SMART CITY APPLICATIONS FOR URBAN TRAFFIC ANALYSIS

Egyetemi doktori (PhD) értekezés

Besenczi Renátó
Témavezető: Dr. Ispány Márton

DEBRECENI EGYETEM
Természettudományi és Informatikai Doktori Tanács
Informatikai Tudományok Doktori Iskola
Debrecen, 2021.

Ezen értekezést a Debreceni Egyetem Természettudományi és Informatikai Doktori Tanács Informatikai Tudományok Doktori Iskola Alkalmazott információ technológia és elméleti háttere programjának keretében készítettem a Debreceni Egyetem természettudományi doktori (PhD) fokozatának elnyerése céljából.
Nyilatkozom arról, hogy a tézisekben leírt eredmények nem képezik más PhD disszertáció részét.

Debrecen, 2021. 10. 24. _____

jelölt

Tanúsítom, hogy Besenczi Renátó doktorjelölt 2020-2021 között a fent megnevezett Doktori Iskola Alkalmazott információ technológia és elméleti háttere programjának keretében irányításommal végezte munkáját. Az értekezésben foglalt eredményekhez a jelölt önálló alkotó tevékenységével meghatározóan hozzájárult. Nyilatkozom továbbá arról, hogy a tézisekben leírt eredmények nem képezik más PhD disszertáció részét.
Az értekezés elfogadását javaslom.

Debrecen, 2021. 10. 24. _____

témavezető

SMART CITY APPLICATIONS FOR URBAN TRAFFIC ANALYSIS

Értekezés a doktori (Ph.D.) fokozat megszerzése érdekében az informatikai tudományágban

Írta: Besenczi Renátó okleveles mérnökinformatikus

Készült a Debreceni Egyetem Informatikai Tudományok Doktori Iskolája (Alkalmazott információ technológia és elméleti háttere programja) keretében

Témavezető: Dr. Ispány Márton

Az értekezés bírálói:

Dr. _____

Dr. _____

A bírálóbizottság:

elnök: Dr. _____

tagok: Dr. _____

Dr. _____

Dr. _____

Dr. _____

Az értekezés védésének időpontja: _____

Contents

List of Figures

List of Tables

Abstract

Currently, intelligent transportation and smart city applications are prominent research areas. These services offered by city administrations can provide valuable information enabling inhabitants to make smarter decisions, thus, making everyday life easier. In recent years, the car industry is facing a paradigm shift due to several groundbreaking technological developments. Electric cars are becoming increasingly widespread as they become more and more capable and cheaper, whilst self-driving cars are becoming smarter. To show a possible connection between the two, we started to develop a complex system, called the Robocar World Championship, which aims to provide an open research platform for the investigation of self-driving cars and smart cities. In the first part of this thesis, we give a brief introduction of the system, show its operation and describe its components.

One of the two main contributions of this thesis is the data collection subsystem, called the Real-Time Traffic Analyzer. It is a complex system, consisting of both a hardware and a software component. The hardware part is an embedded system that can be assembled into cars. Its basic operation is to count cars on a given road segment, aggregate the data, then send this data using mobile internet connectivity.

The other main contribution is a simulation algorithm. The proposed algorithm can simulate traffic in urban areas. It is based on graph theory and a Markov model of probability theory. The implementation uses OpenStreetMap as a geographical data source and an open dataset, called the Taxi Trajectory Prediction. We introduce the concepts of "Markov random walk", which describes the motion of an individual vehicle, and "Markov traffic", which describes the entire traffic on the road network, respectively. The stationary distribution of the Markov traffic will be determined as a multinomial distribution. We present how the ergodic theory of finite Markov chains implies that a dense traffic event can be approximated well by the stationary distribution of a Markov chain on the road network. To estimate the Markov traffic, the weighted least squares estimation as a kind of composite (quasi-) likelihood methods is applied. In this thesis, the data collection subsystem and the simulation algorithm will be presented in detail.

Chapter 1

Introduction

In recent years, the research and development of Smart City applications have become a vivid topic. These applications, offered by city administrations, provide services to inhabitants which can make the everyday life easier [49]. In addition, these services contain solutions such as intelligent city planning, crowdsourcing, as well as crisis and disaster management [103].

There exist several definitions which seek to specify what a smart city is. Technological companies, such as IBM or Cisco, generally define smartness of a city as a complex IT infrastructure. Gartner defines smart cities as "multiple sectors cooperating to achieve sustainable outcomes through the analysis of contextual real-time information shared among sector-specific information and operational technology systems." One of the most accurate definition is given by the European Innovation Partnership on Smart Cities and Communities [87]: "Smart cities should be regarded as systems of people interacting with and using flows of energy, materials, services and fi-

nancing to catalyse sustainable economic development, resilience, and high quality of life; these flows and interactions become smart through making strategic use of information and communication infrastructure and services in a process of transparent urban planning and management that is responsive to the social and economic needs of society." In [101], Z. Karvalics attempts to clarify the operation and characteristics of smart cities.

One good example for a smart city application is a live timetable service for public transportation or a real-time traffic information system, but many other initiatives exist [46], [38], [39]. Besides, many research efforts try to measure the "smartness" of a city [19], [30], [68], [100], [42].

Although the idea behind the smart city originated in the 1990s – *connected cities, intelligent cities, digital cities, etc* –, the recent technological advancements have accelerated the widespread use of these applications. On the one hand, *big data* analytics could arise from the wide availability of cloud computing. On the other hand, complex sensor systems, complemented with participatory (or mobile) sensing, are being implemented and has been aptly called, the Internet of Things (IoT). IoT is envisioned to become real with 50 billion devices in the near future.

Crowd-sensing is a technique with which we are able to measure certain conditions of urban areas. More precisely, we can observe their residents' habits (e.g. travelling) and analyze them. Although the inspection occurs on the individual level, we analyze information on an aggregated (or crowd) level. From the viewpoint of smart cities, crowd-sensing can be used for: emergency management [32], public transportation [86], road traffic monitoring [63] or sports events [5].

By the year 2050, 70% of Earth's population is expected to live in urban areas [11]. City infrastructures will face new challenges in many respects.

One such aspect is urban traffic.

In the past few years, many developments have occurred in the automobile industry. First, prototypes of autonomous or driverless cars are being introduced by several tech giants and car companies. By the end of the 2010s, most cars were equipped with some sort of driving-aid system (e.g. pedestrian observers, lane support systems), in the 2020s, the widespread use of driverless cars is expected. Second, pure electric cars have been on the market for a few years. These vehicles have advanced on-board computers for perception and controlling. Considering all these developments, it is clear, one of the world's leading economic industries is facing a revolution.

In the face of this revolution, the question arises as to what contribution can be made from the viewpoint of information technology: how can a (smart) city administration assist the widespread of these cars? What can a city-controlled IT solution offer to these cars to allow them to be operated more efficiently? Let us consider some more practical questions: Where should a city install chargers for electric cars? On which route should a pure electric car travel to avoid precocious battery discharge?

It could be assumed with reasonable confidence that the answers will come from a centralized IT infrastructure maintained by city administration. The city may possess all the information necessary, such as traffic jams, accidents, detours, etc. to produce optimal routes for driverless and electric cars.

In this thesis, to answer the need for a traffic management system, we present an open-source software and hardware system, called the Robocar World Championship (or OOCWC for short, from RObOCar World Championship) Framework. The primary aim of the OOCWC is to offer a research platform for developing urban traffic control algorithms along with

traffic simulation algorithms that will provide the framework to investigate the relationship between smart cities and autonomous cars. Although we developed the system for a future time when most of the cars are driverless, it is capable of analyzing certain scenarios in the present, as well.

One of the two main contributions of this thesis is the data collection subsystem, called the Real-Time Traffic Analyzer (RTTA). It is a complex system, consisting of both a hardware and a software component. The hardware part is an embedded system that can be assembled into cars. Its basic operation is to count cars on a given road segment, aggregate the data, then send this data using mobile internet connectivity. We have no knowledge of such a device. The other main contribution is a mathematical model that will give the theoretical background of a simulation algorithm. As a novelty in the field of traffic simulation algorithms, we introduce the concepts of "Markov random walk", which describes the motion of an individual vehicle, and "Markov traffic", which describes the entire traffic on the road network, respectively. The stationary distribution of the Markov traffic will be determined as a multinomial distribution. We present how the ergodic theory of finite Markov chains implies that a dense traffic event can be approximated well by the stationary distribution of a Markov chain on the road network. To estimate the Markov traffic from open datasets, the weighted least squares estimation as a kind of composite (quasi-) likelihood methods is applied, for the first time in the field of traffic simulation models.

Several traffic simulation tools exist. Simulation of Urban Mobility [54] (or SUMO) is a portable, microscopic and continuous traffic simulation package. Aimsun[1] is a traffic modeling software environment that includes

[1] `https://www.aimsun.com/`

a mesoscopic and microscopic hybrid simulator. Multi-Agent Transport Simulation [45] or MATSim for short (version 0.8.0) is an open source software for large-scale, agent-based transport simulations. Its primary simulation algorithm is queue-based [23] and aims to reach an "equilibrium state" by co-evolutionary algorithms [74]. PTV Vissim[2] is a microscopic multimodal traffic flow simulation package for traffic patterns. Although the above-mentioned applications are widely used in traffic analysis and planning, the main focus of their simulation algorithms is on microscopic traffic events. In contrast, the traffic simulation component of our software system focuses on the traffic flow of the whole city, or, to be more precise, the traffic graph and can be considered as a macroscopic traffic simulator. Our system can simulate traffic in large-scale, on traffic graphs with more than 40,000 vertices and with more than 50,000 cars.

Our system has many other possibilities. Besides traffic simulation, the system would be able to collect data about cities with on-board sensors. A database can be established to collect these data. Furthermore, the system can raise new questions in the information and communications technology (ICT) research domain: how can a road vehicle communicate with a city administration in an effective way? The OOCWC is a multidisciplinary system that can bring us to the next step in smart cities: the automated city.

The main contribution of this thesis applicable in the smart city and urban traffic analysis field (including the corresponding publications as well) can be summarized as follows:

1. Participation in the development of the OOCWC traffic simulation framework [14], [15].

[2] `http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/`

2. The development of a crowd-sensing tool for data collection [3, real-time-traffic-analyzer subfolder], [7].

3. Further development of OOCWC system to use collected data [15], [3].

4. Elaboration and integration of a traffic simulation algorithm that provides stationary distribution of the cars on the map [9], [18] [17], [4], [3, justine subfolder].

Notwithstanding the thesis is written in plural form, the author has a principal contribution to the presented software and hardware system. The OOCWC framework which will be introduced in this thesis is the outcome of the collaboration of a research group with several members listed at the end of this thesis. The author engaged in the elaboration of the theoretical models, he implemented and tuned up all the introduced methods and finally evaluated them.

The rest of the thesis is organized as follows: in chapter 2, we present the OOCWC framework and its components. We show the overall architecture and basic operations. In chapter 3, we describe the data collector device, and in chapter 4 the simulation algorithm in detail. In chapter 5, we conclude this thesis.

Chapter 2

The OOCWC Framework

With the development of the OOCWC system, our primary aim was to create a common research platform for the investigation of the relation between smart city applications and autonomous cars. In addition, we intended to create a traffic management system that can give optimal routes for autonomous cars in urban areas. During the development, we continuously checked whether the system is working correctly or not. In addition, we organized a programming competition around the system, because competitive programming can improve the software development process [16]. So we developed *Police Edition* to support the software process (see also section 2.2.3).

2.1 Requirements

Prior to the development process, we defined the software requirements specification (SRS)[1]. Because of the high complexity, we wrote two SRSs, one for the Robocar City Emulator (RCE), and another for the Robocar City Cloud (RCC). The RCE will be able to simulate traffic and perform analysis, the RCC will collect, store and process data that will serve as a basis for analysis and traffic simulation of the RCE.

2.1.1 Requirements for the Robocar City Emulator

1) "Global" requirement: we intend to create a system that can work "globally", which means that it can operate in any city of the world. For this, we need a map database that has a good coverage. Moreover, we do not want to exclude a possible commercialization, so we need an open database with the possibility to integrate it into a commercial product. Because of these licensing issues, we have chosen OpenStreetMap (see also section 2.2.1).

2) "High number of cars" requirement: the system must be able to manage a high number of cars. For example, in [83], a simulation experiment used 10^6 cars. Because of this, implementing every simulation entity as, e.g., one individual Linux process is not possible (no operating system can handle this amount of process).

3) "Car size" requirement: the system must take into consideration the actual size of a car. So, we use a modified Nagel-Schreckenberg cell-based simulation model [65] (see also section 2.2.4).

[1] see `https://github.com/nbatfai/robocar-emulator/tree/master/doc/SRS`

4) "Graph" requirement: the map contains many nodes and road segments. The system must be capable of representing the map as a directed graph. To satisfy this requirement, we use the `libosmium` and the `Boost Graph Library` (see also section 2.2.1).

5) "Types" requirement: the system should be able to handle different car types (e.g. public transportation, emergency services). So far, we have created two different car types. The one is called *routine* car; these cars are going on a regular route, the ordinary, everyday route. The other one is called *smart* car that takes some input from the system, e.g., detours, traffic jams, etc.

6) "Data" requirement: an important requirement is the possibility to operate the system using external data. This means that the system must be ready to take nearly any kind of traffic data (trajectories or point measurements).

2.1.2 Requirements for the Robocar City Cloud

The Robocar City Cloud has two main parts: one consists of server side services, the other one is the data collector. At the time of the writing of this thesis, the server side services are not implemented and its requirements are not defined yet, therefore we only describe the data collector part.

We imagine the data collector as a continuously moving "sensor" that can measure certain traffic conditions. Practically, our solution can be assembled into vehicles (buses, taxis, etc.) and during travelling, the data collector can count vehicles coming from the opposite direction.

For such operation, we have developed a device consisting of dedicated hardware and software components. Prior the development, we envisioned

three main ways for its implementation:

1) ARM processor based solution: in this solution, the core of the hardware component is an ARM processor and a field programmable gate array (FPGA), consisting a System-on-a-Chip (SoC). We use the FPGA part for I/O and memory handling, and the ARM processor for basic calculations. We can develop prototypes on a development board (e.g. Digilent Zybo or ZedBoard). The main advantage of these type of SoCs is that we can install Embedded Linux Systems (e.g. PetaLinux) on them, so we can use a standard Linux environment for development and testing. One solution can be a Digilent Zybo development board with the following peripherals: webcamera and GPS module for input and a GSM module for output.

2) Soft-processor based solution: in this case, only a pure FPGA chip is available, no "prewired" processor can be found, so we need to define it. One solution can be the Xilinx MicroBlaze. As an option, we can create our own processor, which may be optimized for our purposes.

3) "Only FPGA" based solution: In this case, every procedure is implemented in raw hardware. Although this might seem the most powerful solution, the development cycle is the longest.

2.2 Components

In this section, we summarize the research and development results that are included in the OOCWC system. We emphasize, this system is currently a simulation platform, no actual traffic intervention or live measurements occur.

The development process of the OOCWC is agile, with fast prototyp-

ing. Competitions can be organized for researchers and university students to test their ideas and compare their knowledge in traffic analysis, route planning and even their programming skills. The system can be downloaded and tested as an open source project. [3]

The tetris plan of the OOCWC system can be seen in Fig. 2.1. The traffic simulation and the competitions are conducted on a map and assigned to a given city. The ASA and the HSA are measuring subsystems (Automatic Sensor Annotations and Human-controlled Sensor Annotations, respectively). The collected data can be used in the Robocar City Emulator. The Monitors can visualize simulations and the results of an analysis. In this section, we give a detailed description of every component of the system.

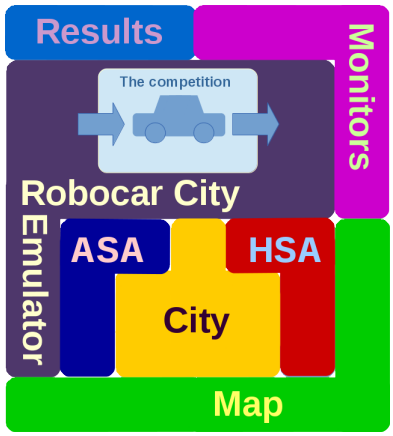


Figure 2.1: This figure shows the tetris plan of the OOCWC system. The overlapping components have a connection with each other. Source: [15].

2.2.1 The Map and the City

In the past few years, numerous map databases have become available for public. These databases can be used for route planning, location discovery and POI search. One example is Google Maps, with which we can navigate on smartphones, plan routes on other devices and has a rich POI database. Another example is Microsoft Bing Maps. In general, these two databases use data from multiple sources created with professional tools. Using these maps are free for private use only.

In contrast, the OpenStreetMap (OSM) is a database that was created and is being developed by a community of volunteers. OSM is free for both private and commercial use. In paper [64], we investigated the usability of OSM in the OOCWC system. The main question was: is a free, community-based map database accurate enough to use for professional purposes?

Since the OOCWC platform has a GNU GPL version 3[2] software license and is free to use and develop as an open source software, the license of OSM (Open Data Commons Open Database License[3]) is suitable for our purposes. In the selection of the map database, one important factor was the license. We needed an open and free database, but we did not want to use a database that has a copyleft license. Therefore, we have the opportunity to close the source of the OOCWC system for commercial use.

Data from the OSM can be downloaded[4] in XML format. The XML

[2] `https://www.gnu.org/licenses/gpl-3.0.en.html`

[3] `https://opendatacommons.org/licenses/odbl/`

[4] Debrecen: `https://www.openstreetmap.org/export#map=13/47.5302/21.6469`

file consists of real-world objects[5]. These objects can be nodes, ways and relations. Every object can have a tag that connects it to its real-world pair. Every object has an ID and coordinates in the WGS 84 geodetic system ("GPS coordinates"). A node can be an individual object (e.g. a building) or it can be a point on a way or a street. Relations define connection between ways and nodes (e.g. bus lanes), but these relations are not used in the OOCWC currently. Source 2.1 shows part of an OSM XML file, a street (Gyöngyösi street, Debrecen) and a node on that street.

```
<way id="24595720" version="7" timestamp="2011-08-16
    T09:49:58Z" changeset="9033539" uid="247611" user="
    SzPaula">
    <nd ref="249781713"/>
    <nd ref="1369686846"/>
    <nd ref="1369686858"/>
    <nd ref="1369672817"/>
    <nd ref="267375701"/>
    <nd ref="267375702"/>
    <nd ref="277120104"/>
    <nd ref="267375703"/>
    <nd ref="267375698"/>
    <tag k="highway" v="residential"/>
    <tag k="name" v="Gyongyosi utca"/>
  </way>
<node id="1369686846" lat="47.5448353" lon="21.6186001"
    version="1" timestamp="2011-07-22T06:58:20Z" changeset="
    8794632" uid="493626" user="tamaas"/>
```

Source 2.1: Short example of OSM XML with a way in Debrecen (Gyöngyösi street) and a node on that way.

[5] https://wiki.openstreetmap.org/wiki/Map_Features

In the OOCWC system, we process the XML file with the Osmium software library as follows. We process every way tagged as *highway* which is accessible by car. Therefore, tags with the *footway*, *cycleway*, *bridleway*, *step*, *path*, *construction* values are not important. We also leave out traffic lights and other signs. From the data, the software creates an adjacency list for every node. This adjacency list consists of key-value pairs, where the key is an OSM node ID, and the value is a list of node IDs of the adjacent nodes of the corresponding node. This adjacency list defines a directed graph (which will be described in chapter 4). In addition, we also store the Haversine distance[6] of the adjacent nodes in a list. For route planning we use the Boost Graph Library (BGL)[7]. We build two alternative data structure, one using a directed routing graph, and another using a BGL graph. The simulation is based on the directed graph, but simulation clients may use the BGL graph as well.

For simulation, we use a rectangular part of the OSM, in most cases, we select a city. This is what we call "City Operating Area". For example, in the case of Debrecen (see Fig. 2.2), the competition area (see section 2.2.3) is bounded with coordinates N47.4095 to N47.652 and E21.4268 to E21.8628.

In Fig. 2.3, the graph of the downtown of Debrecen can be seen. This figure was created with the graph-tool Python library[8]. (For more details on conversion from OSM XMLs to graphs see chapter 4)

[6] `https://docs.osmcode.org/libosmium/latest/haversine_8hpp.html`

[7] `https://www.boost.org/doc/libs/1_66_0/libs/graph/doc/`

[8] `https://graph-tool.skewed.de/`

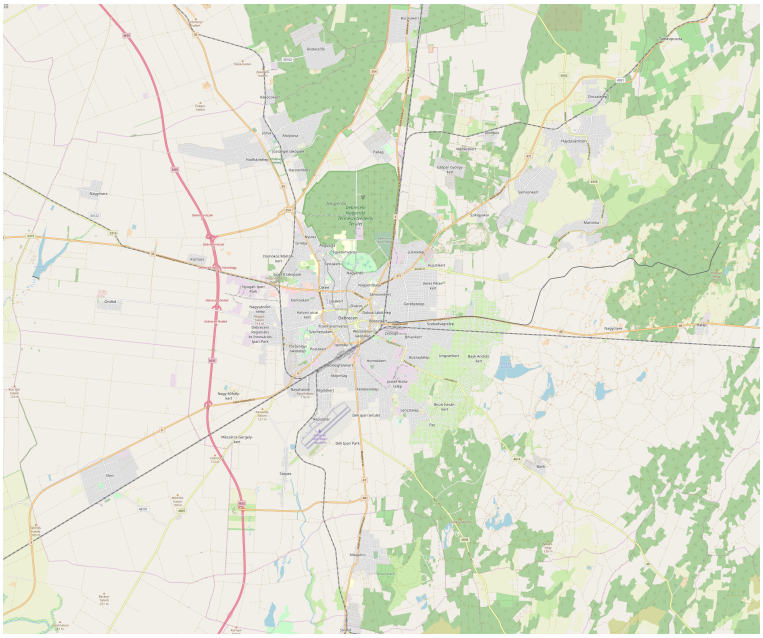


Figure 2.2: The map of Debrecen exported from the OpenStreetMap database. ©OpenStreetMap contributors.

2.2.2 Monitors and Results

Two monitors are included within the system, rcwin and rclog. Both have similar structure, based on the Java Swing GUI library and both are based on the JXMapViewer2 library[9]. In Fig. 2.4, rcwin can be seen in operation. The rcwin can visualize simulations using a local socket or a file, the rclog can replay previously executed, stored simulations.

[9] https://github.com/msteiger/jxmapviewer2

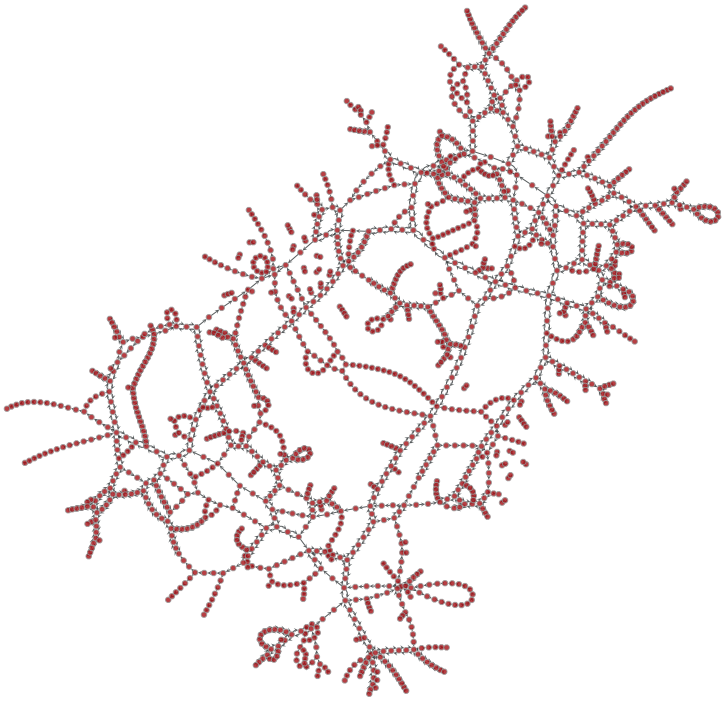


Figure 2.3: The graph of the downtown of Debrecen created with the graph-tool Python library.

2.2.3 The competition aspect

The original main edition of the OOCWC is called the “Police Edition”. The primary aim of this edition is to facilitate research and development of the OOCWC system, simulation algorithms and routing algorithms. We organized competitions around the system, hoping that better and better solutions could arise. A similar approach can be found in other research domains, such as artificial intelligence, where the RoboCup initiative [53]

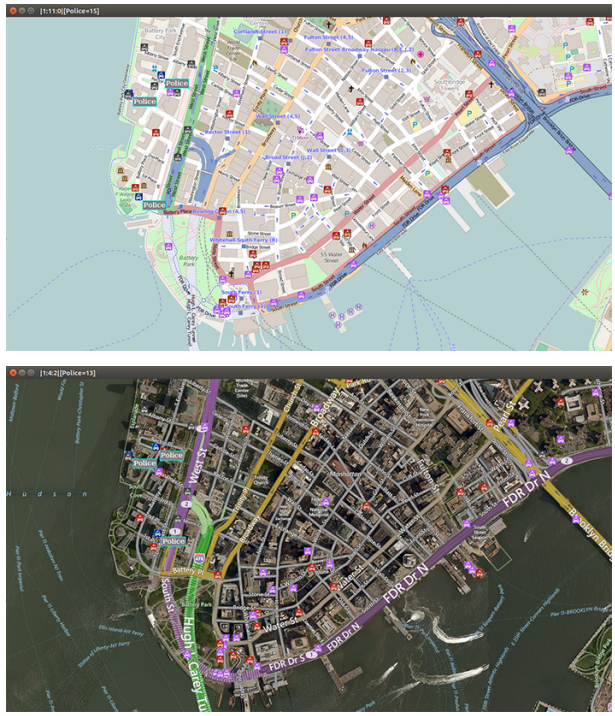


Figure 2.4: The rcwin in operation. Source: [15]. ©OpenStreetMap contributors

organizes competitions for the development of agent-based AI algorithms. In the OOCWC system, three different types of agent exist in the simulation, routine cars, smart cars and guided cars (for more, see section 2.2.4). The aim of the competition is to catch as many gangster agents (smart cars) as possible with 10 police agents (guided cars) in 10 minutes. Competing teams control the police agents. Since in the OOCWC system all cars are multi-agent based entities, every car of a given type has the same controlling algorithm and heuristics (e.g., the closest gangster is pursued).

Paper [6] describes our solution[10]. In this implementation, one cop pursues one gangster. We set cop-gangster pairs based on distance: every cop pursues the closest gangster. First, we used Haversine distance, which gives the distance between two points on a sphere given their latitudes and longitudes, but it did not show satisfying results. Then, we implemented a different solution that takes into consideration the length of the roads and the one-way streets. The basic algorithm in the OOCWC can handle only one cop at a time. The method implemented by us can handle N distance. This gives us a $N \times M$ matrix which contains all cop–gangster distances. (These distances are not symmetrical, because of one-way streets.) In this matrix, we search for the smallest value (M_1, N_1), i.e., the shortest distance between a cop and a gangster, that will be the best pair, so the M_1 cop will pursue the N_1 gangster. The code snippet in Source 2.2 shows this implementation.

```
1  //Calculate the Cop_gangster matrix
2  for (int i = 0; i < 10; i++){
3
4      int *dists = dijkDist(w_cops[i].to, w_gangsters);
5
6      for (int j = 0; j < number_of_gangster; j++){
7          m_cop_gangster[i][j] = dists[j];
8      }
9  }
10 int it = 10;
11 //Sort the matrix
12 while (it--> 0){
13     int MAX = 10000000;
14     int min = MAX;
```

[10]A demonstration video can be found at `https://youtu.be/MzH-0i4VGMk`.

```
15      int i_cop = -1;
16      int j_gang = -1;
17
18      for (int i = 0; i < 10; i++){
19          for (int j = 0; j < number_of_gangster; j++){
20              if (m_cop_gangster[i][j] < min){
21                  min = m_cop_gangster[i][j];
22                  i_cop = i;
23                  j_gang = j;
24              }
25          }
26      }
27      //....//
28  }
29  //Routing
30  for (int i = 0; i < 10; i++){
31      Cop  c = w_cops[i];
32      Gangster g = gcop[i];
33      Dijkstra target = dijkstraAll(c.to, g.to);
34      route ( socket, c.id, target.path );
35  }
```

Source 2.2: Code snippet of the algorithm created for the Police Edition.

About the performance of this algorithm, we can state that on an Apple MacBook Air 11" (Model A1465) laptop (Intel Core i5 1.4 GHz CPU, 2 core, 4 thread, 4 GB RAM) calculating the matrix (i.e., 100×10 routes) took about 200 milliseconds on the map of Debrecen (the graph consists of 77,591 edges and 37,455 vertices).

2.2.4 Robocar City Emulator

The main component of the OOCWC system is the Robocar City Emulator (RCE). This component has several tasks:

- The Smart City Server opens the map database, converts it into a BGL graph.
- The Traffic Server simulates the traffic.
- Simulation agents (e.g., clients) can connect to the server and act.

There exist several approaches for traffic simulation. Realistic traffic simulations have already been used in [28]. The traffic simulation model of the RCE is based on the Nagel-Schreckenberg (NaSch) model [65], because it uses a cell-based approach, as well. Three classes of traffic simulation models exist:

1. Agent-based models (or microscopic models), for example, see [88], [33].
2. Continuum models (or macroscopic models), for example, see [37], [78], [25].
3. Hybrid models, for example, see [79].

The RCE uses the agent-based approach, moreover, it can be considered as a standalone multi-agent system. For more on traffic simulation, see section 4.

The first rapid prototype of the OOCWC system is called Justine. It has 3 main components, the RCE, the rcwin and the rclog. This prototype uses

the OSM database and processes it with the Osmium Library. The result of this processing is a routing map graph and a BGL graph. The routing map graph then placed into a shared memory segment by the smart city server. The traffic server simulates the traffic. A sample client program is provided to the original version that demonstrates how a client can connect to the server and how the shared memory segment should be operated. One implementation of the RCE is the "Police Edition".

Basic operation of the simulation model As we mentioned earlier, the simulation takes place on a rectangular part of the OSM. In the case of the standard competition map of Debrecen, the routing graph has 77,591 edges and 37,455 vertices. We slice all edges for parts that are 3 meters long. Based on NaSch terminology, these parts are called cells. Therefore, the cell length is equal to 3 meters. We set the simulation step time for 200 milliseconds. Edges can contain a given number of cars only (calculated as the edge length divided by the length of the cell, or part, i.e., 3 meters), since one cell can contain only one car. For simplicity, we use only one lane per direction for a road segment.

In the original implementation, the traffic server moves the cars by random walk or ant simulation [43], [71]. In the case of RCE, random walk means that when a car arrives at an intersection (i.e., graph vertex) it chooses its next street (i.e., graph edge) based on uniform distribution. Ant simulation means that the next edge is selected with a probability that increases based on previous selections. In addition, the sample client has an implementation of Dijkstra's and Bellman-Ford algorithms (using the BGL) [80].

One of the biggest challenges during the development was to find a so-

lution that can handle a large number of cars. Obviously, we could not implement cars as individual UNIX processes or Java/CUDA threads. No current operating system can handle this amount of individual processes. So, a routine car only has a probability distribution on the map. The initial distribution of the cars can be measured or estimated. Then, all cars move according to the aforementioned ant simulation or random walk. In chapter 4, we show that this simulation approach is not accurate enough and introduce a mathematically rigorous stochastic model called “Markov traffic”.

Chapter 3

Data collection – The Real-Time Traffic Analyzer

Most smart city applications depend not only on well-designed ICT infrastructure but on real-time data. In most cases, these data are collected with some sort of ICT device, but it can be a viable option to involve inhabitants in data collection. On the tetris plan of the OOCWC system in Fig. 2.1 the ASA (Automatic Sensor Annotations) and HSA (Human-controlled Sensor Annotations) components are considered to implement these corresponding approaches. For data collection in smart cities, we described a framework in paper [86].

The framework is an outline of smart city applications that are based on participatory sensing; this approach involves inhabitants in the data collection actively. With the widespread use of smartphones, more and more computational capacity and sensors are available in the hand of inhabitants. If a community finds an application useful, they will participate in its op-

eration. This context sharing (or crowdsourcing) combined with big data analytics can realize the Internet of Things (see Fig. 3.1). One good example can be the Waze navigation system, where users can annotate certain events of traffic (e.g. accidents, detours, etc). More examples can be seen in papers [86, chapter III] and [5]. Although this framework is developed especially for participatory sensing, crowd-sensing based applications can work following the presented approach, like our system, the Real-Time Traffic Analyzer.

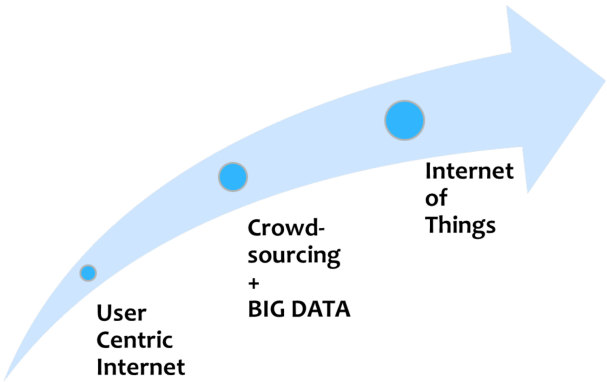


Figure 3.1: Internet evolution in our vision. Source: [86].

An average crowd-sourcing application nowadays consists of two components, one at the user's device and another one in the cloud [40]. This may result in unnecessary developments and slow innovation and development cycle. In our framework, we separate application logics from analytics and communication. To provide development that is independent from other components, the framework must pass extra information, by which its services can be extended. For this, we need an approach based on some sort of extensible messaging service. In addition, in the case of partici-

patory sensing, we must separate information producers and consumers in time, space and synchronization.

Because of the lack of a framework that separates producers from consumers, we proposed a system for participatory sensing based smart city applications. The system (Fig. 3.2) consists of

- a communication framework (including an advanced messaging protocol);
- analytics;
- applications.

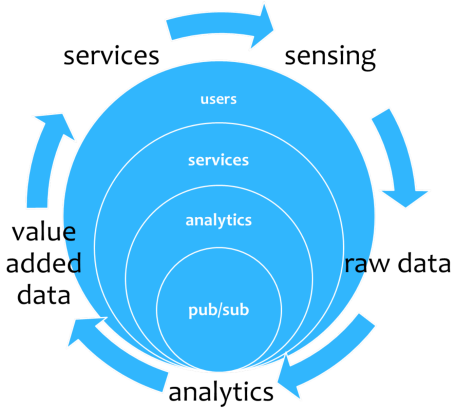


Figure 3.2: Production chain from sensing to services. Source: [86].

In the framework, we originally investigated the Extensible Messaging and Presence Protocol [76] because of its publish-subscribe (pubsub) [62] service. The communication scheme of crowd-sourcing based applications is somewhat similar to the basic scheme of pubsub. Users collect data (or

publish their data), and use services offered by the system (or they *subscribe* to services). Therefore, in this model, we define three roles: *Producers*, *Service Providers* and *Consumers* (see Fig. 3.3). Every entity interacts via pubsub nodes, which are event based. So, this is a type of pubsub pattern with pushing the data towards a broker application.

Producers are the original information sources; they are users who collect data. Consumers are using the data provided. In some cases, they can act as Producers as well. Service Providers analyze, aggregate, and add value to the raw data.

In the model shown in Fig. 3.3, Producers publish (depicted with empty arrowheads) the collected (raw) data to pubsub nodes (depicted with blue circles). Service providers subscribe (shown as filled arrowheads) to these nodes and publish their value-added information to other pubsub nodes (depicted with orange circles). Consumers (who can be Producers as well) subscribe to these nodes, so they can obtain this value-added information. An important property of this pubsub approach is that the information flow occurs asynchronously.

The publish-subscribe architecture (originally introduced in XMPP) can directly map our communication scheme (see Fig. 3.4). Raw data nodes and value-added information nodes (or service nodes) are created by Service Providers. Producers and Consumers with appropriate node rights can publish their collected data to raw data nodes, and only the owner of the node can pull data from these nodes with the subscription service. Service providers publish information to the service nodes. Consumers subscribe to these service nodes and may retrieve value added information. In most cases, consumers can decide which node they subscribe to, so they get appropriate information. XMPP has a data aggregation feature, but we should

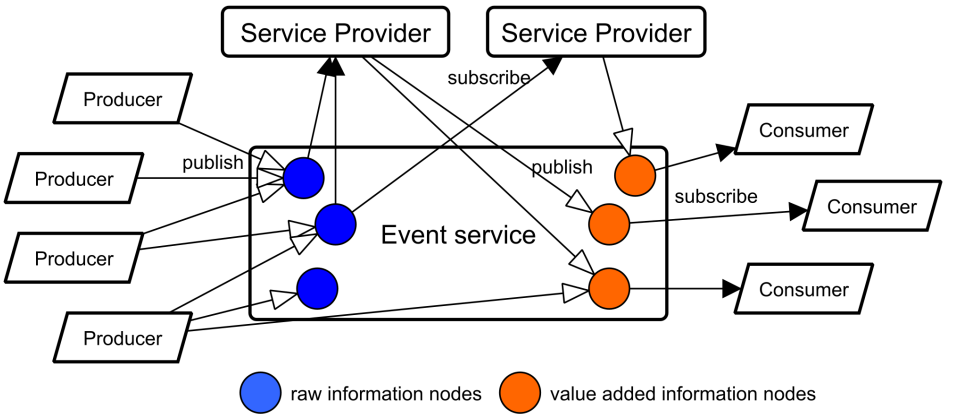


Figure 3.3: Roles and their interaction in the pubsub architecture. Source: [86].

note that this feature cannot filter events. In Fig. 3.4, this aggregation is denoted as dark circles, whilst empty circles indicate aggregation that is implemented as a service logic.

To briefly summarize, crowd-sensing and crowdsourcing (and participatory sensing) are tools with which we can measure certain conditions in a city and we can obtain information about citizens and their environment. The inspection occurs on an individual level and we investigate data on an aggregated level (crowd level). Crowdsourcing or participatory sensing is a method where the user's (or crowd's) active *participation* is required. In the scheme of the OOCWC this component is the HSA. The other data collection type is crowd-sensing where we can use automatic measuring tools. In the scheme of the OOCWC system, this is the ASA component. The devices, developed by us, are collecting data without human interaction. This component is considered as a *Producer*. As a first step of development, we

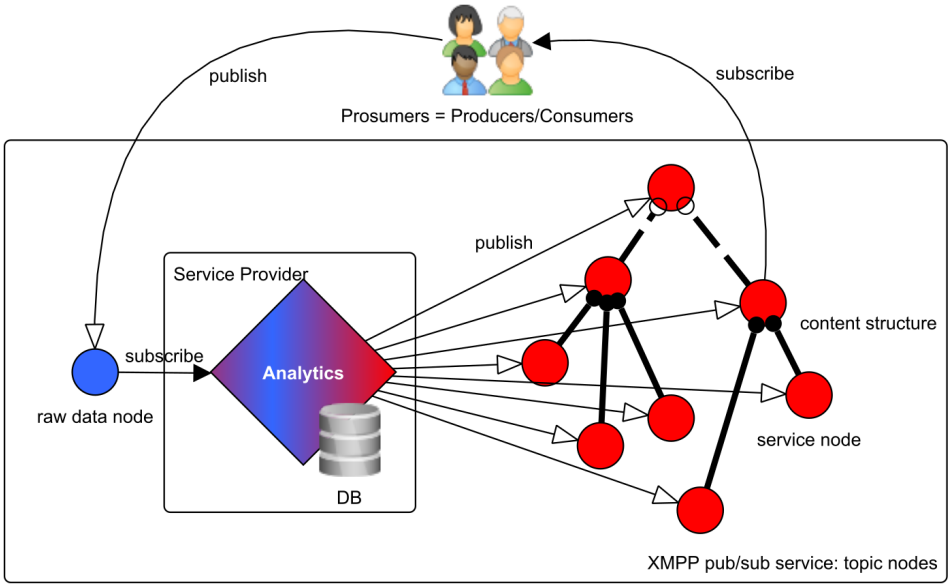


Figure 3.4: Participatory sensing information chain. Source: [86].

described the requirements of this system. The key points are the following:

- Real-time operation, local image processing.
- Real-time connection with a cloud-based service.
- Accurate positioning.
- Ease of assembly into cars.

We imagined the development in three different directions. The main difference is the type of the CPU. We apply a Field Programmable Gate Array (FPGA) in every solution, and all peripherals are the same. As input devices, we use an image sensor and a GPS (Glonass) module; as an

output we use a standard GSM module. We conduct the development on "development boards". The three directions are as follows.

ARM based solution. The fast processing property of the FPGA is used for the I/O subsystem and the memory. The ARM provides a standard platform for calculations, and for image processing. The main advantage of this possibility is that we can use an Embedded Linux System (ELS), so the high-level processes can be implemented in a standard Linux environment. One such ARM based system is the Raspberry Pi single-board computer (but without FPGA).

Soft-processor based solution. In this solution there is no physical, dedicated CPU, but we can add one for the hardware design (e.g. a Xilinx MicroBlaze). In this case, we can define the operation of the CPU, but its instruction set can be a limiting factor. Using an ELS, like the case of ARM based solutions, is possible.

Raw FPGA design. In this case, there is no CPU at all. The I/O system, the memory and the image processing methods are all implemented in the FPGA; technically, it is a pure hardware design. Regarding the computational capacity and execution times, this is the best solution, but its development cycle is longer.

As a conclusion, we can say that the ARM based solutions are more flexible and easier to extend with new functionality, but we are limited to the properties of the ARM (clock frequency and instruction set). Because development boards based on ARM are relatively cheap and easy to access, and the development cycle is shorter, one can build a development community around a project easier. Raw FPGA designs provide faster processing but their development cycle is longer and complicated (i.e., more expensive). In the current state of the project, a prototype of the ARM based

solution is provided. We will describe this prototype in this section.

3.1 Technology overview

In the first phase of development, we implemented the data collection system to an ARM based hardware. For this, we needed a device that provides flexibility during development. For example, the Raspberry Pi single-board computer (its first edition) was not suitable, because of its limited computational capacity and poor periphery handling. Since ARM is a ubiquitous CPU type, compilers are available, and we can use an ELS, as well. We chose to use the Zynq System-on-chip (SoC).

In this section, we describe in details the hardware and software components of the data collector subsystem.

3.1.1 Hardware

The scheme of the system can be seen in Fig. 3.5, and the detailed scheme is shown at `https://bit.ly/3wTsNKF`.

We developed the system on a Digilent Zybo development board[1]. As shown in Fig. 3.5, the central component of the system is the Zynq Processing System, with the Processor System Reset. The devices are connected through the standard Advanced Extensible Interface (or AXI for short, [98]) bus protocol.

The two main components are the two UartLite serial ports. The communication with the GPS module is provided through the *axi_uartlite_0* serial port with 4800 baud (1 PPS data). (The GPS sends measurable

[1] `http://www.digilentinc.com/zybo/`

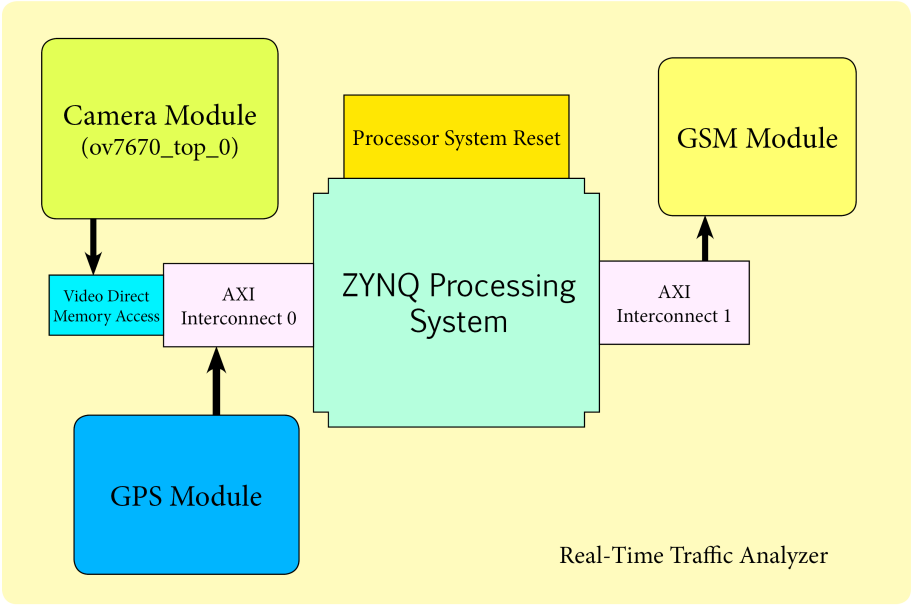


Figure 3.5: The scheme of RTTA. Source: [7].

data once per second.) The connection with the GSM module is provided through the *axi_uartlite_1* serial port with 115,200 baud.

The *ov7670_top_0* reads data from the camera module and converts it into AXI compatible stream. The *axi_vdma_0* module loads camera data into the memory via the *axi_interconnect_0* module.

The communication between the peripheries occur via the AXI bus, which has a 100 MHz base clock frequency. For sending camera data, we use a 150 MHz secondary clock.

Video Direct Memory Access (VDMA) provides a high-speed connection between the memory and the camera module. We load the video stream into memory with the resolution of 640×480 pixel. The VDMA is rela-

tively easy to use, and thanks to its Linux driver, we can easily integrate it into the system.

For our aims, the Vincotech A1080-A GPS module seems suitable. It requires minimal external components and communicates through serial port. The module sends data 1 PPS which is processed by our application on the Linux system (see the next section). The time and the actual position are both important to us. We should note that the power provided by the Peripheral Module (PMOD) is enough for this module.

For internet connection, we use a SIM900 GSM module. Since this module requires 5V voltage to operate, we connected it to the Zybo board's external power supply.

For visual perception, we use the OV7670 camera module that has a maximal resolution of 640×480 pixel. One huge advantage of this unit is that we can set its resolution before operation (among several other properties). With this, we can spare some capacity of the ARM by skipping a pre-processing step of the videostream.

The hardware parts can be seen in Fig. 3.6.

During the development process, we should pay high attention for the type of the vehicle (bus, car, etc) where the device will be used, because this will define the power supply, the case size and other parameters of the device. The current state can be considered as a prototype, mainly for developer testing, however, the final version can be developed easily from this current version.

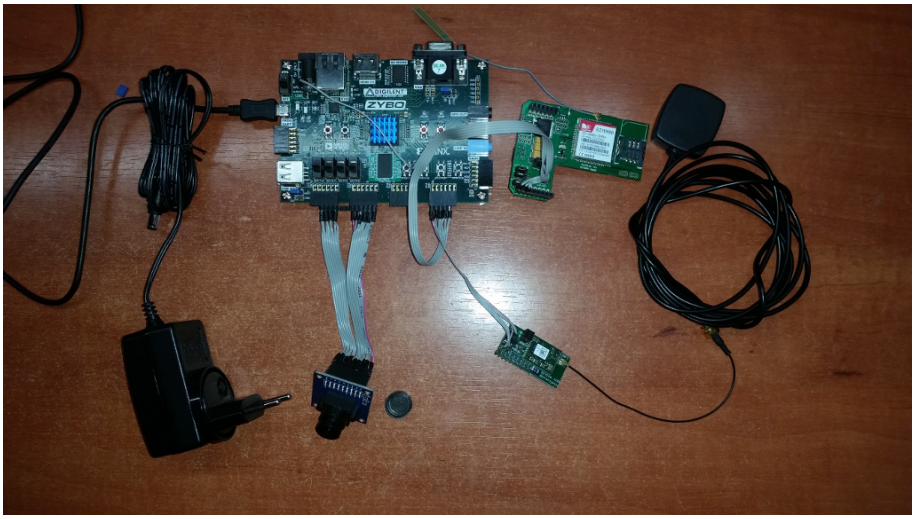


Figure 3.6: The hardware parts of the system.

3.1.2 Software

As we mentioned before, we can run a Linux system on the ARM based development board. Linaro is an open source Ubuntu Linux based operating system, optimized for SoC systems. The software executed on the OS consists of three components:

- GPS component,
- image processing component,
- GSM component.

The program initializes the components and executes it on different threads (as shown on Source 3.1).

```
init_gps();
MQTT_init();
init_video(argv[1]);

thread t_gps(gps_thread);
thread t_mqtt(mqtt_thread);
thread t_opencv(opencv_thread);
```

Source 3.1: Component initialization and execution.

The GPS component reads the data from the GPS module and stores it in a data structure in the memory (see Source 3.2).

```
typedef struct GPGAA_ {
    unsigned char hour, min, sec;
    enum NS lns;
    double latitude;
    enum EW lew;
    double longitude;
    unsigned char gps_fix, sat_num;
} GPGAA;
```

Source 3.2: The data structure for timing and positioning.

For a communication protocol, we use MQTT, which is a protocol developed for *machine-to-machine* communication. It has a low communication overhead and implements the publish-subscribe communication paradigm. It is a very good protocol for low bandwidth communication in low energy environments[2].

The image processing method is based on a Haar-cascade classifier [94] [58]. (The source of the pre-trained cascade file: [77].) After initializing the videostream, we store every frame in a Mat object.

[2] http://www.mqtt.org

The algorithm tries to recognize a vehicle on every frame of the stream. If a vehicle is recognized, a counter (`density`) is incremented with the number of recognized cars (`object->total`) on that frame. A car may be counted more than once if it appears on consecutive frames; this is why we call this value as "density". We should note that if the car stops or its speed is lower than 6 km/h for some reason (e.g. traffic lights or traffic jam) then the counter stops.

The device does not send raw image data to the server, and it performs pre-processing. We send the pre-processed data in Google Protocol Buffers format. The format of the message can be seen on Source 3.3.

```
message TrafficAnalytics{
    required string car_id = 1;
    required string timestamp = 2;
    required LatitudeNS latitudens = 3;
    required double latitude = 4;
    required longitudeEW longitudeew = 5;
    required double longitude = 6;
    required double density = 7;
    required uint32 vehicle_speed_gps = 8;
    optional uint32 vehicle_speed_can = 9;
    optional uint32 fuel_level = 10;
    enum LatitudeNS{
        NORTH = 0;
        SOUTH = 1;
    }
    enum longitudeEW{
        EAST = 0;
        WEST = 1;
    }
```

```
20 }
```

Source 3.3: Protocol Buffers message format.

This pre-processing is useful from two aspects. First, in contrast with other approaches where the videostream is sent to a remote processing system, our application uses the communication channel to send only a few kBs of data every minute. The size of the videostream depends on the resolution, encoding and compression of the videostream, but it definitely requires a network connection with a much higher bandwidth. Second, data arriving from multiple sources that is evaluated and processed in a central system can cause a huge workload, thus requires a lot more computation capacity. This can be considered as a standard approach for IoT applications, see [75].

For testing the system, we developed a desktop application. The program is an MQTT broker application, to which the data collectors are connected. The input of this program is the data arriving from the devices and its output is a data structure that can be used in the OOCWC system. We describe this in details in section 3.2.

We should note that the system in its current version does not cause legal (privacy) issues. In information systems that monitor civilian environment the importance of privacy is crucial. Our system does not forward or record data that have privacy related issues. The system only records and forwards the number of cars (or more precisely its density) counted in its environment. No license plate numbers, faces or any other personal data are recorded or stored.

3.1.3 Development environment

Since the Real-time Traffic Analyzer is a heterogeneous system, different development tools were required. The development environment of the hardware design was the Xilinx Vivado Design Suite version 2014.4. Vivado supports high level design development (but the possibility of module development remains). It supports component-based development on an advanced GUI, where projects can be assembled from modules previously developed or provided by third-party developers. We also followed this high-level development. First, we created an empty project on a Zybo board with ELS, then we added and connected each periphery. It is important to note that not every module had a predefined module description (i.e., a "ready-to-use" module); we had to develop the GPS module.

The operating system running on the device can be added easily, we must make only a few modifications. We should define the *device tree*, which we should add to the Linux kernel. This device tree lists the devices and defines their I/O ports. If we define this successfully, we can use our peripheries just like in any other Linux system; as files that can be read and written. To be more precise, the data arriving from the GPS module can be read from the file assigned by the device tree in the Linux file system. Similarly, if we want to send data via GSM, we should write the file assigned to the "input port" of the GSM module. An example can be seen on Source 3.4. The first part opens the file assigned to the GPS module, the second reads it. The third block opens the file assigned to the camera module.

```
1 if ((gps_fd = open("/dev/ttyUL1", O_RDONLY | O_NOCTTY)) ==
    -1) {
2         perror("Open GPS serial: Failed to Open!");
3         return -1;
```

```
4       }
5   char read_GPS() {
6       char in;
7       if (read(gps_fd, &in, sizeof in) < 0) {
8           perror("Failed to read GPS serial");
9           exit(1);
10      }
11      return in;
12  }
13  if (video_fd = open("/dev/video0", O_RDWR) == -1) {
14          perror("Open Video: Failed to Open!");
15          return -1;
16  }
```

Source 3.4: File operations in the software.

Since our software runs on a Linux system, we could develop in Linux/UNIX environment. For this, we used the KDevelop IDE, the C++11 language, and POSIX threads. First, we tested our image processing methods in desktop environment, because no display is implemented in the RTTA system, so we cannot check its operation visually. The test application uses the same Haar-cascade based object detection algorithm. The operation of the program is shown in Fig. 3.7. On the left side, the video can be seen (recorded with an Apple iPhone 5S), with data written on it (e.g. position, date and time, timestamp, and number of cars on a given street). On the right side, we visualize the density of the traffic using the Google Maps API. In Fig. 3.8, the basic functionality of the object detection can be seen. We should note that the video footage of the iPhone was pre-processed before the testing, so the resolution, frame rate and other properties of the video was the same as of the RTTA device.

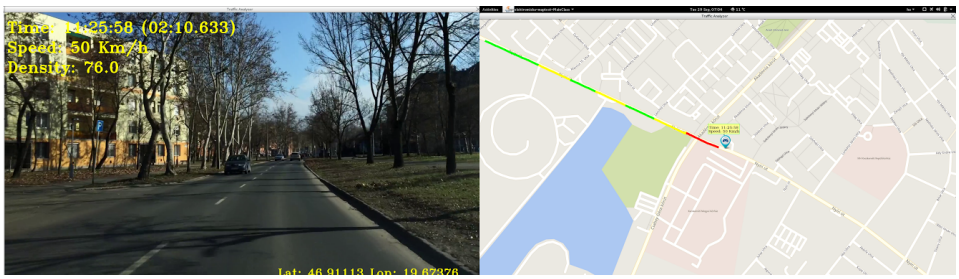


Figure 3.7: Testing the image processing method in desktop environment.

For a demonstration, an MQTT broker application is included in the RTTA system, which was developed in Java 8 in the NetBeans IDE. This app receives the data sent by the data collector devices. This program performs data aggregation using the Osmium library. In addition, the program produces a file, where every street has a vehicle density value. This file will be the input of the RCE.

3.2 Using Data – The Crowd-sourced Traffic Simulator

The aggregated output data of the RTTA was used as an input to the RCE. For this, we created a fork of the original OOCWC, and modified it. The project's name is Crowd-sourced Traffic Simulator (CSTS) and can be downloaded from GitHub [3].

The input of the RCE is a plain text file that contains lines, with each line containing a street name and the car density value on that street. Let denote the street as s_i ($i = 1, \ldots, N$), and the corresponding intensity as v_i. This intensity is calculated in a point or points of the corresponding street.

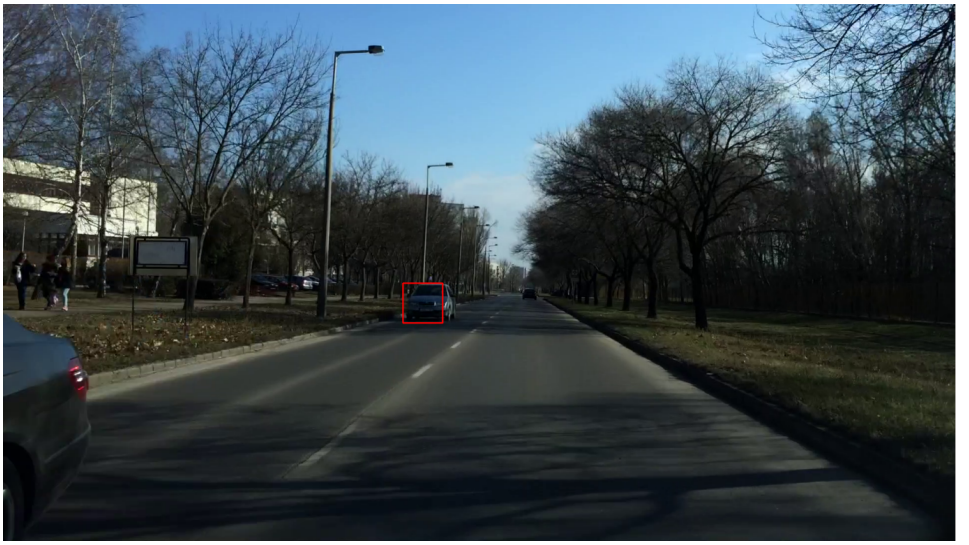


Figure 3.8: The basic functionality of the object detection algorithm.

Later, this intensity will be the average of the measured values in that point. The RCE works on an OSM based graph where the nodes are OSM way node references. Let denote the number of OSM way node references as n_i on the street s_i and s_i^j the jth OSM way node reference on the street s_i ($i = 1, \ldots, N$, $j = 1, \ldots, n_i$). Then, we initialize the vehicle to the s_i^j node according the following probability:

$$P(s_i^j) = \frac{v_i}{\sum_{k=1}^{N} n_k v_k}.$$

One can see easily that $\sum_{i=1}^{N} \sum_{j=1}^{n_i} P(s_i^j) = 1$.

So, we initialize vehicles for the first step of the simulation as measured. Then, we can observe the operation of the simulation algorithm.

To obtain this functionality, we had to extend the original OOCWC im-

plementation with one new class. In this class, we represent the traffic that is based on real measured data. The new class is inherited from the Traffic class. The new members of this class are the measured distribution of cars and a function that initializes the vehicles (described above). As shown in Source 3.5, we create an instance of this class if we want to initialize traffic based on measured data. In the constructor of the class, the *itype* parameter must be set, identifying that we want to use measured data.

```
1 traffic = new justine::robocar::RealTraffic {
2       nrcars,         //Number of cars
3       shm.c_str(),    //Shared memory management
4       catchdist,      //Catch distance
5       type,           //Type of traffic simulation
6       itype,          //Type of car initialization
7       minutes         //Time of the simulation
8 };
```

Source 3.5: Creating a RealTraffic object.

In Source 3.6, the method of the initialization of the cars can be observed. The return value of the function is the Way Node ID of the OSM graph where we initialize the current vehicle. First, the function summarizes the ID-s (see Line 7), then generates a pseudo random number (at Line 9). Based on this number, we decide whether we put the car on that node or not. One example on this type of initialization can be seen in Fig. 3.9.

```
1 osmium::unsigned_object_id_type virtual node(){
2   double no_edges = 0;
3   for ( shm_map_Type::iterator iter=shm_map->begin();
4         iter!=shm_map->end(); ++iter ){
5
6       for ( auto nodeval : RealTraffic::alist[iter->first]
```

```
    )
7           no_edges += nodeval;
8       }
9   double rand = ( double ) std::rand() / ( double )
     RAND_MAX;
10  double sum = 0.0;
11  for ( shm_map_Type::iterator iter=shm_map->begin();
12          iter!=shm_map->end(); ++iter ){
13
14      for ( auto nodeval : RealTraffic::alist[iter->first]
     ){
15          sum += ( double ) nodeval/no_edges;
16          if ( sum >= rand )
17            return iter->first;
18        }
19    }
20 }
```

Source 3.6: Initialization.

3.3 Conclusions on data collection

The RTTA is an important part of the OOCWC system. For traffic simulations, scenario analysis and effective route planning, we need real measured data, so we developed a rapid prototype application for data collection following the crowd-sensing approach. The device is based on ARM which, with the software parts, will be ready to be assembled into vehicles. We performed the initial test of the device in Debrecen and the collected data was used as an input to the RCE. The name of this device is Real-time Traffic Analyzer, the fork implementation of the OOCWC that uses

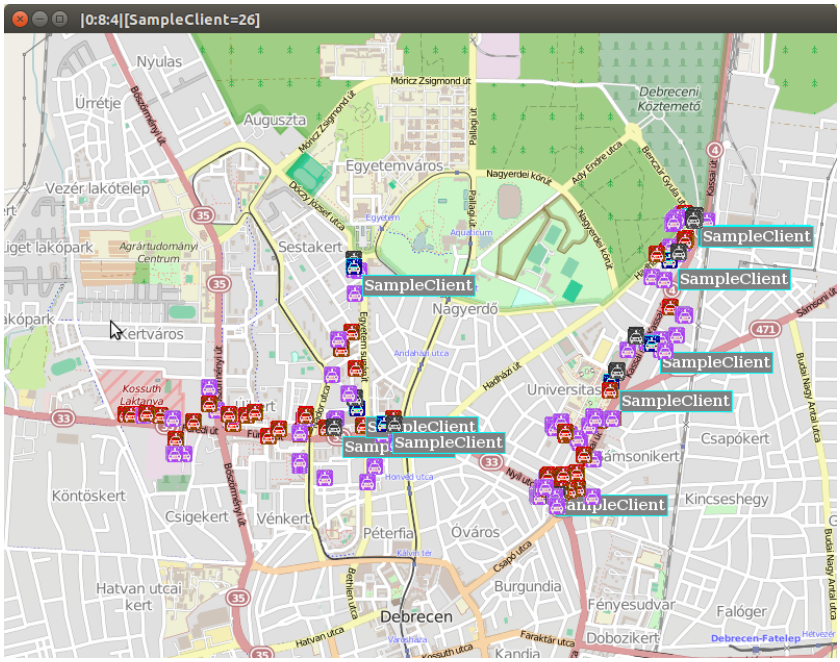


Figure 3.9: The first step of the simulation initialized based on measured data. Source: [15]. Base map and data from OpenStreetMap and OpenStreetMap Foundation. ©OpenStreetMap contributors.

collected data in the Crowd-sourced Traffic Simulator.

It is important to note that the novelty of the system is that the device is moving during measurement. We had to take this into consideration during design planning. The authors have no knowledge about such data collection device. There exist similar devices, but all are designed for fixed position. The central element of the system is the Digilent Zybo development board, which consists of an FPGA and an ARM CPU. The FPGA part handles I/O and memory, the ARM performs image processing. Because the device is moving, we had to extend the device with a GPS module. The image

processing is a Haar-cascade based object classification, the videostream has a resolution of 640×480. The pre-processed data, the street name and the car density value on the given street, is sent to our server application via a standard GSM module.

The measurement is a vehicle counting method that provides a density value on a given road segment or street. The MQTT broker application collects the data, aggregates it, then assigns it to streets. The program creates a file where each line contains a street name and a density value of the given street. In the RCE, we use this density value as an initialization of the simulation. We start the simulation and observe the distribution of cars. OOCWC is a flexible system, i.e., it is relatively easy to modify it to different applications. CSTS proves that we could modify it to operate with measured data.

We should note that the data collector system worked properly during the test phase; the object detection algorithm recognized vehicles coming from the opposite direction, could provide an accurate position and the data arrived at the MQTT broker application properly.

Chapter 4

Simulation algorithms

In this chapter, we introduce the proposed simulation algorithm[1]. The model is based on graph theory and a Markov model of probability theory. First, we show the simulation algorithm that is included in the original OOCWC system and some other simulation tools and models. Then, we describe our model in details. We use a publicly available dataset, so we outline the concept how we use this data. After, we describe the use of OSM data in our simulation environment. Finally, we outline our results and draw some conclusions.

4.1 Introduction

In the CSTS system, the initial state of the simulation can be initialized based on measured data or some prescribed distribution (e.g. uniform one).

[1]Because of the length regulations of the thesis, some parts of the mathematical model are omitted for brevity. For a detailed description, see papers [9], [18] and its longer version on arXiv [8].

The former can be seen in Fig. 3.9; in this case, we can observe how the distribution of the cars changes during the simulation. Fig. 4.1 shows the change of the distribution in the case of the simulation algorithm included in the original OOCWC version. (The simulation occurs on the map of Debrecen represented with a graph.) The x axis shows the streets, the y axis shows the number of cars on that street. Only those streets are shown that have at least one car on it. These histograms show the number of cars on the streets of Debrecen in a descending order. It is interesting to observe that the nature of the distribution did not change during the simulation. However, another important aspect is that the order of the streets should remain the same (we require a sort of stationarity). It can be observed that the order of the streets changed during the simulation. During the test, we initialized 10,000 cars based on the measured data. This can be seen in Fig. 4.2.

Since the application was a prototype, some of its features were not defined or developed satisfactorily. One such feature was the traffic simulation algorithms. In this version, the simulation algorithm moves the cars quite randomly. So, when a car arrives to an intersection, it selects the next street according to uniform distribution.

In papers [26], [36] and [34], authors introduced a stochastic model that can control the traffic in an urban road network. The model is based on discrete time Markov chain on the road graph which plays the role of the state space. A transition probability matrix is introduced that describes the dynamic of the traffic while its unique stationary distribution corresponds to the traffic equilibrium (or steady) state. In this equilibrium state, the distribution of the cars on the graph remains invariant locally in time. So, this stationary distribution of the Markov chain can be interpreted as the

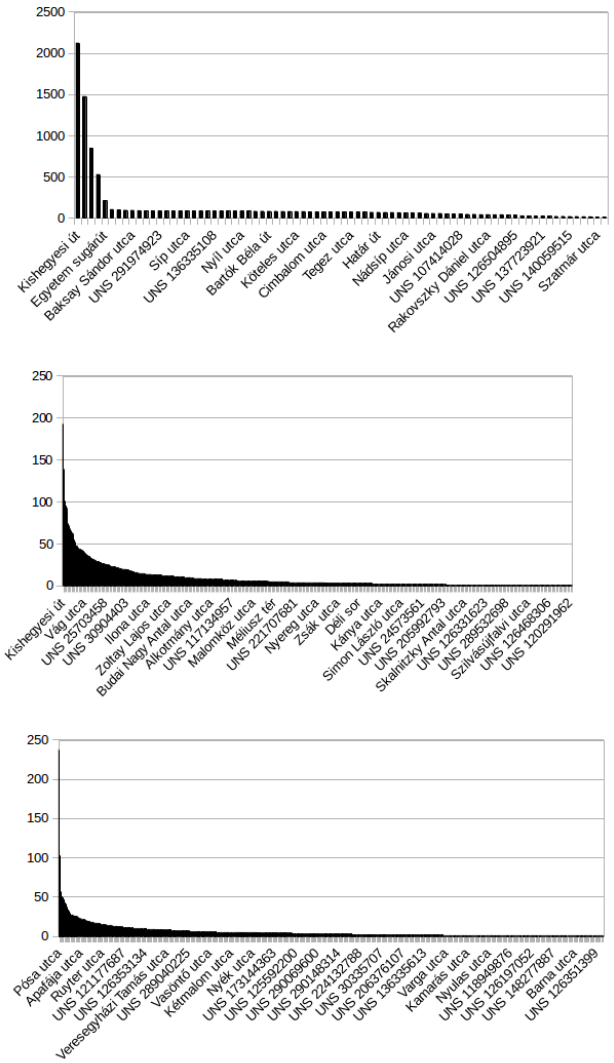


Figure 4.1: The change of the order of the streets. The number of the street in each step is 78, 1,085 and 1,725 respectively. Source: [15].

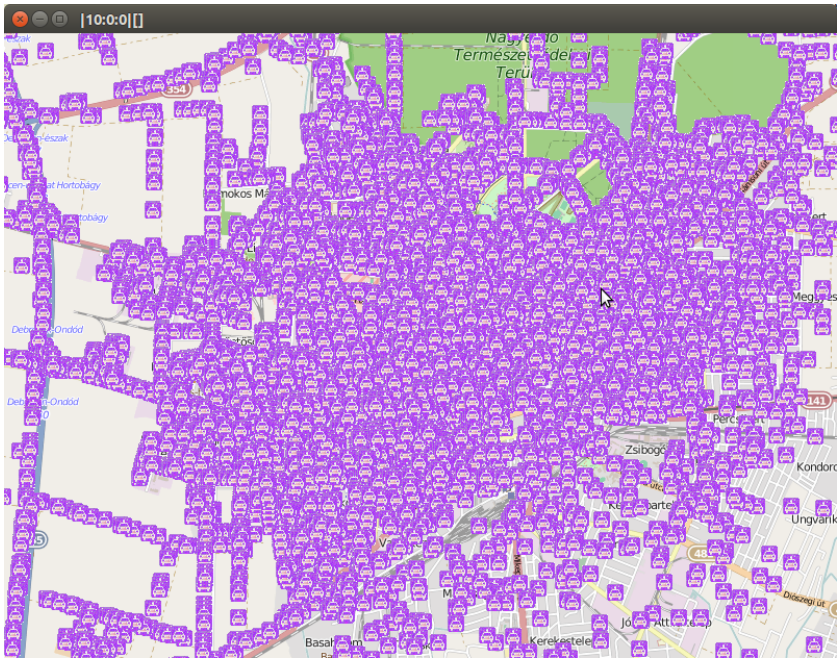


Figure 4.2: Screenshot from the simulation. Source: [15].

momentary true distribution of the vehicles on the road network.

Based on the aforementioned mathematical model, we introduce the concepts of “Markov random walk”, which describes the motion of an individual vehicle, and “Markov traffic”, which describes the entire traffic on the road network, respectively. The stationary distribution of the Markov traffic will be determined as a multinomial distribution. We present how the ergodic theory of finite Markov chains implies that a dense traffic event can be approximated well by the stationary distribution of a Markov chain on the road network. We reparametrize the model by introducing the concept of two-dimensional stationary distribution which possesses equidistributed

marginals that are the unique stationary distribution of the transition probability matrix, respectively. To estimate this parameter matrix the weighted least squares (WLS) estimation as a kind of composite (quasi-) likelihood methods is applied, see [44]. Afterwards, we show that the WLS estimator of the two-dimensional stationary distribution can be expressed explicitly.

4.2 Other simulation models and approaches

Several approaches exist for short-term traffic flow prediction. These models are based on for example Box-Jenkins time-series analyses with ARIMA model [90], [55], [84], [41], [99], Kalman filter theory [96], [69], non-parametric methods (k-NN, kernel, local regression) [29], [82], [89], [81], exponential smoothing [61], [20], spectral analysis [70] or wavelets [52], [97], [24]. In addition, several approaches use machine learning and data mining techniques, such as support vector regression [51], artificial neural networks [22], [73], [31], Bayesian networks [85] or deep learning [59]. Some applications can be found based on computational intelligence techniques, e.g., linear genetic programming [12] or Fuzzy logic [48], [57], [102], but seldom can we find approaches based on Markov models, [67] and [26] mentioned previously. Note that the joint application of Markov chains and large graphs to analyze the behavior of complex systems is well known in several fields, e.g., distributed systems [27], geophysics [21] and biology [56]. Agent-based simulation is applied for the modeling of VANET network information spreading in papers [93], [92], [91] and [47]. A survey on traffic flow prediction from the aspect of Smart Cities can be found in paper [66].

4.3 The proposed model

In this section, we give a brief outline of our mathematical model.

4.3.1 Basic concepts and notation

In this subsection, we introduce the concepts of graph theory and the theory of finite Markov chains that are necessary for our method. A textbook on graph theory is [2], some textbooks on Markov chains are [1] and [13].

Road network, line digraph and degree distributions. Let $G = (V, E)$ be a directed graph (digraph) where V and E denote the set of vertices or nodes and the set of directed edges of the graph, respectively. In the sequel, vertices are denoted by u, v, w, edges are denoted by e, f, g. For a directed edge $e = (v, w) \in E$ we also use the notation $v \to w$. G is a simple digraph, i.e., it does not contain two or more edges that connect the same two vertices in the same direction. The digraph G represents the road system of a city. More precisely, we have the following definition, see [72],

Definition 1. A **road network** G is a simple directed graph, $G = (V, E)$, where V is a set of nodes representing the terminal points of road segments, and E is a set of directed edges denoting road segments.

A **road segment** $e = (v, w) \in E$ is a directed edge in the road network graphs, with two terminal points v and w. The vehicle flow on this edge is from v to w.

Fig. 4.3 presents a simple road network.

On a road network, we allow loops but only to represent that a vehicle remains at the same node or edge after a time step. For $v \in V$ define $v^- :=$

$\{e \in E \,|\, \exists u \in V : e = (u, v)\}$ and $v^+ := \{e \in E \,|\, \exists w \in V : e = (v, w)\}$. v^- and v^+ are the sets of edges in and out of the node v, respectively. Note that $deg^-(v) = |v^-|$ and $deg^+(v) = |v^+|$ is the indegree and outdegree of v, respectively, where $|\cdot|$ denotes the cardinality of a set. Based on this, the degree distributions on our Porto example graph can be seen in Fig. 4.9. One can see clearly that this graph is sparse in a sense that there is no node with higher indegree or outdegree than 6.

For a digraph G another digraph can be associated. Let the set V' of vertices of this new digraph be the set of directed edges E of G and let the set E' of its directed edges consist of the ordered pair (e, f) where $e, f \in E$ such that there exist $u, v, w \in V$ that $e = (u, v)$ and $f = (v, w)$. This associated digraph is called a directed line graph, see Section 4.5 in [2], and it is denoted by $\mathrm{L}(G) = (V', E')$. A digraph assigns the vehicles moving in a city to the vertices while a line digraph to the edges.

Recall a topological property of digraph G. For a pair $u, v \in V, u \neq v$, v is reachable from u if there exists a walk $u = v_1 \to v_2 \to \cdots \to v_\ell = v$ where $v_i \in V$ $(i = 1, \ldots, \ell)$. A digraph G is said to be strongly connected

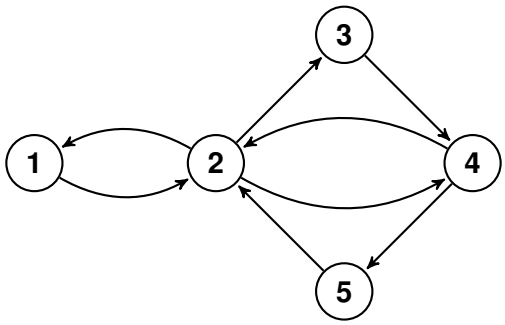


Figure 4.3: A simple road network.

(diconnected) if every vertex is reachable from every other vertex.

Adjacency matrix. We can define vectors (functions) and matrices (operators or kernels) on V in the following way. Let $\boldsymbol{\alpha} : V \to \mathbb{R}$ be a real function on V. Let $T : V \times V \to \mathbb{R}$ be a real function. Then T is called matrix, operator or kernel on V and induces a linear operator in the following way: for each $\boldsymbol{\alpha}$, $T(\boldsymbol{\alpha})$ is defined as $T(\alpha)_u := \sum_{v \in V} t_{uv}\alpha_v$, $u \in V$. If the support of T (the set $\{(u, v) \,|\, u, v \in V : t_{uv} \neq 0\}$ in $V \times V$) is a subset of E then T is called G-subordinated in strong sense.

Let $A = (a_{uv})_{u,v \in V}$ denote the adjacency matrix of the digraph G, i.e., A is a matrix on V and $a_{uv} = 1$ if and only if $(u, v) \in E$ and 0 otherwise. Clearly, the support of A is E, i.e., A is a G-subordinated matrix in strong sense ($a_{vv} = 0$ for all $v \in V$). Note that A is not necessarily symmetric. The indegree and outdegree of a vertex v can be expressed by the adjacency matrix as $deg^-(v) = \sum_{u \in V} a_{uv}$ and $deg^+(v) = \sum_{u \in V} a_{vu}$. Let us introduce the vectors $\boldsymbol{d}^- := (deg^-(v))_{v \in V}$ and $\boldsymbol{d}^+ := (deg^+(v))_{v \in V}$. Then, we have $\boldsymbol{d}^- = A^T\mathbf{1}$ and $\boldsymbol{d}^+ = A\mathbf{1}$ where $\mathbf{1} := (1)_{v \in V}$ is the constant unit function. We should note that we identify every node in V with a unique node ID number, just like in the OSM database, therefore, these nodes can be identified in A as well.

Open road networks and their closures. To model the possibilities that vehicles enter or leave the traffic network, V is augmented by a new ideal vertex 0 which denotes the world outside of the city, see [35]. Let $\overline{V} := V \cup \{0\}$. Then, additional directed edges are also added to E creating E', $(v, 0)$ denotes that the vehicles can leave the city at vertex v, and $(0, v)$ denotes that new vehicles can enter the city at vertex v, where $v \in V$. The augmentation of G is denoted by $\overline{G} = (\overline{V}, \overline{E})$ and it is called the closure

of a road network G. In the rest of this section, it is assumed that the road network is closed, i.e., new vehicles may not enter or leave the city.

Probability distributions and Markov kernels on road networks. A probability distribution (p.d.) on V is the vector $\boldsymbol{\pi} := (\pi_v)_{v \in V}$ where $\pi_v \geq 0$ for all $v \in V$ and $\sum_{v \in V} \pi_v = 1$. We can think of π_v as the proportion of the number of vehicles which drive through the crossing v with respect to the whole number of vehicles in the city at a fixed time period.

Definition 2. A **Markov kernel** or transition probability matrix on V is defined as a real kernel $P := (p_{uv})_{u,v \in V}$ such that $p_{uv} \geq 0$ for all $u, v \in V$ and $\sum_{v \in V} p_{uv} = 1$ for all $u \in V$, i.e., $\boldsymbol{p}_u := (p_{uv})_{v \in V}$ is a p.d. on V for all $u \in V$. The quantity $p_{uv} \in [0, 1]$ is called the transition probability from vertex u to vertex v.

Then, the sum condition for Markov kernel P can be rewritten as (where the case $u = v$ is also allowed):

$$\sum_{w : v \to w} p_{vw} + p_{vv} = 1, \quad v \in V. \tag{4.1}$$

A p.d. $\boldsymbol{\pi}$ on V is a stationary distribution (s.d.) of the kernel P if $\sum_{u \in V} \pi_u p_{uv} = \pi_v$ for all $v \in V$. For a G-subordinated Markov kernel P this formula, the so-called global balance equation, can be expressed as:

$$\sum_{u : u \to v} \pi_u p_{uv} + \pi_v p_{vv} = \pi_v, \quad v \in V. \tag{4.2}$$

Fig. 4.4 presents a Markov kernel with its s.d. From a practical point of view, subordination of a Markov kernel means that a transition probability is positive for an entry p_{uv} only if an edge exists between u and v in the

graph.

From the viewpoint of the evaluation of our model, it will be important to choose the set E for the role of the state space. Our software can only extract the number of cars on the streets, where a street is a set of edges (and it is somewhat more natural to assign vehicles to edges). For a p.d. $(\pi'_e)_{e\in E}$ on E we can think of π'_e as the proportion of the number of vehicles at the road segment e with respect to the whole number of vehicles in the city.

A transition probability matrix (or Markov kernel) on E, i.e., on the line digraph $\mathrm{L}(G)$, can be defined as a real kernel $P' := (p'_{ef})_{e,f\in E}$ such that $p'_{ef} \geq 0$ for all $e, f \in E$ and $\sum_{f\in E} p'_{ef} = 1$ for all $e \in E$. A p.d. $\boldsymbol{\pi}'$

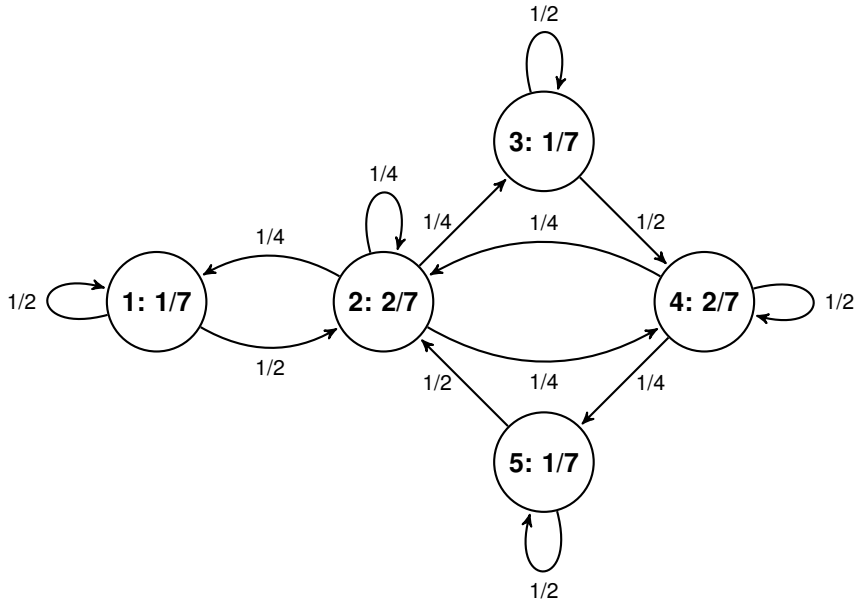


Figure 4.4: A Markov kernel (on edges) with its stationary distribution (on vertices) on the road network in Fig. 4.3.

Table 4.1: An example for a Markov kernel on the minimal line digraph of the road network in Fig. 4.3.

	(1,2)	(2,3)	(3,4)	(4,2)	(2,1)	(2,4)	(4,5)	(5,2)
(1,2)	1/2	1/4	0	0	0	1/4	0	0
(2,3)	0	1/2	1/2	0	0	0	0	0
(3,4)	0	0	1/2	1/4	0	0	1/4	0
(4,2)	0	1/4	0	1/2	1/4	0	0	0
(2,1)	1/2	0	0	0	1/2	0	0	0
(2,4)	0	0	0	0	0	1/2	1/2	0
(4,5)	0	0	0	0	0	0	1/2	1/2
(5,2)	0	1/4	0	0	1/8	1/8	0	1/2

on E is a s.d. of the kernel P' if $\sum_{e\in E} \pi'_e p'_{ef} = \pi'_f$ for all $f \in E$. Since G represents a road system we may suppose that if $e \neq f$ then $p'_{ef} > 0$ implies that $(e, f) \in E'$, i.e., there exist $u, v, w \in V$ such that $e = (u, v)$ and $f = (v, w)$, and hence, $u \to v \to w$ is a walk of length 2. In other words, we may suppose that the Markov kernel P' is $\mathrm{L}(G)$-subordinated. Similarly to Markov kernels on V, P' also induces an associated graph $G_{P'} = (V', E'_{P'})$. Thus, P' is $\mathrm{L}(G)$-subordinated if and only if $E'_{P'} \subseteq E' \cup S'$ where S' denotes here the diagonal $\{(e, e) \,|\, e \in E\}$ in $L(G)$. In this case, we use the notation $p'_{ef} = p'_{uvw}$ as well. In fact, p'_{uvw} denotes the probability that a vehicle on the road segment (u, v) will go further to the road segment (v, w) in the next time point. Moreover, in the case of $e = f = (u, v)$, let $p'_{ee} = p'_{uv}$ be the probability that a vehicle remains on the same road segment in the next time point which can be non-zero as well. Thus, since P' is a Markov kernel, we have that, for all $u \to v$,

$$\sum_{w:v\to w} p'_{uvw} + p'_{uv} = 1 \tag{4.3}$$

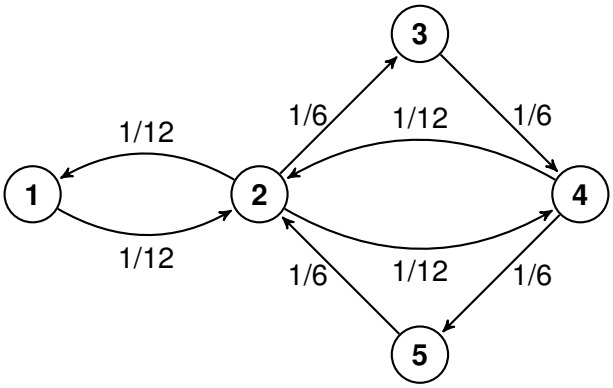


Figure 4.5: The stationary distribution of the Markov kernel in Table 4.1.

and the global balance equation is given as:

$$\sum_{u:u\to v} \pi'_{uv} p'_{uvw} + \pi'_{vw} p'_{vw} = \pi'_{vw} \tag{4.4}$$

for all $v \to w$. (Note that p'_{uv} is the probability that a vehicle remains on the road segment (u, v).)

By deleting all of the unnecessary edges, such that the remaining line digraph be still strongly connected we get a minimal strongly connected line digraph of G. This line digraph is denoted by $\mathrm{ML}(G)$. An example for the Markov kernel P' on the minimal line digraph $\mathrm{ML}(G)$ of the road network G in Fig. 4.3 is shown in Table 4.1. Fig. 4.5 shows the unique stationary distribution $\boldsymbol{\pi}'$ of the Markov kernel P'.

We can define probability distributions and Markov kernels on open road networks as well, see section 2.2 of paper [9].

Uniqueness of stationary distribution.

Definition 3. The Markov kernel P on V is called G-compatible if, for any

$u, v \in V$ such that $u \neq v$, $p_{uv} > 0$ if and only if $(u, v) \in E$. Similarly, the Markov kernel P' on E is called G-compatible if it is $\mathrm{L}(G)$-compatible Markov kernel on $\mathrm{L}(G)$. This is equivalent to that $p'_{uvw} > 0$, $u, v, w \in V$, if and only if $(u, v), (v, w) \in E$. Since $(e, f) \in E'$ if and only if there exist $u, v, w \in V$ such that $e = (u, v)$ and $f = (v, w)$ we can define the G-compatibility of a Markov kernel P' as, for any $e, f \in E$ such that $e \neq f$, $p'_{ef} > 0$ if and only if there exist $u, v, w \in V$ such that $e = (u, v)$ and $f = (v, w)$.

In other words, the compatibility of a Markov kernel means that a positive transition probability for an entry in the Markov kernel is only possible, if and only if there exists an edge between the two corresponding nodes in the graph (i.e., there is a road segment between the two nodes in the OSM).

If P is G-compatible, then the strong connectivity of G implies that the associated graph to the Markov kernel P is also strongly connected. In this case, the Markov kernel (the transition matrix) P is called irreducible. Thus, by Theorem 1 in [50], the following theorem holds. See also Theorem 3.1 and 3.3 in Chapter 3 of [13].

Theorem 1. If a road network G is strongly connected, then there is a unique stationary distribution $\boldsymbol{\pi}$ ($\boldsymbol{\pi}'$) to any G-compatible Markov kernel P (P'). Moreover, this distribution satisfies $\pi_v > 0$ for all $v \in V$ ($\pi'_{uv} > 0$ for all $(u, v) \in E$).

In other words, if a graph G is strongly connected, than the stationary distibution $\boldsymbol{\pi}$ will be unique for a given Markov kernel P.

The importance of the theorem is that, all of the Markov kernels which are defined on a physical road network's closure that has positive transition probability on all roads, have a unique stationary distribution. Therefore,

we can suppose that a real traffic which follows a Markovian dynamic has a local unique stationary distribution in a short time period. This s.d. can be calculated by observing the properties of the traffic.

4.3.2 Markov random walk and Markov traffic on road networks

Let $(\Omega, \mathscr{A}, \mathbb{P})$ be a probability space. Then a V-valued (E-valued) random variable (r.v.) is a $X : \Omega \to V$ ($Y : \Omega \to E$) measurable function, i.e., $X^{-1}(v) \in \mathscr{A}$ for all $v \in V$ ($Y^{-1}(e) \in \mathscr{A}$ for all $e \in E$). In this case, X (Y) is a random function on the set V of vertices (on the set E of edges). For example, X (Y) can be the random position of a vehicle on the road network G, where the position refers to the actual vertex (edge) which the vehicle belongs to. Then, $\mathbb{P}(X^{-1}(v)) = \mathbb{P}(X = v)$ ($\mathbb{P}(Y^{-1}(e)) = \mathbb{P}(Y = e)$) denotes the probability that a vehicle is at the vertex $v \in V$ (at the edge $e \in E$). Clearly, by $\boldsymbol{\pi}_X(v) := \mathbb{P}(X = v)$, $v \in V$, a r.v. X induces a p.d. $\boldsymbol{\pi}_X$ on V. Similarly, by $\boldsymbol{\pi}'_Y(e) := \mathbb{P}(Y = e)$, $e \in E$, a r.v. Y induces a p.d. $\boldsymbol{\pi}'_Y$ on E.

A sequence $\{X_t\}_{t\in\mathbb{Z}_+}$ of V-valued r.v.'s is a Markov chain on the state space V if the Markov property holds:

$$\begin{aligned}\mathbb{P}(X_t = v_t | X_{t-1} = v_{t-1}, \ldots, X_0 = v_0) \\ = \mathbb{P}(X_t = v_t | X_{t-1} = v_{t-1})\end{aligned}$$

for all $t \in \mathbb{N}$, $v_0, \ldots, v_t \in V$. If X, X' are V-valued r.v.'s then for the conditional distribution $P = (p_{vv'})_{v,v'\in V}$, $p_{vv'} := \mathbb{P}(X = v | X' = v')$, $v, v' \in V$, we shall also use the notation $X|X'$. Clearly, $X|X'$ is a Markov

kernel on V. Similarly, a Markov chain $\{Y_t\}_{t\in\mathbb{Z}_+}$ of E-valued r.v.'s can also be defined through the Markov kernel $Y|Y'$ on the state space E.

The Markov random walk and the Markov traffic defined in the following way.

Definition 4. Let the road network G be strongly connected and let P be a G-compatible Markov kernel on V with unique s.d. $\boldsymbol{\pi}$. Moreover, let $\{X_t\}_{t\in\mathbb{Z}_+}$ be a Markov chain on V such that $\boldsymbol{\pi}_{X_0} = \boldsymbol{\pi}$ and $X_t|X_{t-1} \sim P$ for all $t \in \mathbb{N}$.

Then, $\{X_t\}_{t\in\mathbb{Z}_+}$ is called **Markov random walk** on the road network G with Markov kernel P.

The set of k ($k \in \mathbb{N}$) mutually independent Markov random walks on G with Markov kernel P is called **Markov traffic** of size k and it is denoted by the quadruple $(G, P, \boldsymbol{\pi}, k)$.

Similarly, $\{Y_t\}_{t\in\mathbb{Z}_+}$ is a Markov random walk on the line road network if it is a Markov chain on the state space E such that $\boldsymbol{\pi}'_{Y_0} = \boldsymbol{\pi}'$ and $Y_t|Y_{t-1} \sim P'$ for all $t \in \mathbb{N}$.

A Markov random walk is an individual Markov traffic with $k = 1$ in the sense that it describes the movement of a random vehicle which follows the stochastic rules defined by the Markov kernel.

We also have results regarding the ergodicity of Markov traffic, see paper [9] and [18].

Two-dimensional stationary distribution. Introduce the two-dimensional distribution $Q = (q_{uv})$ on $V \times V$ as $q_{uv} := \pi_u p_{uv}$, $u, v \in V$. Then, Q is a two-dimensional stationary distribution on G in the following sense:

Definition 5. A matrix $Q = (q_{uv})_{u,v\in V}$ is called **two-dimensional stationary distribution** on G if (i) $q_{uv} \geq 0$ for all $u, v \in V$ and $q_{uv} = 0$ for all

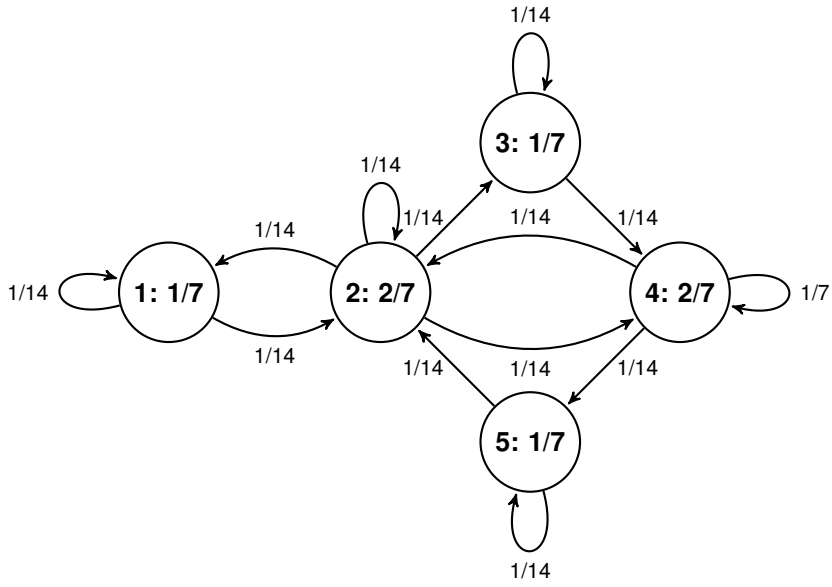


Figure 4.6: The two-dimensional stationary distribution (on edges) with its equidistributed marginals (on vertices) on the road network in Fig. 4.3 for the Markov kernel in Fig. 4.4.

$u, v \in V$ such that $(u, v) \notin E \cup S$ (i.e., Q is weakly G-subordinated); (ii) $\sum_{u,v \in V} q_{uv} = 1$ (i.e., Q is a normalized matrix on V); and (iii) $\sum_{v \in V} q_{uv} = \sum_{v \in V} q_{vu}$ for all $u \in V$ (i.e., Q has equidistributed marginals).

Property (iii) states that the two (row-wise and column-wise) marginal distributions of a two-dimensional stationary distribution on G coincide with each other. The distribution Q can be visualized on the edges, see, Fig. 4.6 for the toy example and Fig. 4.13 in case of the Porto example discussed later.

Denote by $\mathscr{Q}$ the set of two-dimensional stationary distributions on G.

For a positive $Q \in \mathscr{Q}$, let us define

$$\begin{aligned} \pi_u :=& \sum_{v \in V} q_{uv} = \sum_{v \in V} q_{vu}, \quad u \in V, \\ p_{uv} :=& \frac{q_{uv}}{\pi_u}, \quad u, v \in V. \end{aligned} \tag{4.5}$$

Then, $P = (p_{uv})$ defines a G-compatible Markov kernel with stationary distribution $\boldsymbol{\pi}$ on G. Thus, a Markov traffic defined by the quadruple $(G, P, \boldsymbol{\pi}, k)$ can be introduced by an equivalent way through the triplet (G, Q, k). We should note that in 4.5 an alternative way to calculate p_{uv} is given.

4.3.3 Statistical inference for Markov traffic using mobile sensors

The statistical analysis of a traffic system in our case means the estimation of the quadruple $(G, P, \boldsymbol{\pi}, k)$ or the triplet (G, Q, k) using observed data. The exploration of the road network G has already been done by a few organizations, in our example, we use OpenStreetMap data. In this section, we show a method for estimating the two-dimensional stationary distribution Q using mobile sensor data. (By (4.5), the estimators for P and $\boldsymbol{\pi}$ can be easily derived from an estimator of Q.) In this case, we have trajectories data which consists of the sequences of consecutive geographical points, like in the Taxi Trajectory Prediction (TTP) dataset, see section 4.4. It is important to note that it is not a simple task to fit GPS-based trajectory data to the vertices of a road network G, see section 4.5. We suppose that the size k of the traffic is known.

Suppose that, for a Markov traffic, we observed a random sample of trajectories $\{X^i\}$, $i = 1, \ldots, k$, of size k defined by $X_1^i \Rightarrow X_2^i \Rightarrow \ldots \Rightarrow X_{n_i}^i$, $i = 1, \ldots, k$, where n_i denotes the length of the i-th trajectory. (For a pair $u, v \in V$ the notation $u \Rightarrow v$ will mean that $(u, v) \in E \cup S$, i.e., either $u \to v$ or $u = v$.) Let $n := n_1 + \ldots + n_k$ be the total sample size. Define the total two-dimensional consecutive empirical frequencies as:

$$n_{uv} := \sum_{i=1}^{k} n_{uv}^i, \tag{4.6}$$

$u, v \in V$, where the trajectory-wise two-dimensional consecutive empirical frequencies, $i = 1, \ldots, k$, are defined as

$$n_{uv}^i := \sum_{j=1}^{n_i - 1} I(X_j^i = u, X_{j+1}^i = v),$$

$u, v \in V$. Plainly, n_{uv}^i denotes the number of consecutive (u, v) $(u, v \in V)$ pairs in the i-th trajectory. Since $\{X^i\}$ is a proper Markov random walk, we have $n_{uv}^i = 0$ for all $(u, v) \notin E \cup S$. Thus, the support of the two-dimensional frequency matrices $N := (n_{uv})_{u,v \in V}$, $N_i := (n_{uv}^i)_{u,v \in V}$, $i = 1, \ldots, k$, is a subset of $E \cup S$, i.e., they are weakly G-subordinated matrices. Clearly, $N = \sum_{i=1}^{k} N_i$ and we have

$$\sum_{u,v : u \Rightarrow v} n_{uv} = n - k, \tag{4.7}$$

where $n - k$ is the corrected sample size. Introduce

$$s_v := \sum_{i=1}^{k} I(X_1^i = v), \quad e_v := \sum_{i=1}^{k} I(X_{n_i}^i = v),$$

$v \in V$, i.e., s_v denotes the number of trajectories which start at vertex v and e_v denotes the number of trajectories which terminate at vertex v, respectively. Denote the one-dimensional marginal frequencies of N by $n_{v+} := \sum_{u \in V} n_{vu}$ and $n_{+v} := \sum_{u \in V} n_{uv}$, $v \in V$. We obtain that

$$n_{v+} + e_v = n_{+v} + s_v = n_v := \sum_{i=1}^{k} \sum_{j=1}^{n_i} I(X_j^i = v) \tag{4.8}$$

for all $v \in V$, where n_v denotes the number of vertices v in all trajectories. Finally,

$$\sum_{v \in V} s_v = \sum_{v \in V} e_v = k. \tag{4.9}$$

Define the vectors $\boldsymbol{s}$ and $\boldsymbol{e}$ on V as $\boldsymbol{s} := (s_v)_{v \in V}$ and $\boldsymbol{e} := (e_v)_{v \in V}$, respectively. Then, (4.9) implies that $\mathbf{1}^\top (\boldsymbol{e} - \boldsymbol{s}) = 0$, i.e., the vectors $\boldsymbol{e} - \boldsymbol{s}$ and $\mathbf{1}$ are orthogonal.

Let $A = (a_{uv})_{u,v \in V}$ and $B = (b_{uv})_{u,v \in V}$ such that $a_{uv} = b_{uv} = 0$ for all $u, v \in V$ where $u \not\Rightarrow v$, i.e., let A and B be weakly G-subordinated matrices. The distance between A and B is defined as

$$\|A - B\|_G := \left(\sum_{u,v : u \Rightarrow v} |a_{uv} - b_{uv}|^2 \right)^{1/2}.$$

In fact, $\|\cdot\|_G$ is the Frobenius norm of the matrices of dimension $|V| \times |V|$

which vanish on the entries outside of $E \cup S$.

Based on k number of trajectories, using the Frobenius norm, the optimality criterion is defined as the weighted sum of squared errors (SSE):

$$\mathrm{SSE}(M, \boldsymbol{w} \mid \boldsymbol{N}) := \sum_{i=1}^{k} w_i^{-1} \|N_i - w_i M\|_G^2, \tag{4.10}$$

where M is a non-negative parameter matrix satisfying assumptions (i) and (iii) of Definition 5, $\boldsymbol{w} = (w_i)_{i=1,\dots,k}$ are non-negative unknown weights, i.e., $\sum_{i=1}^{k} w_i = 1$, and $\boldsymbol{N} := (N_i)_{i=1,\dots,k}$ denotes the data, where N_i is the two-dimensional consecutive empirical frequency matrix for the ith trajectory, see (4.6). The statistical inference for a Markov traffic means the minimization of the objective function SSE in its parameters M and $\boldsymbol{w}$ deriving the weighted least squares (WLS) estimators $\widehat{M}_{\mathrm{WLS}}$ and $\widehat{\boldsymbol{w}}_{\mathrm{WLS}}$. Then, the WLS estimator of Q is defined as $\widehat{Q}_{\mathrm{WLS}} := n_{\mathrm{eff}}^{-1} \widehat{M}_{\mathrm{WLS}}$ where $n_{\mathrm{eff}} := (\mathbf{1}^\top \widehat{M}_{\mathrm{WLS}} \mathbf{1})$ is the so-called effective sample size.

The main theorem of this section is the following.

Theorem 2. There is a unique pair $(\widehat{M}_{\mathrm{WLS}}, \widehat{\boldsymbol{w}}_{\mathrm{WLS}})$ which minimizes the weighted sum of squared errors SSE defined in (4.10). These WLS estimators are derived as

$$\widehat{w}_{\mathrm{WLS}}^{i} := \frac{\|N_i\|_G}{\sum_{j=1}^{k} \|N_j\|_G},$$

$i = 1, \dots, k$, and

$$\widehat{M}_{\mathrm{WLS}} := N + (\mathbf{1}\boldsymbol{\lambda}^\top - \boldsymbol{\lambda}\mathbf{1}^\top) \circ A,$$

where $\boldsymbol{\lambda} = (\lambda_v)_{v \in V}$ is called Lagrange vector and defined as a unique solution to the linear equation $L\boldsymbol{\lambda} = \boldsymbol{s} - \boldsymbol{e}$ and $\circ$ denotes the entrywise

(Hadamard) product of matrices, L is the symmetric unnormalized graph Laplacian matrix of G, and defined as $L := D - A - A^\top$, see [95], where A denotes the adjacency matrix of G and $D := \mathrm{diag}\{\boldsymbol{d}^+ + \boldsymbol{d}^-\}$.

For the detailed proof of Theorem 2, see paper [9], Supporting information, S1 Appendix. (Due to the length regulations of the thesis, we must omit the proof.)

The estimation theory of finite Markov chains has a wide literature, see [10]. In the traditional maximum likelihood (ML) approach, the estimators have a few problems and they can be applied with limited success for our purposes. Firstly, they are based on only one long trajectory (or realization). In a real traffic dataset this is quite rare. Usually, there is a large number of relatively short trajectories, like in the Taxi Trajectory Dataset, where the number of trajectories is above 80,000, the mean length is 40 and the maximum length is 2,000, see Table 4.2. During the evaluation of our model, we will mimic this on small and medium size road networks, see section 4.7. Secondly, the global balance equation holds only asymptotically, i.e., when the sample size tends to infinity. The inaccuracy in the global balance equation is not too large, but, this small bias can cause significant difference from the "true" stationary distribution in the simulation. Thirdly, the trajectories are biased during a short time period. In the morning, the vehicles are moving from the residential districts to the business districts of the city and they are moving back in the afternoon. In other words, the traffic has a definite direction on the road network. Fig. 4.10(c) shows this behavior clearly by the distribution of the elements of the traffic direction vector $\boldsymbol{s} - \boldsymbol{e}$ while Fig. 4.10(b) shows their spatial distribution. We can handle these problems with our proposed WLS estimator. $\widehat{Q}_{\mathrm{WLS}}$ is

taking account of more than one trajectory with their length. In addition, it can correct the bias caused by the unbalanced sampling of trajectories on the road network.

Our proposed $\widehat{Q}_{\mathrm{WLS}}$ (or $\widehat{M}_{\mathrm{WLS}}$) estimator consists of two parts. First, the naive estimator for the distribution of the consecutive pairs in trajectories based on the empirical frequencies, and second, a correction term ensuring that $\widehat{Q}_{\mathrm{WLS}}$ (or $\widehat{M}_{\mathrm{WLS}}$) has equidistributed marginals. This second part depends on the Laplacian matrix of the road graph and the traffic direction vector. It is important to note that all statistics (two-dimensional consecutive frequencies, starting and ending frequencies) can be computed by counting, which is a very effective way and can be applied for big data, as well.

All mathematical methods presented in this section can be downloaded and tested. See Appendix, [3, model-sources/Markovkernel subfolder] and the rest of this chapter.

4.4 Public datasets

In our terminology, a trajectory (or trace) is a sequence of geographical data (coordinates with timestamp) of the path of a vehicle, moving from a start to an end point. For our experiments, we need a dataset of trajectories that conforms with the following criteria:

- Contains complete trajectories, so trajectories that contain only the start and the end point are not suitable.
- Trajectories must be sampled in a high frequency, the distance between adjacent measured points should not be too large (10 meters is

acceptable, but 100 meters is not).

- The dataset should cover a long enough time period and the number of trajectories per day should be in the order of thousands.
- Since the OOCWC operates on cities, the dataset should cover an urban area well.
- Vehicles should not follow a fixed route, e.g., bus lanes are not suitable.
- Publicly available and can be used for research purposes.

We observed several candidates, many datasets contained only the start and end points, e.g., New York City Taxi Trip Data,[2] TLC Trip Record Data,[3] Uber TLC FOIL Response[4]. Another example that is not suitable is the T-Drive trajectory dataset, because it was created using an average sampling interval of 177 seconds, so the average distance between two points is 623 meters.

We found two suitable datasets, one is the Uber GPS Traces, the other is the Taxi Trajectory Prediction from Kaggle[5]. The Uber dataset covers one week and contains 25,000 trajectories from San Francisco, CA, USA. Its sampling rate is 4 seconds. The Taxi Trajectory Prediction dataset covers one year and is split into training and test sets. The training set contains

[2] http://www.datadrivenstanford.org/features/nyc-taxi-data/

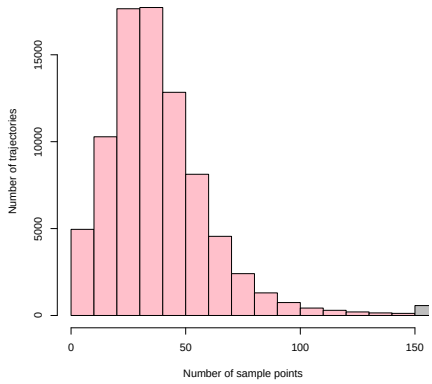
[3] https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[4] https://github.com/fivethirtyeight/uber-tlc-foil-response

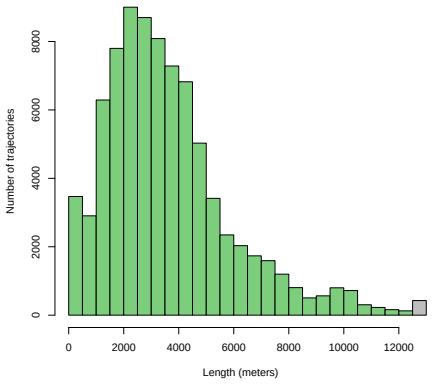
[5] https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i

1,710,670, the test set contains 320 trajectories collected in Porto, Portugal. The sampling rate was 15 seconds. Because of its much larger size, we use the Taxi Trajectory Prediction dataset.

As a first step of processing the data, we filtered the dataset between the coordinates W8.6518, W8.5771, N41.1129, N41.1756 (see Fig. 4.8), and trajectories that have a time of start between 8-9 am. The size of the area is 6.274 km $\times$ 6.963 km = 43.68 km^2, the set contains 82,345 trajectories. Features that are not relevant for us, such as origin of call, costumer IDs, etc, are omitted. Most samples are around the length of 41 coordinates, the longest contains 2,324 points. Some descriptive statistics of the dataset are shown in Fig. 4.7 and Table 4.2, see [17].



(a) Histogram of number of sample points per trajectories. The grey bar on the right represents trajectories with more than 150 sample points. On the average, a trajectory consists of 40 sample points and takes $\approx$10 minutes.



(b) Histogram of trajectory lengths. The grey bar on the right represents trajectories longer than 12,500 meters. The average trajectory length is 3,628.93 meters.

Figure 4.7: Some statistics of the dataset. Source: [9].

Table 4.2: Descriptive statistics of lengths of trajectories. (82,345 total.)

Name of statistics	Distance in points	Distance in meters
Mean	39.53	3,628.93
Median	35	3,176.786
Mode	34	0
Standard Deviation	31.64	2,408.93
Kurtosis	473.28	9.9
Skewness	12.11	1.78
Minimum	2	0
Maximum	2,324	61,055.58

4.5 Graphs from OSM

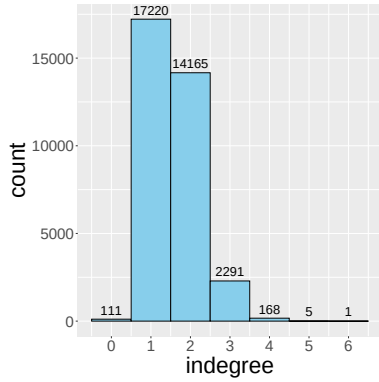
The first step of processing is to create a graph of Porto from the same bounding box that we used for subsetting the data. Since we only need nodes that can be reached by vehicles, we select only specific nodes of the OSM. For every node we store its coordinates and its OSM ID. Nodes will be vertices in the graph. The weight of the edges is the square distance between the two nodes. We used the pyosmium library[6] for processing OSM files and the NetworkX library[7] for creating the graph. The resulting graph has 33,961 nodes and 53,126 edges. The indegree and outdegree distribution (see section 4.3.1) of the Porto traffic graph can be observed in Fig. 4.9. Various distributions of the trajectory points (all, difference of start points and end points, histogram of difference) is shown in Fig. 4.10. We should note that we filter those nodes that has zero indegree or outdegree before we initiate simulations.

When the graph is ready, we process the list of trajectories. First, we

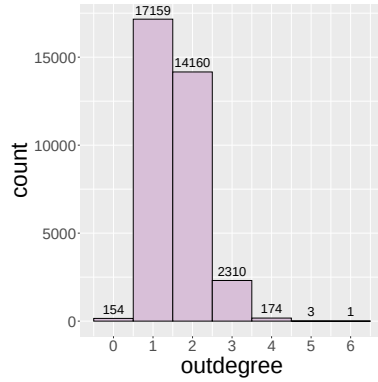
[6] `https://osmcode.org/pyosmium/`
[7] `https://networkx.github.io/`

Figure 4.8: Map of the area covered by the selected subset of the dataset. The size of the area is 6.274 km $\times$ 6.963 km = 43.68 km^2. ©OpenStreetMap contributors.

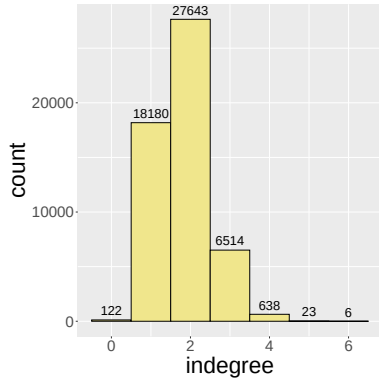
convert the GPS coordinates of the points to OSM node IDs. We simply search for the closest OSM node. Hence, the resulting node will be in the same domain as the graph built earlier. The coordinates are sampled in regular time interval, so the trajectories are not aligned to OSM data (and with the graph built previously). So, we need to interpolate the data to fit our graph. For this, we run Dijkstra's shortest path on our graph between every node ID. In some cases, because of OSM errors, there is no route between two nodes that are extracted from the dataset. In this case, we split that trajectory into two chunks.



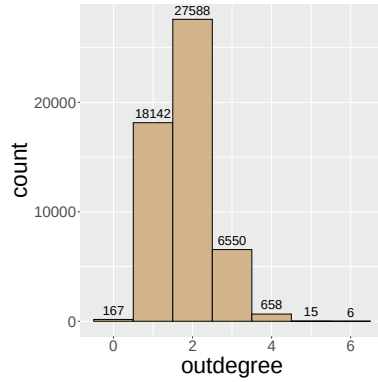
(a) Indegree distribution (vertices).



(b) Outdegree distribution (vertices).



(c) Indegree distribution (edges).



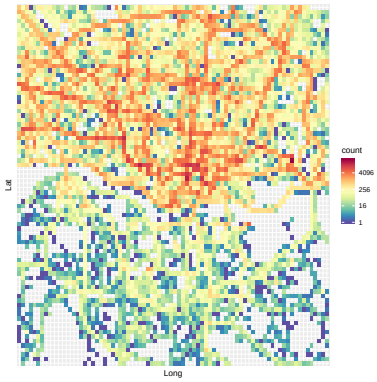
(d) Outdegree distribution (edges)

Figure 4.9: The degree distribution histograms of the Porto map traffic graph. Source: [9].

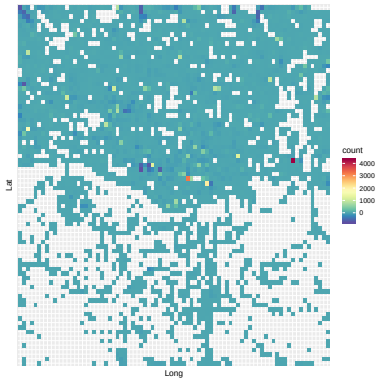
4.6 Implementation details

In this section, we present the implementation details of the model in the OOCWC and in some other related software.

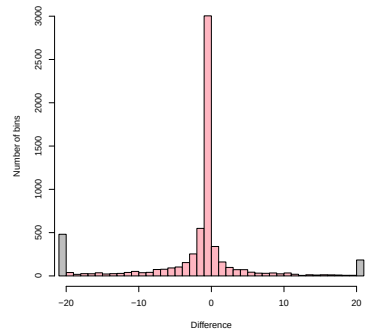
As a first step, we have created an R script for the filtering of the



(a) Distribution of all trajectory points shown in a 2D histogram (number of bins: 80×80).



(b) Difference of trajectory starting and endpoints shown in a 2D histogram (number of bins: 80×80). The color of each bin represents the number of trajectories starting points minus the number of trajectory endpoints that fall in that bin.



(c) Histogram of the difference of trajectory starting and endpoints.

Figure 4.10: Distribution of trajectory points of the filtered dataset. Source: [9].

Taxi Trajectory Prediction dataset and a Python script for the creation of the Markov kernel from the filtered dataset. The preprocessing scripts open the dataset and select the points according to the limitations described in section 4.4. Then, it saves these points in a new file[8]. The Python scripts first open the map database with the `WayNodeHandler` class and build a NetworkX graph by selecting the nodes that are relevant. After that, it converts every point (GPS coordinates) to an OSM node with the `matching_thread` function. This function is highly resource demanding, so we implemented it as a multi-process function, as shown in source 4.1 (some lines are omitted for brevity). One can see that we initiate this function as up to 6 distinct processes, one for each trajectory[9].

```
1  with open('pkdd15-subset-bbox-train.csv') as csvfile:
2      reader = csv.DictReader(csvfile)
3      for row in reader:
4          line = row['polyline'].translate(None,'[]')
5          coordinates = line.split(',')
6
7          threadd = multiprocessing.Process(target=
       matching_thread,args=(coordinates,))
8          threadd.daemon = True
9          threadd.start()
10
11         count = count + 1
12         if(count%6 == 0):
13             threadd.join()
14         ...
```

[8] see [3, /model-sources/Preprocessing]
[9] see [3, /model-sources/Markovkernel]

```
15 trajectories = np.asarray(trajectories)
```

Source 4.1: Multi-process processing of node conversion.

Since most trajectories are not complete in a sense that they are not matched with the OSM graph, i.e., the sampling of trajectories differs from the OSM graph, we interpolated every trajectory to obtain a neighbor-to-neighbor path on the graph, see the `interpolation` function. Finally, we create the Markov kernel as defined in section 4.3.

Regarding the RCE, we performed several modifications, including:

- Extended the OSMReader to read the Markov kernel file.
- Extended the shared memory segment in the Smart City Server to handle probability vectors.
- Modified the Car entity to work according to the probability vectors.

First, we extended the operation of the OSMReader to be able to handle kernel files. For this we added new command line arguments for the Smart City Server and added some basic file reading functionality to the OSM-Reader. We open the file and read every vector from the Markov kernel file into the corresponding shared memory segment.

Second, we extended the shared memory segment to handle probability vectors for every node. The relevant memory segment works as shown in source 4.2. The `AdjacencyList` is a map that consists of (1) the ID of the node (2) a pair that consists of (2a) the list of the IDs of the adjacent nodes and (2b) a list of the transition probabilities to the corresponding adjacent nodes. We read the transition probabilities from the kernel file to the `ProbabilityVect` segment. If there is no probability vector for a node, we initialize this vector according to uniform distribution.

```
typedef std::vector<osmium::unsigned_object_id_type>
    WayNodesVect;
typedef std::vector<double> ProbabilityVect;
typedef std::pair<WayNodesVect, ProbabilityVect>
    WayNodesProbability;
typedef std::map<osmium::unsigned_object_id_type,
    WayNodesProbability> AdjacencyList;
```

Source 4.2: The shared memory segment of a Markov traffic.

Finally, we had to modify the basic operation of the simulation algorithm. In the original implementation, the cars are moving on the map according to random walk or ant simulation. Now, a car selects the next node based on the probability vector of the current node. For this, we use the pseudo-random number generation engine from the Boost Random library[10] that is based on the method presented in paper [60]. The node selection procedure is shown in Source 4.3.

```
  double_vector prv = traffic.getProbabilityVector(
    next_m_from);

  boost::random::mt19937 gen;
  boost::random::discrete_distribution<> dist(prv);
  gen.seed(static_cast<unsigned int>(std::time(0)));

  osmium::unsigned_object_id_type next_m_to = dist(gen);
```

Source 4.3: Operation of the car entity in a Markov traffic simulation.

We illustrate the operation of the implementation of the simulation algorithm with an intersection. Observe Fig. 4.11 and Table 4.3. This in-

[10] https://www.boost.org/doc/libs/1_58_0/doc/html/boost_random.html

tersection has the OSM ID 1110673569 and GPS coordinates 41.1752185, -8.6231927. From the trajectory dataset, we calculated 1,649 total transitions on this intersection, i.e., 1,649 trajectories go through this intersection. The transitions to the neighbor nodes can be observed in Table 4.3. It is important to note that the actual transition probability (TP) is not the same as the frequency (or the maximum likelihood) based TP. The actual TP, which is the WLS based TP, is given by the Markov kernel file, and has been derived by our method. This simple example shows clearly the difference between the two methods.

Table 4.3: Transitions of intersection 1110673569.

Neighbor node	# of transitions	ML based TP	WLS based TP
1471136241	1449	0.879	0.6
1110673512	170	0.103	0.382
1837918561	30	0.018	0.018
Sum	1649	1	1

4.7 Evaluation of the proposed model

To evaluate the performance of the proposed WLS estimation method by comparing it to the traditional ML one, a simulation study was conducted at different sample sizes. The trajectories in the Porto dataset have a relatively short length (around 41 points) compared to the size of the traffic graph (around 34K vertex), so, in the simulations, we somewhat mimic this by keeping the length of trajectories low and the number of trajectories high. The absolute bias of an investigated estimator $\widehat{Q}$ for

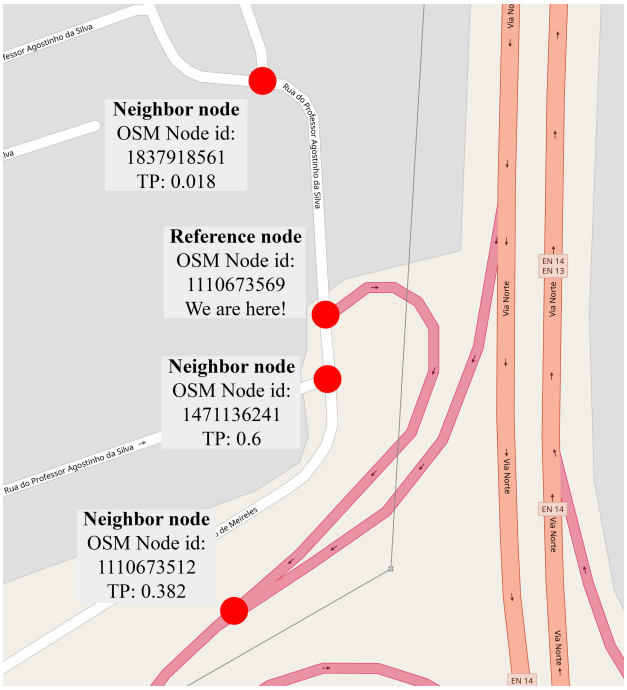


Figure 4.11: A visual explanation of transitions of intersection 1110673569. TP means transition probability, red dots indicate nodes. Source: [9]. (Base map and data from OpenStreetMap and OpenStreetMap Foundation. ©OpenStreetMap contributors, annotated by the authors.)

the two-dimensional stationary distribution Q as a parameter is defined by $\|\widehat{Q} - Q\|_G$. The empirical absolute bias and its standard error (SE) correspond to the mean and standard deviation of absolute biases in 100 replications, respectively. All simulations were carried out using the PyDTMC library[11].

First, we evaluated the model on the small road network in Fig. 4.3

[11] `https://pypi.org/project/PyDTMC/`

using the kernel of Fig. 4.4, see also the Appendix for a toy example. The parameters were $k = 100, 200, 500,$ and 1000 numbers of trajectories with $n = 3, 5$ and 10 lengths. The results can be seen in Table 4.4. The absolute bias and its standard error do not depend on n and, as expected, both are decreasing as k is increasing for the ML and the WLS estimation methods, as well. For small and medium k, the two methods perform similarly, but if k gets higher, the ML estimator outperforms the WLS one a bit. This phenomenon could be due to the asymptotic optimality of the ML estimator because the parameter k is large enough (1000 trajectories) compared to the size of the road graph (5 nodes).

For the second evaluation, we tried to mimic a real traffic network, so we exported a strongly connected subgraph from the OSM map of Porto. This graph contains 1,056 vertices and has somewhat similar properties than a large traffic graph, i.e., can be considered a sparse graph and has the same indegree and outdegree distribution. The GPS coordinates of the bounding box are W8.6137, W8.5991, N41.1573, N41.1437. The entries of Markov kernel were generated randomly, and the simulation parameters were $k = 1000, 3000$ and 5000 with $n = 3, 5$ and 10. It can be seen clearly in Table 4.5 that the absolute bias and the SE is also independent of n. However, the performance of the ML and the WLS estimator show significant difference as we change the parameter k. On the one hand, the absolute bias of the ML estimator is decreasing with higher k, but it is constant for the WLS one. On the other hand, the WLS estimator is better with lower k, but worse with higher k than the ML estimator. In real traffic settings, lower k's are more typical, so this simulation corroborates the superiority of WLS method based on two-dimensional stationary distribution against the traditional maximum likelihood. Finally, the scale of the SE's of the

WLS estimator shows that it is more stable in this scenario.

Finally, we tested the simulation model in the OOCWC. The initialization phase of the simulation adds traffic units to the vertices of the simulation graph. There exist two ways to do this, one is following a prescribed distribution (e.g. uniform), the other is following measured data. In this test setup, we initialized simulations with fictional data. We put units only to the streets Rua de Antero de Quental, Rua da Constituição and Rua da Boavista (25.6%, 51.4%, 23% of the cars, respectively), i.e., the simulation starts from the traffic configuration which is concentrated on three nodes of the road graph. We run simulations with $k = 5,000, 10,000, 20,000, 30,000$ and $50,000$ units. The simulation starts when all simulation units are added to the map. Fig. 4.12 shows the change of the distribution of cars during

Table 4.4: Simulation results, absolute bias and SE (inside parenthesis), for the Markov kernel in Fig. 4.4 on the road network in Fig. 4.3. (k - number, n - length of trajectories)

k	n	ML	WLS
100	3	0.034 (0.0103)	0.034 (0.0109)
100	5	0.035 (0.0106)	0.034 (0.0103)
100	10	0.033 (0.0095)	0.033 (0.0094)
200	3	0.024 (0.0066)	0.026 (0.0066)
200	5	0.023 (0.0064)	0.024 (0.0067)
200	10	0.024 (0.0071)	0.025 (0.0070)
500	3	0.015 (0.0046)	0.017 (0.0049)
500	5	0.015 (0.0041)	0.017 (0.0049)
500	10	0.016 (0.0047)	0.017 (0.0049)
1000	3	0.010 (0.0032)	0.013 (0.0041)
1000	5	0.011 (0.0034)	0.015 (0.0044)
1000	10	0.010 (0.0030)	0.014 (0.0040)

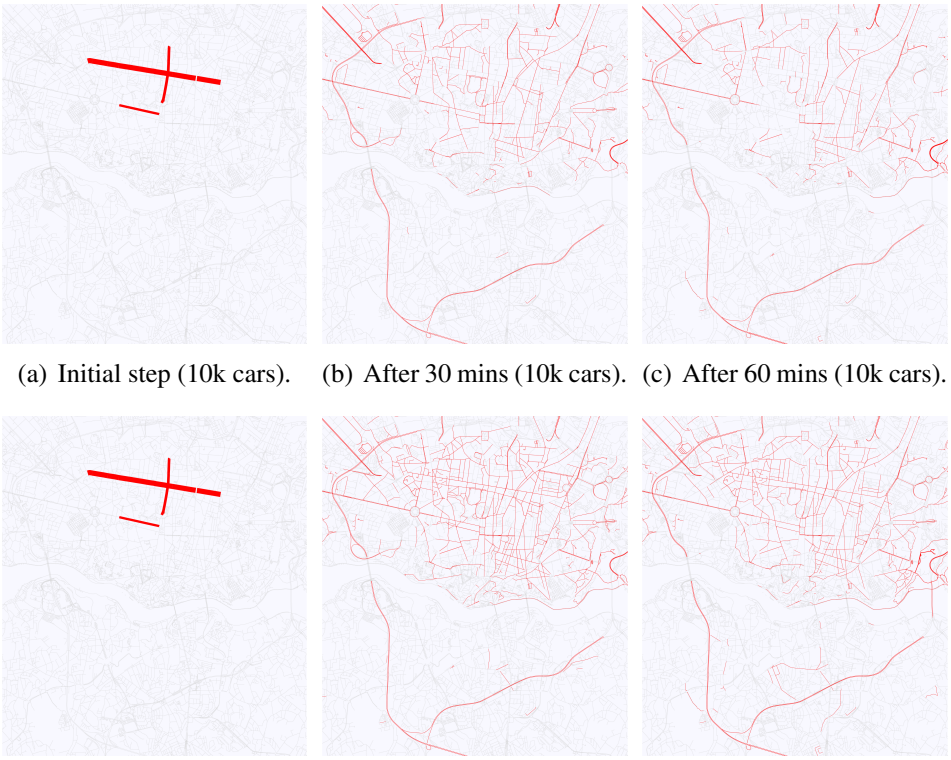
Table 4.5: Simulation results, absolute bias and SE (inside parenthesis), for a part of Porto's map with 1000 vertices. (k - number, n - length of trajectories)

k	n	ML	WLS
1000	3	0.166 (0.0559)	0.025 (0.0007)
1000	5	0.184 (0.1214)	0.025 (0.0007)
1000	10	0.169 (0.0938)	0.025 (0.0008)
3000	3	0.064 (0.1725)	0.023 (0.0005)
3000	5	0.070 (0.1665)	0.023 (0.0005)
3000	10	0.063 (0.1705)	0.023 (0.0005)
5000	3	0.016 (0.0150)	0.023 (0.0004)
5000	5	0.014 (0.0055)	0.023 (0.0004)
5000	10	0.014 (0.0126)	0.023 (0.0003)

the simulation.

We can extract and calculate the number of cars by street in every minute from a file produced by the RCE, so we can observe the change of distribution of the cars. Comparing it with the previously calculated stationary distribution (see Fig. 4.13), it shows us the operation of the implementation of the model. Please note the similarity between Fig. 4.13 and Fig. 4.10(a). The ticker line in Fig. 4.13 corresponds to increasingly hot color in Fig. 4.10(a).

We applied the Pearson's chi-squared test to compare the actual distribution of the cars minute-by-minute with the stationary distribution. We expect, based on the ergodicity of Markov traffic, see section 2.3 of paper [9], that during the simulation, independently from the initial distribution, within a certain time period, the distribution of the cars becomes close to the previously calculated stationary distribution. Fig. 4.14 shows the test results. We can observe that in the first few minutes the test statistic is high.



(a) Initial step (10k cars). (b) After 30 mins (10k cars). (c) After 60 mins (10k cars).

(d) Initial step (20k cars). (e) After 30 mins (20k cars). (f) After 60 mins (20k cars).

Figure 4.12: The change of the distribution of cars during the simulation (10,000 and 20,000 cars). The thickness of the street is proportionate with the number of cars on the street. Source: [9].

This means that the distribution of the cars is still far from the previously calculated steady state. However, after a time period, which depends on the number of cars, the test statistic becomes lower and remains the same, thus the distribution becomes steady. One can observe a reasonable phenomenon that it takes more time to reach the steady state with more traffic units. Another notable trend is the case of 5,000 cars, where the line is

elevating after reaching the steady-state. This can be caused by the low number of cars. The number of individual streets (named or unnamed, e.g., motorway junctions) is 2,194. 5,000 cars are simply not enough statistically to reach and hold a steady state in this type of simulation.

4.8 Conclusions on the simulation algorithms

In this chapter, we have outlined our mathematical model for traffic simulation that is called "Markov traffic". This model is based on tools from graph and Markov chain theories. Our aim was to create a simulation method that is able to keep the distribution of the cars on the traffic graph in a steady state. We have proven that, under general assumptions, the stationary distribution is unique for any Markov transition mechanism on a wide class of road networks.

Figure 4.13: The two-dimensional stationary distribution of cars in Porto based on the TTP dataset. Source: [9].

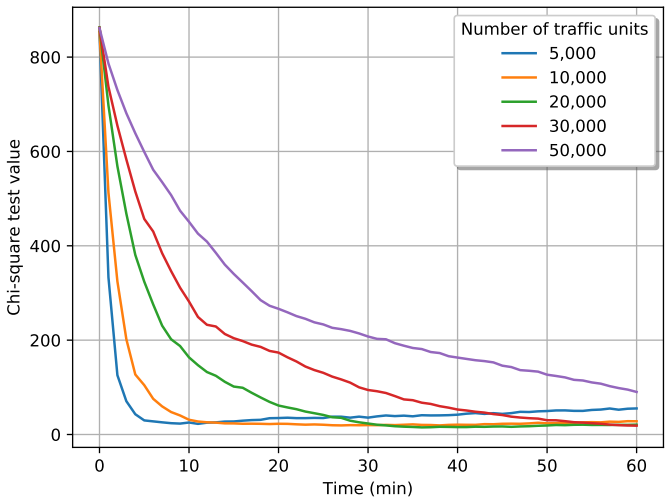


Figure 4.14: Chi-square test results. Source: [9].

This transition mechanism and the stationary distribution can be explored from observed trajectories. We have shown a statistical method with which we can create the Markov kernel which is necessary to obtain a Markov traffic. Using this kernel, we can initiate traffic simulations that provide a stationary distribution of the cars on the map. To provide an example for creating this kernel file, we have used a publicly available dataset, namely the Taxi Trajectory Prediction dataset. Our simulation uses OpenStreetMap, from which we extract the traffic graphs.

Finally, we should note a few minor implementation details that can be drawbacks of this model and may have an impact on its overall performance. In smaller graphs, the proposed algorithm and its implementation performs as it is expected, see the evaluation in section 4.7. But in the case

of large graphs, like our Porto example (33,961 nodes and 53,126 edges) and a very sparse TP matrix (see the degree distribution in Fig. 4.9), numerical problems may occur. First, when we calculate the $\widehat{Q}_{\mathrm{WLS}}$ estimator and then TP matrix using the method described in section 4.3.3, we cannot always solve the linear equation of the Lagrange vector in Theorem 2 numerically. We could only use the least square solution for a numerically stable calculation. In some cases, impossible numbers are present in the TP matrix, e.g., for a node, the TP vector is [1.17489, -0.174894]. It is rather interesting that the sum of these "impossible" TP vectors are 1 all the time, and mostly occurs if the node has a low number of transitions (less than 20). In these cases, we use the frequency-based TP instead. Second, problems can also occur when we calculate the stationary distribution π, namely, negative values may be present in the results. We chose to shift every value of π until we get a sum of 1 for π when we calculate the Pearson's Chi-square test. Finally, we should note that the OSM Porto map and the trajectory dataset do not cover each other perfectly, we only know the stationary distribution π for a subgraph of the whole map. The simulation units can traverse the whole OSM map graph, so, it can happen that a traffic unit reaches an edge which is not a part of the subgraph where we know the stationary distribution π. When we calculate the Pearson's chi-squared test, we consider only those cars that are present on the subgraph, where the stationary distribution π is known.

The whole project (including the RCE) is available for download[12]. Some simulation video is available on the YouTube channel of the author[13].

[12] see `https://github.com/rbesenczi/Crowd-sourced-Traffic-Simulator/blob/master/justine/install.txt`

[13] see `http://bit.ly/2FRpPxL`

Chapter 5

Conclusion

In this thesis, a smart city application is presented. The main function of this application is to simulate traffic in urban environments and to provide a research platform. The main contribution of this thesis is providing two reference implementations for two components of the system.

First, a complex system applied for data collection is presented. It consists of a hardware and a software part. The hardware part is a custom embedded system that can be assembled into cars. Its basic operation is to count cars on a given road segment, aggregate the data, then send data to a MQTT broker application. The Real-Time Traffic Analyzer has been tested and was suitable to conduct measurements and data collection. The RTTA is an important part of the OOCWC system. For traffic simulations, scenario analysis and effective route planning, we need real measured data, so we developed a rapid prototype application for data collection following the crowdsensing approach. The device is based on ARM which, with the software parts, will be ready to be assembled into vehicles. We performed

the initial test of the device in Debrecen and the collected data was used as an input to the RCE. It is important to note that the novelty of the system is that the device is moving during measurement. We had to take this into consideration during design planning. The measurement is a vehicle counting method that provides a density value on a given road segment or street. The application on server side collects the data, aggregates it, then assigns it to streets. After a minor development phase, the RCE could simulate traffic that was initialized with measured data.

Second, a simulation algorithm and its implementation are introduced that can provide a stationary distribution of the cars in a given time period. The proposed algorithm is based on graph theory and a Markov model of probability theory. It is shown that the proposed algorithm can simulate such traffic where the distribution of cars is stationary. We have outlined our mathematical model for traffic simulation that is called "Markov traffic". This model is based on tools from graph and Markov chain theories. Our aim was to create a simulation method that is able to keep the distribution of the cars on the traffic graph in a steady state. We have proven that, under general assumptions, the stationary distribution is unique for any Markov transition mechanism on a wide class of road networks. This transition mechanism and the stationary distribution can be explored from observed trajectories. We have shown a statistical method with which we can create the Markov kernel which is necessary to obtain a Markov traffic. Using this kernel, we can initiate traffic simulations that provide a stationary distribution of the cars on the map. To provide an example for creating this kernel file, we have used a publicly available dataset, namely the Taxi Trajectory Prediction dataset. Our simulation uses OpenStreetMap, from which we extract the traffic graphs.

Our future work will involve extending the simulation environment with different types of simulation units. We are planning to introduce different type of cars (electric, hybrid, gas, diesel) and alternative transportation, like public transit (bus, tram, trolley) or individual transit (bike and walk). Our aim is to observe the changes when travellers choose other mode of transportation. Nowadays, this is crucial from the viewpoint of many environmental and health factors.

Also, we are planning to use other data sources as well. The TTP dataset is not perfect in the sense that it is not matching the real traffic conditions of a city. There are new research initiatives that use mobile phone locations with the close cooperation of mobile phone service providers. These types of datasets are much more realistic and we believe that these can be used for more accurate estimation in our simulation environment.

Acknowledgments

In this thesis, we refer many times for the “original” implementation of the OOCWC (and its RCE component). This initial version was implemented from scratch by Norbert Bátfai, see its GitHub repository[1]. Every further development outlined in this thesis (and the corresponding papers) has been conducted by me in a fork of his software repository[2].
I am grateful to my thesis advisor Márton Ispány for his encouragement and guidance. I am also thankful to the Doctoral School of Informatics of the University of Debrecen for having provided the opportunity to conduct researches and to write my thesis.
I would like to thank all the support of the OOCWC and Smart City research groups at the Department of Information Technology. Members are: Márton Ispány, Norbert Bátfai, Péter Jeszenszky and the former students of the High Level Programming Languages courses.
I would like to express my special thanks to Norbert Bátfai, because without his exceptional straightforwardness this thesis would not exist. This work is dedicated to his kind memory.
I would like to thank Prof. István Oniga that we could use some of the devices of his research group at the early phase of the development.
I could not be grateful enough to my engineer-hacker friends Tamás Katona and Mihály Szilágyi. I cannot be grateful enough for Fanny Monori for her invaluable help and endless support.
Most importantly, I would like to thank all the support to my family and my parents.
Jelen munka a Nemzeti Tehetség Program, Egyedi fejlesztést biztosító ösztöndíjak NTP-EFÖ-P-15-0187 számú pályázat keretei között készült. A projektet az Emberi Erőforrások Minisztériuma Emberi Erőforrás Támogatáskezelő támogatta.
This thesis was supported by the National Talent Programme of Hungary, contract no.: NTP-EFÖ-P-15-0187.

[1] https://github.com/nbatfai/robocar-emulator/
[2] https://github.com/rbesenczi/Crowd-sourced-Traffic-Simulator

Bibliography

[1] S. Asmussen. *Applied Probability and Queues*, volume 51 of *Applications of Mathematics (New York)*. Springer-Verlag, New York, 2003.

[2] J. Bang-Jensen and G. Z. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer Science & Business Media, Berlin Heidelberg New York, 2008.

[3] R. Besenczi. Crowd-sourced Traffic Simulator software repository, 2019. URL `https://github.com/rbesenczi/Crowd-sourced-Traffic-Simulator`.

[4] R. Besenczi. Markov modell alapú közlekedés-szimulációs algoritmusok. In D. Tokody, E. Balla, and K. Németh, editors, *Okos Közlekedés Tudományos Konferencia 2019*, page 13. Doktoranduszok Országos Szövetsége, Műszaki Tudományok Osztálya, 2019. ISBN 978-963-318-570-4.

[5] R. Besenczi, G. Kövér, and M. Smajda. Közösségi alapú adatgyűjtő rendszer implementációja, 2013.

[6] R. Besenczi, T. Katona, and M. Szilágyi. A fork implementation of the Police Edition of the OOCWC system. In *Cognitive Infocommunications (CogInfoCom), 6th IEEE Conference on*, pages 163–164, 2015.

[7] R. Besenczi, M. Szilágyi, N. Bátfai, A. Mamenyák, I. Oniga, and M. Ispány. Using crowdsensed information for traffic simulation in the Robocar World Championship framework. In *Cognitive Infocommunications (CogInfoCom), 6th IEEE Conference on*, pages 333–337, 2015.

[8] R. Besenczi, N. Bátfai, P. Jeszenszky, R. Major, F. Monori, and M. Ispány. Large-scale analysis and simulation of traffic flow using Markov models, 2020. URL `https://arxiv.org/abs/2007.02681`.

[9] R. Besenczi, N. Bátfai, P. Jeszenszky, R. Major, F. Monori, and M. Ispány. Large-scale simulation of traffic flow using Markov model. *PLOS ONE*, 16(2):1–31, 02 2021. doi: 10.1371/journal.pone.0246062. URL https://doi.org/10.1371/journal.pone.0246062.

[10] P. Billingsley. *Statistical Inference for Markov Processes*. The University of Chicago Press, Chicago, 1961.

[11] P. Bocquier. World Urbanization Prospects: an alternative to the UN model of projection compatible with the mobility transition theory. *Demographic Research*, 12:197–236, 2005.

[12] M. F. Brameier and W. Banzhaf. Basic concepts of linear genetic programming. *Linear Genetic Programming*, pages 13–34, 2007.

[13] P. Brémaud. *Markov chains. Gibbs fields, Monte Carlo simulation, and queues*, volume 31 of *Texts in Applied Mathematics*. Springer-Verlag, New York, 1999.

[14] N. Bátfai, R. Besenczi, A. Mamenyák, and M. Ispány. OOCWC: The Robocar World Championship initiative. In *Telecommunications (ConTEL), 13th International Conference on*, pages 1–6, 2015.

[15] N. Bátfai, R. Besenczi, A. Mamenyák, and M. Ispány. Traffic simulation based on the Robocar World Championship initiative. *Infocommunications Journal*, 7(3): 50–59, 2015.

[16] N. Bátfai, P. Jeszenszky, A. Mamenyák, B. Halász, R. Besenczi, J. Komzsik, B. Kóti, G. Kövér, M. Smajda, C. Székelyhídi, T. Takács, G. Róka, and M. Ispány. Competitive programming: A case study for developing a simulation-based decision support system. *Infocommunications Journal*, 8(1):24–38, 2016.

[17] N. Bátfai, R. Besenczi, M. Ispány, P. Jeszenszky, R. S. Major, and F. Monori. Markov modeling and simulation of traffic flow. In *Data Science, Statistics & Visualisation, DSSV 2018*, page 61, 2018. URL http://cstat.tuwien.ac.at/filz/BoA.pdf.

[18] N. Bátfai, R. Besenczi, P. Jeszenszky, M. Szabó, and M. Ispány. Markov modeling of traffic flow in smart cities. *Annales Mathematicae et Informaticae*, (53):21–44, 2021. URL https://doi.org/10.33039/ami.2021.04.008.

[19] R. Carli, M. Dotoli, R. Pellegrino, and L. Ranieri. Measuring and managing the smartness of cities: A framework for classifying performance indicators. In *Systems, Man, and Cybernetics (SMC), IEEE International Conference on*, pages 1288–1293, 2013.

[20] M. Castro-Neto, Y.-S. Jeong, M.-K. Jeong, and L. D. Han. Online-SVR for short-term traffic flow prediction under typical and atypical traffic conditions. *Expert Systems with Applications*, 36(3):6164–6173, 2009.

[21] M. Cavers and K. Vasudevan. Spatio-temporal complex Markov chain (SCMC) model using directed graphs: Earthquake sequencing. *Pure and Applied Geophysics*, 172(2):225–241, 2015.

[22] K. Y. Chan, T. S. Dillon, J. Singh, and E. Chang. Neural-network-based models for short-term traffic flow forecasting using a hybrid exponential smoothing and Levenberg–Marquardt algorithm. *IEEE Transactions on Intelligent Transportation Systems*, 13(2):644–654, 2012.

[23] D. Charypar, K. W. Axhausen, and K. Nagel. Event-driven queue-based traffic flow microsimulation. *Transportation Research Record*, 2003(1):35–40, 2007. doi: 10.3141/2003-05.

[24] Y. Cheng, Y. Zhang, J. Hu, and L. Li. Mining for similarities in urban traffic flow using wavelets. In *2007 IEEE Intelligent Transportation Systems Conference*, pages 119–124, 2007.

[25] R. Chrobok, J. Wahle, and M. Schreckenberg. Traffic forecast using simulations of large scale networks. In *ITSC 2001. IEEE Intelligent Transportation Systems. Proceedings*, pages 434–439, 2001.

[26] E. Crisostomi, S. Kirkland, and R. Shorten. A Google-like model of road network dynamics and its application to regulation and control. *International Journal of Control*, 84(3):633–651, 2011.

[27] C. Dabrowski and F. Hunt. Using Markov chain and graph theory concepts to analyze behavior in complex distributed systems. Technical report, U.S. National Institute of Standards and Technology, 2011. https://www.nist.gov/sites/default/files/documents/itl/antd/DabrowskiHunt-Paper129-updated.pdf.

[28] J. Dallmeyer, A. D. Lattner, and I. J. Timm. From GIS to mixed traffic simulation in urban scenarios. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, pages 134–143, 2011. URL http://eudl.eu/pdf/10.4108/icst.simutools.2011.245680.

[29] G. A. Davis and N. L. Nihan. Nonparametric regression and short-term freeway traffic forecasting. *Journal of Transportation Engineering*, 117(2):178–188, 1991.

[30] R. De Santis, A. Fasano, N. Mignolli, and A. Villa. Smart city: fact and fiction, 2014. URL https://mpra.ub.uni-muenchen.de/54536/.

[31] H. Dia. An object-oriented neural network approach to short-term traffic forecasting. *European Journal of Operational Research*, 131(2):253–261, 2001.

[32] C. Du and S. Zhu. Research on urban public safety emergency management early warning system based on technologies for the Internet of Things. *Procedia Engineering*, 45:748–754, 2012.

[33] G. Duncan and J. Littlejohn. High performance microscopic simulation for traffic forecasting. *IET Conference Proceedings*, pages 4–4(1), 1997.

[34] M. Faizrahnemoon. *Real-data modelling of transportation networks*. PhD thesis, Hamilton Institute, National University of Ireland Maynooth, 2016.

[35] M. Faizrahnemoona, A. Schlote, E. Crisostomi, and R. Shorten. A Google-like model for public transport. In *International Conference on Connected Vehicles and Expo (ICCVE)*, pages 612–613, 2013.

[36] M. Faizrahnemoona, A. Schlote, L. Maggi, E. Crisostomi, and R. Shorten. A big-data model for multi-modal public transportation with application to macroscopic control and optimisation. *International Journal of Control*, 88(11):2354–2368, 2015.

[37] S. Fan, M. Herty, and B. Seibold. Comparative model accuracy of a data-fitted generalized Aw-Rascle-Zhang model. *arXiv preprint arXiv:1310.8219*, 2013.

[38] Fi-Ware. Fi-Ware, 2015. URL http://www.fiware.org/.

[39] Future City. Future City Glasgow, 2015. URL http://futurecity.glasgow.gov.uk/.

[40] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *IEEE communications Magazine*, 49(11):32–39, 2011.

[41] B. Ghosh, B. Basu, and M. O'Mahony. Multivariate short-term traffic flow forecasting using time-series analysis. *IEEE Transactions on Intelligent Transportation Systems*, 10(2):246, 2009.

[42] R. Giffinger, C. Fertner, H. Kramar, R. Kalasek, N. Pichler-Milanovic, and E. Meijers. Smart cities-Ranking of European medium-sized cities. Technical report, Vienna University of Technology, 2007.

[43] D. M. Gordon. The development of organization in an ant colony. *American Scientist*, 83(1):50–57, 1995.

[44] N. L. Hjort and C. Varin. ML, PL, QL in Markov chain models. *Scandinavian Journal of Statistics*, 35(1):64–82, 2008.

[45] A. Horni, K. Nagel, and K. W. Axhausen. *The multi-agent transport simulation MATSim*. Ubiquity Press London, 2016.

[46] iCity. iCity Project, 2015. URL `http://www.icityproject.eu/`.

[47] A. Ilyés, T. Kovács, G. Tisza, and I. Varga. Spatial characteristics of communication in urban vehicular system. In *COMPLEXIS*, pages 108–112, 2020.

[48] T. Iokibe, N. Mochizuki, and T. Kimura. Traffic prediction method by fuzzy logic. In *Second IEEE International Conference on Fuzzy Systems*, pages 673–678, 1993.

[49] E. Ismagilova, L. Hughes, Y. K. Dwivedi, and K. R. Raman. Smart cities: Advances in research—an information systems perspective. *International Journal of Information Management*, 47:88–100, 2019.

[50] J. Jarvis and D. R. Shier. Graph-theoretic analysis of finite Markov chains. *Applied mathematical modeling: a multidisciplinary approach*, pages 85–102, 1999.

[51] Y.-S. Jeong, Y.-J. Byon, M. M. Castro-Neto, and S. M. Easa. Supervised weighting-online learning algorithm for short-term traffic flow prediction. *IEEE Transactions on Intelligent Transportation Systems*, 14(4):1700–1707, 2013.

[52] X. Jiang and H. Adeli. Dynamic wavelet neural network model for traffic flow forecasting. *Journal of Transportation Engineering*, 131(10):771–779, 2005.

[53] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of the First International Conference on Autonomous Agents*, pages 340–347, 1997.

[54] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138, 2012. URL `http://elib.dlr.de/80483/`.

[55] S. Lee and D. B. Fambro. Application of subset autoregressive integrated moving average model for short-term freeway traffic volume forecasting. *Transportation Research Record*, 1678(1):179–188, 1999.

[56] A. Lesne. Complex networks: from graph theory to biology. *Letters in Mathematical Physics*, 78:235–262, 2006.

[57] L. Li, W.-H. Lin, and H. Liu. Type-2 fuzzy logic approach for short-term traffic forecasting. In *IEE Proceedings-Intelligent Transport Systems*, volume 153, pages 33–40. IET, 2006.

[58] R. Lienhart, A. Kuranov, and V. Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *Pattern Recognition*, volume 2781, pages 297–304. Springer, 2003.

[59] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang. Traffic flow prediction with big data: a deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):865–873, 2015.

[60] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[61] C. J. Messer. Advanced freeway system ramp metering strategies for Texas. Technical report, Texas Transportation Institute, College Station, TX, 1993.

[62] P. Millard, P. Saint-Andre, and R. Meijer. XEP-0060: publish-subscribe. *XMPP Standards Foundation*, 1:13, 2010. URL `https://xmpp.org/extensions/xep-0060.pdf`.

[63] P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, pages 323–336, 2008.

[64] F. Monori, R. Besenczi, and N. Bátfai. Forgalom szimulációs platform nyílt térképi adatbázisokon. In *Balázs, Boglárka (Eds.) Az elmélet és a gyakorlat találkozása a térinformatikában VII. = Theory meets practice in GIS*, pages 319–324, 2016.

[65] K. Nagel and M. Schreckenberg. A cellular automaton model for freeway traffic. *Journal de physique I*, 2(12):2221–2229, 1992.

[66] A. M. Nagy and V. Simon. Survey on traffic prediction in smart cities. *Pervasive and Mobile Computing*, 50:148–163, 2018. ISSN 1574-1192. doi: https://doi.org/10.1016/j.pmcj.2018.07.004. URL https://www.sciencedirect.com/science/article/pii/S1574119217306521.

[67] E. Necula. Dynamic traffic flow prediction based on GPS data. In *IEEE 26th International Conference on Tools with Artificial Intelligence*, pages 922–929, 2014.

[68] P. Neirotti, A. De Marco, A. C. Cagliano, G. Mangano, and F. Scorrano. Current trends in Smart City initiatives: Some stylised facts. *Cities*, 38:25–36, 2014.

[69] D. Ngoduy. Low-rank unscented Kalman filter for freeway traffic estimation problems. *Transportation Research Record*, 2260(1):113–122, 2011.

[70] H. Nicholson and C. Swann. The prediction of traffic flow volumes based on spectral analysis. *Transportation Research*, 8(6):533–538, 1974.

[71] V. Nugala, S. J. Allan, and J. W. Haefner. Parallel implementations of individual-based models in biology: bulletin- and non-bulletin-board approaches. *Biosystems*, 45(2):87–97, 1998.

[72] B. Pan, Y. Zheng, D. Wilkie, and C. Shahabi. Crowd sensing of traffic anomalies based on human mobility and social media. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 344–353. ACM, 2013.

[73] B. Park, C. J. Messer, and T. Urbanik. Short-term freeway traffic volume forecasting using radial basis function neural network. *Transportation Research Record*, 1651 (1):39–47, 1998.

[74] E. Popovici, A. Bucci, R. P. Wiegand, and E. D. De Jong. Coevolutionary principles. In G. Rozenberg, T. Bäck, and J. Kok, editors, *Handbook of Natural Computing*, pages 987–1033. Springer, Berlin, Heidelberg, 2012.

[75] A. Rayes and S. Salam. Internet of things from hype to reality. *Springer*, 2017.

[76] P. Saint-Andre. Extensible messaging and presence protocol (XMPP): Core. Technical report, Internet Engineering Task Force, 2011. URL http://www.rfc-editor.org/info/rfc6120.

[77] School of Computing, University of Utah. Pre-trained Cascade Database, 2015. URL http://www.cs.utah.edu/~turcsans/DUC/.

[78] J. Sewall, D. Wilkie, P. Merrell, and M. C. Lin. Continuum traffic simulation. In *Computer Graphics Forum*, volume 29(2), pages 439–448. Wiley Online Library, 2010.

[79] J. Sewall, D. Wilkie, and M. C. Lin. Interactive hybrid simulation of large-scale traffic. In *ACM Transactions on Graphics (TOG)*, volume 30(6), page 135. ACM, 2011.

[80] J. Siek, A. Lumsdaine, and L.-Q. Lee. *The Boost Graph Library: user guide and reference manual*. Addison-Wesley, 2002.

[81] B. L. Smith and M. J. Demetsky. Traffic flow forecasting: comparison of modeling approaches. *Journal of Transportation Engineering*, 123(4):261–266, 1997.

[82] B. L. Smith, B. M. Williams, and R. K. Oswald. Comparison of parametric and nonparametric models for traffic flow forecasting. *Transportation Research Part C: Emerging Technologies*, 10(4):303–321, 2002.

[83] K. Spieser, K. Treleaven, R. Zhang, E. Frazzoli, D. Morton, and M. Pavone. Toward a systematic approach to the design and evaluation of automated mobility-on-demand systems: a case study in Singapore. In *Road Vehicle Automation*, Lecture Notes in Mobility, pages 229–245. Springer, 2014. URL http://dspace.mit.edu/handle/1721.1/82904.

[84] A. Stathopoulos and M. G. Karlaftis. A multivariate state space approach for urban traffic flow modeling and prediction. *Transportation Research Part C: Emerging Technologies*, 11(2):121–135, 2003.

[85] S. Sun, C. Zhang, and G. Yu. A Bayesian network approach to traffic flow forecasting. *IEEE Transactions on Intelligent Transportation Systems*, 7(1):124–132, 2006.

[86] R. Szabó, K. Farkas, M. Ispány, A. Benczúr, N. Bátfai, P. Jeszenszky, S. Laki, A. Vágner, L. Kollár, C. Sidló, R. Besenczi, M. Smajda, G. Kövér, T. Szincsák, T. Kádek, M. Kósa, A. Adamkó, I. Lendák, B. Wiandt, T. Tomás, A. Nagy, and G. Fehér. Framework for smart city applications based on participatory sensing. In *Cognitive Infocommunications (CogInfoCom), IEEE 4th International Conference on*, pages 295–300, 2013.

[87] The European Innovation Partnership on Smart Cities and Communities. EIPSCC, 2013. URL http://ec.europa.eu/eip/smartcities.

[88] M. Treiber, A. Hennecke, and D. Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62(2):1805, 2000.

[89] R. E. Turochy and B. D. Pierce. Relating short-term traffic forecasting to current system state using nonparametric regression. In *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems*, pages 239–244, 2004.

[90] M. Van Der Voort, M. Dougherty, and S. Watson. Combining Kohonen maps with ARIMA time series models to forecast traffic flow. *Transportation Research Part C: Emerging Technologies*, 4(5):307–318, 1996.

[91] I. Varga. A complex sis spreading model in ad hoc networks with reduced communication efforts. *Advances in Complex Systems*, 23(04):2050009, 2020. URL https://doi.org/10.1142/S0219525920500095.

[92] I. Varga and G. Kocsis. Statistical properties of VANET-based information spreading. *Advances in Systems Science and Applications*, 20(4):36–44, 2020.

[93] I. Varga, A. Némethy, and G. Kocsis. Agent-based simulation of information spreading in VANET. In G. Mauri, S. El Yacoubi, A. Dennunzio, K. Nishinari, and L. Manzoni, editors, *Cellular Automata*, pages 166–174, Cham, 2018. Springer International Publishing. ISBN 978-3-319-99813-8.

[94] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, Proceedings of the IEEE Computer Society Conference on*, volume 1, pages I–511–I–518 vol.1, 2001.

[95] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4): 395–416, 2007.

[96] Y. Wang and M. Papageorgiou. Real-time freeway traffic state estimation based on extended Kalman filter: a general approach. *Transportation Research Part B: Methodological*, 39(2):141–167, 2005.

[97] Y. Xie and Y. Zhang. A wavelet network model for short-term traffic volume forecasting. *Journal of Intelligent Transportation Systems*, 10(3):141–150, 2006.

[98] Xilinx, AXI. Reference guide. *Xilinx Inc*, 2012.

[99] J. Xue and Z. Shi. Short-time traffic flow prediction based on chaos time series theory. *Journal of Transportation Systems Engineering and Information Technology*, 8(5):68–72, 2008.

[100] T. Yamauchi, M. Kutami, and T. Konishi-Nagano. Development of quantitative evaluation method regarding value and environmental impact of cities. *Fujitsu Sci. Tech. J*, 50(2):112–120, 2014.

[101] L. Z Karvalics. Okos városok: a dekonstrukciótól a hiperkonstrukcióig. *Információs Társadalom: Társadalomtudományi Folyóirat*, 16(3):9–22, 2017.

[102] Y. Zhang and Z. Ye. Short-term traffic flow forecasting using fuzzy logic system methods. *Journal of Intelligent Transportation Systems*, 12(3):102–112, 2008.

[103] Y. Zheng, L. Capra, O. Wolfson, and H. Yang. Urban computing: Concepts, methodologies, and applications. *ACM Trans. Intell. Syst. Technol.*, 5(3):38:1–38:55, 2014.

Appendix

A Toy Example

In order to demonstrate the main concepts and methods of section 4.3 we present a simple toy example[3]. Consider the road network $G = (V, E)$ in Fig. 5.1 where $V := \{1, 2, 3, 4, 5\}$ and
$E := \{(1, 2), (1, 3), (2, 3), (3, 1), (3, 4), (3, 5), (4, 1), (5, 3)\}$. Then $|V| = 5$ and $|E| = 8$. The adjacency matrix A_G of G, where we denote the vertices

[3]For a Python implementation, see: `https://github.com/rbesenczi/Crowd-sourced-Traffic-Simulator/blob/master/model-sources/Markovkernel/example_graph_thesis.py`

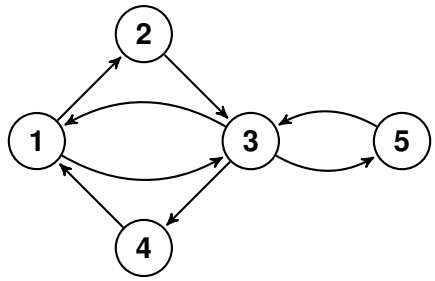


Figure 5.1: A simple road network.

as well, can be derived as:

$$A_G := \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 \\ 3 & 1 & 0 & 0 & 1 & 1 \\ 4 & 1 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 1 & 0 & 0 \end{array}.$$

Clearly, G is a strongly connected digraph. One can see that the indegree and outdegree of vertices are given as:

$$\boldsymbol{d}^- = \boldsymbol{d}^+ = \begin{bmatrix} 2 & 1 & 3 & 1 & 1 \end{bmatrix}^T.$$

The symmetric unnormalized graph Laplacian matrix L of the road network G is given as:

$$L = \begin{bmatrix} 4 & -1 & -2 & -1 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ -2 & -1 & 6 & -1 & -2 \\ -1 & 0 & -1 & 2 & 0 \\ 0 & 0 & -2 & 0 & 2 \end{bmatrix}$$

The eigenvalues and eigenvectors of the symmetric unnormalized graph

Laplacian matrix L are given as:

$$S = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1.72 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 4.46 & 0 \\ 0 & 0 & 0 & 0 & 7.82 \end{bmatrix}$$

and

$$O = \begin{bmatrix} 0.447 & -0.21 & 0 & 0.76 & -0.41 \\ 0.447 & -0.36 & -0.71 & -0.4 & -0.07 \\ 0.447 & 0.12 & 0 & 0.23 & 0.86 \\ 0.447 & -0.36 & 0.71 & -0.4 & -0.07 \\ 0.447 & 0.82 & 0 & -0.19 & -0.29 \end{bmatrix}$$

where S contains the eigenvalues in its diagonal and O is the orthonormal matrix of eigenvectors in its columns. The multiplicity of the smallest eigenvalue 0 is 1 which shows that the road network is strongly connected. The inverse of L on the subspace $\mathscr{S}$ which is the Moore-Penrose inverse of L, can be derived as

$$L_{\mathscr{S}}^{-1} = OS^{-1}O^{\top} = \begin{bmatrix} 0.18 & -0.02 & -0.02 & -0.02 & -0.12 \\ -0.02 & 0.36 & -0.05 & -0.14 & -0.15 \\ -0.02 & -0.05 & 0.11 & -0.05 & 0.013 \\ -0.12 & -0.13 & -0.05 & 0.36 & -0.15 \\ -0.12 & -0.15 & 0.013 & -0.15 & 0.41 \end{bmatrix}$$

where S^{-1} is the generalized inverse of S. $\mathscr{S}$ is the linear subspace of vectors which are orthogonal to $\mathbf{1}$.

Let the following trajectories be observed in the road network G:

Trajectory	Length	Count
$1 \to 2 \to 3 \to 5$	4	150
$1 \to 2 \to 3 \to 4$	4	100
$3 \to 4 \to 1$	3	350
$5 \to 3 \to 1$	3	250
$5 \to 3 \to 4$	3	50
$4 \to 1 \to 2$	3	100
		1000

Then, the total sample size is $n = 2250$, the number of trajectories is $k = 1000$ and the two-dimensional consecutive frequency matrix N is given by

$$N = \begin{bmatrix} 0 & 350 & 0 & 0 & 0 \\ 0 & 0 & 250 & 0 & 0 \\ 250 & 0 & 0 & 500 & 150 \\ 450 & 0 & 0 & 0 & 0 \\ 0 & 0 & 300 & 0 & 0 \end{bmatrix}.$$

The statistics for the starting and ending points of the trajectories are:

Node	Start	End	Diff
1	250	600	-350
2	0	100	-100
3	350	0	350
4	100	150	-50
5	300	150	150
Sum	1000	1000	0

The Lagrange multiplicators are:

$$\boldsymbol{\lambda} = \begin{bmatrix} -85 & -64.16 & 56.66 & -39.16 & 131.66 \end{bmatrix}^\top.$$

Thus, the correction matrix R is

$$R = \begin{bmatrix} 0 & 20.83 & 141.66 & 0 & 0 \\ 0 & 0 & 120.83 & 0 & 0 \\ -141.66 & 0 & 0 & -95.83 & 75 \\ -45.83 & 0 & 0 & 0 & 0 \\ 0 & 0 & -75 & 0 & 0 \end{bmatrix}$$

Since $\boldsymbol{d}^- = \boldsymbol{d}^+$ we have $n_{\text{eff}} = (n-k) = 2350$, and

$$N + R = \begin{bmatrix} 0 & 370.83 & 141.66 & 0 & 0 \\ 0 & 0 & 370.83 & 0 & 0 \\ 108.33 & 0 & 0 & 404.16 & 225 \\ 404.16 & 0 & 0 & 0 & 0 \\ 0 & 0 & 225 & 0 & 0 \end{bmatrix}$$

$$\widehat{Q}_{\text{WLS}} = \begin{bmatrix} 0 & 0.16 & 0.06 & 0 & 0 \\ 0 & 0 & 0.16 & 0 & 0 \\ 0.04 & 0 & 0 & 0.179 & 0.1 \\ 0.179 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 \end{bmatrix}$$

The stationary distribution is

$$\widehat{\boldsymbol{\pi}}_{\text{WLS}} = \begin{bmatrix} 0.227 & 0.165 & 0.328 & 0.18 & 0.1 \end{bmatrix}^\top$$

and one can easily check that $\boldsymbol{\pi}$ is indeed the stationary distribution of the estimated Markov kernel:

$$\widehat{P}_{\text{WLS}} = \begin{bmatrix} 0 & 0.723 & 0.277 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0.147 & 0 & 0 & 0.548 & 0.305 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Summary

In this thesis, a Smart City application is presented, called the Robocar World Championship. The application is in the research domain of smart cities and autonomous cars and can offer a platform to investigate theories and conduct new research ideas. The system itself consists of several components written in C++11 using shared memory segments and several external libraries, such as the Boost Graph Library. One component can visualize traffic in urban areas, another makes the system suitable to organize programming competitions. Two very important components of the system are the Robocar City Emulator, which can simulate traffic in urban areas, and the Automatic Sensor Annotations, which can measure traffic in cities.

The Automatic Sensor Annotations component collects data in urban areas, it is called the Real-Time Traffic Analyzer. It consists of a hardware and a software part. The hardware part is a custom embedded system that can be assembled into cars. Its basic operation is to count cars on a given road segment, aggregate the data, then send data to a central server infrastructure. The part consists of a Digilent Zybo development board, a camera module, a GPS module and a GSM module. The method used for the counting of the cars is a Haar-cascade based object detection algorithm. The development of the component conducted in the Vivado IDE and the software running on the board was developed in C++11. The collected data is sent to the server with the use of the MQTT protocol as a Google Protocol Buffer message. The software part is an MQTT broker application that can visualize and analyze incoming data. The Real-Time Traffic Analyzer has been tested and was suitable to conduct measurements and data collection.

The output of the Real-Time Traffic Analyzer is an aggregated data that

can be served as an input for the Robocar City Emulator. This component can simulate traffic in urban areas. The original version of the system was inspected and was proven to be wrong regarding its simulation algorithm. So, we developed an algorithm that can provide a stationary distribution of the cars in a given time period. The proposed algorithm is based on graph theory and a Markov model of probability theory. The solution uses OpenStreetMap as a geographical data source and an open dataset, called the Taxi Trajectory Prediction. Based on these data sources a graph and a transition matrix on it can be built. This matrix contains the transitions between adjacent nodes, i.e., a probability vector for each node. This probability vector shows the probability of transition to the adjacent node for each node. In this thesis, the method of the construction of such a matrix is given. In addition, it is shown that the proposed algorithm can simulate such traffic where the distribution of cars is stationary and this stationary distribution is unique for a given graph and transition probability matrix.

Összefoglaló

Jelen PhD disszertációban egy Smart City alkalmazást mutatunk be, melynek neve Robocar World Championship. Az alkalmazás a smart city és önvezető autók kutatási terület között helyezkedik el, fő célja, hogy egy kutatási platformot kínáljon különböző ötletek, elméletek tesztelésére, kutatások lefolytatására. A rendszer több komponensből épül fel, melyet C++11-ben írtunk, osztott memóriás környezetet használva, több külső függvénykönyvtár alkalmazásával, mint például a Boost Graph Library. Az egyik komponens képes a városi forgalmat megjeleníteni, míg egy másik lehetővé teszi, hogy a rendszert programozási versenyeken is alkalmazzuk. Két kiemelendő része a Robocar Emulator, mely forgalmat képes szimulálni, illetve az Automated Sensor Annotations, mellyel méréseket tudunk végezni városokban.

Az Automatic Sensor Annotations komponens adatgyűjtésre alkalmazható, a referencia implementáció neve Real-Time Traffic Analyzer. Hardware és software részekből épül fel. A hardware rész egy beágyazott rendszeren alapuló, egyedileg készített eszköz, mely különböző járművekbe építhető. Alapvető működése, hogy egy útszakaszon megszámolja a szemből érkező járműveket, ezzel egy képet alkotva az adott útszakasz terheltségéről. Ezután az adatokat aggregálja, majd továbbítja egy központi szerveralkalmazás felé. Az eszköz alapja egy Digilent Zybo fejlesztői kártya, perifériái egy kamera modul, egy GPS modul, valamint egy GSM modul. A futó algoritmus egy Haar-cascade alapú objektumfelismerés. A fejlesztést Vivado IDE környezetben végeztük, az eszközön futó software-t C++11-ben írtuk. A gyűjtött adatot MQTT protokollon keresztül küldjük a szerveralkalmazásnak Google Protocol Buffer üzenetként. Az alrendszer software

része vizualizációra és elemzésre szolgál. A Real-Time Traffic Analyzert teszteltük, és alkalmasnak találtuk városi környezetben történő adatgyűjtésre.

A Real-Time Traffic Analyzer kimenete olyan aggregált adathalmaz, mely a Robocar City Emulator bementeként szolgál. Ez a komponens forgalomszimulációra alkalmas. A rendszer eredeti verziójában megvizsgáltuk a szimulációs algoritmust, és hibásnak találtuk. Szükséges volt tehát egy új algoritmus kifejlesztése. Az algoritmus fő erénye, hogy a járművek eloszlását tartja a szimulációs gráfon, tehát a járművek eloszlása stacionárius egy adott időablakban. Az algoritmus gráfelméleten és véges Markov-láncokon alapszik. Térképi adatként az OpenStreetMapet használjuk, forgalmi adatokat a nyílt Taxi Trajectory Prediction adatbázisból szereztünk. Ezen adatok alapján fel tudunk építeni egy gráfot és azon egy átmenetvalószínűségi mátrixot. E mátrix elemei átmenetvalószínűségek lesznek, a gráf minden egyes csomópontjára meghatároznak egy átmenetvalószínűség vektort. Ez a vektor tartalmazza, hogy adott csomópontból milyen valószínűséggel lépünk át a szomszédos csomópontokba. Jelen PhD dolgozatban ennek a mátrixnak a készítését írjuk le. Ezen felül megmutatjuk, hogy a szimulációs algoritmus képes olyan szimulációkra, mely stacionárius eloszlást biztosít, illetve megmutatjuk, hogy adott gráfon, adott átmenetvalószínűségi mátrix mellett ez az eloszlás egyértelmű.

List of Publications

Journal papers

R. Besenczi, N. Bátfai, P. Jeszenszky, R. Major, F. Monori, M. Ispány: **Large-scale simulation of traffic flow using Markov model**. PLoS ONE 16(2), 1-31, 2021. doi: https://doi.org/10.1371/journal.pone.0246062

N. Bátfai, R. Besenczi, P. Jeszenszky, M. Szabó, M. Ispány: **Markov modeling of traffic flow in Smart Cities**. Annales Mathematicae et Informaticae. 53 (2021) pp. 21-44. doi: https://doi.org/10.33039/ami.2021.04.008

N. Bátfai, R. Besenczi, A. Mamenyák, M. Ispány: **Traffic simulation based on the Robocar World Championship initiative**. Infocommunications Journal 7:(3) pp. 50-59. 2015.

Conference proceedings

N. Bátfai, R. Besenczi, P. Jeszenszky, M. Szabó and M. Ispány. **Markov modeling of traffic flow in Smart Cities**. 1st Conference on Information Technology and Data Science, 2020.

Besenczi R. **Markov modell alapú közlekedés-szimulációs algoritmusok**. In: Tokody D., Balla E., Németh K. (eds.) Okos Közlekedés Tudományos Konferencia 2019. Doktoranduszok Országos Szövetsége, Műszaki Tudományok Osztálya. pp. 13. 2019. (ISBN: 978-615-5586-38-5)

N. Bátfai, R. Besenczi, M. Ispány, P. Jeszenszky, R. S. Major, and F. Monori. **Markov modeling and simulation of traffic flow**. In Data Science, Statistics & Visualisation, DSSV 2018, pp. 61, 2018. URL http://cstat.tuwien.ac.at/filz/BoA.pdf.

Monori F., Besenczi R., Bátfai N. **Forgalom szimulációs platform nyílt térképi adatbázisokon**. In: Balázs Boglárka (ed.) Az elmélet és a gyakorlat találkozása a térinformatikában VII. = Theory meets practice in GIS. pp. 319-324. 2016. (ISBN: 978-963-318-570-4)

R. Besenczi, M. Szilágyi, N. Bátfai, A. Mamenyák, I. Oniga, M. Ispány. **Using crowdsensed information for traffic simulation in the Robocar World Championship framework**. In Cognitive Infocommunications (CogInfoCom), 6th IEEE Conference on, pp. 333-337, 2015.

R. Besenczi, T. Katona, M. Szilágyi. **A fork implementation of the Police Edition of the OOCWC system**. In Cognitive Infocommunications (CogInfoCom), 6th IEEE Conference on, pp. 163-164, 2015.

N. Bátfai, R. Besenczi, A. Mamenyák, M. Ispány. **OOCWC: The Robocar World Championship initiative**. In Telecommunications (ConTEL), 13th International Conference on, pp. 1-6, 2015.

R. Szabó, K. Farkas, M. Ispány, A. Benczur, N. Bátfai, P. Jeszenszky, S. Laki, A. Vágner, L. Kollár, C. Sidló, R. Besenczi, M. Smajda, G. Kövér, T. Szincsák, T. Kádek, M. Kósa, A. Adamkó, I. Lendak, B. Wiandt, T. Tomas, A.Z. Nagy, G. Fehér. **Framework for smart city applications based on participatory sensing**. In Cognitive Infocommunications (CogInfoCom), 4th IEEE International Conference on, pp. 295-300, 2013.

Other papers (not part of thesis)

Bátfai, N., Papp, D., Besenczi, R., Bogacsovics, G., Veres, D. **Benchmarking Cognitive Abilities of the Brain with the Event of Losing the Character in Computer Games**. Studia Universitatis Babeș-Bolyai Informatica 64:(1), pp.. 15-25. 2019.

Bátfai N., Besenczi R., Jeszenszky P., Szabella O., Abai A., Kocsis D., Bozsányi A., Szabó Á., Ispány M. **Az oktatás és az esport szinergiája**. In: Kerülő, Judit; Jenei, Teréz (szerk.) Új kutatások a neveléstudományokban 2017 - "Pedagógusképzés és az inklúzió", pp. 136-151. 2018.

Bátfai N., Besenczi R., Szabó J., Jeszenszky P., Buda A., Jármi L., Lovas R. B., Pál M. K., Bogacsovics G., Tóthné Kovács E. **DEAC-Hackers: játszó hackerek, hackelő játékosok**. Információs Társadalom: Társadalomtudományi Folyóirat 18:(1) pp. 132-146. 2018.

Bátfai N., Bogacsovics G., Paszerbovics R., Antal A., Czevár I., Kelemen V., Besenczi R. **E-sportolók mérése**, Információs Társadalom: Társadalomtudományi Folyóirat 18:(1) pp. 147-155. 2018.

Bátfai N., Bersenszki M., Lukács M., Besenczi R., Bogacsovics G., Jeszenszky P. **Az e-sport és a robotpszichológia közös jövője**. Információs Társadalom: Társadalomtudományi Folyóirat 16:(4) pp. 26-39. 2017.

N. Bátfai, R. Besenczi. **Robopsychology manifesto: Samu in his prenatal development.** Carpathian Journal of Electronic and Computer Engineering 10:(1) pp. 3-12. 2017.

N. Bátfai, A. Mamenyák, P. Jeszenszky, G. Kövér, M. Smajda, R. Besenczi, B. Halász, Gy. Terdik, M. Ispány. **Avatar-based sport science soccer simulations**. Annales Mathematicae et Informaticae 46: pp. 13-36. 2016.

R. Besenczi, J. Tóth, A. Hajdu. **A review on automatic analysis techniques for color fundus photographs**. Computational and Structural Biotechnology Journal (14), pp. 371-384, 2016.

N. Bátfai, P. Jeszenszky, A. Mamenyák, B. Halász, R. Besenczi, J. Komzsik, B. Kóti, G. Kövér, M. Smajda, Cs. Székelyhídi, T. Takács, G. Róka, M. Ispány. **Competitive programming: a case study for developing a simulation-based decision support system**. Infocommunications Journal 8:(1) pp. 24-39. 2016.

A. Hajdu, B. Harangi, R. Besenczi, I. Lázár, G. Emri, L. Hajdu, R. Tijdeman. **Measuring regularity of network patterns by grid approximations using the LLL algorithm**. 23rd International Conference on Pattern Recognition (ICPR), pp. 1524-1529, 2016.

R. Besenczi, K. Szitha, B. Harangi, A. Csutak, A. Hajdu. **Automatic optic disc and optic cup detection in retinal images acquired by mobile phone**. In Image and Signal Processing and Analysis (ISPA), 9th International Symposium on, pp. 193-198, 2015.

R. Besenczi, K. Szitha, A. Hajdu. **A framework for distributed processing on an offline cell phone network**. In Cognitive Infocommunications (CogInfoCom),

5th IEEE Conference on, pp. 257-262, 2014.

R. Besenczi, K. Szitha, A. Hajdu. **Distributed eye lesion detection on an offline cell phone network**. In Cognitive Infocommunications (CogInfoCom), 5th IEEE Conference on, pp. 467-467, 2014.