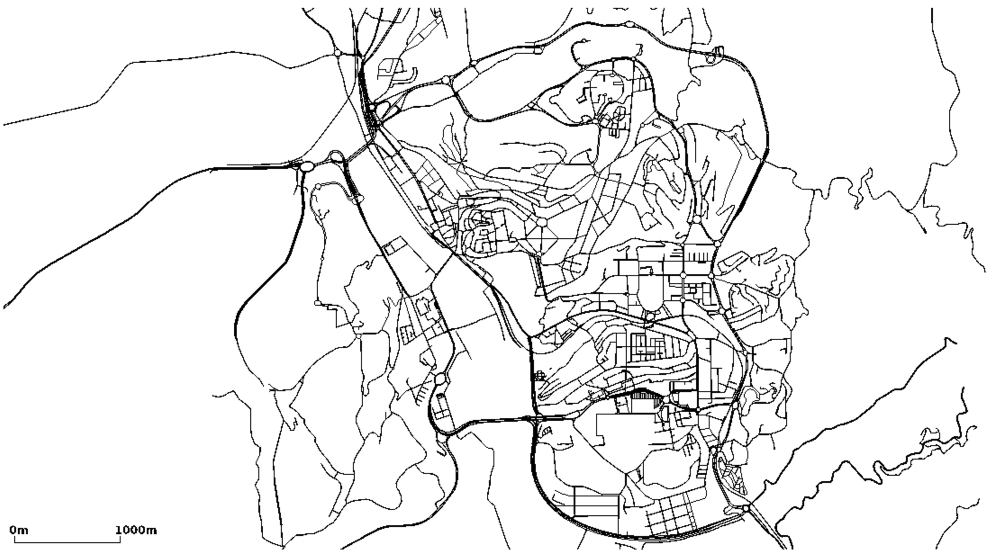


Figure 9 – Global view of the network of Coimbra in a SUMO graphical simulation, using the implemented extension

The drivers whose vehicles are represented in green, visible in Figure 10, are the ones that relied on traffic optimization algorithms and changed their routes.

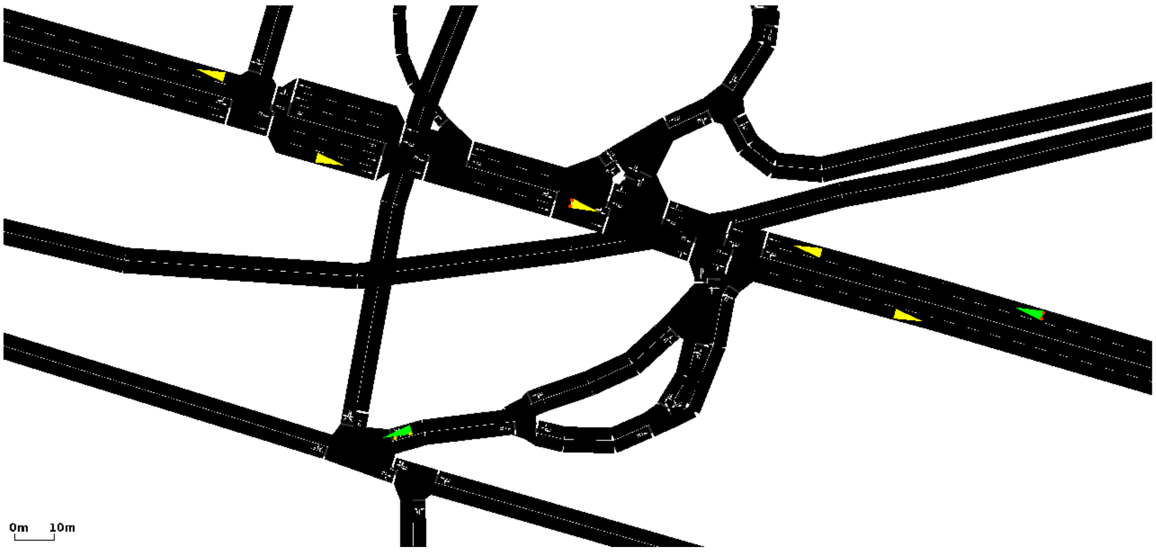


Figure 10 – Example of agents who changed their routes during the simulation, depicted as green triangles

Finally, the red triangle in Figure 11 indicates a stopped agent, as a result of an accident, already causing a disruption in the usual traffic flow.

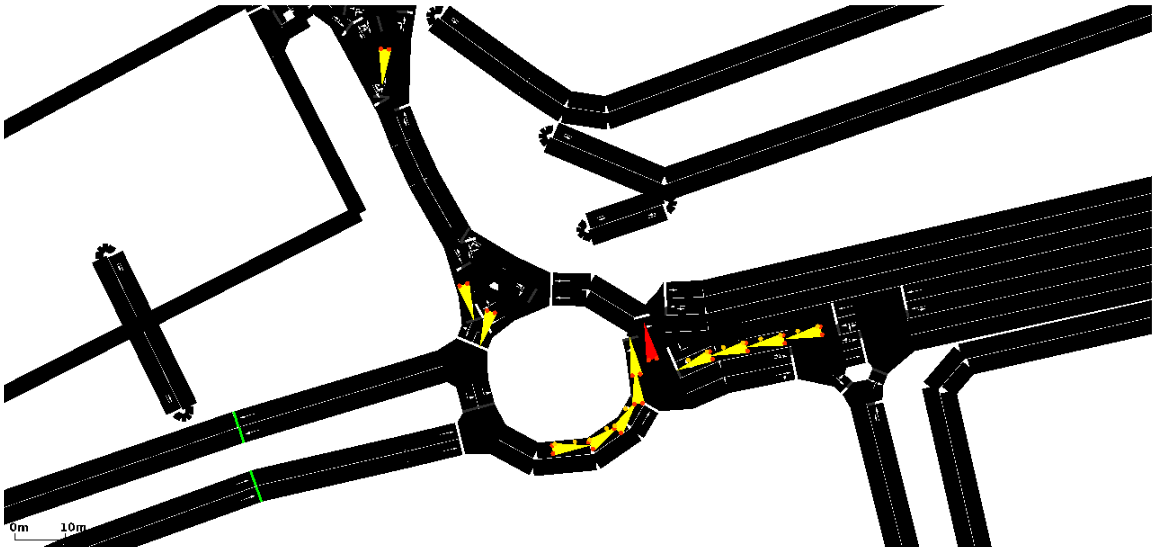


Figure 11 – Example of an accident during the simulation

4.2. Algorithm comparison

The testing of the four traffic flow optimization algorithms consisted in the simulation of two separate sets of plans, for 1000 and 2500 drivers, in two different networks, the lattice and the radial and ring, illustrated in Figures 11 and 12, respectively. Both plans only generate 1 driver per step/second. Each of the four combinations between the networks and the plan sets was simulated 40 times, that is to say, 10 times for each algorithm, to enable the averaging of the results and thus ensuring a minimum of statistical significance. The results are available in the Appendix B section of this report.

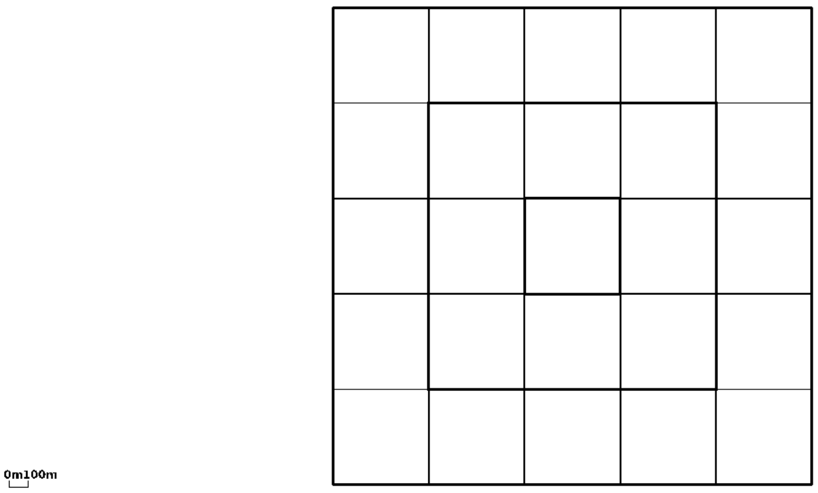


Figure 12 - Snapshot of the lattice network

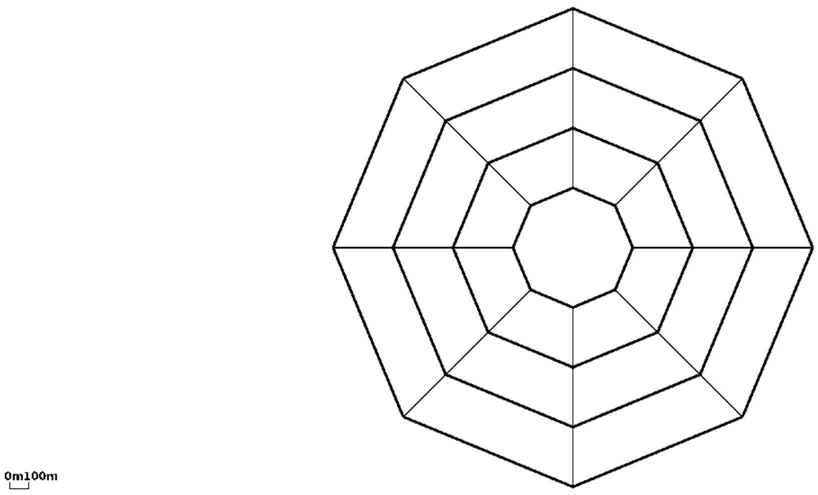


Figure 13 – Snapshot of the radial and ring network

The results show that the road blocking algorithm, for these specific networks and for an occupancy threshold of 25%, is as just as bad as the shortest path one, in other words, it is the same of not using any traffic information. On the other hand, the weight altering algorithm, with a 50% weight variation, shows no advantage towards the one that uses the real link occupancies. This means that, with its present implementation, it fails to boost traffic flow beyond what it is currently being achieved by individual vehicle GPS devices. Consequently, the use of private location information proves irreplaceable.

Although these outcomes seem rather disappointing, there is still much space for improvement. Each of the perception changing algorithms can be greatly enhanced, both in terms of implementation and calibration. Plus the testing could be more extensive. For instance, it is possible that better results would be obtained with a larger network overload, i.e., with a greater and/or faster generation of drivers to the simulation.

Even if it is impossible to achieve the same results of algorithms that use private location information, with a little more effort, it is certainly possible to work towards it, and maybe prove the creation of privacy issues unnecessary.

Chapter 5
Conclusions

This report described the implementation process of a platform for testing traffic flow optimization algorithms, as well as the conception of a few of these algorithms themselves. The test bed was built on top of the road traffic micro-simulator SUMO, allowing real-time interaction with the simulation. The rerouting of vehicles during the simulation and the replication of accidents are examples of added functionality.

The results obtained from preliminary algorithm testing were not as good as expected, having failed to obtain a tangible advantage of perception altering algorithms over simple individual routing algorithms with general traffic information. Nevertheless, this project serves essentially as the basis for future research. That is its true value, a stepping stone in an area that still has much potential to explore.

A lot of work remains to be done, both on the test bed and on the traffic flow optimization algorithms. Some examples of possible developments are:

- implementation of dynamic traffic lights to help with traffic management. This is a well-researched approach in the area, and there is no reason not to add it to the general solution;
- enhancement of the coded communication process and its use as additional help to traffic management. For instance, by broadcasting real-time accident information among the drivers, these can avoid areas with potential future congestion;
- general improvement of the platform implementation, to better simulate reality. For example, the mechanism responsible for the route checking by drivers, or route check probability, can be altered to include other functions. The increase in probability with the number of visited links is just one among many possibilities;
- improving existing algorithms, perfecting their execution and/or fine-tuning their adjustable variables, as well as designing new ones;
- use real-world data, for both networks and trips, validating and calibrating real-life scenarios.

To conclude, a possible opposing argument to this project is in the fact that it would be unethical to pass unfairly uneven information to the clients, providing a better service to some than others, even if randomly. Some would say that the communication of a distorted perception alone, even ultimately improving driving experience for everyone, is totally absurd. There are two main reasons to contradict these statements:

- the first is that even if one agent is not receiving the most accurate information possible, it will be ultimately helping the better functioning of the network. If main objectives of Cosmo are accomplished, this will end up generally improving the trips for every single agent, anyway. In a community everyone has to give something for it to flourish and, if the benefits are worth it, the contributions will be made willingly and gladly;
- secondly, this cannot be compared to a mandatory tax pay, since its use will be completely optional. If one person decides that it is better served by following the shortest route, there will be no rules against it.

Many of the greatest advancements in History came from ideas classified as preposterous, by society. All that is needed is new and innovative concepts and designs, and the ability to believe in them.

References

Adler, J., Satapathy, G., Manikonda, V., Bowles, B., & Blue, V. (2005). A multi-agent approach to cooperative traffic management and route guidance. *Transportation Research Part B: Methodological*, *39*(4), 297-318. doi:10.1016/j.trb.2004.03.005

Balmer, M., Rieser, M., Meister, K., Charypar, D., Lefebvre, N., & Nagel, K. (2009). MATSim-T: Architecture and Simulation Times. (A. L. C. Bazzan & F. Klugl, Eds.) *MultiAgent Systems for Traffic and Transportation Engineering*, 57-78. Information Science Reference

Bazzan, A.L.C.; Nagel, K.; Klügl, F. (2009). Integrating MATSim and ITSUMO for Daily Replanning Under Congestion. *Proceedings of the 35th Latin-American Informatics Conference, CLEI*. Pelotas, Brazil

Ben-Akiva, M., Davol, A., Toledo, T., & Koustopoulos, H.N. (2002). Calibration and evaluation of MITSIMLab in Stockolm. *81st Transportation Research Board Annual Meeting*. Washington, D.C

Chen, B., & Cheng, H. H. (2010). A Review of the Applications of Agent Technology in Traffic and Transportation Systems. *IEEE Transactions on Intelligent Transportation Systems*, *11*(2), 485-497. doi:10.1109/TITS.2010.2048313

Damiani, M., Bertino, E., & Silvestri, C. (2008). Protecting location privacy through semantics-aware obfuscation techniques. *Proc. Of IFIPTM 2008,* 231–245. Springer Boston

Gawron, C. (1998). An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model. *International Journal of Modern Physics C*, *9*(3), 393–408. WORLD SCIENTIFIC PUBLISHING CO PTE LTD.

Gruteser, M., & Grunwald, D. (2003). Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. *Proceedings of the 1st international conference on Mobile systems, applications and services - MobiSys '03*, 31-42. New York, New York, USA: ACM Press. doi:10.1145/1066116.1189037

Krajzewicz, D., Bonert, M., & Wagner, P. (2006). The Open Source Traffic Simulation Package SUMO. *RoboCup 2006 Infrastructure Simulation Competition*. Bremen, Germany

Krajzewicz, D., Hertkorn, G., Wagner, P., & Rössel, C. (2002). An example of microscopic car models validation using the open source traffic simulation SUMO. *Proceedings of Simulation in Industry 14th European Simulation Symposium,* 318–322

Krauß, S. (1998). Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics. *Thesis and dissertation*. Forschungsbericht, Deutsches Zentrum für Luft- und Raumfahrt; zugl. Diss., Universität Köln

Kuriyama, H., Murata, Y., Shibata, N., Yasumoto, K., & Ito, M. (2007). Congestion Alleviation Scheduling Technique for Car Drivers Based on Prediction of Future Congestion on Roads and Spots. *2007 IEEE Intelligent Transportation Systems Conference*, 910-915. IEEE. doi:10.1109/ITSC.2007.4357704

Meister, K., Rieser, M., Ciari, F., Horni, A., Balmer, M., & Axhausen, K. W. (2008). Anwendung eines agentenbasierten Modells der Verkehrsnachfrage auf die Schweiz. *Proceedings of Heureka '08*. Stuttgart, Germany

Paruchuri, P., Pullalarevu, A. R., & Karlapalem, K. (2002). Multi agent simulation of unorganized traffic. *Proceedings of the first international joint conference on Autonomous Agents and Multi-Agent Systems*, 176-183. ACM, New York

Pursula, M. (1999). Simulation of traffic Systems-An Overview. *Journal of Geographic Information and Decision Analysis*, *3*(1), 1-8.

Yamashita, T., Izumi, K., Kurumatani, K., & Nakashima, H. (2005). Smooth traffic flow with a cooperative car navigation system. *Proceedings of the first international joint conference on Autonomous Agents and Multi-Agent Systems*, 478-485. ACM, New York

Yokoo, M., & Suzuki, K. (2002). Secure Multi-agent Dynamic Programming based on Homomorphic Encryption and its Application to Combinatorial Auctions. *Proceedings of the first international joint conference on Autonomous Agents and Multi-Agent Systems*

MATSim, Multi-Agent Transport Simulation. Available at http://matsim.org/

SUMO, Simulation of Urban Mobility. Available at http://sumo.sourceforge.net/

Appendix A

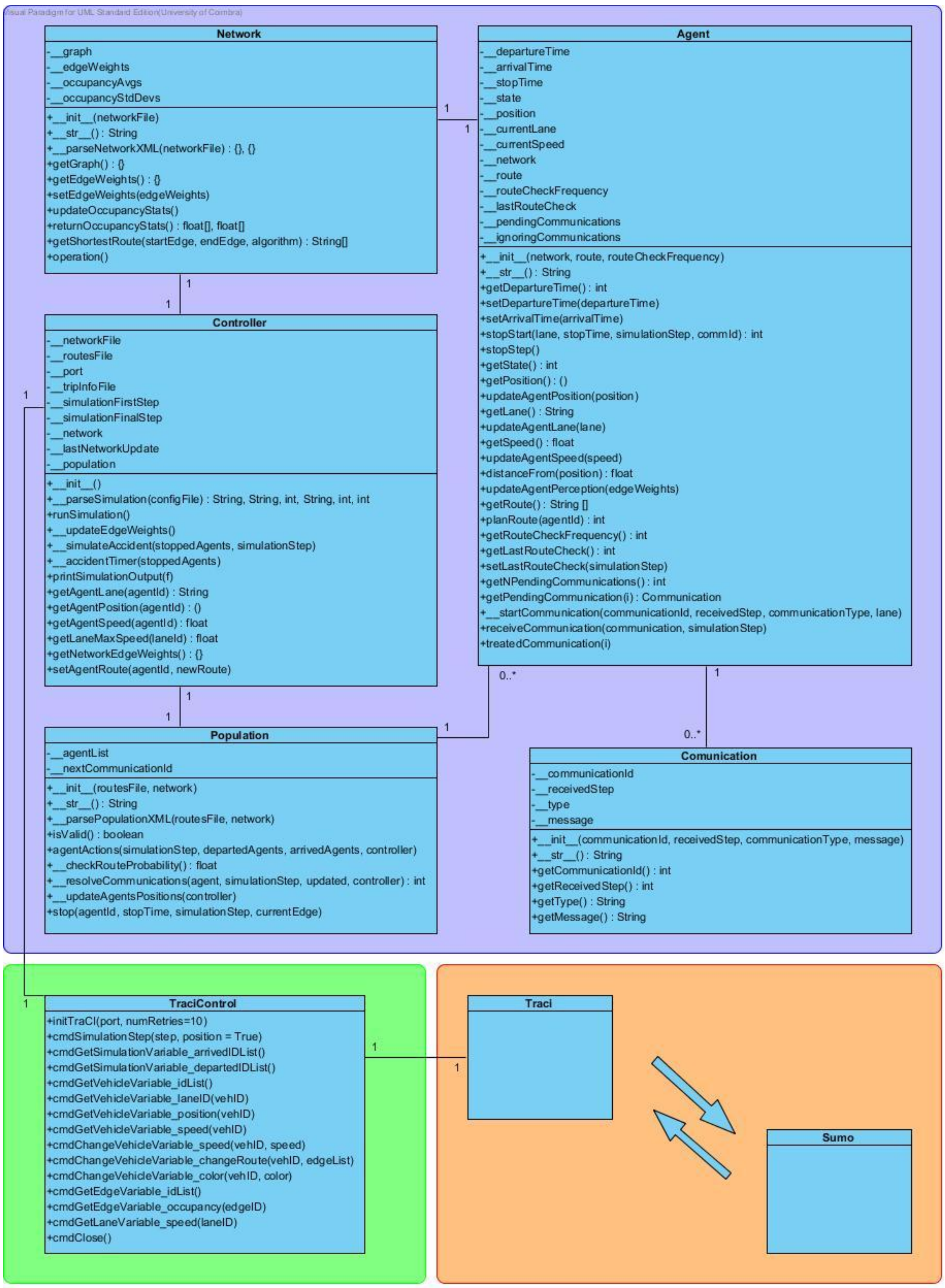

Visual Paradigm for UML Standard Edition(University of Coimbra)
Network
-__graph
-__edgeWeights
-__occupancyAvgs
-__occupancyStdDevs
+__init__(networkFile)
+__str__() : String
+__parseNetworkXML(networkFile) : {}, {}
+getGraph() : {}
+getEdgeWeights() : {}
+setEdgeWeights(edgeWeights)
+updateOccupancyStats()
+returnOccupancyStats() : float[], float[]
+getShortestRoute(startEdge, endEdge, algorithm) : String[]
+operation()
Agent
-__departureTime
-__arrivalTime
-__stopTime
-__state
-__position
-__currentLane
-__currentSpeed
-__network
-__route
-__routeCheckFrequency
-__lastRouteCheck
-__pendingCommunications
-__ignoringCommunications
+__init__(network, route, routeCheckFrequency)
+__str__() : String
+getDepartureTime() : int
+setDepartureTime(departureTime)
+setArrivalTime(arrivalTime)
+stopStart(lane, stopTime, simulationStep, commId) : int
+stopStep()
+getState() : int
+getPosition() : ()
+updateAgentPosition(position)
+getLane() : String
+updateAgentLane(lane)
+getSpeed() : float
+updateAgentSpeed(speed)
+distanceFrom(position) : float
+updateAgentPerception(edgeWeights)
+getRoute() : String []
+planRoute(agentId) : int
+getRouteCheckFrequency() : int
+getLastRouteCheck() : int
+setLastRouteCheck(simulationStep)
+getNPendingCommunications() : int
+getPendingCommunication(i) : Communication
+__startCommunication(communicationId, receivedStep, communicationType, lane)
+receiveCommunication(communication, simulationStep)
+treatedCommunication(i)
Controller
-__networkFile
-__routesFile
-__port
-__tripInfoFile
-__simulationFirstStep
-__simulationFinalStep
-__network
-__lastNetworkUpdate
-__population
+__init__()
+__parseSimulation(configFile) : String, String, int, String, int, int
+runSimulation()
+__updateEdgeWeights()
+__simulateAccident(stoppedAgents, simulationStep)
+__accidentTimer(stoppedAgents)
+printSimulationOutput(f)
+getAgentLane(agentId) : String
+getAgentPosition(agentId) : ()
+getAgentSpeed(agentId) : float
+getLaneMaxSpeed(laneId) : float
+getNetworkEdgeWeights() : {}
+setAgentRoute(agentId, newRoute)
Population
-__agentList
-__nextCommunicationId
+__init__(routesFile, network)
+__str__() : String
+__parsePopulationXML(routesFile, network)
+isValid() : boolean
+agentActions(simulationStep, departedAgents, arrivedAgents, controller)
+__checkRouteProbability() : float
+__resolveCommunications(agent, simulationStep, updated, controller) : int
+__updateAgentsPositions(controller)
+stop(agentId, stopTime, simulationStep, currentEdge)
Comunication
-__communicationId
-__receivedStep
-__type
-__message
+__init__(communicationId, receivedStep, communicationType, message)
+__str__() : String
+getCommunicationId() : int
+getReceivedStep() : int
+getType() : String
+getMessage() : String
TraciControl
+initTraCI(port, numRetries=10)
+cmdSimulationStep(step, position = True)
+cmdGetSimulationVariable_arrivedIDList()
+cmdGetSimulationVariable_departedIDList()
+cmdGetVehicleVariable_idList()
+cmdGetVehicleVariable_laneID(vehID)
+cmdGetVehicleVariable_position(vehID)
+cmdGetVehicleVariable_speed(vehID)
+cmdChangeVehicleVariable_speed(vehID, speed)
+cmdChangeVehicleVariable_changeRoute(vehID, edgeList)
+cmdChangeVehicleVariable_color(vehID, color)
+cmdGetEdgeVariable_idList()
+cmdGetEdgeVariable_occupancy(edgeID)
+cmdGetLaneVariable_speed(laneID)
+cmdClose()
Traci
Sumo
1
1
1
1
1
1
1
1
1
0..*
1
0..*
1
1
1
1

Figure 14 - Class diagram for the Cosmo extension and its communication with SUMO

Appendix B

		Lattice				Radial and Ring			
		Durations	Route Lengths	Occup. Avgs	Occup. StdDevs	Durations	Route Lengths	Occup. Avgs	Occup. StdDevs
1000 Trips	Alg. 1	259,16	2471,346	1,58%	2,04%	188,51	1879,809	1,76%	2,21%
	Alg. 2	243,64	2453,440	1,51%	1,55%	192,66	1922,470	1,69%	2,04%
	Alg. 3	259,19	2471,289	1,58%	2,04%	187,96	1880,117	1,76%	2,18%
	Alg. 4	244,78	2457,344	1,52%	1,54%	194,64	1934,868	1,72%	2,07%
2500 Trips	Alg. 1	259,89	2466,144	1,99%	2,49%	187,81	1860,091	2,20%	2,70%
	Alg. 2	243,52	2443,470	1,79%	1,75%	194,62	1919,774	2,09%	2,49%
	Alg. 3	260,18	2466,757	1,99%	2,48%	187,98	1860,652	2,20%	2,71%
	Alg. 4	245,92	2453,103	1,81%	1,75%	195,17	1924,023	2,10%	2,48%

Figure 15 – Table of results for the testing of traffic flow optimization algorithms