Simulation, optimization and visualization of transportation data

Dancho Panovski

INSTITUT POLYTECHNIQUE DE PARIS

TELECOM SudParis

IP PARIS

Simulation, optimization and visualization of transportation data

Simulation, optimisation et visualisation de données de transport

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom SudParis, Dep.ARTEMIS

École doctorale n°626
Ecole Doctorale de l'Institut Polytechnique de Paris
Spécialité : Signal, Images, Automatique et robotique

Thèse présentée et soutenue à Telecom SudParis - Evry, le 4 Novembre 2020, par

Dancho Panovski

Composition du Jury :

Anissa Mokraoui Professeure, Université Paris 13 Sorbonne Paris Cité	Président du Jury
Ioan Tabus Professeur, Tampere University	Rapporteur
Danco Dvacev Professeur, Ss. Cyril and Methodius University	Rapporteur
Lydie Nouvelière MC HDR, Université d'Évry	Rapporteur
Catherine Achard MC HDR, Université Pierre et Marie Curie - Sorbonne	Examinateur
Veronica Scurtu Maître Assistant, Télécom SudParis	Membre invité
Titus Zaharia Professeur, Institut Polytechnique de Paris, Télécom SudParis	Directeur de thèse

NNT : 2020IPPAS016

Thèse de doctorat

Titre : Simulation, optimisation et visualisation de données de transport

Mots clés: Apprentissage automatique, Apprentissage en profondeur, Algorithmes d'optimisation, Transports publics, Simulation du trafic

Résumé : Aujourd'hui, toutes les grandes métropoles de France, d'Europe et du reste du monde souffrent de graves problèmes de congestion et de saturation des infrastructures routières, qui concernent à la fois les transports individuels et publics. Les systèmes de transport actuels atteignent leurs limites de capacité et semblent incapables d'absorber l'augmentation des flux de passagers à l'avenir. Proposer des solutions adaptées et dont la mise en œuvre peut être réalisée dans engendrer des coûts exorbitants pour les municipalités peut conduire à une amélioration significative de la qualité de vie des habitants.

Dans ce contexte, l'un des principaux défis à relever concerne la création de méthodologies dédiées pour l'analyse des données de transport géo-localisées pour le stockage instantané, l'analyse, la gestion et la diffusion de flux de données massives. Les algorithmes associés doivent être capables de gérer des listes d'événements de plusieurs dizaines de minutes pour calculer des trajectoires réelles, des occupations instantanées, des cycles de changement de feux de circulation ainsi que des prévisions de flux de circulation de véhicules.

Dans cette thèse, nous abordons deux différentes problématiques liées à ce sujet.

Une première contribution concerne l'optimisation des systèmes de feux tricolores. L'objectif est de minimiser le temps de trajet total des véhicules présents dans une certaine partie d'une ville. Dans ce but, nous proposons une technique d'optimization de type PSO (*Particle Swarm Optimization*). Les résultats expérimentaux obtenus montrent qu'une telle approche permet d'obtenir des gains importants (5.37% - 21.53%) en termes de temps de parcours moyen global des véhicules.

La deuxième partie de la thèse est consacrée à la problématique de la prédiction des flux de trafic. En particulier, nous nous concentrons sur la prédiction de l'heure d'arrivée des bus dans les différentes stations présentes sur un itinéraire donné. Ici, nos contributions concernent tout d'abord l'introduction d'un nouveau modèle de données, appelé TDM (*Traffic Density Matrix*), qui capture dynamiquement la situation du trafic tout au long d'un itinéraire de bus donné. Ensuite, nous montrons comment différentes techniques d'apprentissage statistique peuvent exploiter une cette structure de données afin d'effectuer une prédiction efficace. L'analyse des résultats obtenus par les méthodes traditionnelles (régression linéaire, SVR avec différents noyaux…) montre que l'augmentation du niveau de non-linéarité permet d'obtenir des performences supérieures. En partant de ce constat, nous proposons différentes techniques de *deep learning* avec des réseaux conçus sur mesure, que nous avons spécifiquement adaptés à nos objectifs. L'approche proposée inclut des réseaux de neurones récurrents, des approches de type LSTM (*Long Short Time Memory*), des réseaux entièrement connectés et enfin convolutionnels. Deux différents modèles de prédiction sont proposés. Un premier, dit *opérateur*, concerne une prédiction globale, s'appliquant à l'ensemble des stations de bus sur un itinéraire donné. Le deuxième modèle cible plutôt une application *client* et vise à prédire le temps d'attente du prochain bus dans une station et pour un instant temporel donnés. L'analyse des résultats expérimentaux obtenus confirme notre intuition initiale et démontre que ces techniques hautement non-linéaires surpassent les approches traditionnelles et sont capables de prendre en compte les singularités qui

apparaissent dans ce type de données et qui, dans notre cas, correspondent à des embouteillages localisés qui affectent globalement le comportement du système.

En raison du manque de disponibilité de ce type d'informations géo-localisées qui très sensibles et soumises à des réglementations variées, toutes les données prises en compte dans nos expériments ont été générées à l'aide du simulateur microscopique SUMO (*Simulation of Urban Mobility*). Nous montrons notamment comment SUMO peut être exploité pour construire des scénarios réalistes, proches de situations réelles et exploitables à des fins d'analyse.

Enfin, une dernière contribution concerne l'élaboration et la mise en œuvre de deux applications de visualisation différentes, une première dédiée aux opérateurs et la seconde aux clients. Pour assurer le déploiement et la compatibilité de ces applications sur différents terminaux (PC, ordinateurs portables, smartphones, tablettes…), une solution scalable est proposée.

Title: Simulation, optimization and visualization of transportation data

Keywords: Machine Learning, Deep Learning, Optimization algorithms, Public transport, Traffic simulation

Abstract: Today all major metropolises in France, Europe and the rest of the world suffer from severe problems of congestion and saturation of infrastructures, which concern both individual and public transport. Current transportation systems are reaching capacity limits and appear unable to absorb increases in passenger flows in the future. The transport of the future is part of the various so-called Smart City initiatives and should be ”intelligent”, that is to say not only react to the demands but anticipate them, relying on the data exchanged between the end user and the information system of transportation operators.

Within this context, one of the main challenges is the creation of appropriate methodologies for analysis of geo-localized transport data for instantaneous storage, analysis, management and dissemination of massive (typically thousands of instant geo-localizations , with a refresh rate of the order of a few seconds) data flows. The related algorithms must be capable of managing event lists of several tens of minutes to calculate real trajectories, instantaneous occupations, traffic lights changing cycles as well as vehicular traffic flow forecasts.

In this thesis, we address two different issues related to this topic.

A first contribution concerns the optimization of the traffic lights systems. The objective is to minimize the total journey time of the vehicles that are present in a certain part of a city. To this purpose, we propose a PSO (*Particle Swarm Optimization*) technique. The experimental results obtained show that such an approach makes it possible to obtain significant gains (5.37% - 21.53%) in terms of global average journey time.

The second part of the thesis is dedicated to the issue of traffic flow prediction. In particular, we focus on prediction of the bus arrival time in the various bus stations existent over a given itinerary. Here, our contributions first concern the introduction of a novel data model, so-called TDM (*Traffic Density Matrix*), which captures dynamically the situation of the traffic along a given bus itinerary. Then, we show how different machine learning (ML) techniques can exploit such a structure in order to perform efficient prediction. To this purpose, we consider first traditional ML techniques, including linear regression and support vector regression with various kernels. The analysis of the results show that increasing the level of non-linearity can lead to superior results. Based on this observation, we propose various deep learning techniques with hand-crafted networks that we have specifically adapted to our objectives. The proposed approach include recurrent neural networks, LSTM (Long Short Time Memory) approaches, fully connected and convolutional networks. The analysis of the obtained experimental results confirm our intuition and demonstrate that such highly non-linear techniques outperform the traditional approaches and are able to deal with the singularities of the data that in this case correspond to localized traffic jams that globally affect the behavior of the system.

Due to the lack of availability of such highly sensitive type of geo-localized information, all the data considered in our experiments has been produced with the help of the SUMO (*Simulation of Urban Mobility*) microscopic simulator. We notably show how SUMO can be exploited to construct realistic scenarios, close to real-life situations and exploitable for analysis purposes.

Interpretation and understanding the data is of vital importance, nevertheless an adequate visualization platform is needed to present the results in a visually pleasing and understandable manner. To this purpose, we finally propose two different visualization application, a first one dedicated to the operators and the second one to clients. To ensure the deployment and compatibility of such applications on different devices (desktop PCs, Laptops, Smartphones, tablets) a scalable solution is proposed.

Acknowledgment

In the beginning, I would like to express my gratitude to my Ph.D. supervisor Prof. Titus ZAHARIA for the guidance and help during my Ph.D. thesis. Many thanks for all the advice and hints during the thesis and for very useful and elaborated discussions. Sincerely, thank you for helping and supporting me on this journey. I would also like to express my sincere thanks to Veronica Scurtu, assistant Professor at Télécom SudParis, who co-supervised for a period of this research work.

I would like to address special thanks to the members of my Ph.D. defense comity.

First I would like to express all my gratitude to Professor Ioan Tabus from Tampere University, Professor Danco Davcev from Ss. Cyril and Methodius University and Professor Lydie Nouvelière at Université d'Évry. Thank you all for the reviews, comments and suggestions that help me to improve the manuscript.

To Profesor Anissa Mokraoui from Université Paris 13 Sorbonne Paris Cité and Professor Catherine Achard from Université Pierre et Marie Curie, I would like to express my gratitude for their interest in my research work.

This work has carried out in Telecom SudParis, ARTEMIS department.

I would like to thank Professor Catalin Fetita, Nicolas Rougon, Mihai Mitrea, Marius Preda, within the ARTEMIS department at Telecom SudPairs. Thank you for your collaboration and shared expertise on the topics form this thesis. Also, many thanks to all of my colleagues from the ARTEMIS department for their collegial atmosphere and collaboration. Furthermore, thanks to Madame Evelyne Taroni for her help and guidance in solving all administrative problems. And, I would like to thank Madame Veronique Guy from the doctoral school ED IP de Paris.

Finally, I would like to thank all my friends and family for their support and motivation during this period. Specially I want to express my love and gratitude to my wife Simona for all the support, love and understanding during this challenging period.

Table of contents

List of illustrations

List of tables

Chapter 1. Introduction

Dear reader, I would like to invite you to embark on this journey with me. This journey promises to be exciting and exploratory in nature. Like any journey, there will be some ups and downs on the road, some new ideas will emerge and some old solutions will become obsolete, but eventually, we will arrive at the end and conclude our itinerary. In the end, I hope you will learn something new and find the work achieved in this Ph.D. thesis, that I researched and wrote passionately, inspiring and worthy of your time. Let us start with some considerations that motivated this research.

The central topic that will be addressed, discussed, and developed in this Ph.D. thesis relates to a very important subject of today's modern life, which concerns improving public transportation and traffic flow management in urban areas by exploring different cost-effective solutions.

This is an important subject for everyone, it concerns and has a potential implication on the planetary level. At some point in our life, all of us have been in a situation (traffic jams, pollution, parking space, public transport …) where some questions naturally arise. Is there a better way (solution) for the transportation problem? Can we do something more to address this issue? As always, there is no simple answer to such a question.

With all that in mind, our journey starts here…

The introduction part will continue with three sections. The first two-sections are introducing the general context concerning two distinct questions: "What is SmartCity and what it stands for?" and "What can be an Intelligent Transportation System (ITS)?". The answers will allow us to position our research work in these particular domains. The last part concerns the problem statement with all the implications and objectives addressed in this thesis.

1.1. What is SmartCity and what it stands for?

The 21st century is the era of computation and connectivity, so discussion about the modern city cannot pass without first answering the question: what is a SmartCity? How it can influence the daily life of its inhabitants?

Historically, the first SmartCity initiatives can be traced back to the 1960s – 70s. It all started when the communities started to collect data using database and cluster analysis. As a result, they started to issue reports of the findings to government agencies and letters to the general public. Since then, three different generations of SmartCitie's have emerged [1]. SmartCity 1.0 was focused on implementing technologies to the cities without fully understanding the possible implications. This was the very beginning of the SmartCity and, as stated, all the technology was implemented and adopted without even knowing the use cases and potential benefits. SmartCity 2.0 (the initiative

officially started at beginning of the 21st century, more effectively after 2010) was led by the cities themselves. At this stage, the municipalities made several attempts to analyze/understand the data, to determine the future of the city and to establish the way that "smart" technologies can be incorporated. SmartCity 3.0 initiative (which represents the natural evolution of SmartCity 2.0) has emerged recently in very few areas and involves both cities and citizens to co-create models together. It is inspired by issues of equity and a desire to create a smart community with social inclusion. Basically, it explores the SmartCity initiative in a more democratic way. Nowadays it is possible to create and collaborate a SmartCity project without even a government agency. The idea is all about sharing, collaborating and contributing for the benefit of all.

As illustrated in Figure 1-1, the concept of SmartCity involves various aspects of the current life in the urban areas, including urbanism, health, energy, resilience, and of course transportation/mobility.

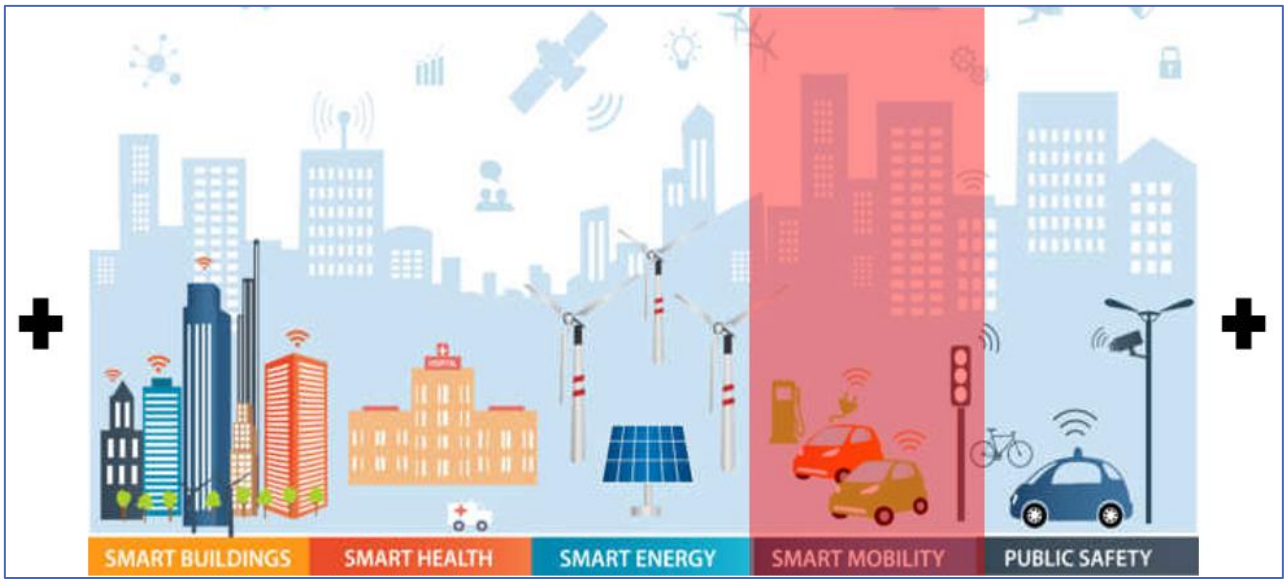


Figure 1-1. SmartCity initiative [2]

Let us note that we lack a generic and unanimously accepted definition of the concept SmartCity. Each new definition that comes is different from the previous one because the SmartCity is an evolving process. The conceptualization of SmartCity, therefore, varies from city to city and country to country, depending on the level of development, willingness to change and reform, resources and aspirations of the city residents. Here-below are some definitions:

[1] TechTarget - "*A smart city is a municipality that uses information and communication technologies (ICT) to increase operational efficiency, share information with the public and improve both the quality of government services and citizen welfare.*"

[3] Techopedia - "*A smart city is a designation given to a city that incorporates information and communication technologies (ICT) to enhance the quality and performance of urban services such as energy, transportation, and utilities in order to reduce resource consumption, wastage and overall costs. The overarching aim of a smart city is to enhance the quality of living for its citizens through smart technology."*

[4] - *"SmartCity should be a city that responds to the peculiar needs of the inhabitants of such a city. This is due to the different social-cultural characteristics of people. Hence a smart city should be the application of Technological solutions in solving city peculiar problems. Whichever solution adopted by any city, it should be sustainable."*

As stated in such definitions, the most important goal of the SmartCity initiative is to enhance the quality of life of the citizens through smartly connected technologies and devices. Let us also underline that information and communication technologies play a central role in the development of smart cities. This correlates with our work, particularly with the so-called *Intelligent Transportation Systems* (ITS) concept, introduced in the following section.

1.2. ITS Intelligent Transportation System

Intelligent Transportation Systems (ITS) represents a sub-part of the SmartCity initiative. This research domain has emerged at the beginning of the 1970s [5], aiming at *dictating the future direction of the modern transportation systems*.

Figure 1-2. Intelligent Transportation Systems [6]

ITS brings a wide area of advanced techniques and technologies into the transportation systems [7] [8], including electronic sensors and technologies, data transmission technologies, intelligent control technologies, public transportation, and more. Data can be obtained from different

and diverse sources such as smart cards, GPS, sensors, video streams, images, social media and so on (Figure 1-2). The main purpose is to provide better services for drivers and riders and to improve the whole transportation system [9].

The importance and impact of this domain grow rapidly, so much so that in 7th July 2010 the European Union create a directive called 2010/40/EU [10], and stated:

"*Intelligent Transport Systems (ITS) are advanced applications which without embodying intelligence as such aim to provide innovative services relating to different modes of transport and traffic management and enable various users to be better informed and make safer, more coordinated and 'smarter' use of transport networks."*

In our research, we are investigating in the area of public transportation systems (particularly buses) and traffic light management as a part of the ITS. Our aim is to develop effective solutions and algorithms that can improve already existing and established systems, without major investment and drawbacks.

1.3. Problem statement

Congestion, traffic flow management, and mobility are the most commonly used words these days describing urban areas (Figure 1-3). From small and not urbanized areas to big and highly dense cities, all are trying to address the same problems: traffic jams, pollution, security, public transportation, parking, and many others. Nowadays, cities invest important resources in solving this issue. Addressing and minimizing this problem will also have some potential benefits for the society in terms of safety for both pedestrians and vehicles, better traffic management, fuel, and energy consumption and environmental benefits like minimizing pollution.

In addition, let us underline that vehicular transport in the urban areas performed by individuals or public transportation vehicles contributes heavily to the carbon footprint, called the greenhouse gas. The data [11] obtained from the French government of the year 2015 provides an extensive overview of the key numbers about the production of greenhouse gas. Transport, in general, is responsible for 27 % of the production of greenhouse gas. Transport operated by road represents 94,8 % of these emissions. From these numbers, it is clear that vehicles operated by roads are the main contributors to this number. The logical step in order to improve this number is to decrease the number of vehicles, specifically the vehicles only with one passenger. In our view, that requires the elaboration of more reliable public transportation systems. The optimization of the traffic lights systems can also help to this purpose.

The question of reliability is highly important since the user acceptance of using public transportation strongly depends on it. Within this framework, disposing of efficient traffic prediction tools and

traffic lights optimization, dedicated to both personal transportation vehicles and public transportation, is a challenging issue.

With the rapid growth of vehicle numbers in modern cities, the problem becomes more and more complex and difficult to solve. Different approaches attempt to address this issue.

One traditional solution relates to the infrastructure and proposes building bigger lanes, bridges, roundabouts, or underground passages. Such an approach is applicable only in areas where the extension of the infrastructure is possible, excluding hence most of the densely populated cities as well as places with cultural centers and historical sites. In addition, it is also the most expensive solution.

Another approach concerns the creation of restricted areas for specific types of vehicles like public transport or electric vehicles and creating areas with low-density traffic flow. As the most recent example, we can point out the center of London, where vehicles with a high carbon footprint are charged an extra fee for driving in this area. This is all implemented in order to discourage the vehicles to pass in this area, explicitly so as to create low-density traffic. Even though the proposed solutions are achievable, they are not suitable for many areas and notably in the case of densely populated cities or areas where such a cost is not justified, or the implementation is just not possible.

Figure 1-3. Modern cities [12]: road infrastructures and traffic congestion [13]

This Ph.D. thesis investigates a different approach, based on computer simulation, optimization techniques and machine learning algorithms. Such methods are applied over existing systems in an artificial environment (traffic simulator software) that can simulate real-life scenarios. Our

contributions are two-fold. First, we propose a technique that can optimize efficiently the traffic lights cycles in city intersections and improve the overall traffic flow. Our second contribution concerns the prediction algorithms that can improve the reliability of public transportation over existing systems/infrastructures.

Let us underline that such methods are more cost-effective with respect to the traditional ones since they do not require any new infrastructural investment.

Within this context, our objectives are defined as follows:

1. Improve traffic flow management in urban areas: optimization of the vehicular traffic flow by optimizing the traffic lights configuration in order to maximize the number of passing vehicles in a minimum amount of time.

By optimizing the traffic lights system, it becomes possible to decrease the traffic jams, while increasing the overall efficiency of the system. Such an approach is possible and can be effective because it is cheap to integrate and can upgrade easily the existing systems. It concerns only the traffic lights cycle program which is regulated by software and algorithms.

2. Improve the public transportation system: Short and long-term prediction of public transportation (buses in particular) using machine learning algorithms.

Here, the objective is to predict, with the help of machine learning (ML) algorithms, the bus location in a given area for a certain period of time, while taking into account the global traffic flow and local traffic jams. The prediction can also take into account different elements, such as the number of people per bus station, velocity, distance... In order to perform good and accurate predictions, the algorithm must learn the past state (historical data) of the observed system. There are two families of predictions that we have considered, including *short-term* and *long-term* predictions. A related web interface is also proposed that can serve as a hub and visual center (representation) for all the obtained results. Client-operator web services and applications are also developed to present the results in a more visually pleasing manner. The client application represents the short-term prediction results, which is a prediction up to one or two hours from the moment of the observation. This application provides the real-time representation and prediction of the bus arrival time at the considered station. On the other hand, the operator application displays long-term prediction results, which concerns and predictions of at least one or more days. The operator application can be natively used by the operator, in order to perform the analysis of the current state, identify areas that raise problems and propose improvements of the overall system.

The rest of the manuscript is organized as follows. Chapter II presents a state of the art of the traffic simulation platforms, both open-source and commercial, as well as a detailed description of the SUMO framework further adopted in our work. Chapter III details the optimization techniques proposed for the traffic lights cycling program. Chapter IV gives an overview of the public

transportation prediction with machine learning algorithms. In chapter V the proposed TDM (*Traffic Density Matrix*) model is presented in detail, together with the various prediction approaches proposed. The experimental results obtained are also presented and discussed here. Finally, Chapter VI describes the visualization platform developed specifically for this work. In the final part (Chapter VII), we conclude the manuscript and open some perspectives of future work.

Chapter 2. Traffic simulation platforms

Abstract

This section presents a state of the art of existing traffic simulation platforms. Both open-source and commercial systems are here discussed and compared, with related advantages and limitations. The analysis of the state of the art shown that the most complete set of functionalities, adapted to the objecties addressed in our work is the open-source microscopic simulator called SUMO (*Simulation of Urban MObility*).

Hence, in the following we detail the SUMO platform, focusing on the main functionalities exploited in our work, which include the simulation of real-life scenarios, the traffic lights system and the SUMO public transportation modeling. The chapter concludes with a short synthesis of the simulation platform and scenarios.

2.1. Introduction

Traffic simulation can be defined as the mathematical model of transportation systems, implemented through the application of dedicated computer software. The first computer simulation started in 1955 in Los Angeles, California, when D.L Gerlough published his dissertation "Simulation of freeway traffic on a general-purpose discrete variable computer" [14]. Since then, numerous traffic simulations researches have been conducted and many software packages proposed.

There are two main categories of traffic simulation solutions, which include both macroscopic [15] and microscopic [16] approaches. While a microscopic traffic simulator focuses on the mobility of each individual entity in the system, a macroscopic traffic simulator provides a complete traffic flow of the system taking into account more global constraints, such as general traffic density and vehicle distribution. We are going to focus on microscopic traffic simulation programs because they are able to provide a detailed image (with location, time, speed...) of each vehicle in the system that can be later exploited for both optimization and prediction. Table 2-1 summarizes the most representative microscopic platforms available today.

TABLE 2-1. TRAFFIC SIMULATION FRAMEWORKS

Traffic Simulators	
Open-source software	[17] SUMO
	[18] MATSIM
	[19] REPAST SIMPHONY
	[20] MAINSIM
Commercial software	[21] QUADSTONE PARAMICS
	[22] VISSUM
	[23] CUBE
	[24] VISSIM
	[25] AIMSUN
	[26] SATURN

In the lower part of Table 2-1, for information purposes only, some commercial solutions are cited. We will not consider them further in our work, despite their certain advantages (real-time support, enhanced graphical user interfaces) since the idea is to set up an open framework that can be exploited in the future by the research community.

The upper part of Table 2-1 presents open-source, free software packages that are usually adopted by the research community and individual developers. In order to further investigate each software platform, let us present a short description of each solution and discuss its pros and cons.

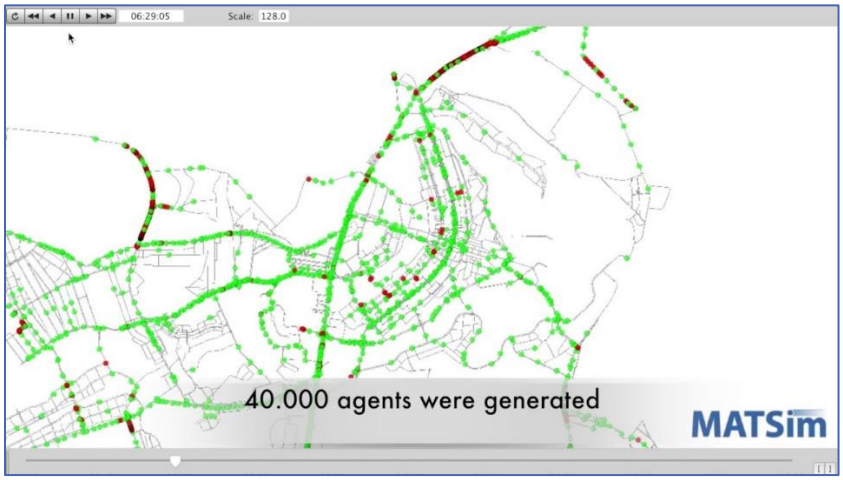


Figure 2-1. MATSim interface

MATSim [18] is an open-source framework, used to implement large-scale agent-based transport simulations, where a large number of individual, synthetic persons (so-called "agents") are simulated (Figure 2-1). The framework is designed for large-scale scenarios, meaning that all features of the model are stripped down to efficiently handle the targeted functionality. Multi-core processors are utilized here for parallel computing and for minimizing the required memory resources. Even though MATSim offers a powerful traffic simulation platform, the main drawback (for our use-case) comes from the global approach adopted, since it is constructed to perform simulations in large-scale scenarios like big cities or even countries.

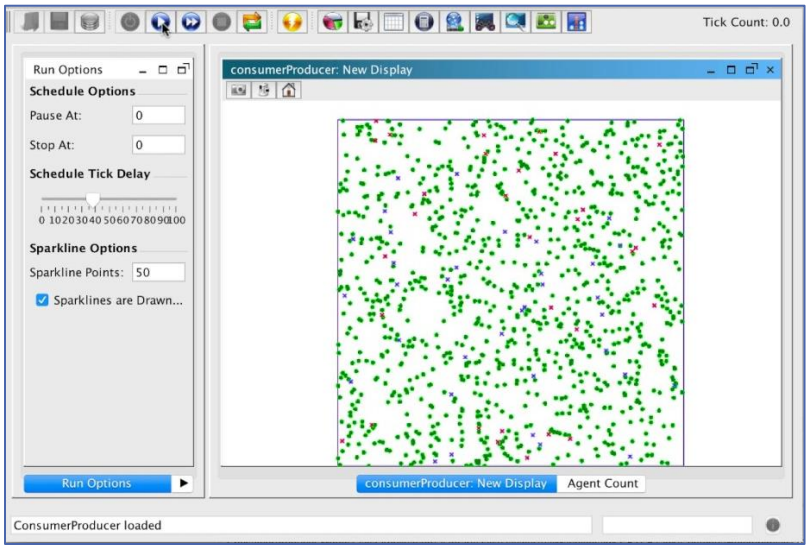


Figure 2-2. RePast Simphony interface

The RePast Simphony [19] platform (Figure 2-2) is a free and open-source agent-based modeling toolkit, which provides a library of classes for creating, running, displaying, and collecting data from an agent-based simulation. Its core and programming interface is based on Java programing language, which makes it less intuitive and syntactically heavy. In addition, the proposed graphical user interface is outdated and visually poor.

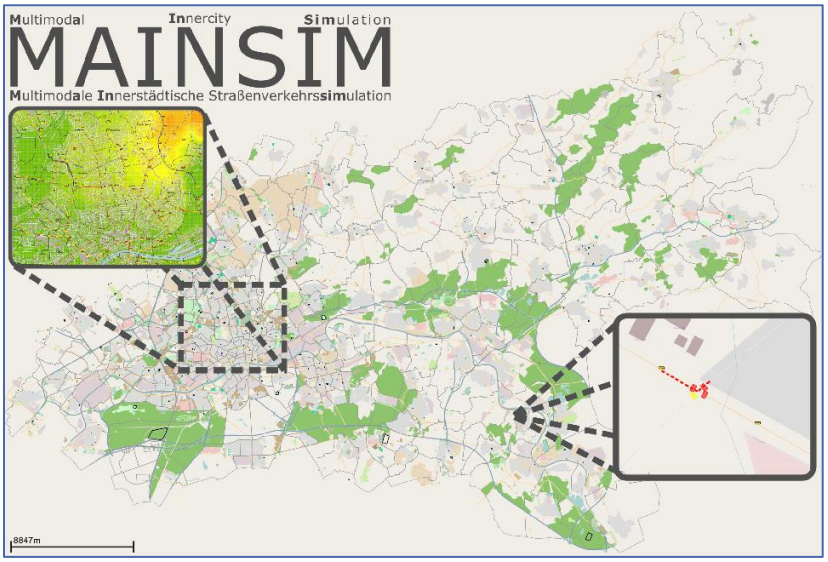


Figure 2-3. MAINSIM interface

Another microscopic simulator is MAINSIM (Figure 2-3), introduced in [20]. Here, the traffic simulator generates simulation graphs from cartographical material in an entirely automatic way. MAINSIM is capable of simulating entire cities with vehicles, bicycles, and pedestrians. The exclusion of public transportation facilities and the poor availability of online documentation makes it unsuitable for our work.

Such traffic simulation platforms are capable of simulating a large variety of scenarios. Our particular requirements need a simulation platform that can successfully simulate both small and large-scale scenarios and provide the possibility of simulating public transportation while having good and understandable documentation and technical support. For this reason, in our work, we have adopted the SUMO [27] open-source platform, described in the following section.

2.2. SUMO traffic simulator framework

Let us start by introducing the SUMO platform and some basic related definitions. SUMO [17] (*Simulation of Urban MObility)* is a microscopic multi-modal traffic simulator that is able to simulate different types of traffic data and can provide nice visual and understandable output for future tests and analysis.

SUMO allows simulating a given traffic on demand, which consists of a set of individual vehicles that move through a given road network. The simulation permits addressing a large set of traffic management topics and is purely microscopic: each vehicle is modeled explicitly, has an own route, and moves individually through the network.

In SUMO, simulations are deterministic by default but there are various options for introducing randomness [28]. The platform was first introduced in 2001 by the German Aerospace Center (DLR), [29]. The last release is "version 1.6.0" (at the moment of writing the thesis) and all releases are publicly available for downloading as install or source package with test data included [30]. All the improvements and releases of SUMO have been made with the purpose of enhancing the user experience and introducing additional functionalities to the environment [31]. Numerous functionalities are available today, including the use of external maps, shortest route algorithms, and different types of vehicles.

Today, SUMO is developed under the form of a suite of applications [31] which makes it possible to prepare and perform traffic simulation in a fully personalized manner.

SUMO is developed as an open-source package since 2001 and remains open until today. There are two main reasons why it is and will remain open-source [27]. The first reason is the lack of stable open-source simulators with good support, documentation, and community, and also to give support to the research community with a free tool that can incorporate and test different algorithms. The second reason is to gain support from other institutions.

Since the beginning of our thesis, we know that acquiring data that includes real traffic information (vehicles, public transportation, pedestrians) is very difficult, due to the sensitivity and the privacy of such data, which may be submitted to various reglementations, more or less strict, from one country to another. For this reason, we have adopted and used the SUMO microscopic traffic simulator in order to create the desired scenarios, perform the simulations, and acquire synthetic data. More precisely, two different scenarios have been created: an *optimization scenario* and a *predictive scenario*. They will be described in detail in the concerned chapters.

In the following section, let us describe and analyze the main features of the SUMO suite that we have exploited and adapted to the construction of our scenarios.

2.2.1. The SUMO suite

SUMO today is developed as a suite of applications [31] which help to prepare and perform traffic simulation. As summarized in Table 2-2, the SUMO package offers a wide range of different components and features.

TABLE 2-2. SUMO SUITE: MAIN FEATURES

SUMO Features	
Main Tasks (SUMO's main features)	Microscopic simulation - vehicles, pedestrians and public transport are modeled explicitly
	Online interaction – control the simulation with TraCI (Traffic Control Interface)
	Simulation of multimodal traffic, *e.g.*, vehicles, public transport and pedestrians
	Time schedules of traffic lights can be imported or generated automatically by SUMO
	No artificial limitations in network size and the number of simulated vehicles
	Supported import formats: OpenStreetMap, VISUM, VISSIM, NavTeq
	SUMO is implemented in C++ and uses portable libraries
Included applications (SUMO's apps)	SUMO: command line simulation
	GUISIM: simulation with a graphical user interface
	NETCONVERT: road network importer
	NETGEN: abstract road networks generator
	OD2TRIPS: converter from O/D matrices to trips
	JTRROUTER: routes generator based on turning ratios at intersections
	DUAROUTER: routes generator based on a dynamic user assignment
	DFROUTER: route generator with use of detector data
	MAROUTER: macroscopic user assignment based on capacity functions

Since the beginning, the SUMO development team has open the code to the public community in reach of improving traffic simulations experience by introducing many new features and components. A lot of different applications and diverse use-cases are presented where SUMO was used as a simulation tool.

In Table 2-3, the SUMO features that are included (out of the box) are presented, as well as some features that are still missing from the SUMO suite.

TABLE 2-3. SUMO FEATURES (INCLUDED AND NON-INCLUDED)

SUMO suite		
Simulation	Included	**Space-continuous and time-discrete vehicle movement**
		Different vehicle types
		Multi-lane streets with lane changing
		Different right-of-way rules, traffic lights
		A fast openGL graphical user interface
		Manages networks with several 10.000 edges (streets)
		Fast execution speed (up to 100.000 vehicle updates/s on a 1GHz machine)
		Interoperability with other application at run-time
		Network-wide, edge-based, vehicle-based, and detector-based outputs
		Supports person-based inter-modal trips
	Non-included	**Accidents**
		Road works
		Parking
		Bus stop schedule
		Traffic jam detection
Network import and routing	Included	**Data transver from Visum simulator**
		Data transfer from MATsim simulator
		XML-descriptions
		Missing values are determined via heuristics
		Microscopic routes - each vehicle has its own one
		Different Dynamic User Assignment algorithms
	Non-included	**Network import does not import properly the map all the time, some modification and adjustment has to be made**
		Importing public transport schedule is not supported
		Some vehicle disappear from the system when they routed
OS	Included	**Windows**
		Main distribution on Linux are supported (Ubuntu, Fedora, openSUSE)
	Non-included	**Some Linux distributions**
		MacOS binaries must be build separately

Table 2-3 confirms the notion that SUMO is presented not only as a simulator but rather as a suite of applications that prepare and perform traffic scenario simulation. Creating successful simulation in SUMO can be achieved by different development stages, divided by their specific purpose. In the

following sections, let us detail the main steps that are necessary for creating useful simulation scenarios.

2.2.2. Road network generation

The first stage of the simulation process consists of creating a road network. The roads presented in SUMO represent real-world networks under the form of graphs, where nodes are intersections and roads are edges [18]. This representation technique is well-known and widely adopted by other network formats, such as Vissim [32] and OpenDrive [33].

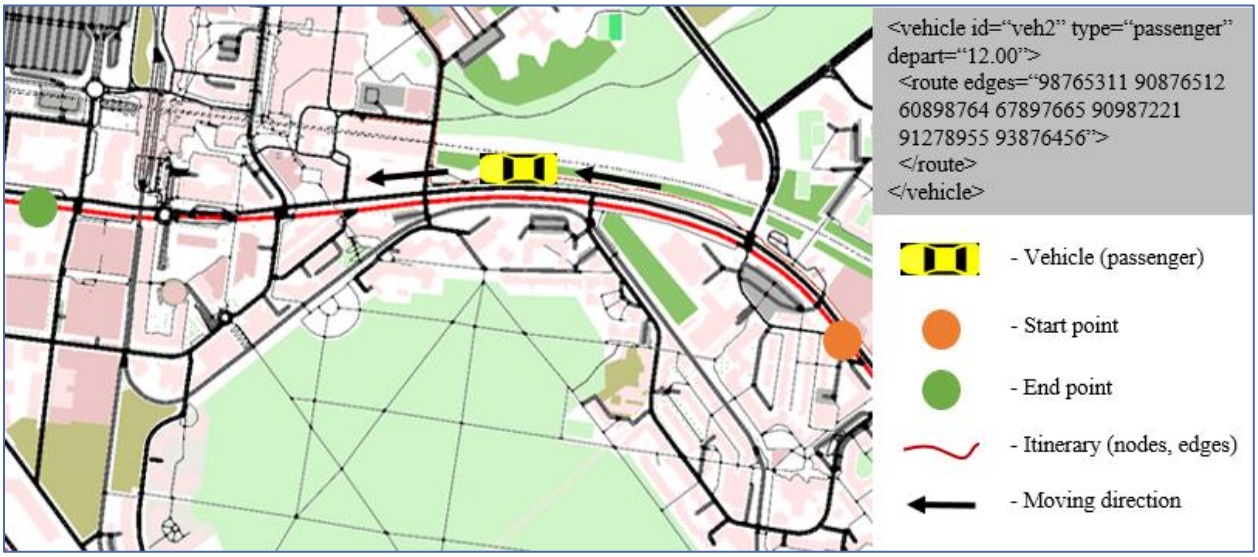


Figure 2-4.Vehicle itinerary presented in SUMO GUI interface

Figure 2-4 shows a sub-part of the digital road network of the city of Evry, France. In a more general manner, in our work, we have considered arbitrary road networks that are freely available in open-source platforms such as Open Street Map (OSM) [34]. The OSM road structure is represented in XML format. SUMO makes it possible to import such OSM maps and convert them into the native graph structure[1].

However, most of the available digital maps are usually used for navigation purposes. Thus, they often lack the detail needed by microscopic road simulation, more specifically in what concerns the

[1] From the implementation point of view, this is done with the help of the *netgenerate* and *netconvert* procedures included in the SUMO suite.

representation of the intersections, with the corresponding traffic lights. When importing a digital map, SUMO tries to determine the missing values using various heuristics. Even with the given functionality, preparing the real-world network for microscopic simulation is a time-consuming task, that often needs to be corrected manually.

2.2.3. Traffic flow generation: vehicles and itineraries

The second stage to be considered in creating a simulation scenario concerns the traffic flaw generation. To do so, a set of different rules needs to be considered. First, we need to define the various types of vehicles that will be inserted into the system.

All vehicles presented within the system are explicitly defined by a unique identifier, a departure time, and a route through the network. One vehicle route in the network is a complete list of connected edges between the initial and final destination. In SUMO, edges are represented by *lanes* and *junctions*. The lanes represent the actual roads, while the junctions define the connectors of the roads and model the crossroads. There are several different attributes that can be associated with each vehicle, including velocity, vehicle type, pollution and noise emission level. This allows the specification of a detailed vehicle representation, illustrated in Figure 2-5. This representation includes the vehicle types (*e.g.*, cars, trucks, bikes, motorcycles…) and defines the associated parameters with velocity ranges, CO2 emission and level of noise.

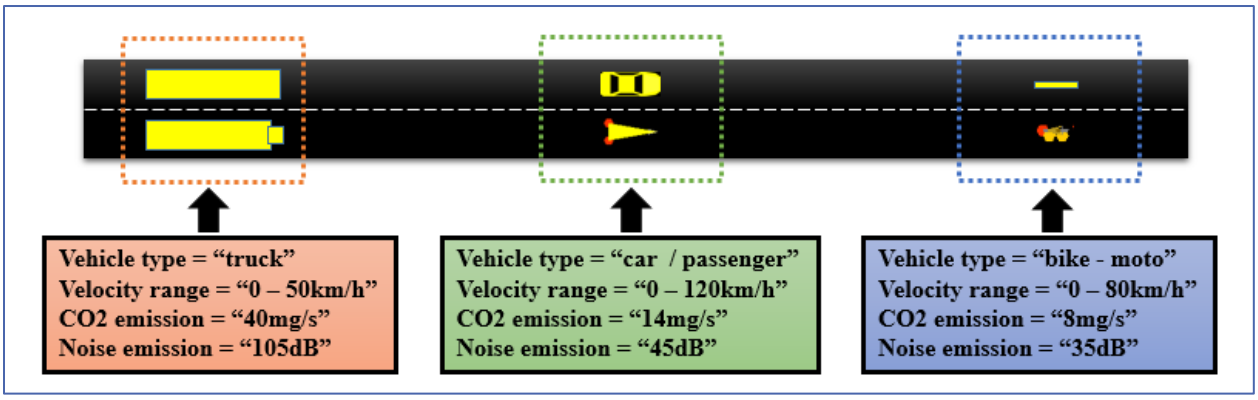


Figure 2-5. Road structure in SUMO with different vehicles types

Concerning the vehicles road map itineraries, they are presented with the help of the so-called O/D (origin/destination matrices. Thus, each randomly generated vehicle has a starting and ending point on the map, within the O/D matrix. In this way, it becomes possible to simulate large-scale scenarios that can easily scale up.

Starting from a given O/D entry and a corresponding departure time, a trip is automatically generated by the construction of an itinerary, represented as a succession of edges on the road structure that relies on origin to destination. The routes are computed with the help of the so-called DUA (*Dynamic User Assignment*) procedure, which is an iterative process employing the shortest path routing procedure calculated under different cost functions.

Table **2-4** summarizes the various routing algorithms supported in SUMO.

TABLE 2-4. SUMO ROUTING ALGORITHMS

SUMO Routing algorithms	
Algorithms	Dijkstra (time dependent)
	A star (A*) (time dependent)
	CH - Contraction hierarchies (Not strictly time depended, new hierarchy is built for each time interval according to the conditions at the departure)
	CHWrapper (Similar like "CH", for each vClass – vehicle class)

Four different routing algorithms are supported, including Dijkstra [35], A* [36], CH (*Contraction hierarchies*) [37] and CHWrapper [37]. The Dijkstra algorithm is the default and most commonly used algorithm that we have also adopted in our work.

2.2.4. SUMO output

The outputs of the simulation generated by SUMO are provided in different formats containing information related to each entity in the system. The most commonly used outputs are the following: trip-info, raw-output, summary, co2, and fcd-output. The power of SUMO comes from its ability to provide detailed and rich output representations. It can present each vehicle in the simulation in a detailed form, including the starting and ending positions of vehicle, longitude, latitude, the velocity of the vehicle, the fuel consumption, the generated noise and pollution, as well as the journey timestamps. All the outputs generated by SUMO are represented in XML format, which makes them highly convenient to read and use.

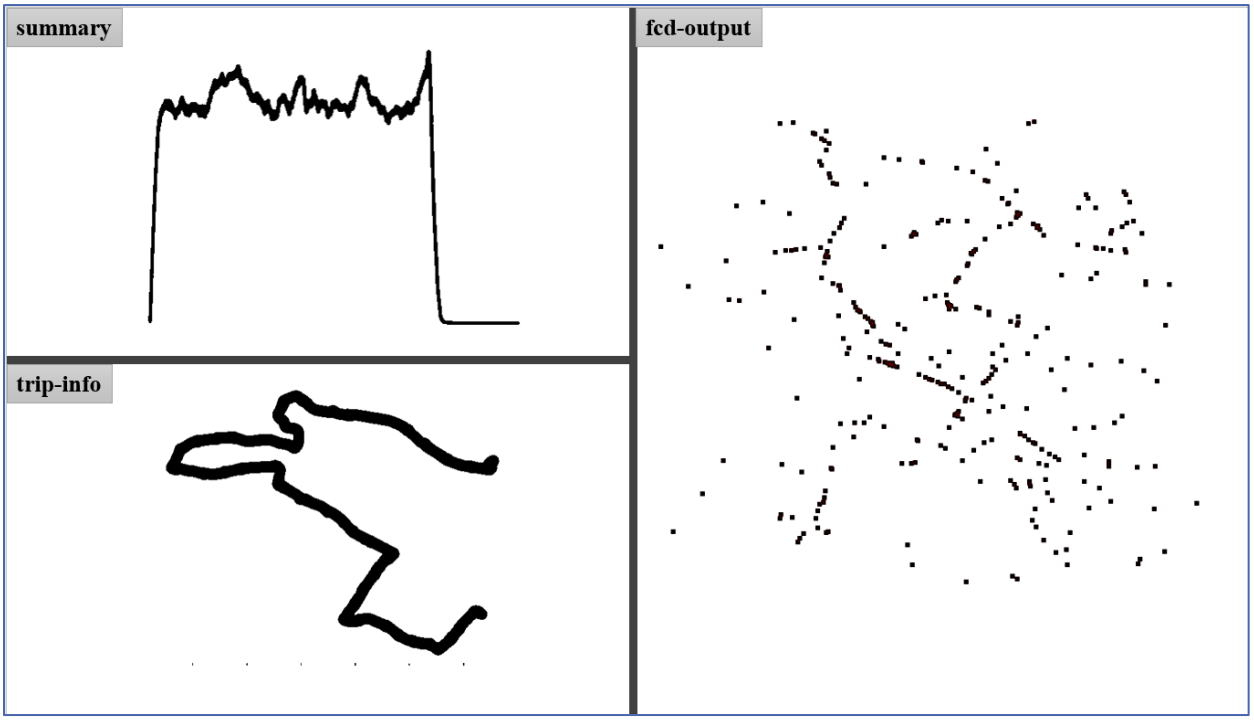


Figure 2-6. Different SUMO outputs: trip-info, summary, and fcd-output

The three different outputs retained in our work, illustrated in Figure 2-6, are the following:

- *trip-info*: for each vehicle in the system, it provides the sequence (latitude, longitude, v_t) $_t$, where lat_t and lon_t are respectively the latitude and longitude of the vehicle at time t, while v_t denotes its speed (velocity). The *trip-info* structure allows us to output detailed data relative to a set of vehicles used for evaluation, called probes. A *probe* is by definition a vehicle with a pre-defined itinerary that is used to test and measure the performance of the simulation.
- *summary*: it provides the global number of vehicles in the system at each second in the simulation interval (or with respect to a given, pred-defined temporal sampling rate). It shows the distribution of the vehicles globally through the whole simulation and can help us to determine the region of measurement or region of interest.
- *fcd-output(Floating Car Data)*: this output globalizes the information from *trip-info* over the whole systems and provides, at each second, the identifier and GPS position of all the vehicles that are present in the system. Such data is particularly useful for prediction purposes, as will be shown in Chapter 5.

An important aspect of the simulation process concerns the tuning of the traffic lights cycles. This issue is presented in the following section.

2.2.5. Traffic lights scheduling in SUMO

Traffic lights, as part of SUMO, are located in the intersections (so-called junctions in SUMO) and are used to control the flow of vehicles in the system. The TLS (*Traffic Lights System*) is composed of two components: a program of their color state (r-red, g-green, and y-yellow) and the cycle duration expressed in seconds. It is important to mention that traffic lights located at the same intersection are governed by a common, synchronized program, for obvious safety reasons.

A traffic light cycle is defined by a succession of phases that are repeated in a loop during the simulation. Each phase is defined by an assignment of color states and corresponding durations, associated with all the traffic lights that are present in the considered intersection.

In SUMO, the traffic lights cycles are simply described in a highly convenient XML format. A phase is encoded as a suite of characters of the form (c_0, c_1,…, c_{n-1}), where n is the number of traffic lights in the intersection and $\boldsymbol{c_i} \in \{\boldsymbol{r-red, g-green, y-yellow}\}$ is the color of the i^{th} traffic light in the considered intersection. The execution of the traffic lights cycle is performed successively, line by line. A new phase will start only when the current one is completed, *i.e.*, when the corresponding phase duration is achieved.

Figure 2-7 illustrates an intersection with four traffic lights. The traffic lights are executed in SUMO with the traffic lights logic program. This program consists of a set of commands that are executed in a timely manner, usually with a refreshment rate of 1 second.

The XML document describing the traffic light cycle starts with the intersection ID, followed by the phase duration (expressed in seconds) and a detailed state of each light. For this intersection, there are 4 different traffic lights in total. Their color states are specified for all the phases that are necessary to obtain a complete traffic light cycle.

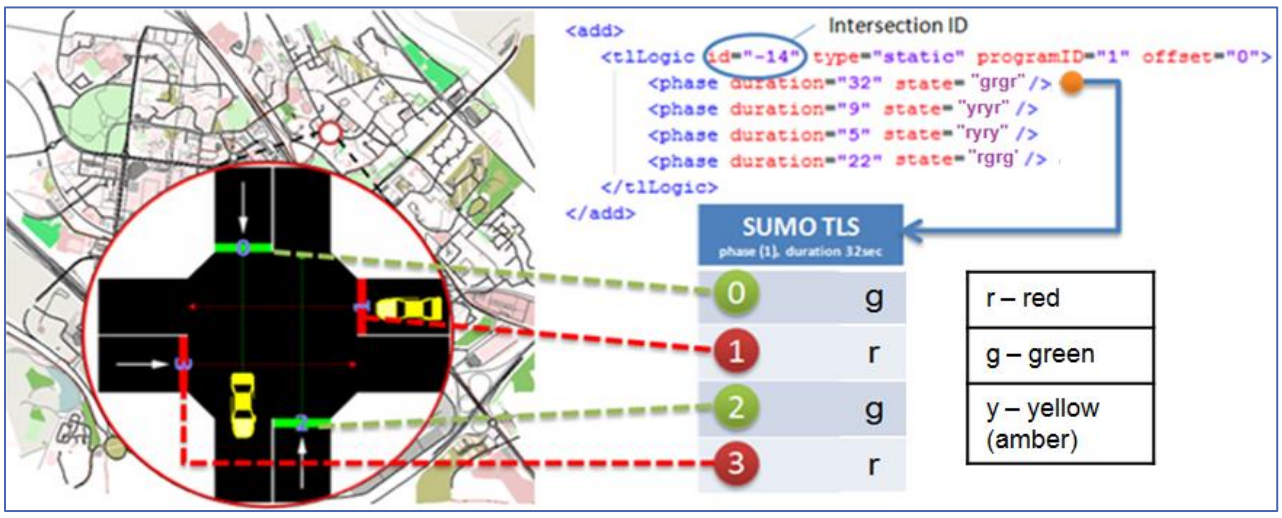


Figure 2-7. Intersection with traffic lights in SUMO

Each phase is encoded by a string of four characters (since we have here four traffic lights). In these examples, the first phase has a duration of 32 seconds and corresponds to the following state:

- traffic light 0: green
- traffic light 1: red
- traffic light 2: green
- traffic light 3: red

This information is encoded by the following string representation: 'grgr'. The second phase starts when the first is completed and is encoded by the sequence "yryr", with a duration of 9 seconds. The third phase has a duration of 5 seconds and a color state "ryry". The second and third phases correspond actually to transitions. Finally, the fourth phase (with a duration of 22 seconds) is the opposite of the first one and is encoded by the sequence "rgrg".

SUMO provides by default a valid combination of traffic lights cycles generated by the so-called *netconvert* or *netgenerate* algorithms [38]. These deterministic algorithms, based on prior (heuristic) tests, generate traffic lights cycle programs with phase durations within the range of [0, 60] seconds. For more information about the traffic lights cycle program, please refer to [39]. From here onwards, the native SUMO algorithm for generating cycle programs of the traffic lights system will be referred to as the *SUMO traffic lights system* (STLS).

2.2.6. SUMO for public transportation

In SUMO, public transportation (buses in our case) can be created only manually, since there is no automatic way to import such information.

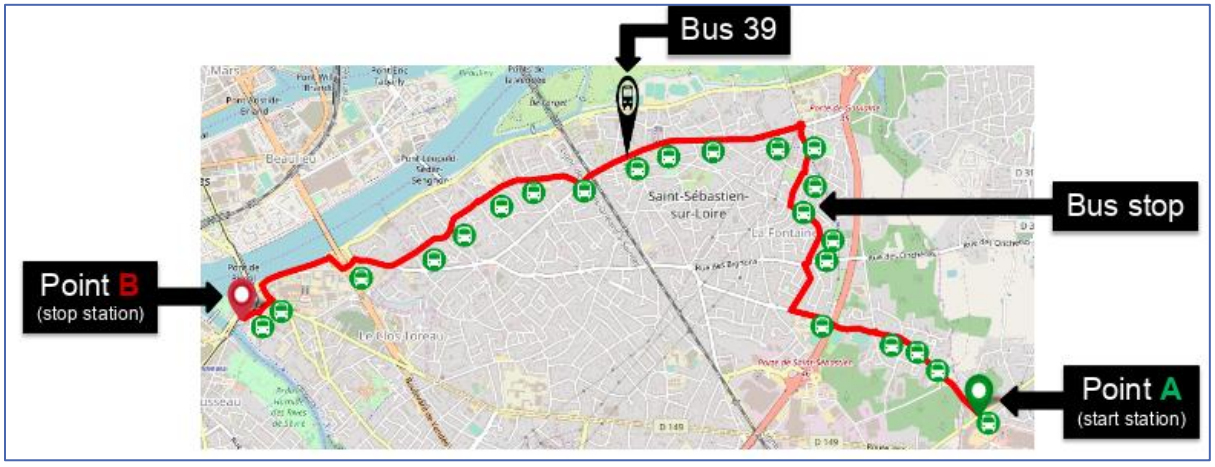


Figure 2-8. Bus itinerary with bus stops presented in SUMO

The process of creating a bus line is illustrated in Figure 2-8, for a real-life example which concerns the bus line 39 of the city of Nantes (France).

The first step is to create each bus stop as a 2D point on the map with corresponding longitude/latitude (Figure 2-8). Then the bus itinerary needs to be specified, *i.e.*, the exact road where the bus operates, from starting point A to terminus point B, via all the bus stops. The main difficulty to overcome here concerns the necessity to ensure that the bus will always operate on the same pre-defined road. Actually, the SUMO simulator uses the Dijkstra algorithm to recalculate for each vehicle the shortest path needed to reach the given destination. Consequently, in certain situations, depending on the traffic conditions, the bus may be re-roaded between two consecutive bus stops in order to link them in the fastest manner. In order to solve this issue, we have manually specified each lane of the bus itinerary.

A lane is defined as a piece of road between a pair of successive intersections. By specifying all the lanes over the itinerary, the bus will be forced to strictly follow the specified itinerary, whatever the traffic conditions, since no deviation is possible on a given lane.

A deeper explanation, including with real examples, will be further provided in Section 5.2, with more details on the bus itinerary creation, the measurement of the real bus location, the number of occupied lanes, the number of bus stops, the bus itinerary length/direction and so on.

2.3. Conclusion

SUMO is an open-source simulator and brings a lot of “on the shelf” functionalities. It is described as a suite of applications that can simulate traffic scenarios of various sizes, from two streets with one intersection to a city with millions of inhabitants. This makes SUMO a powerful tool for observing and testing traffic simulation. It is not “The perfect simulator”, there is still a lot of space for improvement but SUMO is well built, with a large community that contributes and is suited for experimental and research purposes. That is why is adopted in many projects worldwide, including ours.

Chapter 3. Traffic Lights Optimization

Abstract

In this chapter, we tackle the issue of traffic flow management in urban areas by proposing as a solution a traffic light optimization method. The analysis of the state of the art, which is first presented, shows that biologically-inspired optimization tecchniques, such as neural networks, genetic algorithms or particle swarm optimization, can bring useful solutions for the considered traffic lights optimization objectives.

The proposed approach uses the SUMO simulator as a cost-effective solution and a PSO (Particle Sworm Optimization) technique for traffic lights cycle program. The experimental setup concerns a real-life scenario, which includes a sub-part of the city of Evry in France with two intersections and six different itineraries.

A series of experimental simulations has been performed with different number of vehicles in different time tables. The results obtained show significant improvements in terms of increasing the number of vehicles that complete the simulation (4,65% to 10,45% gain) and average journey time necessary for the vehicles to reach their destination (5,37% to 21,53% less time loss).

The chapter concludes with an analysis highlighting the advantages and limitations of the proposed optimization techniques and also discusses the shift in our research orientations, from optimization to prediction.

3.1. Introduction

The issue of traffic flow management is of crucial importance in the case of modern cities, which are facing significant problems related to traffic jams, pollution, security, mobility, parking, and so on. Solving such problems requires today the investment of significant resources, most of them aiming at enhancing the existing infrastructures.

Ongoing research programs within the field of traffic control systems are mainly divided into two great families. The first one concerns the so-called *Adaptive Traffic Control Systems* (ATCSs) [40], which changes the cycle program duration depending on vehicles in queues in real-time. The operations involved are highly dependent on the associated sensor systems that are directly connected to the existing traffic lights. Many cities around the world successfully implement and use such systems that are integrated into the infrastructure. However, the daily management of a traffic network in real-time requires a high operational cost. Moreover, in case of malfunction, appropriate and qualified human intervention is needed to maintain the system and solve the problems.

A second family of approaches concerns the usage of modern simulators [41], [42] to simulate real-life scenarios in microscopic or macroscopic [43] traffic views. Such simulators allow researchers to recreate real scenarios in a cost-effective manner, and test them under various parameter settings. Within this framework, numerous research works are using simulators in order to create test environments adapted for determining optimal traffic light cycles.

Creating traffic scenarios in a microscopic traffic simulation environment is not an easy task. There are a lot of elements to be considered, and all the components need to be perfectly implemented in order to have a successful simulation. Simulations are computed in previously prepared scenarios with additional preparation of each component, like network map, types of vehicle, traffic lights systems, lane width, junctions, crossroads, and others, as discussed in Chapter 2. They all influence heavily the final outcome.

Our main objective is to maximize the number of vehicles that are passing in a certain urban area in a minimum amount of time. To achieve this goal, we are going to address the problem of traffic light system optimization with the PSO algorithm. The approach proposed in this chapter concerns the setup of a finely optimized traffic lights system that can adapt to the conditions of traffic. Such an approach is the most cost-effective, since it solely requires the integration/update of already existing systems, without imposing restrictions concerning the vehicle movement.

3.2. Traffic Lights Optimization algorithms: Related work

Traffic control is mainly divided into two sections. The first one concerns the so-called *Adaptive Traffic Control Systems* (ATCSs) [40], also known as real-time traffic control systems. A second

family of approaches (what we have adopted) exploits computer simulators [17] to optimize and improve traffic control.

Adaptive traffic signal systems have been operating successfully in many countries since the early 1970s. The two most widely deployed systems in the world are SCATS (*Sydney Coordinated Adaptive Traffic System*) [44] and SCOOT (*Split Cycle and Offset Optimization Technique*) [45]. The *Intelligent Transportation Systems* (ITS) provide users with information required to make a better, safer, and smarter use of the transport network. In [46], authors introduce SCOOT as an intelligent traffic control system, explore the system's functionalities, present some applications and provide various comparisons with other states of the art techniques. An adaptive traffic control system has been introduced in the city of Dublin [47], specifically getting empirical insights from the SCATS approach.

Such models provide good ideas and directions on how to deal with the problem, but they are mostly expensive as they most often require human intervention. For the scope of this research, we are going to adopt the second family of approaches, which is exclusively based on computer simulators.

In our work, we have retained the SUMO simulator [16], described in detail in the previous chapter, which offers a large set of available functionalities that can be directly exploited for our simulation purposes. Let us now consider the issue of simulation-based optimization of traffic light cycles [48] and analyze the related work.

In [49], authors introduce a traffic simulator based on fuzzy logic for managing the traffic lights at isolated junctions. The experimental results reported show that the fuzzy logic traffic light controller is able to significantly reduce the waiting time at red lights since the controller adapts itself to the traffic density. The main drawback here is the experimental environment since this system is solely tested on one isolated junction and the question of scalability is not addressed. Nevertheless, the results obtained are highly promising and deserve further investigation.

In [50], an adaptive reinforcement learning (RL) algorithm [51], [52] is proposed for learning to control the traffic lights system. The algorithm learns the expected waiting times of cars for red and green lights at each intersection and sets the traffic lights to green for the configuration maximizing individual car gains. The decision of the algorithms is based on the amount of traffic measured in the vicinity of the considered intersection. Simulation results show that RL algorithms reduces the average waiting time for more than 25% when compared to the standard, non-adaptive controllers.

In [53], biologically-inspired techniques are introduced, such as CA (*cellular automata*) [54] and NN (*neural networks*) [55]. Such techniques perform a combinatorial optimization for traffic lights staging problems.

Another popular biologically-inspired algorithm for solving the optimization problem in the traffic light systems concerns the GA (*genetic algorithm*s) approaches [56]. Let us note that genetic algorithms have also been used in [57] for the optimization of traffic flow control. Authors demonstrate the ability of such algorithms to determine the optimized solution in a self-tuning

process, by tracking the current queue length in the considered intersection. The queue length is here estimated from the incoming traffic flow rate. The green time is then generated by the genetic algorithm for the optimization of the traffic flow within the intersection. One of the main characteristics of the GA approach concerns its ability to evolve through generations of chromosomes by choosing the maximum fit chromosomes and letting them self-optimizing through successive generations. Results reported show that the traffic lights cycle can be successfully optimized by using a genetic algorithm.

In the case of "La Almozara" [58] district in Saragossa, three key techniques have been applied for solving the problem of congestion: 1) A genetic algorithm was used for the optimization task; 2) A cellular automata, based on micro-simulators for evaluating all possible solutions for traffic programming times; 3) A Beowulf Cluster, which is a multiple-instruction-multiple-data (MIMD) multicomputer of excellent price/performance ratio. For testing purposes, ten hypothetical congestion situations have been defined, with four different fitness functions, including the number of Vehicles (NoV), the Mean Travel Time (MTT), the Time of Occupancy / State of Occupancy (TOC/SOC) and the global mean speed. The most interesting results occurred in extreme cases, corresponding to an increasing number of vehicles introduced into the system.

The GA approaches have also been retained for traffic lights optimization purposes in [59], [60], [61], showing here again highly promising performances.

Another biologically-inspired algorithm is the so-called PSO (*Particle Swarm Optimization*) approach [62]. PSO is defined as a computational method that optimizes a problem by iteratively attempting to improve a candidate solution with regard to a given measure of quality. A first introduction of the algorithm was proposed by Kennedy, Eberhart, and Shi [63], [64], and defined as a form of social behavior of movement of organisms in a bird flock (swarm).

Since then, many authors try to use PSO as an optimization algorithm for traffic lights optimization. In [65], PSO has been used as an optimizer for determining a successful traffic light cycle program. Two scenarios are here introduced and 18 different numerical instances were generated depending on the number of vehicles into the system, as well as the number of traffic lights. During the test phase, several analyses have been carried out, concerning the performance of the optimization technique, the scalability issues, the computational effort required, and the quality of the solution. The results from these analyses are compared with other solutions, including the RANDOM algorithm [66], differential evolution [67] and standard PSO [63] algorithms, showing that the proposed PSO approach outperforms the other proposed methods with respect to two highly important features: a higher number of vehicles reaching the final destination in a given time and the average journey time.

In [68], the road network in the Kumamoto city has been modeled and considered. Several tests have been performed for solving the traffic congestion problem. A PSO algorithm was proposed as a traffic light optimization technique. The algorithm was implemented and attached to Aimsun 6.1 simulator via a dedicated API. Four junctions with different formations and traffic lights cycles were tested. Experimental results were compared with the base-line ME-GA (*Multi-Element Genetic Algorithm*)

method. The comparative evaluation performed shows that the PSO algorithm outperforms the ME-GA algorithm, in terms of both decreases in the total number of vehicles inside the system and vehicle delay time.

Another interesting study has been presented in [69], where three GA and PSO algorithms have been compared. The three GA algorithms are slightly different in terms of chromosome selection strategies: (1) randomly, (2) the best two and (3) best half respectively. The simulation has been conducted in SUMO simulator with two lanes and four intersections. The obtained results showed that GA in its version 3 and PSO algorithm are more efficient to achieve near-optimal solutions than other algorithms. Results also showed that the topology of the problem's function may be the central point in selecting the best-suited algorithm.

The above-mentioned algorithms presented provide good and promising results for further research in the domain, while leaving space for further improvement and new propositions. In our work, we have adopted a PSO approach, developed under the SUMO simulation platform. The proposed optimization method is described in the following section.

3.3. Particle Swarm Optimization

The proposed traffic light cycle optimization strategy is based on the PSO algorithm. The approach is illustrated in Figure 3-1.

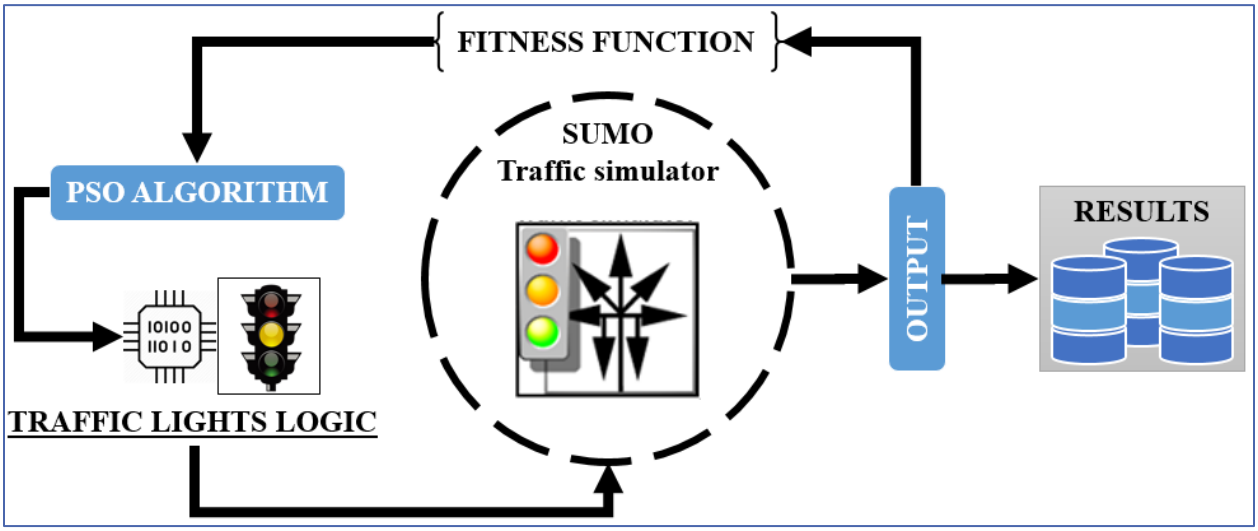


Figure 3-1. Traffic lights simulation and optimization: an overview

The initialization stage starts with a set of random values representing the phase durations, which will constitute the particles in the swarm. The corresponding values range in the [5, 60] (seconds) interval for the green and red lights. The amber light is a constant of 5 seconds.

At each stage, for the current set of parameters, a SUMO simulation is performed. The outcome of this simulation is used to compute the target fitness function. Each cycle program is statically loaded for each simulation. Within one cycle program the various variables including time duration, number of vehicles and map remain unchanged.

Let us now introduce the PSO algorithm used in our work.

3.4. PSO (Practical Swarm Optimization) algorithm

By definition, *Particle Swarm Optimization* (PSO) is a population-based metaheuristic algorithm [62] inspired by the behaviors of bird flocking (swarm). The algorithm has been initially designed for continuous optimization problems.

In PSO, each single candidate solution represents a "particle" or a "bird" in the space described by a point in a D – dimensional vector space, where D is the number of parameters to be optimized. Thus, the i^{th} particle $\boldsymbol{X^i}$ is defined as a D – dimensional vector, denoted by:

$$\boldsymbol{X^i} = [\boldsymbol{X_1^i}, \boldsymbol{X_2^i}, \boldsymbol{X_3^i} \ \cdots \ \boldsymbol{X_D^i}] \tag{3.1}$$

A swarm is then defined as a set of N candidate particles and denoted by $\boldsymbol{X_{swarm}}$:

$$\boldsymbol{X_{swarm}} = \{\boldsymbol{X^1}, \boldsymbol{X^2}, \ \ldots, \boldsymbol{X^N}\}. \tag{3.2}$$

The initialization of the algorithm starts with a swarm, *i.e*. a group of N random particles that represent solutions. The PSO algorithm is then iteratively searching for the optimal solutions by iteratively updating successive generations. At a given iteration, each particle is updated by two best values that are measuring the appropriateness of the considered solution (particle), with respect to a given *fitness function*.

A first value, denoted by *pbest* (*personal best*), represents the best solution achieved so far by an individual particle, with respect to the considered fitness function. A second value, denoted by *gbest* (*global best*) is the best value achieved so far by *all* the particles in the population.

After obtaining the *pbest* and *gbest* values, each particle can update its velocity and positions, as described in equations (3.3) and (3.4), respectively.

More precisely, the i^{th} particle position $\boldsymbol{X_k^i}$ at iteration k is updated as follows:

$$\boldsymbol{X_{k+1}^i} = \boldsymbol{X_k^i} + \boldsymbol{V_{k+1}^i} \ , \tag{3.3}$$

where $\boldsymbol{V_k^i}$ is the particle's velocity at iteration k.

The velocity is an important component of the process that governs the way that particles move across the search space and is defined as follows:

$$\boldsymbol{V_{k+1}^i = w^i V_k^i + U[0, \varphi_1]\left(pbest_k^i - X_k^i\right) + U[0, \varphi_2]\left(gbest_k - X_k^i\right)}, \quad (3.4)$$

where:

- $\boldsymbol{V_k^i}$ – is the velocity of the particle *i* at iteration *k*,
- $\boldsymbol{pbest_k^i}$ – is the personal value of the best solution that particle *i* has reached so far, up to iteration *k*,
- $\boldsymbol{gbest_k}$ – is the value of the global best particle (leader) in the entire swarm, up to iteration *k*
- $\boldsymbol{w^i}$ – is the inertia weight of the particle,
- $\boldsymbol{\varphi_1}$ and $\boldsymbol{\varphi_2}$ – are the acceleration coefficients that control the relative effect of *pbest* and *gbest* values,
- $\boldsymbol{U[0, \varphi_n]}$ – is a uniform random value in $\boldsymbol{U[0, \varphi_n]}$, $\boldsymbol{n \in 1, 2}$, which is sampled for each component, *i.e.*, velocity vector, particle, and iteration.

The fitness function is a particular type of objective function that measures how close a given solution is with respect to the considered objective. In our case, the two main objectives are considered when defining the fitness function. Firstly, we want to maximize the number of vehicles that reach the destination during a given simulation time. Secondly, we aim at minimizing the vehicle's journey time.

For this reason, we have adopted the fitness function introduced in [65], defined as described in equation (3.5):

$$\boldsymbol{F(X_k^i) = \frac{\left(\sum_{v=0}^{V} j_v(X_k^i)\right) + \left(\sum_{v=0}^{V+C} \omega_v(X_k^i)\right) + (C(X_k^i) \cdot St)}{V^2(X_k^i) + Cr}}, \quad (3.5)$$

where, for the considered i^{th} particle at iteration *k* ($\boldsymbol{X_k^i}$):

- $\boldsymbol{V}$ – is the number of vehicles that reach their destination in a given period of time,
- $\boldsymbol{S_t}$ – is the total simulation time,
- $\boldsymbol{C}$ – is the number of vehicles that do not reach their destination and remain in simulation,
- $\boldsymbol{j_v}$ – is the overall duration of the vehicle's journeys (for the vehicles that reach their destination),
- $\boldsymbol{\omega_v}$ – represents the time that each vehicle must stop and wait.

Here, $\boldsymbol{Cr}$ is a parameter that measures the ratio of colors in each phase state, averaged over all traffic lights in the system, defined as described in the following equation:

$$\boldsymbol{Cr = \sum_{k=0}^{tl} \sum_{h=0}^{ph} S_{k,h} \cdot \left(\frac{G_{k,h}}{R_{k,h}}\right)}, \quad (3.6)$$

where:

- $\boldsymbol{h}$ - phase state of the traffic light,

- $\boldsymbol{k}$ – the k^{th} traffic light,
- $\boldsymbol{G_{k,h}}$ - is the number of traffic lights in green,
- $\boldsymbol{R_{k,h}}$ - is the number of traffic lights in red (minimum value is 1 to avoid division by 0),
- $\boldsymbol{S_{k,h}}$ - is the duration of the phase state h for traffic light k

The $\boldsymbol{Cr}$ parameter plays a regularization role. More precisely, a prolonged state in the green light phase may be introduced for some specific traffic lanes. This improves the traffic flow for the considered lane but also makes traffic flow worst on other lanes. In this regard, a balanced system for traffic lights phases is needed, that is able to control the lights phase duration. This system will increase the green-light duration in lanes with a higher number of vehicles, and also increase the red light duration with lanes with a low number of vehicles.

The first objective, *i.e.* maximizing the number of vehicles that reach their destination ($\boldsymbol{V}$) during the simulation time ($\boldsymbol{S_t}$), is related to minimizing the number of vehicles ($\boldsymbol{C}$) that do not reach their destination and remain blocked somewhere in the simulation.

The second objective*, i.e.* minimizing the vehicle journey time ($\boldsymbol{j_v}$), concerns solely the vehicles that reach their destination in the simulation time period. An additional penalty term is computed by multiplying two factors $\boldsymbol{C}$ and $\boldsymbol{S_t}$, in order to take into account the vehicles with incomplete journeys.

Figure 3-2 provides a pseudo-code representation of the PSO algorithm.

```
1: initialization()
        init(X^i)
        init(gbest)
2: while k < max(iteration)
        updateV(V^i_{k+1})
        updateP(X^i_{k+1})
        evaluate(F(X^i_k))
        update(pbest, gbest)
    3: update(gbest)
```

Figure 3-2. Pseudocode of PSO

In the specific case of traffic lights optimization, the PSO algorithm is adapted as follows. Let us consider a set of I intersections in a given urban area. Each intersection i is governed by a certain number of traffic lights and is characterized by a set of $n(i)$ phase durations denoted by $\{\boldsymbol{d_1^i}, \boldsymbol{d_2^i}, \ldots, \boldsymbol{d_{n(i)}^i}\}$. The concatenation of all phase durations for all the traffic lights considered in the system defines a given particle. We obtain thus a particle vector of size $\sum_{i=1}^{I} \boldsymbol{n(i)}$.

This completes the theoretical description of the PSO algorithm. Let us now detail the environment and simulation protocol used to experimentally evaluate the proposed approach.

3.5. Environment and simulation setup

The main goal of this experiment is to develop a traffic light optimization technique that can be scalable and applicable to real-life scenarios.

3.5.1. Environment

In order to develop an optimizer that can be implemented and tested in real-life scenarios, first, we have to create a smaller model and test the proposed optimization technique. The considered simulation environment includes lanes, intersections, traffic lights, vehicles, and simulation time. Buildings and surrounding areas are excluded from the simulation since they are not really pertinent to our objectives and add useless computational burden. The distribution of vehicles is random through all the simulations and is obtained with the help of the native random generator offered by SUMO. All vehicles are randomly generated, with associated start and endpoints, under the form of O/D (Origin / Destination matrix). The vehicle's velocity is limited to 50 km/h maximum, which is normal for urban areas.

Concerning the testing location, we have chosen two different intersections situated in the city of Evry, France. They are illustrated in Figure 3-3. These intersections have been retained since they are quite representative, corresponding to a configuration that can be easily encountered in most European cities.

Figure 3-3. Two intersections in the city of Evry, France

A small scale scenario was created, including six different probe itineraries. The trial runs (probes) itineraries were planned in advance, as illustrated in Figure 3-4. Here, black lines schematically represent the road network scenario and red lines the considered probe itineraries. The first three itineraries are passing through 2 intersections and the last three itineraries are passing through only one intersection.

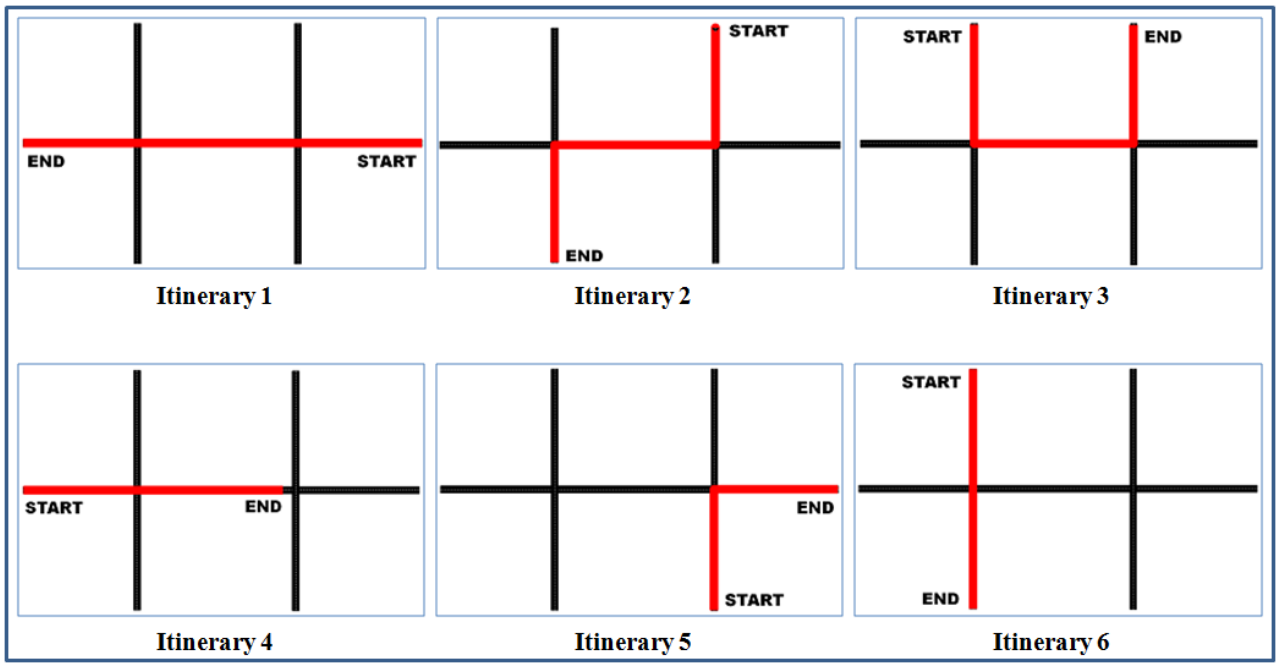


Figure 3-4. Probe itineraries

The goal of creating these different scenarios is to take to account various possible vehicle movements and to get an accurate measure of the performances of the proposed method. The total length of the itineraries varies between 1200 and 2800 meters, while the real distance between the two intersections is 400 meters.

3.5.2. Simulation setup

The simulation scenario was conducted with the following setup. Concerning the number of vehicles, three different test simulations have been conducted, each involving a different number of vehicles. In our experiments, we have considered 500, 1000, and 2000 vehicles. For each simulation, 30 to 60 vehicles were imported as *probes* that are forced to cross the predefined itineraries in different periods of time, with 10 or 20 vehicles per group (Table 3-1).

Let us recall that a probe is, by definition, a vehicle that performs a predefined itinerary, under the current global traffic conditions. Each probe is characterized by a given itinerary and a starting time. It serves as an observation to perform the necessary measurements, *i.e.*, time of completion of the itinerary. When a vehicle reaches the finishing point, it disappears from the system and does not reappear again. The total simulation time was set to 1600s for 500 and 1000 vehicles, and 3200s for 2000 vehicles.

TABLE 3-1. SIMULATION GROUPS AND TIME

Total No. Vehicles	500	1000	2000
Total simulation time	1600s	1600s	3200s
Total No. Probes	30	30	60
RED Group	300s – 500s	300s – 500s	600s – 1000s
BLUE Group	700s – 900s	700s – 900s	1400s – 1800s
GREEN Group	1100s – 1300s	1100s – 1300s	2200s – 2600s

All simulations performed by SUMO have an even distribution of vehicles for a fixed period of time as illustrated in Figure 3-5. In order to test and measure the proposed method, it is important to know the number of vehicles in each period of time. Figure 3-5 also shows the distribution of 500 vehicles in the period of 1600 seconds, with 30 probes launched in different periods of time: red group (300s-500s), blue group (700s-900s), green group (1100s-1300s). This allows us to benchmark the method in the three different time intervals presented in Table 3-1.

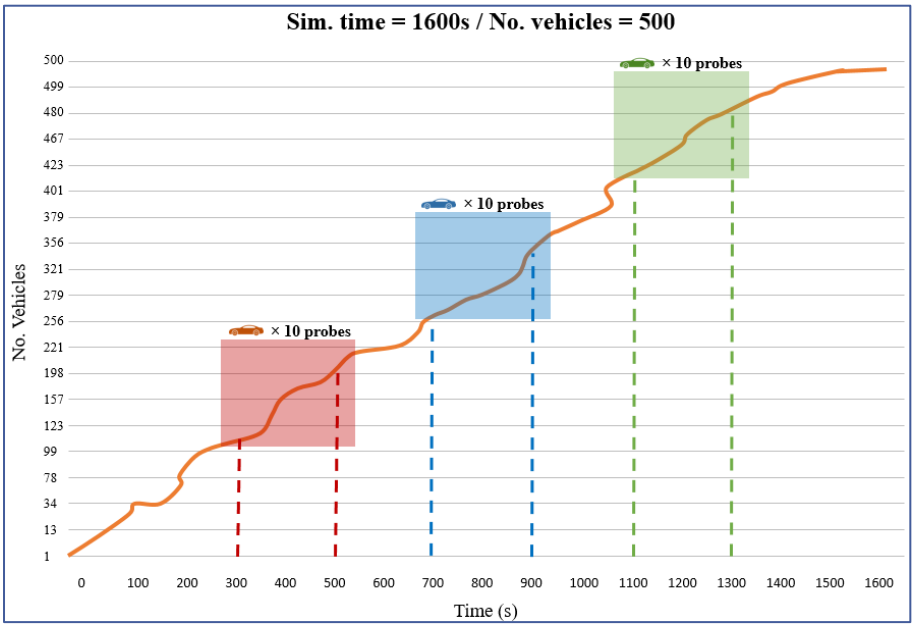


Figure 3-5. Launching 30 probes in three different time periods (RED, BLUE and GREEN)

We can observe a quasi-linear increase in the number of vehicles actually present in the simulation. In order to capture the behavior of the observed probes, we have divided the probes starting time into three groups, corresponding to the beginning, the middle, and the end of the simulation. They are respectively denoted by RED, BLUE, and GREEN. However, there are some differences that are worthy to be pointed out between the cases with 500/1000 vehicles and the one that involves 2000 vehicles:

- (500/1000 vehicles - 1600s) When the total number of the vehicle in the system is 500 or 1000, for the RED, BLUE and GREEN groups, the starting point of the corresponding probes is uniformly sampling the [300-500s], [700-900s] and [1100-1300s] intervals, respectively. An equal number of 10 probes are generated for each group, which leads to a total number of 30 probes. Thus, in this case, a probe is launched every 20 seconds.
- (2000 vehicles - 3200s) When the number of vehicles is 2000, the global time of simulation is increased up to the 3200s. Consequently, in this case, the corresponding RED, BLUE and GREEN intervals are set to [600-1000s], [1400-1800s] and [2200-2600s], respectively. In addition, since the length of these intervals is two times greater than in the previous scenarios, a number of 20 probes has been generated in each RED/BLUE/GREEN group. This leads to a total number of 60 probes, with a probe launched every 40 seconds.

The total simulation time was set to 1600s for 500 and 1000 vehicles, and 3200s for 2000 vehicles. Let us note that it is necessary to increase the simulation time when a greater number of vehicles are considered, in order to make sure that a sufficient number of probes finish the simulation.

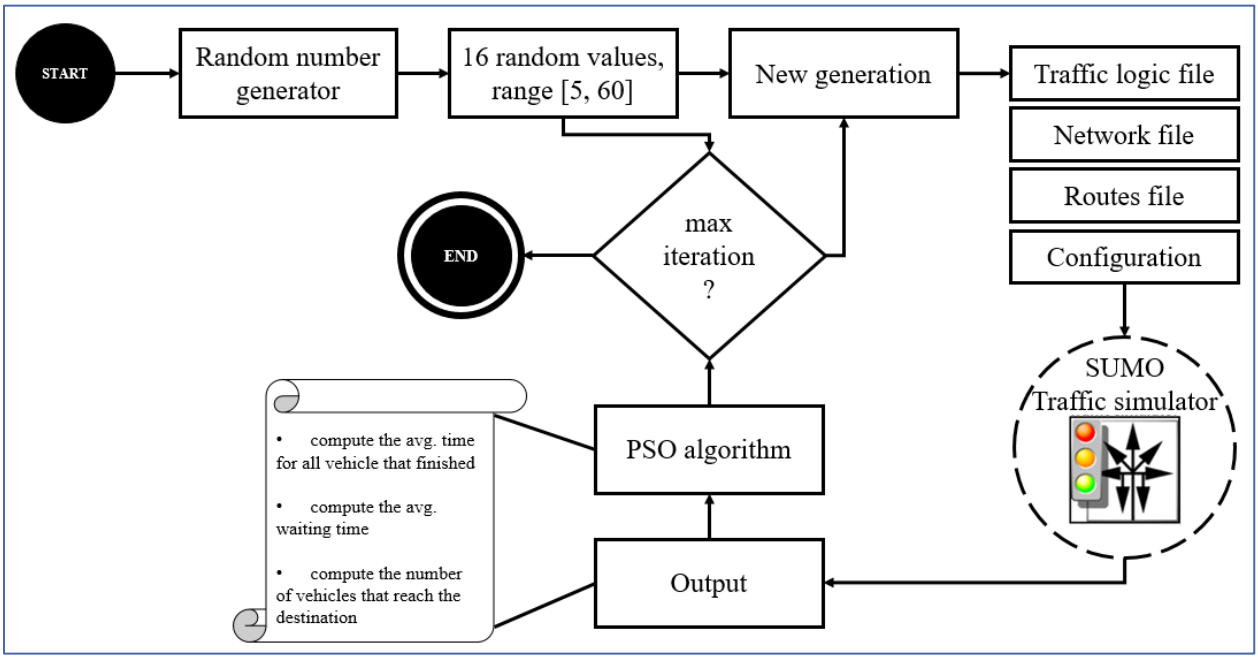


Figure 3-6. Simulation and optimization cycle

Figure 3-6 illustrates with diagram the full simulation cycle performed with the PSO algorithm integration.

The algorithm execution involves the following steps:

- Initialize the random number generator.
- A number of 16 different values for the traffic lights are associated with the corresponding lights in the traffic light structure considered (*i.e.*, two intersections, with 8 different traffic lights phases per intersection).
- Import the current traffic light generation in SUMO and run the simulation.
- Output the SUMO simulation result and feed it to the PSO algorithm in order to evaluate the fitness function.
- Compute and compare the *pbest* and *gbest* values, then update the set of particles (new traffic lights cycle program) as described in equations (3.3) and (3.4).
- Check if the number of iterations reaches the (6000 per simulation). If yes, stop the algorithm. Otherwise, continue the process.

TABLE 3-2. SIMULATION PARAMETERS, STLS AND PSO PROPERTIES

Parameters	Name	Value
Simulation	Simulation time (t)	1600s / 3200s
	Area	2 Intersections
	Number of vehicles	500 / 1000 / 2000
	Vehicle speed	0 - 50 km/h
	Number of probes	30 / 60
	Number of vehicles	500 / 1000 / 2000
	Map location	Evry
Traffic Lights		
	Traffic lights	16
	Number of intersections	2
PSO		
	Maximum number of evaluation per simulation	6000
	Number of independent runs	3
	Number of generations	20
	Swarm size	100
	Particle size (traffic lights)	16

Table 3-2 summarizes the simulation setup considered for the testing environment, with STLS and PSO algorithm properties.

The simulation scenario was performed by SUMO suite version 0.25.0. The PSO algorithm was implemented using the C++ library called "MALLBA" [40]. Experiments have been carried out on a virtual machine running Linux, version 14.04 with the following specifications: 8192MB RAM, 100GB Internal storage, 4 CPU units (Intel Xeon 3.60 GHz). The GPU was not used.

3.6. Experimental results

The results obtained for the traffic lights cycle (phase) program both PSO and STLS are presented in Table 3-3. The obtained results for each algorithm have been used as traffic lights cycle programs for each performed simulation.

They are divided into two groups, corresponding to each intersection: the first group (numbered from 1 to 8) corresponds to intersection 1, while the second (numbered from 9 to16) to intersection 2.

The results obtained by STLS are always the same, whatever the number of vehicles inserted in the system since the STLS approach does not take into account the traffic conditions. For the PSO optimization method, we can observe that different values are obtained, in function of the number of vehicles present in the system.

TABLE 3-3. PSO AND STLS TRAFFIC LIGHTS CYCLE PROGRAM: OBTAINED PHASE DURATIONS

Phase duration in seconds				
	PSO			STLS
No Vehicles	500	1000	2000	500/1000/ 2000
Intersection 1 / 2				
1 / 9	13 / 13	25 / 37	23 / 16	31
2 / 10	5 / 5	5 / 5	5 / 5	5
3 / 11	6 / 5	5 / 5	5 / 5	6
4 / 12	5 / 5	5 / 5	5 / 5	5
5 / 13	31 / 14	50 / 50	50 / 22	31
6 / 14	5 / 5	5 / 5	5 / 5	5
7 / 15	5 / 5	5 / 5	5 / 5	6
8 / 16	5 / 5	5 / 5	5 / 5	5

The computational times required for computing the PSO algorithm and determining the best traffic lights program are presented in Table 3-4. From the presented results it is obvious that increasing the number of vehicles in the system leads to a natural increase in computational time.

TABLE 3-4. COMPUTATIONAL EFFORT FOR THE PSO ALGORITHM

	500 vehicles / 1600s	1000 vehicles / 1600s	2000 vehicles / 3200s
Time (minutes)	36	72	235

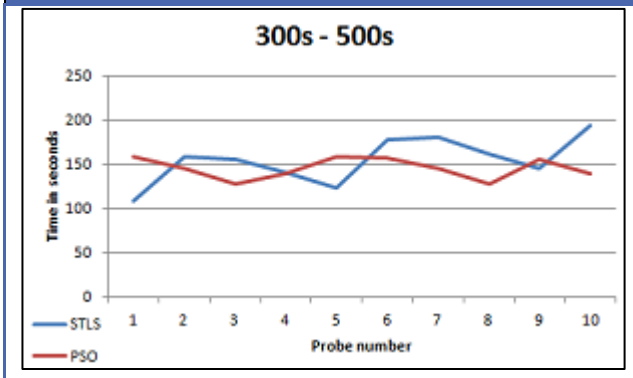
Let us underline that here, the simulation has been created in a small-scale scenario, hence excluding the unnecessary elements that may be included in SUMO, such as buildings, parks, houses. In this case, the main computational burden is balanced between the calculation of the PSO algorithm and the time required for performing the corresponding SUMO simulation. Let us also observe that the simulation must be executed sequentially and not in parallel because the PSO algorithm takes as an input the results of the last simulation (traffic lights cycle program).

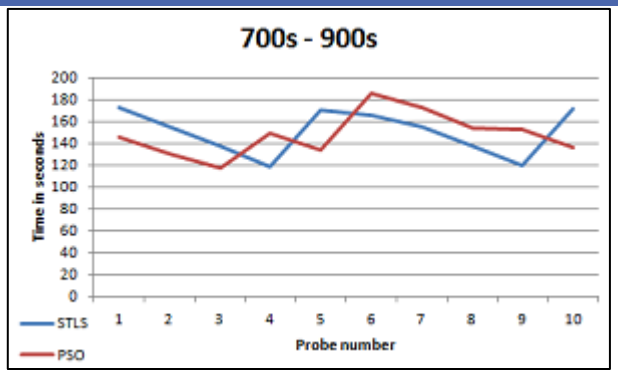
Simulation results were collected in 3 different groups (number of vehicles), for all six experimental itineraries (from 1 to 6) in two different times:

- Group 1 - 500 vehicles, 30 probes, 1600 seconds
- Group 2 - 1000 vehicles, 30 probes, 1600 seconds
- Group 3 - 2000 vehicles, 60 probes, 3200 seconds

The obtained results for the probes in Groups 1, 2 and 3 are respectively presented in Figure 3-7, Figure 3-8 and Figure 3-9, for both STLS (Blue) and PSO (Red) approaches.

Group 1: 500 vehicles in 1600 seconds, 3 different time slots, 30 probes





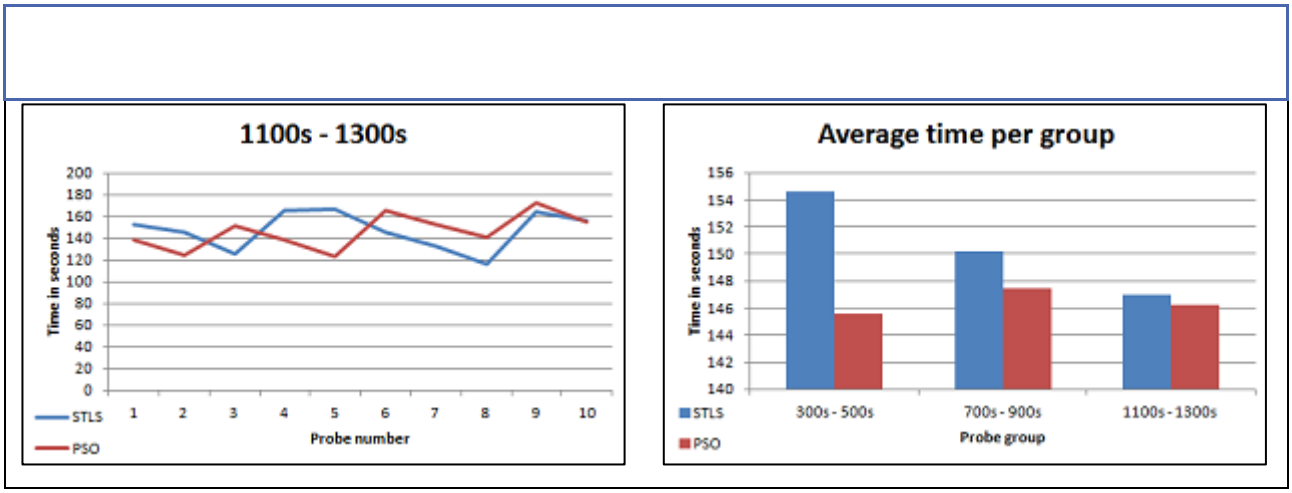


Figure 3-7. Obtained journey times for the probes in Group 1

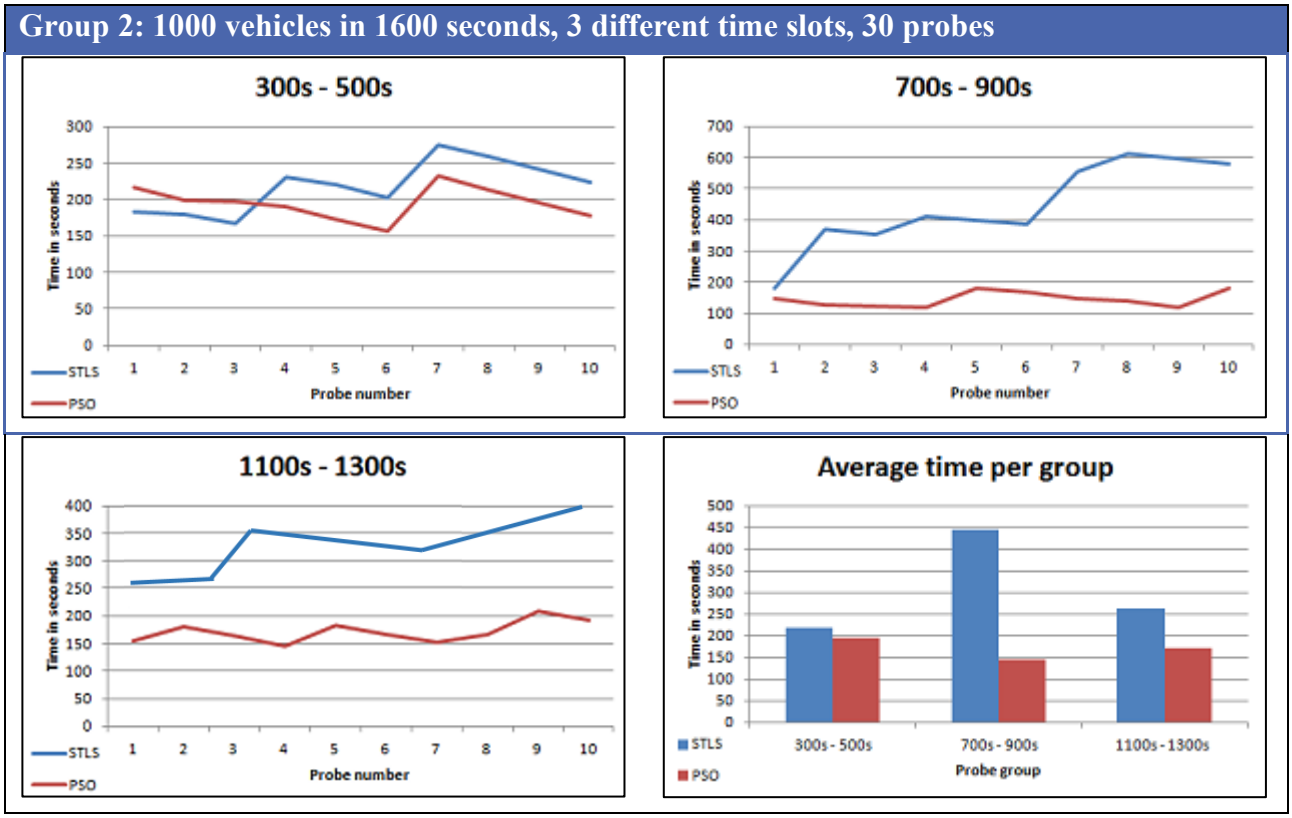


Figure 3-8. Obtained journey times for the probes in Group 2

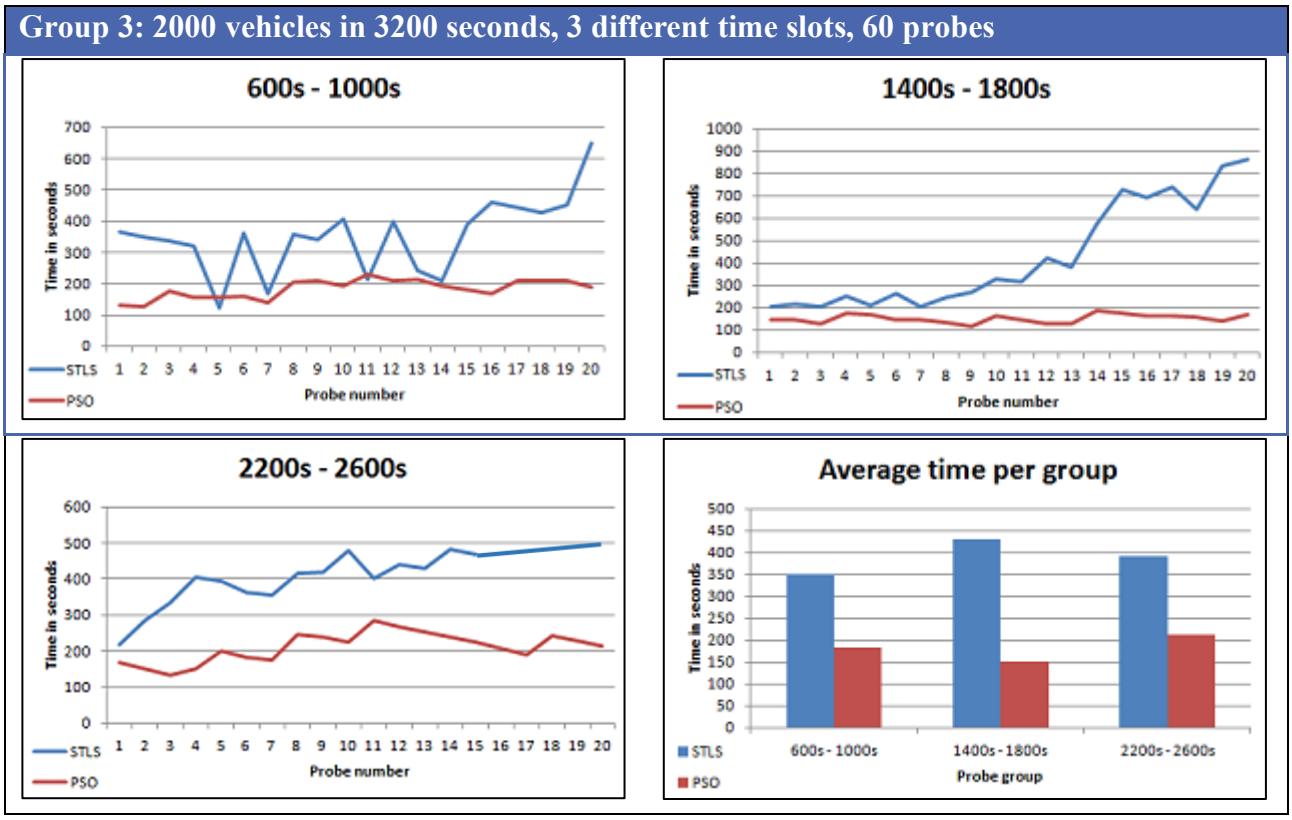


Figure 3-9. Obtained journey times for the probes in Group 3

The results obtained show a significant improvement in terms of probe journey time, notably in the cases of Group 2 and Group 3, where a more important number of vehicles is inserted in the system. This effect can be explained by the fact that in Group 1 the system is not heavily saturated with vehicles. The final plot of each group gives the average time of all tested probes (lower is better, meaning less time). This behavior is also confirmed by the results presented in Figure 3-10 and Figure 3-11, which respectively report the average time loss and the average journey times obtained.

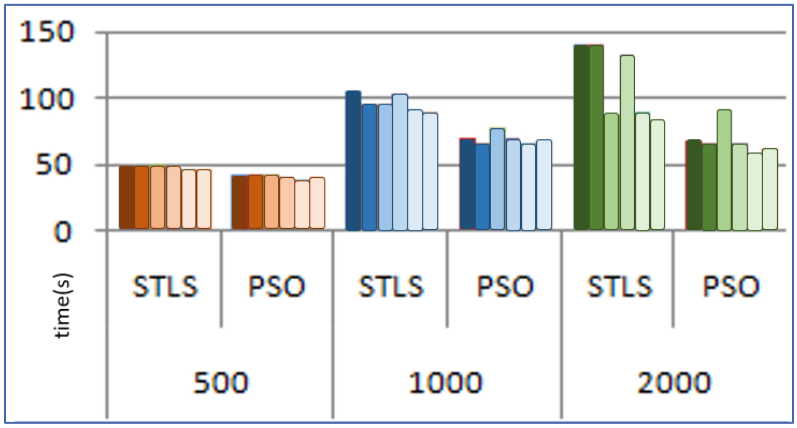


Figure 3-10. Average time loss in different vehicle groups

Here, the 6 different colors represent the 6 itineraries considered, grouped per number of vehicles. The average time loss refers to the time that probes wait, spent in the traffic jam, or in an intersection waiting for the green light. The average journey time is the time needed for a probe to complete the requested itinerary.

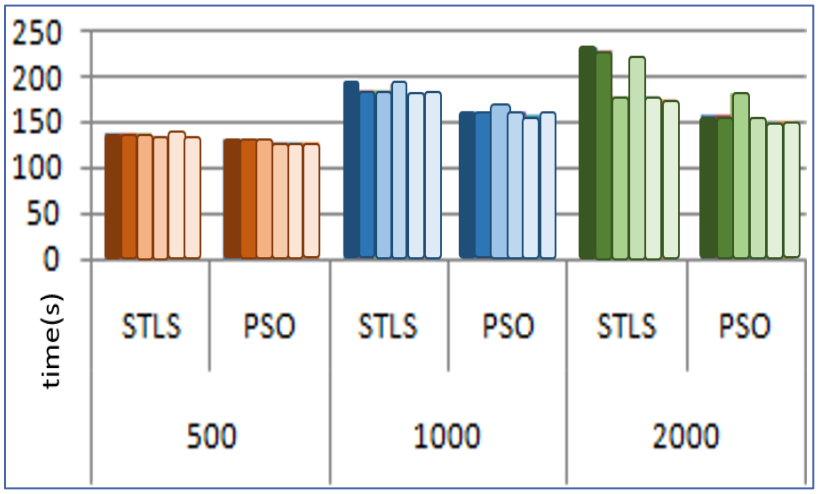


Figure 3-11. Average journey time in different vehicle groups

We can observe that the proposed optimization technique (PSO) performs better than STLS, in all the experimental scenarios considered. The most important gains in performance are in the range of 1000 to 2000 vehicles, for the 1600s or 3200s simulation time, respectively.

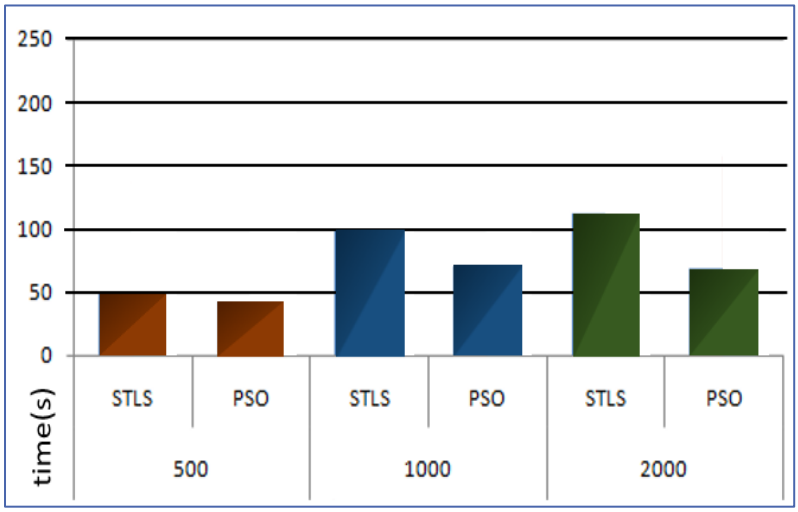


Figure 3-12. STLS and PSO performance: average time loss

Finally, Figure 3-12 and Figure 3-13 respectively show the average time loss and average journey time, obtained under PSO and STLS algorithms, globalized over all probes and itineraries involved and for all the three groups considered.

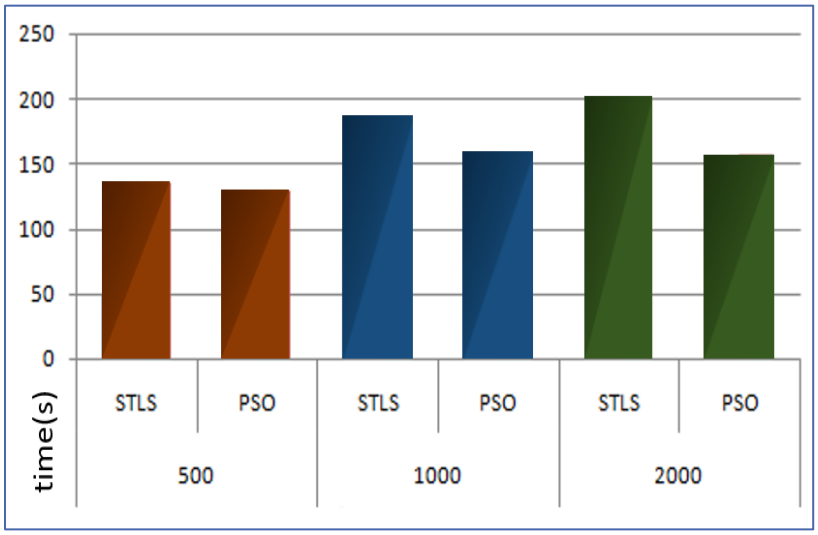


Figure 3-13. STLS and PSO performance (AVG journey time)

In all experiments presented PSO outperforms STLS solution, with the following performances in terms of:

Average time loss

- 500veh/1600sec - 14,39% less time loss;
- 1000veh/1600sec - 28,19% less time loss;
- 2000veh/3200sec - 38,67% less time loss;

Average journey time

- 500veh/1600sec - 5,37% less time;
- 1000veh/1600sec - 14,63% less time;
- 2000veh/3200sec - 21,53% less time;

We can note that the gain of the average loss time that probes are loosing in waiting, is higher when compared with the overall journey time (average journey time). This observation is interesting because it shows that there is a direct correlation between the traffic lights cycle program and the waiting time.

Another measurement that is interesting is the total number of vehicles that reach their destination in the given simulation time (*i.e.*, vehicles that complete the simulation successfully). Here the PSO algorithm outperforms the STLS approach in groups 2 and 3 while offering equivalent performances in group 1, as illustrated in Figure 3-14.

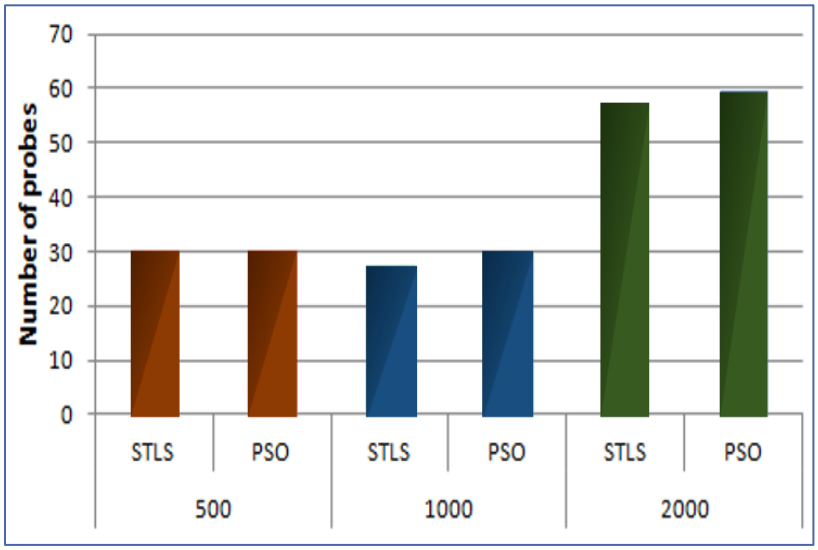


Figure 3-14. STLS vs PSO probes that complete the simulation in a given time (1600s/3200s)

The results obtained from all the experimental groups show clearly that PSO outperforms STLS for the three objectives considered:

- an increasing number of vehicles succeed to successfully finish the itineraries;
- the waiting time at the intersections is significantly reduced;
- the overall journey time for each vehicle is reduced.

3.7. Conclusion and discusion

In this Chapter, we have addressed the issue of traffic flow management and congestion in urban areas by proposing an optimization method as a solution. The proposed method uses the SUMO simulator and a PSO technique for optimizing the traffic lights cycle program.

Experiments have been conducted on two intersections with 6 different itineraries on a different number of vehicles in different simulation times. The obtained results have been analyzed and benchmarked against the native STLS approach proposed by the SUMO simulator.

The obtained experimental results presented and discussed showed that PSO outperforms STLS solution in all instances. For measuring the quality of the solution, a statistical comparison between the solutions has been conducted. We observed a significant decrease in vehicle waiting time at the intersections and duration of the overall vehicle journey time, as well as an increased number of vehicles that are able to complete the simulation. This shows that the proposed solution can lead to real improvement in real-life traffic situations. The computational time for completing the optimization process was between 30 minutes and 4 hours.

For further analyzes and developments, we may explore the proposed method on different traffic simulation platforms available, in order to observe the behavior of the optimization technique. It would also be interesting to investigate how the proposed method performs in different test environments. The scalability of the method is a highly important feature to be considered since we aim at performing a simulation of a whole city in real conditions. For this reason, we are interested in testing the method on large scale scenarios. Let us elaborate on this notion.

The process of optimization in urban environments that we have adopted, takes to account not only the considered intersection with the traffic lights configuration program but also the previous and following ones, in a recursive manner. By increasing the number of intersections, the amount of possible combinations also increases exponentially. Consequently, the complexity of the system becomes computationally intractable. So, this kind of solution should be taken into consideration only in situations where it is possible, meaning, where the city infrastructure allows it, and on small and relatively independent parts of a city. As an example, the green waves (green flow zones), can be observed and discussed. Here, the community consciously decides on optimizing/synchronizing several consecutive intersections, usually in the busiest or the main road of the city. This is done to create low and fluid traffic flow in the city, with the ease to cross the city relatively fast. The traffic lights are optimized in such a manner that allows the vehicles to pass the whole itinerary relatively fast and possibly without stopping on the traffic lights.

In the next chapters of the Ph.D. thesis, we are going to shift our focus from optimization to prediction. The focus is on improving public transportation in urban areas, predicting bus arrival time at each bus stop station, both for the operator and consumer usage.

Chapter 4. Public transportation prediction with machine learning algorithms: an overview

Abstract

This chapter proposes an overview of the state of the art in public transportation prediction. A large variety of methods is identified and discussed, including model-driven, data driven and machine learning-based approaches. They concern both short time approaches, which are client-oriented and adrress a temporal horizon of up to several hours, and long-term predictions, which are dedicated to the transportation operators, take into account longer periods of time and are useful for analyzing potential problems and investigate pertinent solutions. A particular focus is put on the machine-learning-based approaches, with both traditional and neural network-based techniques that are analyzed and described. The comparative analysis proposed shows that the neural networks outperform today the traditional methods, in terms of prediction accuracy.

The final part of this chapter briefly recalls the basic underlying principles of the most popular machine learning approaches, including linear regression, Support Vector Regressors (SVRs), fully connected, convolutional and recurrent neural netorks.

4.1. Context and objectives

Public transportation, public transit, or mass transit is defined as transport of passengers by group travel systems available for use by the general public. Public transportation is managed on a pre-defined schedule, operated on established routes and charges a fee for each trip, with some exceptions where the public transport is free of charge.

All around the world, from small and relatively poorly urbanized areas to big and highly dense cities, all are trying to address the problem of mobility and transportation. A good potential solution will have an impact on many domains, like avoiding traffic jams, decrease pollution levels, increase security, ensure reliable public transportation, offer more parking spaces... Improving such issues may have a huge impact on society, as well as on the everyday's life of the inhabitants. However, with the rapid growth of the number of vehicles in modern cities, the problem becomes more complex and difficult to solve.

In this thesis, the main objective is to develop prediction algorithms that can improve the efficiency of transportation over existing systems/infrastructures without physical changes or intervention.

More precisely, we aim at predicting, with the help of machine learning (ML) algorithms, the bus location in a given area for a certain period of time, while taking into account the global traffic flow and local traffic jams. In order to be able to perform accurate predictions, the algorithm must learn the past state (historical data) of the observed system and produce reliable prediction results.

Let us first analyze the state of the art in the field, described in the following section.

4.2. State of the Art

There are two main categories of vehicular traffic prediction approaches:

(1) Short-term predictions [70], which usually reflect a prediction horizon of up to one or two hours (typical example: predict the next bus arrival time information at the bus station). Such predictions are usually *client-based*, they take place in consumer applications (Google maps, Citymaper, Transilien…) and board notifications (located at the bus stop stations).

(2) Long-term predictions [71], which correspond to at least one or more days (example: prediction of the bus arrival times at the bus stops for an entire day). Such predictions are mostly *operator-based*, used by the company that provides these services for analysis and system improvements.

In the mainstream literature for road traffic prediction (in the scope of ITS and public transportation), there are two main families of approaches, so-called model-driven and data-driven.

4.2.1. Model-driven approaches

The model-driven approaches [72] attempt, usually through simulations of the whole urban region under analysis, to compute, analyze and predict the behavior or the performance of the observed entities in the system. Performing such simulations is a complex task, which requires a large variety of information as input, such as traffic flow data, number of vehicles, total population number, population average age, social status, economic status, working days, living and working areas and other similar information.

Among the well-known model-driven frameworks reported in the literature [73], [74], [75] let us notably mention the DynaMIT (Dy*namic Network Assignment for Management of Transportation to Travelers*) system. [72], developed by MIT's Intelligent Transportation System Laboratory. DynaMIT requires both offline and online (real-time) data in order to work properly. As road network input, the method requires the specification of a full network topology including links, nodes, and loading elements. Associated socio-economic historical data includes age, income, gender, trip purpose, car ownership, and other additional information obtained from census and questionnaires. Finally, the user needs to provide real-time information about the traffic situation (surveillance data) and incident reports (location, start-end time). The system is able to provide estimation, travel information and prediction as well as traffic density, flow speed, and travel time. Methods like the Extended Kalman Filter (EKF) [76] or Bayesian Additive Linear Model (BALM) [77] are used for dynamic adjustment of parameters based on real-time data. In addition, a genetic algorithm [78] is employed to optimizing the network control parameters.

However, the various variables involved are making this type of prediction highly complex and computationally heavy. In addition, it is difficult to collect/obtain such type of data, whose availability may also depend on the regulations existing in different countries. Such approaches will rest out of the scope of this thesis, which is exclusively focused on data-driven approaches, described in the following section.

4.2.2. Data-driven approaches

In the case of data-driven approaches, the decisions are based on data analysis and interpretation [79]. Such approaches [80] use both historical and real-time traffic data to perform analysis with data-mining inspired techniques in order to provide possible predictions for future traffic situations. Nevertheless, several challenges need to be overcome. Thus, traffic measurement data is not available for all traffic links, the data may be incorrect or incomplete, and traffic flows are complex and exhibit highly non-linear behaviors.

Well-known forecast techniques [81], [82], still widely used today, mainly include time series analysis [83], exponential smoothening [84], Kalman filtering [85], machine learning (SVMs) [86], or ANNs (Artificial Neural Networks) [87].

On the other hand, the parametric methods rely on a relatively reduced set of generic parameters that are governing the states and the output of the model, under a pre-defined data structure [88]. A long variety of parametric models are today available. Among the most popular ones, let us cite the fully-fledged traffic models like traffic simulators with OD (Origin-Destination) matrices, extended Kalman filters [76], time series analysis approaches [89], or linear regression [90].

The Box-Jenkins' *Autoregressive Integrated Moving Average* (ARIMA) model [91], is today one of the most known and popular approaches. The proposed model overcomes the issues of incomplete or noisy data by proposing a prediction scheme using a SARIMA model for traffic flow prediction that can be applied when only limited data input is available.

4.2.3. Machine Learning algorithms

With the rapid data growth and expansion of ITS in the recent years, machine learning becomes an important tool for solving complex problems like prediction, analytics and deriving patterns from a large amount of data [92].

In order to understand more about this trend, a general explanation may be first required. The Machine Learning paradigm can be divided into three main categories: supervised learning, unsupervised learning and reinforcement learning algorithms. Each of these categories has a unique approach in solving certain problems. Supervised machine learning algorithms [93], which are commonly adopted machine learning techniques, use a variety of functions to learn the connections between the input data and the labeled output. Then, a pre-trained model can perform a prediction over an unseen input. The main issue that needs to be ensured concerns the generalization capabilities of the model.

The unsupervised learning algorithms learn the connection without given labels, the objective here being to create its own labels during the learning process. One of the most widely adopted unsupervised algorithms in highway transportation planning is the well-known *k*-means approach [94].

Other use cases in ITS concern the self-driving vehicles, where the system continuously learns and improves on each new situation. Reinforcement learning is a particular type of machine learning algorithm which makes it possible to perform a suitable action in order to maximize the reward in a particular situation. This type of method is often used in control and optimization theory and is specifically adapted for optimizing the traffic signal control in ITS [95].

In our work, we have considered supervised machine learning algorithms that include both traditional approaches (linear regression, SVRs) and neural network-based approaches. The supervision of the

system is here again managed with the help of the SUMO simulator already introduced and used in the previous Chapters.

4.2.3.1. Traditional Machine Learning models

Traditional machine learning (ML) algorithms are today still widely used and adopted by both commercial companies and research communities. Let us recall some well-known and established ML algorithms involved in the public transportation prediction domain.

As the most simple approach, let us mention the case of linear regression [96], also known as OLS (*Ordinary Least Squares*). Here, the prediction model is based on linear combinations of its covariates, and the parameters indicate the contribution of the covariate to the outcome. The model is computationally light, and in some relatively simple cases can produce satisfactory results.

Support Vector Machine (SVMs) and Support Vector Regression (SVRs) [97] are well-known machine learning algorithms for solving classification and regression problems respectively. Over the years, they have been continuously gaining in popularity and adoption, for various applications, since they offer convenient implementation and testing facilities.

SVMs algorithms are still today one of the most popular and widely adopted methods. In many cases, they achieve good and accurate prediction results that can be used as standalone solutions or as a baseline for further improvements.

Let us cite some recent use cases and implementations reported in the literature. In [98], different features including road length, weather and speed have been used in order to successfully predict the bus arrival time. In a different manner, in [99], the SVM algorithm was used to predict multiple bus arrival times at one bus station. An application for bus time prediction was also proposed in [100]. Here, the distance between each bus station and the time needed for the bus to reach the station has been considered.

Among other algorithms used for public transportation prediction, let us also mention the *k*-nearest neighbors [101], or random forests [102].

Traditional ML algorithms have proven to be powerful tools for solving particular problems. Nevertheless, there are some key issues and challenges that need to be overcome.

First, traffic measurement data is not available for all traffic links so the data may be incorrect or incomplete. Second, traffic flows are complex and dynamic so they exhibit highly non-linear behaviors. For this reason, it is essential to look for solutions that can overcome such problems and produce better results. One potential solution concerns the neural network approaches, which have made in recent years a spectacular breakthrough in various fields of applications, including computer vision, object recognition, natural speech processing and so on. The following section examines how

such artificial intelligence-related approaches have been employed for the objectives of public transportation prediction.

4.2.3.2. Neural Network-based models

Artificial neural networks [103] algorithms are specific types of machine learning algorithms, that are able to learn both the model structure and the corresponding parameters directly from the data. Such models manage to find a certain robustness and prove to be less sensitive to incorrect or incomplete data. The main advantage is that the complexity, dynamics, and non-linearity encountered in various traffic conditions can be intrinsically taken into account. The development, in recent years, of affordable GPUs (*Graphical Processing Units*) with high processing power and intrinsic parallelization abilities, required for accomplishing the learning stage, has largely catalyzed the emergence of deep neural-networks approaches. Let us detail some recent achievements and research works carried out in the field of public transportation prediction.

In [104], authors employ a standard, Back Propagation Neural Network (BPNN) [78] for traffic forecast purposes. Geo-spatial bus data from the city of Guangzhou as well as weather report information have been here exploited. The obtained results are satisfying and through a ten-fold cross-validation experimental procedure, the model demonstrates its suitability for traffic forecast purposes.

Zheng *et al*. [105] introduce the so-called Bayesian-combined neural network (BCNN), which combines radial basis functions and a BPNN. The network assigns a credit value to each predictor and adapts accordingly its behavior. The obtained results show that the hybrid model outperforms a single neural network most of the time.

More recently, a new deep-learning-based traffic flow prediction method [106] has been proposed, which inherently takes into account both spatial and temporal correlations. A stacked auto-encoder, trained in a greedy, layer-wise fashion, is here used to learn generic traffic flow features.

In [107], the proposed method focuses on building a framework not for “normal” but for what they call “extreme” traffic conditions. A Deep LSTM (*Long Short Term Memory*) neural network is here used to forecast peak hours and identify unique characteristics for the traffic data. Further enhancing the model for post-accident forecasting by mixing Deep LSTMs provides a joint model for both normal traffic conditions and patterns of accidents. The experimental evaluation shows huge improvements over the baseline.

Ma *et al.* [101] propose a Convolutional Neural Network (CNN)-based method that puts the traffic data in an image/matrix-based representation and predicts large-scale, network-wide traffic speed. The proposed method is benchmarked against 4 different predictive methods, including ordinary least squares, k-nearest neighbors, artificial neural networks and random forests. Concerning the neural networks three deep learning architectures, namely, stacked auto-encoder, recurrent neural network,

and long-short-term memory have been retained. The obtained results show that the CNN-based approach manages to predict the traffic more accurately.

In a less conventional manner, in [108] authors model the traffic flow as a diffusion process performed on a directed graph, with the help of the so-called DCRNN (*Diffusion Convolutional Recurrent Neural Network*) approach. The method incorporates both spatial and temporal dependencies of the traffic flow. The spatial dependencies are captured with the help of bidirectional random walks on the graph. The temporal dependency is taken into account by using the encoder-decoder architecture with scheduled sampling. The framework has been evaluated on two real-world datasets and shows significant improvement over the state of the art.

4.2.3.3. Machine Learning algorithms: comparison and performances

In order to better position the research in the domain of ITS-public transportation prediction (buses in particular), it is important to narrow down the subject and explore deeper the most recent publications in the field. To this purpose, Table 4-1summarizes the existing approaches, recalling the underlying principles, and attempts to propose a fair comparison.

Let us first mention the approach introduced in [109], where one traditional ARIMA method and two deep learning techniques (LSTM and GRU which stands for Gated Recurrent Unit) are considered for predicting the traffic flow. Test and analysis have shown that the LSTM network performs better than the baseline ARIMA by improving the MAE (*Mean Absolute Error*) score of 5%, while the GRUs MAE is improved by 10%. The MAE score represents the mean average error of prediction.

A more recent paper [110] specifically tackles the issue of bus transportation prediction. The bus travel time is here jointly predicted for multiple bus stops. Several techniques are used for prediction purposes, including Historical Average, Linear Regression, SVR, as well as various deep neural network variants. Authors conclude that the proposed solution, so-called PCF - *Partitioning and Combination Framework* (which is a combination of ANNs) outperforms heavily other algorithms.

A long-term algorithm for bus arrival time prediction is introduced in [111]. The prediction algorithm is based on a standard BPNN (Back-Propagation Neural Network), which is a regular ANN algorithm. When benchmarked against a simple historical average approach, the proposed technique achieved a 5,7% improvement over the global MAE.

In [112], the authors propose to perform both short and long-term predictions based on the GPS coordinate of the bus movement. An OLS (Ordinary Least Squares) and FCNN (*Fully Connected Neural Network*) are here used for prediction purposes. The buses are located in the city of Bangkok in Thailand and the study was conducted because of the newly opened bus lines in the region. After the initial experiment, the obtained results showed a 55% improvement for the FCNN technique over the baseline OLS prediction.

A very recent study [113], conducted in the touristic city of Macao in China, also addresses the issue of bus arrival time prediction. Three different simple models, based on historical and real-time on-line bus locations are proposed, namely SMA (Simple Moving Average), ANN and HM (Hybrid Model). The HM is consist of a mixture between the SMA and ANN, which combines historical data and on-line data to improve and fine-tune the prediction results. The results reported have shown that the Hybrid Model can improve the MAE score by 17%.

In [114], the prediction of bus arrival time based on GPS data is proposed. A method called CK-means (K means Clustering) is used for prediction. The K-Means clustering method is used to cluster the historical traffic data and calibrate the running state of the road section. Compared with the BPNN (Back-Propagation Neural Network) model, HA (History Average) model, and FVDM (Front Vehicle Data Model) based on multi-directional data, the experimental results show that the prediction effect is better in the peak period with a 34,57% improvement in terms of MAE.

Efficient bus arrival time prediction based on spark streaming platform is described in [115]. Here, an OPFA (*Optimized Particle-Filtering Algorithm*) is used to predict the bus arrival time. The prediction model is improved by introducing the latest bus speed for collaborative data analysis, which improves the accuracy of the prediction process by taking into account the actual traffic conditions and that can simultaneously predict the arrival time of multiple buses. The model was benchmarked against HA and improves the MAE by 8.35%.

In [116], the authors exploit the GPS data collected by the bus company in real-time. Format conversion, missing value processing, data cleaning and denoising of GPS data and finally prediction is performed with the help of a GRU (*Gated Recurrent Unit*) neural network and compared with the LSTM approach. The results show that the accuracy of the GRU algorithm is 7.84% higher than that of LSTM.

In the recent works of Heghedus *et al*. [117], an overview and useful comparison between different techniques and algorithms for public transportation prediction are described. Here, also a comparison of the performances between the two most popular deep learning frameworks (PyTorch and TensorFlow) is discussed.

The survey proposed in [118] provides a comprehensive literature review of how deep learning models are applied in various transportation applications. An extensive study of a variety of techniques and methodologies is discussed as well as a direct comparison between different projects.

The analysis of the state of the art shows that machine learning and big data analysis techniques are highly helpful for allowing the transportation system to become smarter.

TABLE 4-1. (PART 1) COMPARISON OF MACHINE LEARNING MODELS FOR PUBLIC TRANSPORTATION PREDICTION.

Article	Year	Algorithms ML/DL	MAE (seconds)	Improvement In %	Prediction window	Veh. types	Dataset
Fu et al. [109]	2016	ARIMA GRU LSTM	9.175 18.127 17.211	0 5.47% 10.24%	Short-term	Veh / Bus	PeMS California USA
He et al. [110]	2018	HA LR SVR FCNN PCF	4.220 3.684 3.601 4.093 1.978	0 12.7% 14.67% 3.01% 53.13%	Long-term	Bus	Land transport authority Singapore
Wu and Tan [119]	2016	GBRT FCNN SAE LSTM CNN+LSTM	22.52 20.61 20.36 21.53 19.37	0 8.48% 9.59% 4.4% 13.99%	Short-term	Veh / Bus	PeMS California USA
Wu et al. [120]	2018	LASSO StoS SAE BPNN DNN-BTF	29.586 27.562 27.173 25.672 25.189	0 6.96% 8.16% 13.23% 14.86	Short-term	Veh / Bus	PeMS California USA
Pan et al. [111]	2012	HA BPNN	None	0 5.7%	Long-term	Bus	Privat com China
Yu et al. [121]	2017	LR RR LASSO FCNN LSTM Mix LSTM	1.631 1.629 1.445 3.643 1.003 0.970	0 0.12% 11.4% -123.36% 38.5% 40.53%	Long-term	Veh / Bus	CalTrans California USA
Yamaguchi et al. [122]	2018	LR ANN RF SVR GBDT	P values (T-test to MAE)	None (Best GBDT)	Short / Long-term	Bus	Nishitetsu Furuoka Japan

MAE - Mean Absolute Error, ML – Machine Learning, DL – Deep Learning, HA – Historical Average, LR – Linear Regression, RR – Ridge Regression, LASSO – Regression, OLS – Ordinary Least Squares RF – Random Forest, ARIMA - AutoRegressive Integrated Moving Average, SVR – Support Vector Regression, StoS – Sequence to sequence learning, SMA – Simple Moving Average, FVDM – Front Vehicle Data Model, PCF – Partitioning and Combination Framework, GBRT – Gradient Boosting Regression Tree, HOD – Historical Origin Destination, SAE – Stacked AutoEncoder, GA – Genetic Algorithm, GBDT – Gradient Boosting Decision Tree, CK-means – K means Clustering, OPFA – Optimized Particle Filtering Algorithm, HM – Hybrid Model, ANN – Artificial Neural Network, DNN-BTF – Deep Neural Network Based Traffic Flow, GRU – Gated Recurrent Unit, FCNN – Fully Connected Neural Network, BPNN – Back-Propagation Neural Network, LSTM – Long-Short Term Memory; PeMS – Performance Measurement System; CalTrans – California Transportation; BMTA – Bangkok Mass Transit Authority; DSAT – Bus Agency in Macao China; GPS – Global Positioning System

TABLE 4-2. (PART 2) COMPARISON OF MACHINE LEARNING MODELS FOR PUBLIC TRANSPORTATION PREDICTION.

Article	Year	Algorithms ML/DL	MAE (seconds)	Improvement In %	Prediction window	Veh. types	Dataset
Treethidtaphat et al. [123]	2017	OLS FCNN	None	0 55%	Short / Long-term	Bus	BMTA Bangkok Thailand
Lam et al. [113]	2019	SMA ANN HM	None	0 17% None	Short-term	Bus	DSAT Busline Macao China
Huang et al. [78]	2019	HOD GA LSTM	None	0 7.5% 14%	Short-term	Bus	GPS Guangzhou China
Zhang and Liu [114]	2019	HA FVDM BPNN CK-means	29.45 28.04 78.11 19.27	0 4.79% -165.23% 34.57	Short / Long-term	Bus	GPS Qingdao China
Liu and Xiao [115]	2019	HA OPFA	78.2 71.67	0 8.35%	Short-term	Bus	GPS Hohhot China
Bohan and Yun [116]	2019	LSTM GRU	0.682 0.761	0 7.84%	Short-term	Bus	Hohhot Bus Corporation China

MAE - Mean Absolute Error, ML – Machine Learning, DL – Deep Learning, HA – Historical Average, LR – Linear Regression, RR – Ridge Regression, LASSO – Regression, OLS – Ordinary Least Squares RF – Random Forest, ARIMA - AutoRegressive Integrated Moving Average, SVR – Support Vector Regression, StoS – Sequence to sequence learning, SMA – Simple Moving Average, FVDM – Front Vehicle Data Model, PCF – Partitioning and Combination Framework, GBRT – Gradient Boosting Regression Tree, HOD – Historical Origin Destination, SAE – Stacked AutoEncoder, GA – Genetic Algorithm, GBDT – Gradient Boosting Decision Tree, CK-means – K means Clustering, OPFA – Optimized Particle Filtering Algorithm, HM – Hybrid Model, ANN – Artificial Neural Network, DNN-BTF – Deep Neural Network Based Traffic Flow, GRU – Gated Recurrent Unit, FCNN – Fully Connected Neural Network, BPNN – Back-Propagation Neural Network, LSTM – Long-Short Term Memory; PeMS – Performance Measurement System; CalTrans – California Transportation; BMTA – Bangkok Mass Transit Authority; DSAT – Bus Agency in Macao China; GPS – Global Positioning System

4.2.3.4. Consumer applications and frameworks

As discussed in the previous section numerous different algorithms, techniques and applications have been introduced in order to address the issue of traffic prediction.

Some mainstream conventional commercial applications have also integrated real-time traffic conditions and provide a prediction of public transportation. Among the most popular solutions, let us cite Google Maps [124], Bing Maps [125] and Citymapper [126], which have been widely adopted

by the general public. Among other applications, let us also mention some local applications for specific cities, like Transilien [127] for Paris, MVV [128] for Munich, or RTC [129] for Las Vegas.

Even though these applications and software solutions are widely available and highly accepted, there is still a lot to be done (particularly for buses) in terms of accuracy for both long and short term prediction.

4.3. Retained machine learning algorithms

This section provides an overview of the machine learning techniques that we have retained and exploited in our work.

The baseline machine learning techniques considered and adopted in the thesis are the following:

- OLS (*Ordinary Least Squares*), also known as Linear Regression (LR) [96]
- SVR (*Support Vector Regression*) [97] with different kernels (*linear, rbf and poly*)

For these first ML algorithms, we have retained the Scikit-learn machine learning toolbox with their respective implementations.

Concerning the neural network-based approaches, we propose the following algorithms:

- FNN - Feedforward Fully Connected Neural Network
- CNN - Convolutional Neural Network
- RNN and LSTM – Recurrent Neural Network / Long Short Term Memory Neural Network (Special case of RNN)

For all of them, we have proposed a hand-crafted solution, adapted to our purposes, which will be further described in Section 5.6.

The deep learning algorithms have been implemented in a python programing language as a programming environment, with its respective data libraries (NumPy, pandas, torch), and for the ANNs implementation, a PyTorch [130] deep learning framework was adopted.

Let us now present the underlying principle of each of the prediction methods retained.

4.3.1. OLS (Ordinary Least Squares)

Ordinary least squares (OLS), or linear regression is a statistical method of analysis that estimates the relationship between one or more independent variables, with respect to a dependent one. Thus, the method estimates the relationship by minimizing the sum of the squares in the difference between the observed and predicted values of the dependent variable configured as a straight line.

The general goal of regression analysis is to learn some relationship between a variable to predict $\boldsymbol{y} \in$ R and some covariates $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p)\, T \in R^p$, with $p \geq 1$. In this case, the target value is expected to be a linear combination of the features involved. If $\boldsymbol{y}$ denotes the predicted value, then:

$$\boldsymbol{y} = \boldsymbol{\omega_0} + \boldsymbol{\omega_1 x_1} + \cdots + \boldsymbol{\omega_p x_p} \ , \tag{4.1}$$

Where $\boldsymbol{\omega} = (\boldsymbol{\omega}_0, \ldots, \boldsymbol{\omega}_p)$ are the parameters of the linear link function. To learn the values of these parameters, $(\boldsymbol{\omega}_0, \ldots, \boldsymbol{\omega}_p)$, we observe $n \geq 1$ pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ that are supposed to come from the same generating mechanism. The OLS approach minimizes the sum of squares of the distances between the observed values $\boldsymbol{y}_i$ and the predicted values at $\boldsymbol{x}_i$ under the linear model considered.

For $i = 1,\ldots,n$, (where $n \geq 1$ observations and $p \geq 1$ covariates) we consider $\boldsymbol{y}_i \in$ R and $x_i = \left(x_{i0}, \ldots, x_{ip}\right)^T \in R^{p+1}$ with x_{i0}=1. This is only to include the intercept in the same way as the other coefficients. The OLS estimator is the coefficient vector $\hat{\omega} = \left(\hat{\omega}_0, \ldots, \hat{\omega}_p\right)^T \in R^{p+1}$ such that:

$$\hat{\omega} = argmin_{\omega \in R^{p+1}} \sum_{i=1}^{n} (y_i - x_i^T \omega)^2 \tag{4.2}$$

This principle is illustrated in Figure 4-1.

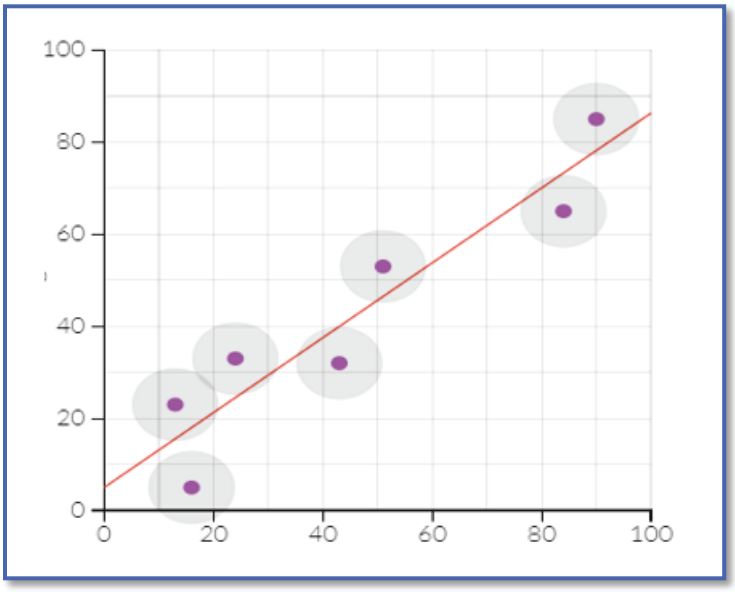


Figure 4-1. OLS principle: the points in the data set are approached by a straight line

Linear regression is the simplest machine learning algorithm that was implemented for prediction purposes in this work. Nevertheless, this technique can produce somewhere good enough results in a

very reasonable time and is convenient to implement. These make it a perfect candidate as a starting point and for benchmarking purposes.

4.3.2. SVR (Support Vector Regression)

SVR (*Support Vector Regression*), is an extension of the SVM (*Support Vector Machine*). As the name suggests, the SVR is a regression algorithm that can be used for working with continuous values instead of classification like SVM.

Figure 4-2 shows the main functionality of the SVR algorithm. The hyperplane basically defines the separation line between the data, similar to SVM (separating the data between classes). The Boundary line is the region of a problem space in which the output label of a classifier is ambiguous. In SVR these are also called decision boundary lines.

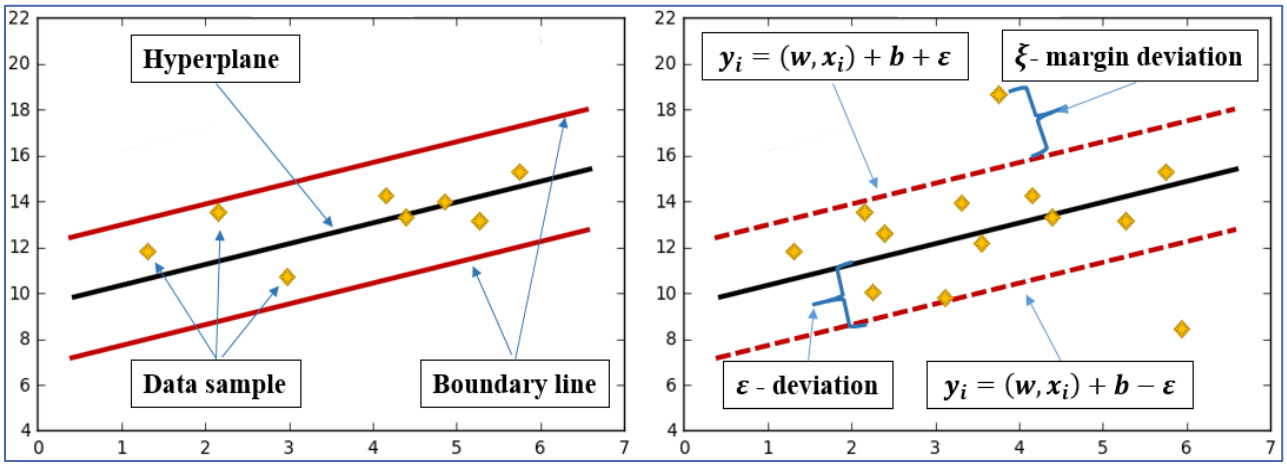


Figure 4-2. Hyperplane and boundary line (left); SVR concept (right)

The real difference between linear regression and SVR is that linear regression solely tries to minimize the error rate, while the SVR tries to fit the error within a certain threshold. In other words, SVR considers only the points that are within the decision boundary, and the best fit line is the hyperplane that has the maximum number of data.

Let us assume that the hyperplane is a straight line going through the Y-axis, then the equation of the hyperplane is:

$$\boldsymbol{w}^{\circ}\boldsymbol{x} + \boldsymbol{b} = \boldsymbol{Y} \ , \tag{4.3}$$

and the equation for the decision boundary is:

$$\boldsymbol{w}^{\circ}\boldsymbol{x} + \boldsymbol{b} = +\boldsymbol{\varepsilon} \text{ or } \boldsymbol{w}^{\circ}\boldsymbol{x} + \boldsymbol{b} = -\boldsymbol{\varepsilon} \ . \tag{4.4}$$

Thus, any hyperplane that satisfies:

$$-\boldsymbol{\varepsilon} < \boldsymbol{Y} - \boldsymbol{w}^{\circ}\boldsymbol{x} + \boldsymbol{b} < +\boldsymbol{\varepsilon} \ , \tag{4.5}$$

Hence, it can take only those points that are within the decision boundary and have the least error rate within the margin of tolerance. This gives us a better fitting model, which is determined by minimizing the quantity defined in equation (4.6):

$$\boldsymbol{min}\frac{\boldsymbol{1}}{\boldsymbol{2}}\left|\left|\boldsymbol{w}\right|\right| \ , \tag{4.6}$$

under the following constraints:

$$|\boldsymbol{y_i} - \boldsymbol{w}_{\circ}\boldsymbol{x_i}| \leq \boldsymbol{\varepsilon} \ , \tag{4.7}$$

From Figure 4-2 we can observe that not all the data falls into the boundary plane, so we need to account for the possibility of errors that are larger than $\boldsymbol{\varepsilon}$, also known as slack variables. Slack variables can be defined as follows: for any value that falls outside of $\boldsymbol{\varepsilon}$, we can denote its deviation from the margin as $\boldsymbol{\xi}$.

In order to minimize as much as possible such potential outliers, the idea is to integrate their corresponding deviation within the objective function, as described in equation (4.9).

$$\boldsymbol{min}\frac{\boldsymbol{1}}{\boldsymbol{2}}||\boldsymbol{w}||^{\boldsymbol{2}} + \boldsymbol{C}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}}|\boldsymbol{\xi}| \ , \tag{4.8}$$

under the following constraints:

$$|\boldsymbol{y_i} - \boldsymbol{w} \ ^{\circ}\boldsymbol{x_i}| \leq \boldsymbol{\varepsilon} + |\boldsymbol{\xi}| \ . \tag{4.9}$$

So now, there is an additional parameter *C* that can be tuned. As *C* increases, the tolerance for the points outside of $\boldsymbol{\varepsilon}$ also increases. On the contrary, as *C* approaches 0, the tolerance is consequently decreasing. This procedure makes it possible to obtain certain robustness with respect to outlier points.

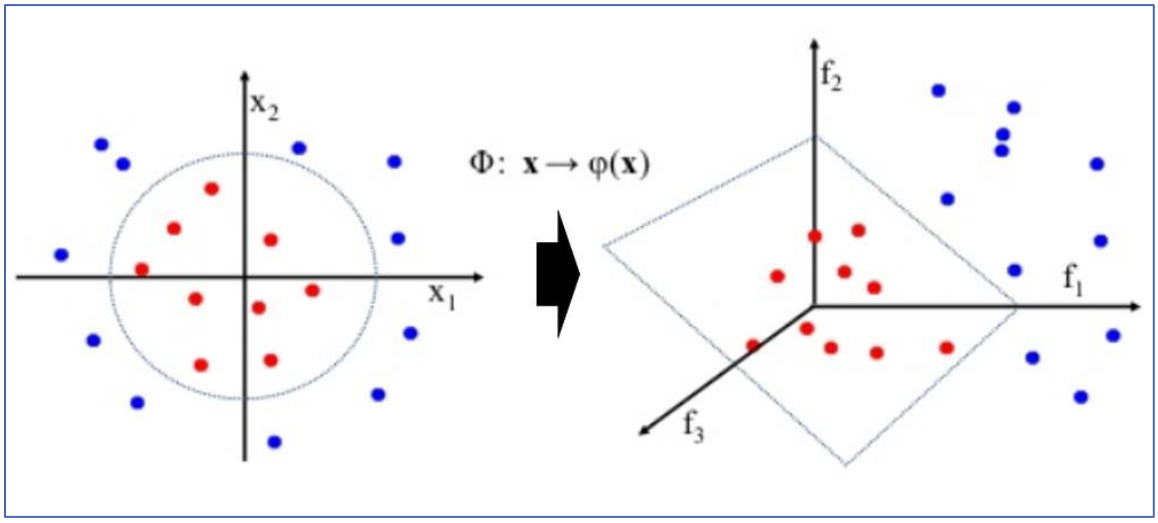


Figure 4-3. The kernel trick: SVR mapping into a higher dimensional feature space

A support vector machine constructs a hyper-plane or set of hyper-planes in a high or infinite-dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier. Figure 4-3illustrates the decision function. Here, the problem is not linearly separable in the original 2D space, but becomes separable when a third additional dimension is introduced.

Another fundamental aspect related to SVR is the so-called *kernel trick*. The kernel functions make it possible to transform the data into a higher dimensional feature space. In this way, problems that are not linearly separable in the original representation spaces become separable. This principle is also illustrated in Figure 4-3. Different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

A Kernel function transforms the training data so that a non-linear decision surface is transformed into a linear equation in a higher number of dimensions. Linear discriminant functions can provide very efficient 2-class classifiers, provided that the class features can be separated by a linear decision surface. For many domains, it is easier to separate the classes with a linear function if you can transform your feature data into space with a higher number of dimensions. One way to do this is to transform the features with a “kernel” function.

There are various choices of kernels that can be considered. Among the most popular, let us mention the following.

The linear kernel is used when the data is linearly separable. It is the simplest kernel that can be used and is defined as the scalar product between the two vectors :

$$\boldsymbol{k(x, x') =< x, x' > \ .} \tag{4.10}$$

The polynomial kernel represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables, allowing the learning of non-linear models. The polynomial kernel looks not only at the given features of input samples to determine their similarity but also at combinations of these. The polynomial kernel of degree *d* is defined as described in equation (4.11):

$$\boldsymbol{k(x, x') =< x, x' >^{d}} \quad . \tag{4.11}$$

The Radial Basis Functions (RBF) kernel is a popular kernel function widely used today. The RBF kernel on two samples ($\boldsymbol{x}$ ***and*** $\boldsymbol{x'}$) represented as feature vectors in some input space is defined as:

$$\boldsymbol{k(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right)} \quad , \tag{4.12}$$

where, $\boldsymbol{||x, -x'||^2}$ is the squared Euclidian distance between two feature vectors and σ is the dispersion parameter.

The SVMs algorithms are still heavily exploited today, with a multitude of applications where they are adopted and produce interesting results.

However, they have been surpassed in recent years by the emergence of deep learning techniques, briefly recalled in the following section, which led to a spectacular increase in classification/regression performances.

4.3.3. FNNs (*Feedforward Fully Connected Neural Networks*)

Historically, the so-called Feedforward Neural Networks are the oldest type of artificial neural networks that are designed to mimic the neural connections in the human brain. They are called Feedforward because the information propagates (travels) forward through the network. Basically, there are three main layers: input, hidden and output layers, as illustrated in Figure 4-4.

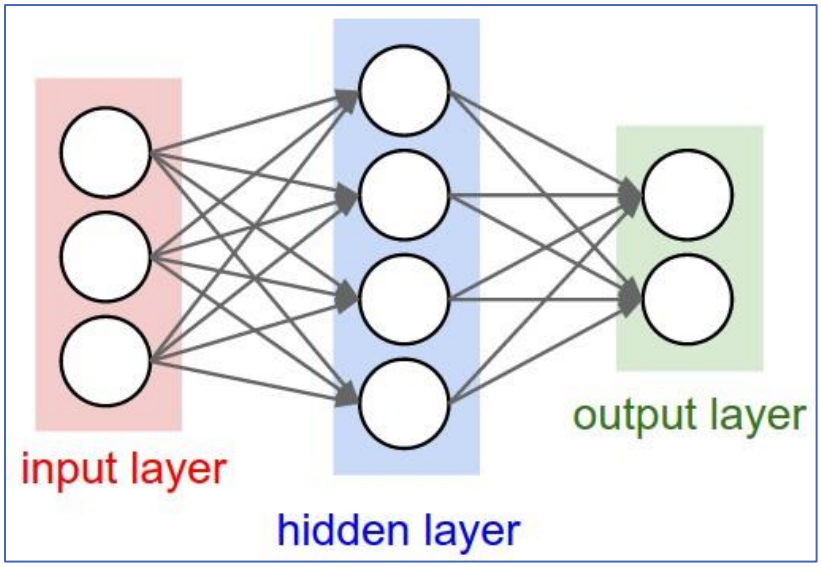


Figure 4-4. The architecture of a simple fully connected neural network, information moves from left to right.

The notion of Feedforward NN starts with SLP (Single-layer perceptron) and MLP (Multi-layer perceptron). The single-layer perceptron is the simplest form Feedforward neural network with no hidden units. In the case of a single output neuron, the network performs a binary classification task. Its output is thus computed directly as the scalar product between the set of weights added to a bias term, as described equation (4.13):

$$f(\boldsymbol{x}) = \begin{cases} \mathbf{1} \; \boldsymbol{if} \; \boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b} > \mathbf{0}, \\ \mathbf{0} \qquad \boldsymbol{otherwise} \end{cases}, \tag{4.13}$$

where the $\boldsymbol{w}$ is the weights vector, $\boldsymbol{x}$ is the input vector and $\boldsymbol{b}$ is the bias term.

The SLP model can thus be simply interpreted as a neural network-based formulation of the linear classification method.

On the other hand, MLP (*Multi-Layer Perceptron*) offers interesting capabilities for solving non-linear classification and regression problems. A MLP network introduces, between the input and the output layers a succession of intermediate layers, so-called hidden layers. Each neuron is governed by a nonlinear activation function that is applied to its output value. Figure 4-5 illustrates some of the popular activation functions that can be utilized, including sigmoid, tanh, ReLU, MAxout, LeakyReLU and ELU.

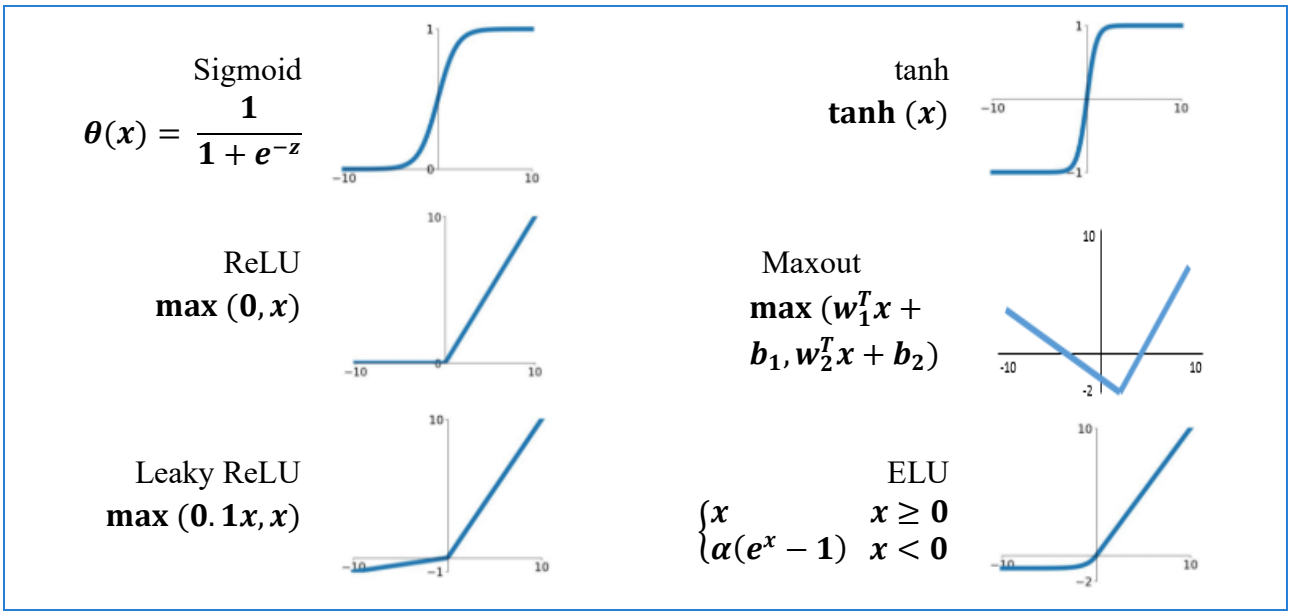


Figure 4-5. Deep learning activation function

Individual neurons in ANNs are only able to classify (solve) linearly separable data, however, combining neurons together, essentially means combining their decision boundaries.

Figure **4-6**). Therefore, ANNs that is composed of many neurons are able to learn complex, non-linear decision boundaries.

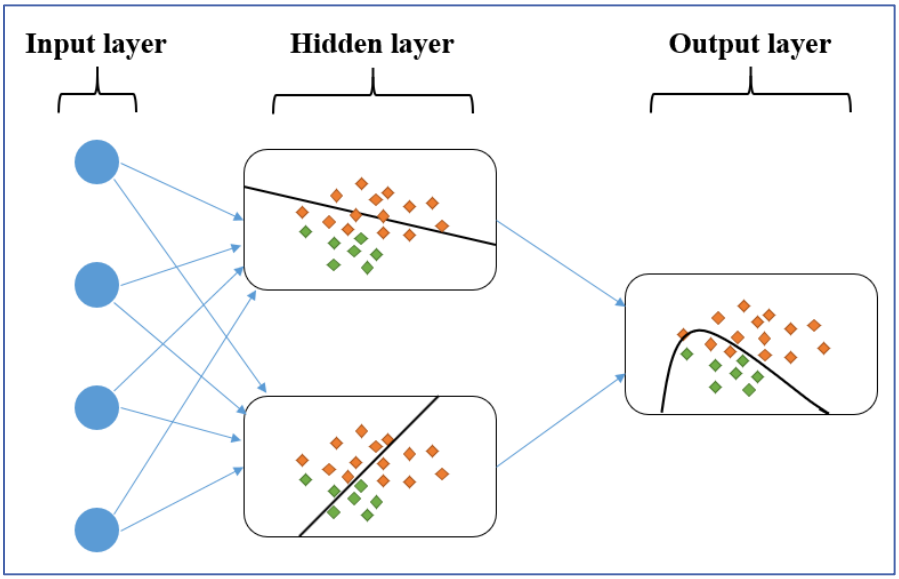


Figure 4-6. ANNS learning non-linear decision boundaries

A fully connected neural network connects all the possible combinations of neurons between successive layers. One important distinction is that the neurons in the same layer do not connect with each other. The number and the size of the hidden layers then define the network architecture.

ANNs typically starts with an input layer, containing the number of neurons usually equal to the dimensions of the input feature vectors that are characterizing the data. The second part is the succession of hidden layer, containing various numbers of neurons. Considering multiple hidden layers allows the network to learn more complex patterns, thus the network is considered as a deep neural network (Figure 4-7). The output layer is the final layer, containing the number of neurons equal to the number of classes or variables to be estimated (*i.e.*, the set of possible solutions).

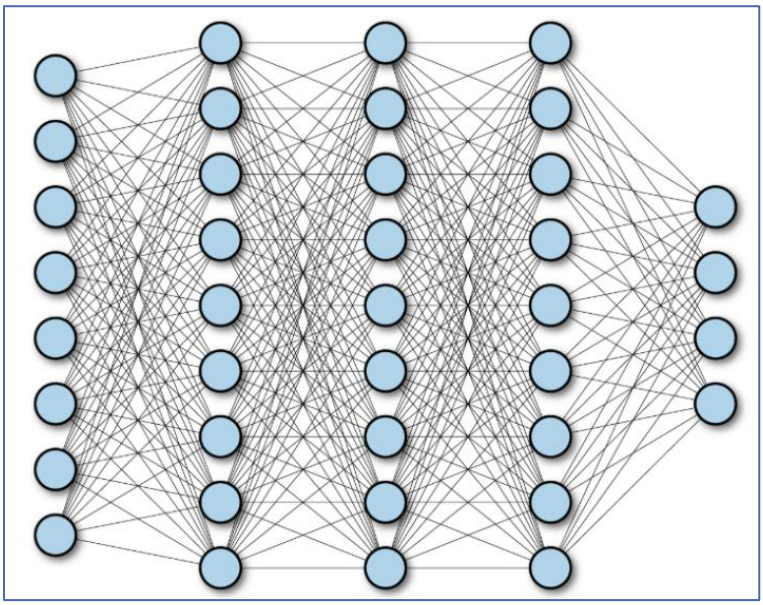


Figure 4-7. Multilayer (deep) fully connected neural network

Each connection between neurons is controlled by a weight parameter. The set of weights need to be learned and governs then the behavior of the whole network. There are various techniques that make it possible to learn the weights based on a supposed available learning data set. The objective is to minimize a loss function, measuring the error between the predictions performed by the neural network and the ground truth values from the learning data set. A typical example of a loss function is the L_2 distance, described in the following equation:

$$\boldsymbol{loss} = \left\| \boldsymbol{y}_{\_predict} - \boldsymbol{y}_{\_original} \right\|^2 , \tag{4.14}$$

The gradient descent is the most popular optimization technique for feedforward neural networks. The term "gradient" refers to the quantity change of output obtained from a neural network when the inputs change a little. Technically, it measures the updated weights concerning the change in error.

The gradient descent makes it possible to set up the so-called backpropagation technique, which makes it possible to iteratively update the network weights. The predicted value of the network is compared to the expected output, and an error is calculated using the loss function. This error is then propagated back within the whole network, one layer at a time, and the weights are updated according to the value that they contributed to the error. This clever bit of math is called a backpropagation algorithm. The process is repeated for all of the examples in the training data. One round of updating the network for the entire training dataset is called an *epoch*. A network may be trained for tens, hundreds or many thousands of epochs.

The batch size is a term used in deep learning and refers to the number of training examples utilized in one iteration. The batch size can be one of three options: batch mode:

- where the batch size is equal to the total dataset thus making the iteration and epoch values equivalent,
- mini-batch mode: where the batch size is greater than one but less than the total dataset size. (usually, a number that is a devisor of the total dataset size is used), and
- stochastic mode: where the batch size is equal to one. In this case, the gradient and the neural network parameters are updated after each sample.

As any gradient descent-based techniques, such optimization approaches suffer from the possibility of geting stucked in local minima. In order to avoid this phenomenon, more advanced optimization techniques have been proposed. They include the stochastic gradient descent [131] or the ADAM optimizer [132], which are today extensively used for learning the network parameters.

4.3.4. CNN (Convolutional Neural Networks)

Convolutional Neural Networks (ConvNet / CNN) are a specific type of neural network architectures that are adapted for image-like structures. The specificity of the images comes from the high dimension of the data. It would be inconceivable to build fully connected neural networks for such data since this would require billions of neural connections and thus parameters that need to be learned.

In order to deal with such complexity-related issues, the principle consists of replacing the fully-connected layers by a set of convolution operators that govern the interactions between successive layers in the network. Each convolution operator is defined by a filtering kernel function, that has to be learned, and that replaces the weights encountered in the case of fully connected networks. The number of convolutional filters used at a given layer defines the depth of the network at the considered layer.

Additionally, pooling operators can be used in order to lower the dimensionality of the data. Such pooling operators correspond to sub-sampling processes and allow also to achieve invariance with respect to different forms of variability in the input data.

An example of CNN architecture is illustrated in Figure 4-8.

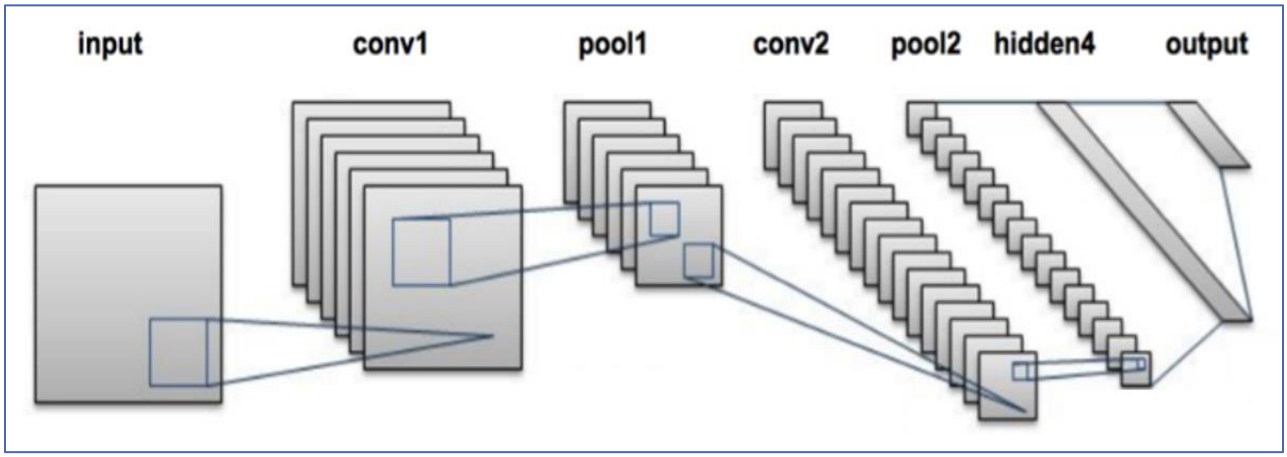


Figure 4-8. Example of a Convolutional Neural Network

Most often, images pixels are represented with three depth color channels (RGB). So, an image can be represented as a matrix of size (H×W×3), which is (height ×width ×depth). The convolutional layer makes use of a set of learnable filters. In the original image (input), a filter is used to detect the presence of specified features or patterns, usually presented with matrix (N×N×3) with smaller dimensions but the same depth (Figure 4-9).

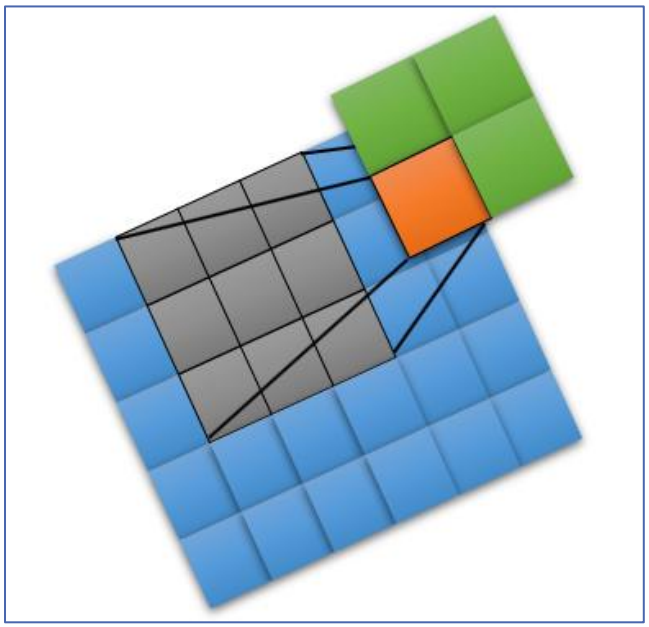


Figure 4-9. CNN with sliding filter

The convolution filter is slided across the image, and a dot product is computed to yield an activation map. There may be different filters convoluted which detects different features on the input image and a set of activation maps are outputted.

The pooling layers are generally inserted between the convolution layers in a CNN architecture. Such layers basically reduce the number of parameters and computation in the network, controlling overfitting by progressively reducing the spatial size of the network.

4.3.5. RNNs and LSTMs (Recurrent Neural Networks and Long Short Term Memory Neural Networks)

Recurrent Neural Network (RNNs) represent a family of ANNs specifically designed to recognize the data sequential characteristics. They use the output from the previous step to fed as an input to the current step, as illustrated in Figure 4-10(a). This feature makes them algorithms with internal memory.

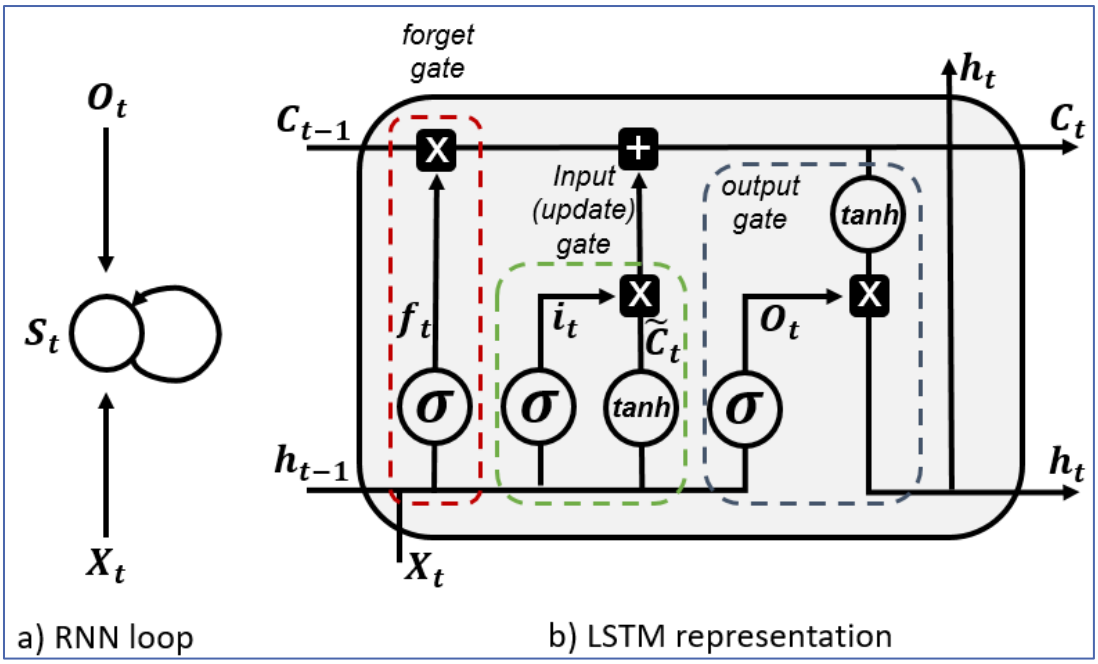


Figure 4-10. RNN loop (a); graphical representation of LSTM (b)

On the other hand, Long Short Term Memory (LSTM), like the name suggests is a special kind of recurrent network, capable of capturing long-term dependencies in the data. First introduced in 1997 by Hochreiter and Schmidhuber [133], the LSTMs have been recently popularized and refined. They

were designed specifically to avoid long-term dependency problems, by remembering the information for a longer period of time.

Similarly to RNNs, the LSTMs also have a chain-like structure, but the repeating model has a different structure, as illustrated in Figure 4-10(b). Here, four neural network layers are interacting in a specific way.

The equations that govern the forward pass of an LSTM are the following:

$$\begin{aligned} \boldsymbol{f_t} &= \boldsymbol{\sigma}(\boldsymbol{W_f} \cdot [\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + \boldsymbol{b_f}) \\ \boldsymbol{i_t} &= \boldsymbol{\sigma}(\boldsymbol{W_i} \cdot [\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + \boldsymbol{b_i}) \\ \boldsymbol{O_t} &= \boldsymbol{\sigma}(\boldsymbol{W_O} \cdot [\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + \boldsymbol{b_O}) \\ \widetilde{\boldsymbol{C}}_{\boldsymbol{t}} &= \boldsymbol{tanh}(\boldsymbol{W_C} \cdot [\boldsymbol{h_{t-1}}, \boldsymbol{x_t}] + \boldsymbol{b_C}) \\ \boldsymbol{C_t} &= \boldsymbol{f_t} * \boldsymbol{C_{t-1}} + \boldsymbol{i_t} * \widetilde{\boldsymbol{C}}_{\boldsymbol{t}}) \\ \boldsymbol{h_t} &= \boldsymbol{O_t} * \mathbf{tanh}\,(\boldsymbol{C_t}) \ , \end{aligned} \tag{4.15}$$

where:

- $\boldsymbol{x_t}$: input vector to LSTM unit
- $\boldsymbol{C_t}$: cell state vector
- $\boldsymbol{h_t}$: hidden state vector is also known as output vector of the LSTM unit
- $\boldsymbol{f_t}$: forget gate activation vector
- $\boldsymbol{i_t}$: input gate activation vector
- $\boldsymbol{O_t}$: output gate activation vector
- $\boldsymbol{W}$: weight matrix
- $\boldsymbol{b}$: bias vector

The core concepts behind the LSTMs concern the cell states and their various gates. In theory, the cell state can carry relevant information through the entire processing of the sequence (Figure 4-10(b)). One LSTM cell consists of four gates as follows:

- Forget gate: after getting the input from the previous state h_(t-1) takes the decision of what must be removed or forget, thus keeping only the relevant information.
- Input gate: add new information from the input to the present cell.
- Update (g) gate: tanh layerthat creates a vector for a new candidate C̃_t.
- Output gate, provides the output from the cell state.

4.3.6. ML accuracy parameters

Machine learning algorithms have been implemented and exploited, with both classical and deep learning approaches. In order to measure the accuracy of the considered algorithms, we have considered, traditional metrics, including MAE and MSE scores:

- MAE (*Mean Absolute Error*) – measures the average magnitude of the errors in a set of predictions, without considering their direction. The MAE is defined as the average over the test sample of the absolute differences between prediction and actual observation, where all individual differences have equal weight. The MAE score is defined as described in equation (4.16).

$$\boldsymbol{MAE} = \frac{\mathbf{1}}{\boldsymbol{n}}\sum_{\boldsymbol{j=1}}^{\boldsymbol{n}} |\boldsymbol{y_j} - \boldsymbol{\hat{y}_j}| \ , \tag{4.16}$$

- MSE (*Mean Square Error*) – measures how close a regression line is with respect to a set of points, by computing the distances from the points to the regression line. It also gives more weight to larger differences. The MSE is expressed as described in equation (4.17).

$$\boldsymbol{MSE} = \frac{\mathbf{1}}{\boldsymbol{n}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} (\boldsymbol{y_i} - \boldsymbol{\hat{y}_j})^{\mathbf{2}} \ , \tag{4.17}$$

For learning the neural network considered within our work, we have applied the SGD (*Stochastic Gradient Descent*) optimizer [131], an iterative method for optimizing an objective (loss) function with suitable smoothness properties.

4.4. Discussion

The analysis of the state of the art shows a multitude of techniques and algorithms used for traffic prediction and public transportation in particular. Many classical approaches like Linear Regression, SVMs, Random Walk, Random Forest, Kalman filtering, ARIMA models, are still widely used today. The emerging computational power in recent years allowed the exploitation of artificial neural networks.

The deeper analysis presented in the SoA has shown that the neural networks approach can significantly increase the prediction accuracy over the other forecasting techniques, in most cases. One of the reasons is their capacity for designing highly complex models with non-linear dependencies. Another very important reason is the ability to process and learn from incomplete and redundant data.

Today the acceptance of ANNs in commercial companies is still not optimal, at least not in the level where the classical algorithms are fully replaced. A major reason for that is the inability to

mathematically describe the background operation of the model and clearly describe the function of each individual variable. Thus, the performances are strongly dependent on the available data set used for the learning stage.

One other barrier can be related to hardware requirements and associated maintenance/cost since ANNs machine learning algorithms are notably designed to run on high-performance GPU units.

The next chapter will present the prediction approach proposed, which is based on the concept of TDM (*Traffic Density Matrix*).

Chapter 5. TDM (*Traffic Density Matrix*) – prediction approach

Abstract

This section introduces the proposed prediction approach, which is based on a novel concept, so-called Traffic Density Matrix (TDM). The TDM gathers the traffic information in a given urban area that can be defined by the user. We notably show the necessity of disposing of such data for successful, both short-term and long-term prediction objectives, by demonstrating that a global prediction approach cannot be a feasible solution. Several different prediction approaches are then introduced, described and experimentally tested on various simulation scenarios. They include traditional machine learning techniques, such as linear regression and support vector machines, but also more advanced, highly non-linear neural network-based approaches.

Within this context, various network architectures are retained and evaluated, including fully connected networks, convolutional neural networks, recurrent and LSTM networks.

The experimental evaluation is carried out under two types of differents scenarios, corresponding to both long term (ODM–Operator Data Model) and short-term (CDM–Client Data Model) predictions. They show that increasing the degree of non-linearity of the predictors is highly benefactory for the accuracy of the obtained predictions. They also show that significant improvement can be achived over state of the art techniques. In the case of long-term prediction, the FNN method perfoms the best when compared with the baseline technique OLS with a significant increase in accuracy (more then 66%). For short-term prediction, the FNN method was also the best performer compared with the baseline OLS (with more then 15% increase in accuracy).

5.1. Introduction

Public transportation and mobility in urban areas is a subject that receives considerable attention from both academic and industrial research. Public transport can be defined as a system of vehicles such as buses and trains that operate at a regular time in fixed pre-determined routes, used by the general public and governed by the government or private sector.

The main objective of the proposed approaches is to improve the public transportation system, making it more accurate, reliable and convenient. An accurate public transportation system is by definition a system that provides valid, correct information about the real situation with the help of some predictive techniques. In order to achieve such a goal, various techniques and algorithms are considered. The proposed methodology consists of several stages. It begins with the creation of real-life simulation scenarios. The second stage concerns mining and extraction of useful and information-rich data, that is further processed and converted under the form of the TDM structure. Then, we detail the various machine learning algorithms retained and their adaptation to our objectives. An extensive experimental study is finally described.

Let us first elaborate on the simulation scenario considered.

5.2. Simulation scenario

The simulation scenario was developed and computed with the help of the SUMO traffic simulation framework described in Chapter 2. Let us further describe the geographical area considered as well as the corresponding simulation parameters.

5.2.1. Geographical area considered

In order to set up and evaluate the predictive algorithms in a real-life scenario, the scenario needs to be carefully planned and executed. Our initial goal was to create a scenario that is complex, close to real-life situations, but not excessively computationally heavy, and still leave space for future extension and improvement.

This work has been carried out within the framework of the ETS project (*Electronic Ticketing System*), a French national project that aims to improve the public transportation system. One of the partners in the project (ZenBus), cooperates with the bus network operators in the city of Nantes, France. For this reason, we have taken the decision of developing a scenario in the city of Nantes, which may have potential benefits for both parties, and can be validated by the industrial partners.

More precisely, two bus lines from the bus network of the city of Nantes have been considered. The bus lines retained are numbered by 79 and 89, and lead to 4 different itineraries in total (both directions for each line), with 25 and 36 bus stops, respectively. The real bus stop locations have been recovered from the TAN (*Transports de l'Agglomération Nantaise*) [134] public traffic provider in Nantes and with the help of the OSM (*Open Street Map*) transport editor [135]. The retained bus lines 79 and 89 are illustrated in Figure 5-1.

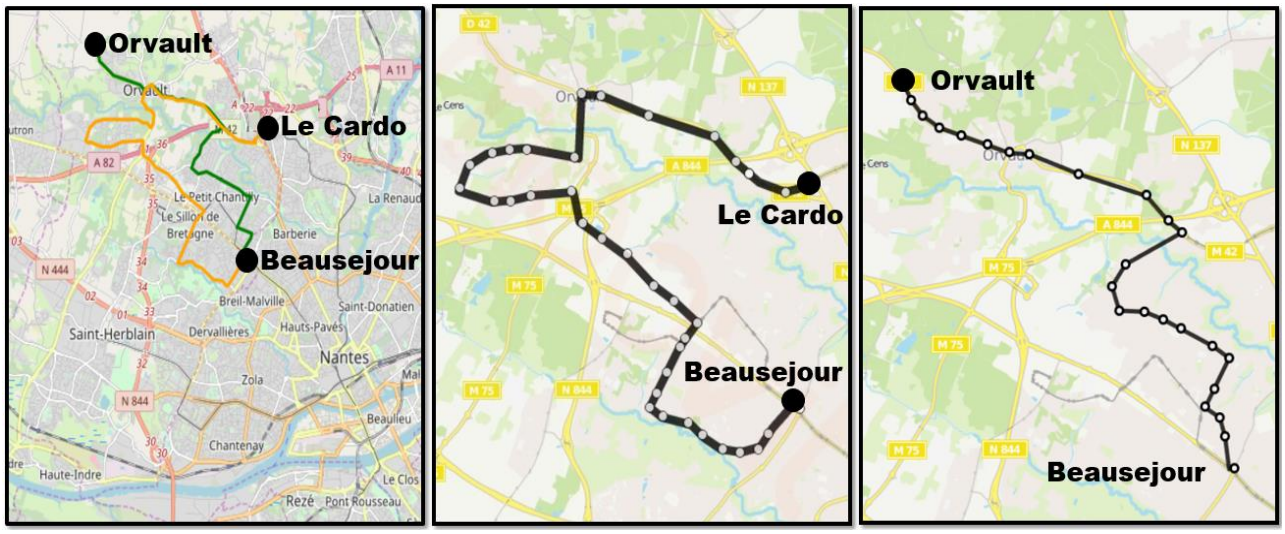


Figure 5-1. A geographical area in the city of Nantes, France showing bus lines 79 and 89, within the same geographical region (left). Bus line 89 (middle), bus line 79 (right)

Figure 5-1 depicts the two bus lines (79 and 89), which are located in the same geographical area. This choice allows us to create a simulation in such a way that the bus lines are computed in a single simulation instead of two different ones. So, from the start, the simulation time and computational effort are divided by two.

5.2.2. Simulation parameters

The simulation scenario has been carefully prepared and executed by the SUMO traffic simulator. We would like to underline that this process was highly time-consuming and retained our attention for about 6 months during which various trials have been performed. The time also includes data export and data transformation in suitable formats. The main challenge concerns the generation of large enough training and test data set, that can be effectively exploited for machine learning purposes. The main bottleneck in terms of computational burden is related to the SUMO simulator since a single simulation requires between 500 and 1300 seconds (approximately 8,5 and 21,5 minutes).

The SUMO simulation does not support GPU integration into the programming level and parallelization between the simulation is impossible, so the whole computational burden falls on the CPU. And since this is a real-life simulation, the CPU resources were utilized to maximum most of the time. On the other hand, the verification process of successfully computed simulations, which contains appropriate data is also time-consuming, since the whole simulation process needs to be rechecked regularly (continuously) in order to avoid possible errors or miss-simulations that may occur.

Figure 5-3 illustrates the whole cycle of the simulation scenario constructed, with global parameters involved (Table 5-1), SUMO simulator, and resulting output data.

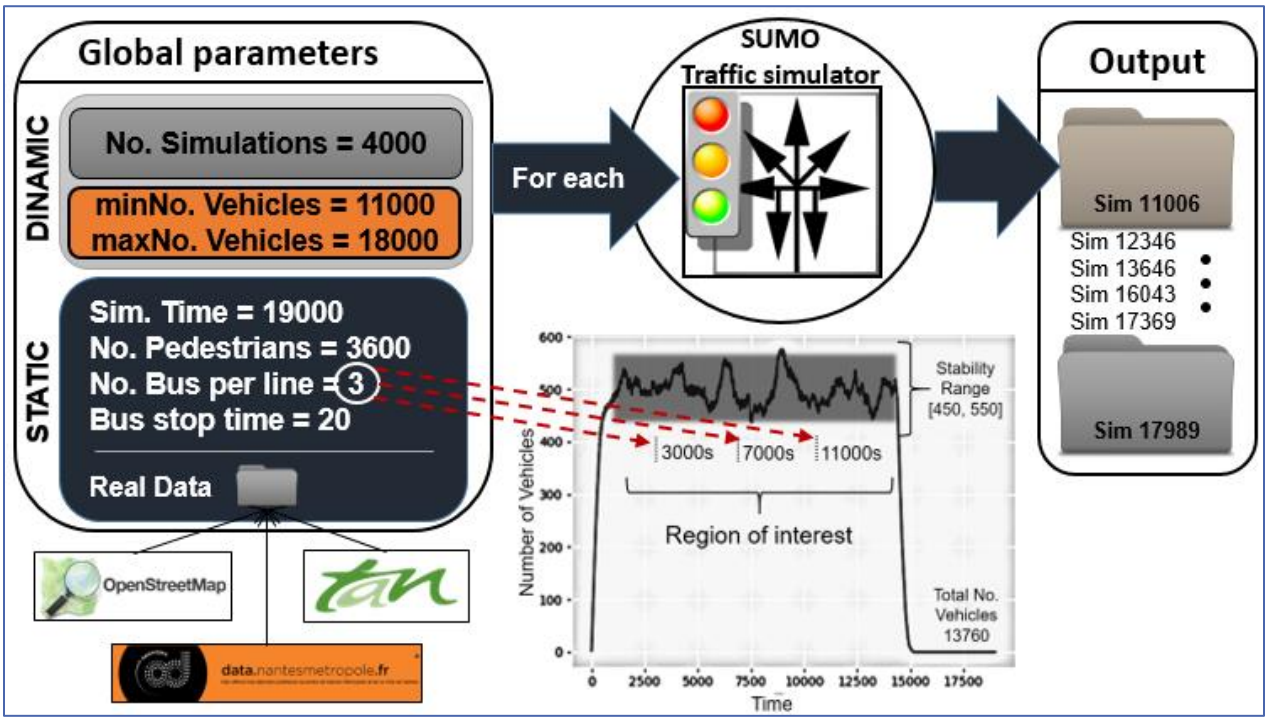


Figure 5-2. The full cycle of the simulation scenario and data aggregation

Let us note that in order to successfully compute the simulation with all of the instances certain parameters need to be planned in advance.

TABLE 5-1. SIMULATION PARAMETERS

No. simulations	No. Vehicles	No. Pedestrians	Bus itineraries	Time (s)
4000	[11000 - 18000]	3600	4	19000

More precisely, the following parameters have been considered:

- The 2D map of the considered region in the city of Nantes, imported from OSM (*Open Street Map*), was converted to the SUMO format.
- The total number of simulations performed was set to 4000.
- Each simulation performs 3 bus runs per bus itinerary in different time slots, as illustrated in Figure 5-3. Then, each bus run per simulation is considered as a separate run. This leads to a total number of 4000 simulations ×3 buses = 12000 bus runs.
- The simulation is controlled by a global macro-parameter, which is the total number of vehicles inserted into the system (*i.e.*, the whole number of both public and private vehicles that are present in the city). In order to simulate various traffic conditions, this parameter ranges within the [11000, 18000] interval with a random sample step (between 1 and 10) of vehicles.
- Each vehicle itinerary is calculated using the shortest path algorithm of Dijkstra [35], presented with the O/D (Origin / Destination) matrix.
- The total number of pedestrians was set for all simulations to 3600. Let us note that it is important to use pedestrians in the simulation due to the impact of pedestrian traffic lights. This makes it possible to increase the degree of realism of the entire simulation.
- SUMO does not support direct import of public transportation from an external source, so bus itineraries and the bus stops have been manually added to the simulation, following the process described in Chapter 2Section 2.2.6.
- The bus waiting time at each bus stop station was fixed to 20 seconds.
- The time for each bus run was limited to 4000 seconds since after extensive measurement it was concluded that 99% of all bus runs complete the itinerary under 4000 seconds.
- The position of each bus was sampled every 10 seconds,
- The total simulation time was set to 19000 seconds, which ensures that all the vehicles will get in and out of the system, as illustrated in Figure 5-3.

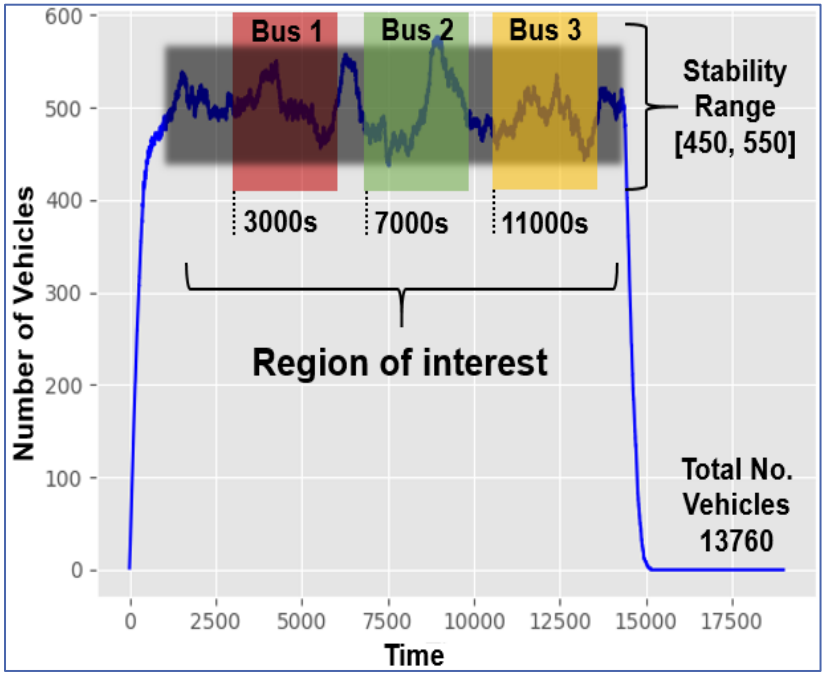


Figure 5-3. Simulation, bus runs and stability range (region of interest)

Please note that the simulation time considered ensures that the simulations are performed within a certain stability range, where the number of total vehicles in the system is relatively stable. This aspect is illustrated in Figure 5-4, where the actual, real number of vehicles that are present in the system over time is presented. We can observe that, in the beginning, the number of vehicles is gradually increasing, the vehicles being successively inserted within the system. On the contrary, at the end of the simulation interval, the number of vehicles is rapidly decreasing, since the vehicles that are going out of the system are not replaced by new ones. This behavior is due to the intrinsic functioning mode of the SUMO simulator. A stability range, with a relatively constant number of vehicles that are present in the system, is achieved within the time interval [2500 - 15000] seconds. For this reason, for a given simulation, 3 different buses are launched for each of the 4 itineraries at the following starting times: 3000, 7000 and 11000 seconds. This makes it possible to obtain consistent simulations.

Once the simulation data available, we have started to investigate the prediction techniques. In the first stage, we have conducted a relatively simple experiment, concerning a global prediction approach. The question that we have addressed here is the following: given the total number of vehicles inserted in the system, would it be possible to predict, for each bus, the time of arrival at the terminus station (*i.e.*, time of completion of the itinerary)? This experiment is described in the following section.

5.3. Global approach

In order to simplify the prediction, we have investigated the possibility to perform a global prediction of the bus arrival time at the terminus station (which corresponds to the time of completion of the itinerary), given the total number of vehicles inserted within the system. *A priori*, it is a reasonable assumption that knowing the total number of vehicles in the system we can successfully predict the bus arrival time at the final destination.

In order to investigate the validity of this assumption, we have studied the correlation between the time of completion of a given itinerary and the total number of vehicles inserted in the system. The results obtained, presented in Figure 5-4, are quite surprising.

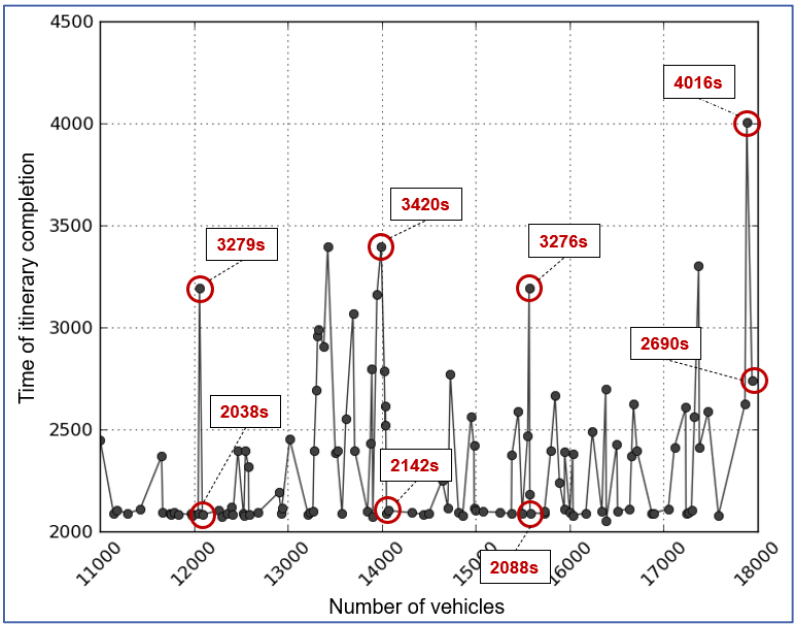


Figure 5-4. Total number of vehicles over the time of itinerary completion

The itinerary completion times are varying in a chaotic manner, and no correlation between the total number of vehicles within the system and the time of completion of the bus itinerary can be established.

We can observe that there are huge disparities from one simulation to another even though the number of vehicles is increased by small units. Moreover, in some cases, the time of completion of the itinerary is inferior when a significantly superior number of vehicles is present in the system. This phenomenon can be explained by the fact that a certain number of singularities can appear. They correspond to traffic jams occurring locally in some parts of the region under analysis. Such singularities are completely out of control, and they depend more on the initial, random distribution

of the vehicles instead of the global number of vehicles parameter. This situation is illustrated in Figure 5-5. Here, the same round-about is fluid when a total number of 15450 vehicles (Figure 5-5/A) is present in the system, but completely saturated for a lower number of 11990 vehicles (Figure 5-5/B).

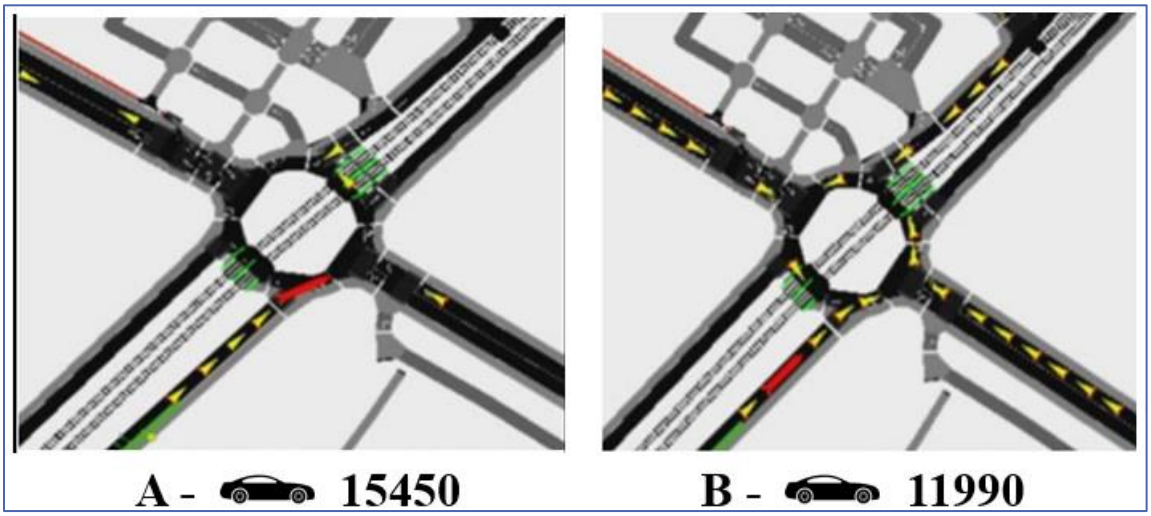


Figure 5-5. Simulation screen-shoot, same round-about different number of vehicles

This initial analysis shows that attempting to perform a global prediction of the time of completion of the itinerary, based on the global number of vehicles is not possible. Let us mention that our partners from the ETS project have confirmed that this situation, which in our case occurred in a computer-simulated scenario, often arrives in practice, in real-life conditions, making thus the predictions highly difficult.

Considering the initial simulation results and this irregular traffic behavior it is clear that a more elaborate and granular solution is necessary in order to be able to perform relevant predictions. The analysis of the results also confirms that no reliable prediction can be performed when solely a global parameter is taken into account. This is mainly due to some singular events, which mainly concern localized traffic jams hazardously occurring in some given points. It is then necessary to consider a finer level of granularity, in order to characterize the state of the system in a more local and reliable manner. The next section describes in greater detail the proposed TDM (*Traffic Density Matrix*) solution.

5.4. TDM (Traffic Density Matrix)

In order to overcome the problem of global traffic prediction, a new solution is needed. This solution should consider a more granular approach, which can offer a localized representation of the traffic over time.

Our objective is to improve the performance of the public transportation system – bus arrival time prediction at each bus stop while taking to account the local traffic situation of the bus itinerary. Doing so will require to compute the whole dataset and extract valuable information about the traffic situation of the whole bus itinerary. Each simulation is used to create this specific, image-like TDM structure that can be utilized for prediction purposes.

The Traffic Density Matrix is illustrated in Figure 5-6.

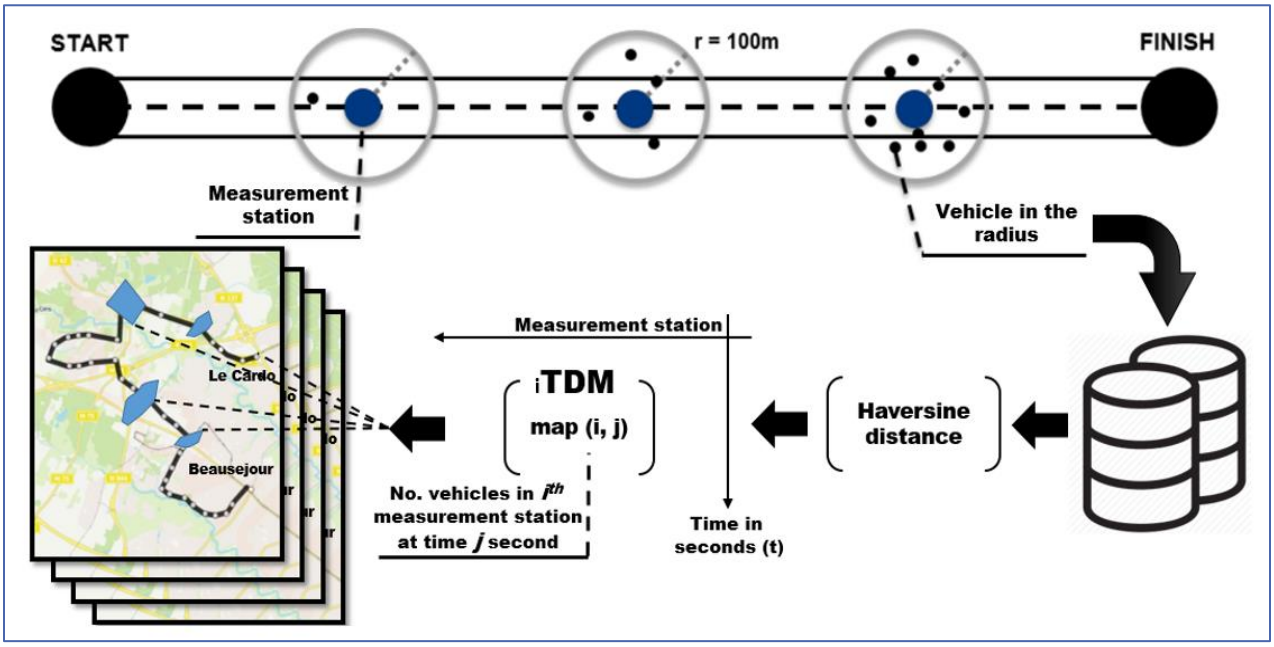


Figure 5-6. TDM (*Traffic Density Matrix*)

Its main function is to successfully represent the traffic situation at each measurement station. Measurement stations are points in space that detect and count how many vehicles are passing at a certain moment of time. In real-life scenarios these sensors are most of the time the loop detectors (magnetic counters), but other sensors can perform a similar task, like a camera for vehicle counting. Of course, social data form GPS devices can also be used to achieve this purpose. For the convenience of the whole simulation, each bus stop location was considered as a measurement station. In this particular scenario, the detection radius around the considered measurement station was set to 100 meters. This radius represents a "reasonable" guess, since the physical distance between the bus stop

stations is generally between 400 and 1000 meters. This representation can be interpreted as a traffic density matrix evolving over time, which is traversed by the considered buses.

For each bus generated at each simulation, we create in this way an individual traffic density matrix per simulation, denoted by iTDM, of size (M_{stations} x T), where M_{stations} is the number of measurement stations and T is the number of time instances measured. In order to simplify and speed up the process, we have evenly sampled the simulation interval with a step of 10 seconds. This helps reducing the amount of data and consequently the related storage/computational requirements.

As the bus stops and vehicle coordinates are expressed in SUMO in terms of GPS coordinates (latitude and longitude), the Haversine distance [136] between two points has been computed in order to establish the radius of interest. The Haversine distance is expressed as described in equation (5.1):

$$\begin{gathered} a = \boldsymbol{sin^2}\left(\frac{\Delta\boldsymbol{\varphi}}{\mathbf{2}}\right) + \mathbf{cos}\,\boldsymbol{\varphi_1} \cdot \boldsymbol{cos\varphi_2} \cdot \boldsymbol{sin^2}\left(\frac{\Delta\boldsymbol{\lambda}}{\mathbf{2}}\right) \\ \boldsymbol{c} = \mathbf{2} \cdot \boldsymbol{arctan}(\sqrt{\boldsymbol{a}}, \sqrt{(\mathbf{1}-\boldsymbol{a})}) \\ \boldsymbol{d} = \boldsymbol{R} \cdot \boldsymbol{c}\ , \end{gathered} \tag{5.1}$$

By definition, the Haversine formula determines the geodesic distance between two points on a sphere, given their longitude and latitude.

The simulation yields the following two outputs, controlled by a global parameter which is the number of vehicles injected within the system:

- A vector $a = (a_1, a_2, \ldots a_{\text{Mstations}})$ of size M_{stations} storing the bus arrival times in all the stations of the itinerary,
- The density matrix *iTDM*, of size ($M_{\text{stations}} \times T$),

Here, M_{stations} is equal to 25 and 36 for buses 79 and 89 respectively, while T is the number of measured time instances. Since each simulation lasts 4000 seconds, and the measures are performed every 10 seconds, the parameter $T = 400$.

A final, global 3D matrix is then constructed, with shape ($M_{\text{stations}} \times T \times S_{total}$), where S_{total} is the total number of simulations (*cf.* Section 5.2.2).

In order to compute and successfully predict the bus arrival time at each bus stop station, a suitable data model needs to be developed. The next section discusses the preparation and development of the data model (input/output), as well as the data normalization processes involved.

5.5. Data models and data normalization

The data obtained from the simulation process needs to be prepared in a suitable format called the data model. The data containing the traffic information under the form of the TDM matrix, also called input data (in the machine learning process) was processed and stored in either 2D (per individual simulation – *iTDM* matrix) or 3D (for all simulations – *TDM* matrix) data structures, as discussed in the previous section.

The bus arrival time at each bus stop station (which represents the output data) was stored either as a 1D vector or as a scalar value. It contains the timestamps as values of each bus arrived at the bus stop stations.

All the data has been structured and stored in .csv format, for practical use (easy to read and write) and for speeding up the data reading process. Furthermore, in order to reduce the data size and the corresponding memory requirements, all the data was compressed with the MsgPack [137] standard library.

The next step is of crucial importance for the learning stage. It involves the data split and the cross-validation, which were achieved as follows:

- Total number of simulations 12000.
- Train dataset - 80 % (9600 simulations).
- Test dataset – 20 %, (2400 simulations), from witch: 10 % (240) were used for validation, and 90 % (2160) for evaluation purposes.
- A 10-fold cross-validation has been used for dealing with the randomness of the data.

5.5.1. Data normalization

The data normalization procedure is performed as described in the following equation:

$$\boldsymbol{\gamma_{normalized}} = \frac{\boldsymbol{\gamma - \mu}}{\boldsymbol{\sigma}} \ , \tag{5.2}$$

where:

- γ: is the data considered (in our case, representing both the density map *TDM* and the arrival time vector *a*),
- $\boldsymbol{\mu}$: is the mean of (γ).
- $\boldsymbol{\sigma}$: is the standard deviation of (γ).

The parameters $\boldsymbol{\mu}$ and γ are computed on the training dataset.

The data normalization procedure has two benefits. On the first hand, in many cases, it makes it possible to speed up the convergence rate of the training procedure. On the other hand, in the case of neural networks-based approaches, it allows increasing the number of neurons in the hidden layers.

For this reason, the data normalization procedure will be used in all of the machine learning techniques considered.

Let us now detail the two data models considered in our experiments.

5.5.2. Operator Data Model (ODM)

The operator data model (ODM), is structured in a manner that attempts to maximize the input information in the learning process. The ODM data model uses the Traffic Density Matrix as a whole input, without removing or compressing any information.

Thus, the ODM contains the TDM information from all 12000 simulations under the form of a 3D matrix, illustrated in Figure 5-7. The horizontal axis presents the measurement stations ($M_{stations}$) that are used to measure the traffic density on the itinerary. The vertical axis is the temporal one and presents the time in seconds (every 10 seconds). The last axis represents the number of simulations, which in our case was set to 12000. Basically, the TDM stores an integer number for each measurement station at a certain period of time. Each individual simulation is stored in an image-like structure called iTDM. The So-called iTDM contains the spatiotemporal data ($M_{stations} \times T$) obtained per individual simulation.

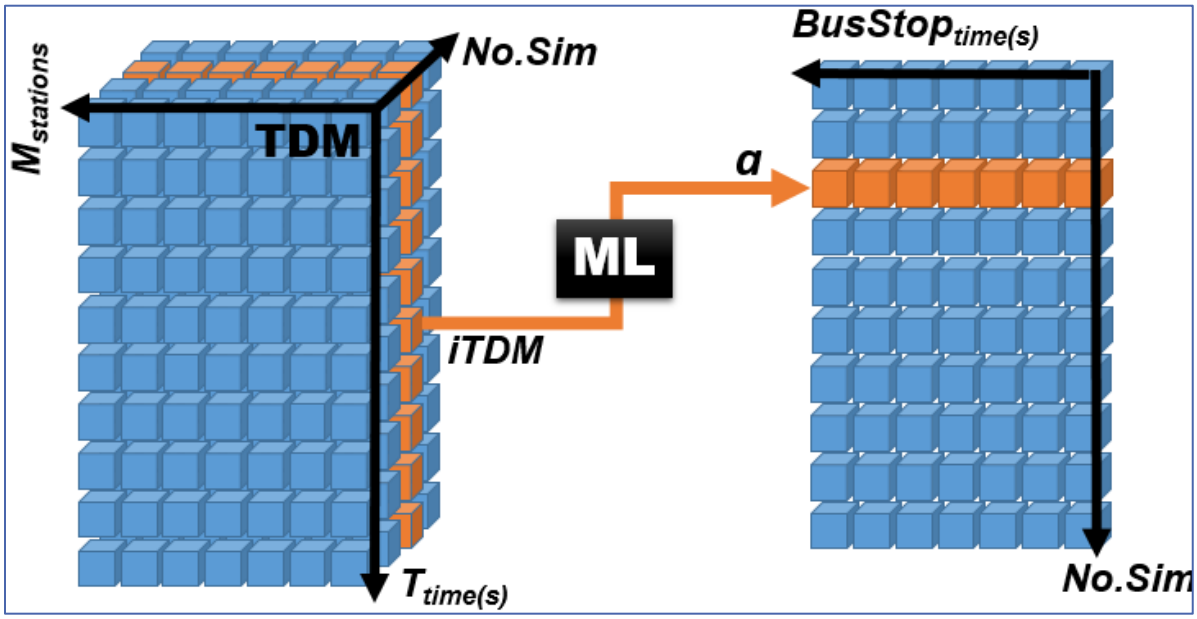


Figure 5-7. Operator Data Model (ODM)

The objective is then to predict the arrival vector *a*, given as input the iTDM. The input data is a matrix of size ($M_{stations} \times T$), and the output is a vector of size $1 \times M_{stations}$ (with $M_{stations}$ being equal to

25 and 36, for buses 79 and 89, respectively) containing the bus arrival times at all the bus stop stations.

5.5.3. Client Data Model (CDM)

The client data model (CDM) concerns the prediction of the bus arrival time at a particular station, based on the current position of the bus over the itinerary and taking into account the traffic situation of the whole itinerary, as illustrated in Figure 5-8.

In order to develop this particular data model, some additional data processing and development must take place.

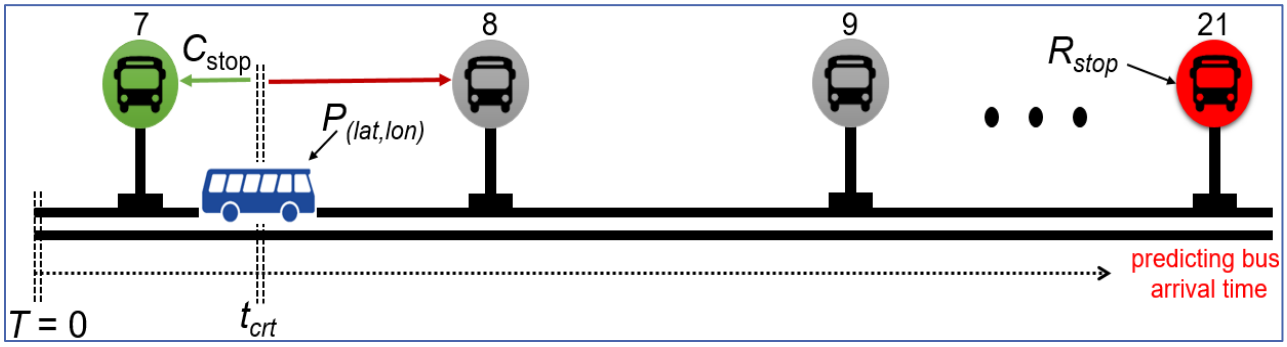


Figure 5-8. Predicting bus arrival time at the desired station

More precisely, for this particular predictive scenario, data modeling has been created in the following way:

- 5 different bus stop stations have been selected, evenly sampling the itinerary, and denoted by R_{stop}. In the case of the bus line 89, the bus stops no 11, 16, 21, 26 and 31 are considered for predicting the corresponding bus arrival times, so $R_{stop} \in \{\mathbf{11, 16, 21, 26, 31}\}$).
- For each selected bus stop, 10 different bus runs are extracted from the TDM. This leads to a total number of **12000 (simulations)** $\times$ **(**$\boldsymbol{R_{stops} = 5}$**)** $\times$ **10 (runs) = 600000 inputs**.

The first part of the CDM data model concerns the vectors generated from the TDM. The bus runs are stored in the so-called called **Input vector**, illustrated in Figure 5-9. A second value is a scalar one, called R_{stop}, which corresponds to the current bus stop station (*i.e.*, the bus station where the user requested the information for). The last part is also a scalar value denoted by C_{stop} (closest bus stop). This station is defined as the bus stop station that is the closest to the current bus position at the given, requested time t_{crt}.

During the construction of the CDM data model, we make sure to retain only valid trials, *i.e.*, values for which in the simulations, at the current time, there exists a bus prior to the current bus stop location. Basically, we make sure to retain valid runs for which the C_{stop} (current bus position) is anterior to the considered bus station where the prediction is required (R_{stop}).

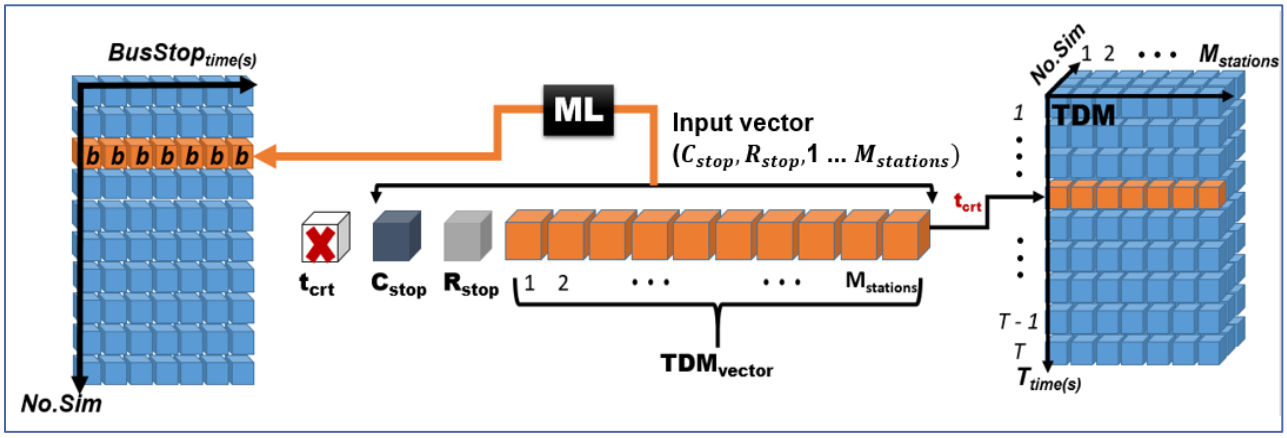


Figure 5-9. Client Data Model (CDM)

To summarize, the input vector (Figure 5-9) includes three different variables, including the TDM_{vector} (which includes the real traffic information over the whole itinerary), the C_{stop} for the current time (t_{crt}), and the R_{stop} (the bus stop station used to predict the bus arrival time). This input vector is used for predicting the bus arrival time at the requested station. The output value is a scalar, representing the expected arrival time, denoted by *b*, in seconds.

5.6. Retained machine learning algorithms

The machine learning algorithms adopted in our work are summarized in Table 5-2, for each of the operator and client data models considered. Two machine learning frameworks have been considered, namely Scikit-learn [138] and PyTorch [139]. The coding environment was chosen to be the Python programing language since both libraries are based on it.

TABLE 5-2. MACHINE LEARNING ALGORITHMS AND THEIR IMPLEMENTATION FRAMEWORKS

Data Model 1 (ODM)	Data Model 2 (CDM)	Framework
OLS (*Ordinary Least Squares*)	OLS (*Ordinary Least Squares*)	Scikit-learn
SVR(*Support Vector Regression*)	SVR(*Support Vector Regression*)	Scikit-learn
FNN (*Feedforward Fully Connected Neural Network*)	FNN (*Feedforward Fully Connected Neural Network*)	PyTorch
CNN (*Convolutional Neural Network*)	/	PyTorch
RNN (*Recurrent Neural Network*)	/	PyTorch
LSTM (*Long-Short Term Memory*)	/	PyTorch

5.6.1. OLS (Ordinary Least Square)

OLS or Ordinary Least Squares is the simplest machine learning technique retained, which performs linear regression (*cf.* Section 0.0.4.3.1). Implementing OSL in Scikit-learn library is a straight forward task, for both ODM and CDM data models. No extra manipulations of any additional parameters are here needed. The input and the output data are prepared in specific formats ready to execute.

5.6.2. SVR (Support Vector Regression)

The Scikit-learn library supports the SVR implementation with different kernels (*cf.* Section 4.3.2). In our implementation, we have considered two SVR kernels: RBF (*Radial Basis Function*), and Polynomial (2^{nd} degree – quadratic kernel). We have not considered the SVR with a linear kernel since in this case, the results are equivalent to those achieved by the OLS technique.

Radial Basis Function (RBF) kernel (*cf.* Section 4.3.2) was used with the following parameters:

$\alpha = \frac{1}{n\_features}$, $C = 1.0$.

This SVR implementation was used for both operator (ODM) and client (CDM) data models.

5.6.3. FNN (Feedforward Fully Connected Neural Network)

In the case of feedforward Fully Connected Neural Network (FNN), two different, hand-crafted architectures have been considered, dedicated to the ODM and CDM data models respectively. The

networks were developed while taking into account the shape and structure of each individual data model.

The proposed FNN for the ODM case is presented in Figure 5-10. It includes 5 fully connected layers, with 4 hidden layers of sizes (1000 – 100 – 1000 - 100). A ReLU activation function has been considered for all of them. The input layer iTDM is of size ($M_{\text{stations}} \times T$) and output is a vector of size ($M_{\text{stations}} \times 1$), representing the bus arrival times at all the bus stop stations considered.

For example, for bus 89, the iTDM is a 2D matrix of size [36 x 400], which corresponds to 36 measurement stations and 400 temporal instants. The output is a 36-valued vector that represents bus arrival times at each bus stop station.

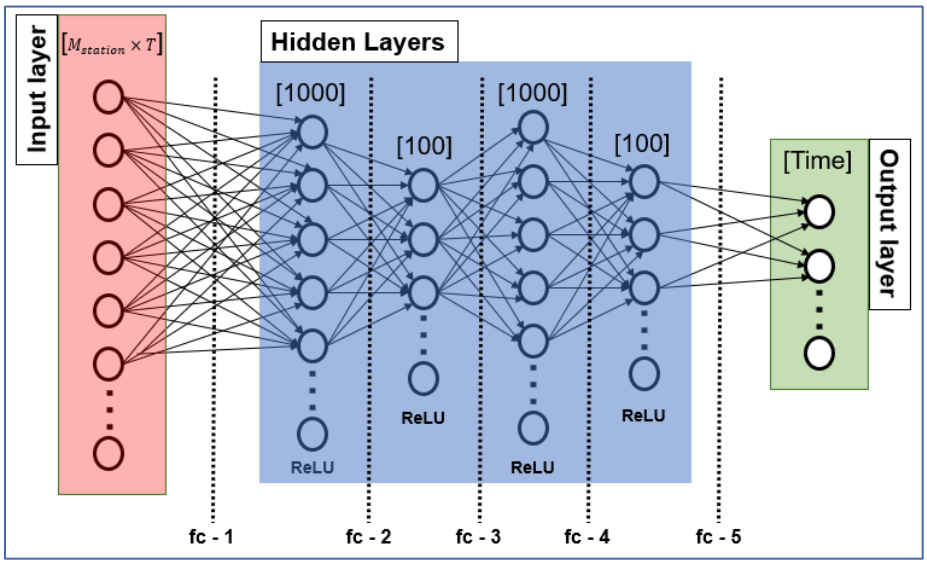


Figure 5-10. The proposed FNN architecture for ODM

In this particular configuration, the hidden layers aim at reducing the dimensionality of the features, up to obtaining the output vector of size 36 (time at each bus stop station). The double alternation between 1000 and 100 layers makes it possible to spread and mix as much as possible the data between layers. Let us also note that adding supplementary hidden layers and thus obtaining a deeper architecture is possible. However, in our case, we have privileged a simpler architecture with only 4 hidden layers. As it will be shown in Section 5.7, this is sufficient for obtaining accurate prediction results.

An MSE (*Mean Square Error*) loss function has been adopted and the SGD (*Stochastic Gradient Descent*) algorithm (with a *learning rate* of 0.001) used for learning. The number of epochs for training this model was set to 416 and the batch size to 10.

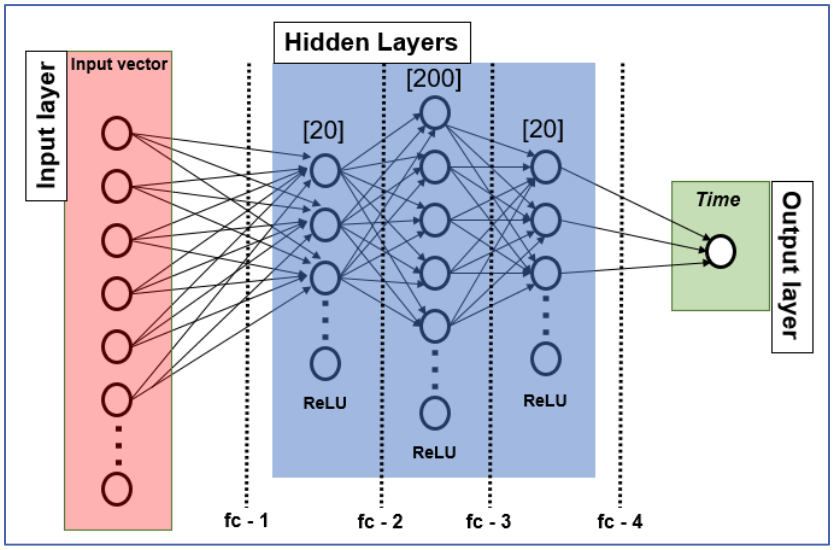


Figure 5-11. The proposed FNN architecture for the CDM case

The second proposed FNN architecture concerns the CDM data model and is presented in Figure 5-11. It includes 3 hidden layers of size (20 – 200 - 20). The ReLU activation function was also considered in this case. The **Input vector** is of size [1, M_{stations} +2] and the output value is a scalar (time), representing the expected bus arrival time at the considered bus stop station.

The MSE metric was used as a loss function and the SGD (*Stochastic Gradient Descent*) algorithm (with a learning rate of 0.001) for optimization in the learning stage. The number of epochs was set to 25. This network is obviously smaller than the previous network due to the reduced complexity of the data.

5.6.4. CNN (Convolutional Neural Network)

A Convolutional Neural Network (CNN) has also been proposed. Particularly adapted for an image-like structure, the CNN aims at taking into account the particular structure of the iTDM data. In particular, let us underline that it makes sense to perform convolutions on both dimensions (time and number of stations) of the iTDM structure, as illustrated in Figure 5-12.

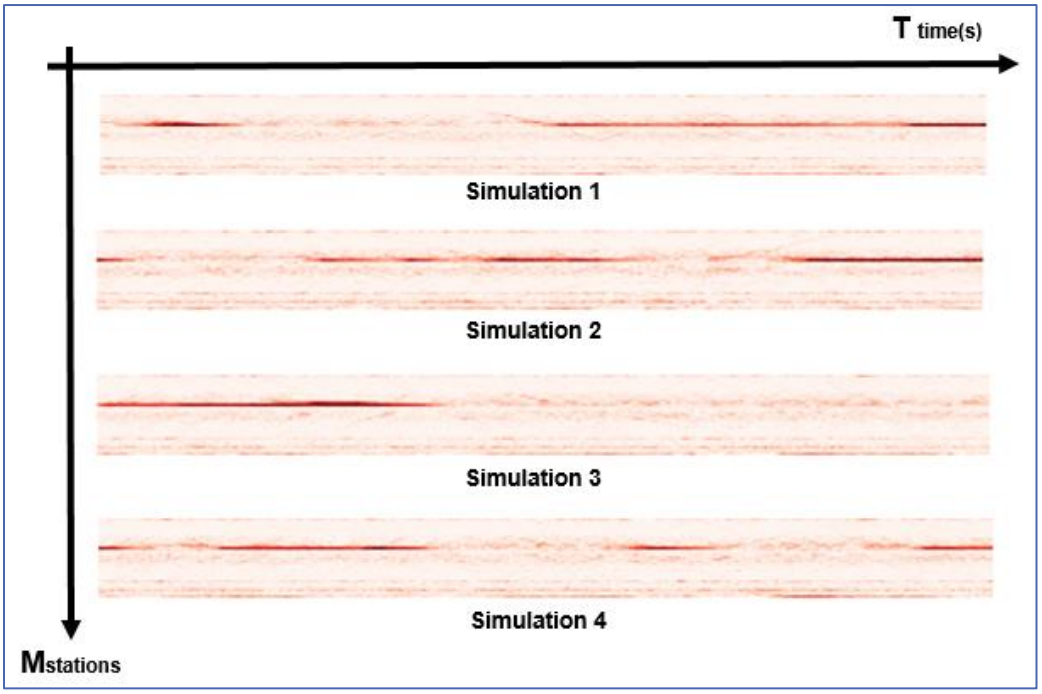


Figure 5-12. Input data, traffic data as an image-like structure iTDM

Here, 4 different randomly selected simulations are presented. Each individual grayscale image (iTDM) depicts the traffic condition of one simulation, which is used as input data for the ODM data. The Y-axis represents the measurement stations, while the X-axis concerns the temporal dimension (in seconds).

The images illustrated in Figure 5-12 were obtained by applying a min/max grayscale normalization procedure, described in equation (5.3):

$$I(\boldsymbol{x},\boldsymbol{y}) = \mathbf{255}\,\frac{\boldsymbol{X}(\boldsymbol{x},\boldsymbol{y})-\mathbf{min}\,(\boldsymbol{X})}{\mathbf{max}(\boldsymbol{X})-\mathbf{min}\,(\boldsymbol{X})}\ , \tag{5.3}$$

where:

- *X(x,y):* represents the value from the original iTDM matrix at (*x*,*y*) coordinates,
- *I(x,y):* is the resulting grayscale value obtained after normalization,
- *min*(*X*) and *max*(*X*) respectively represent the minimum and maximum values of the X matrix.

The normalization process renders the data particularly adapted for CNN networks. The proposed CNN architecture is presented in Figure 5-13.

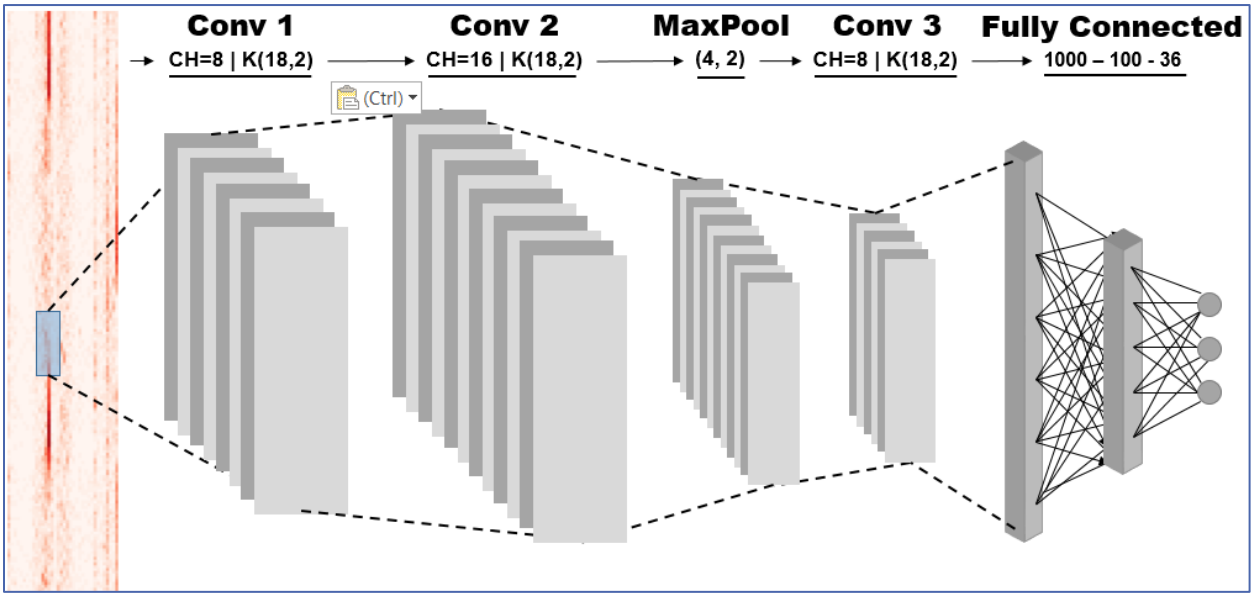


Figure 5-13. Proposed CNN architecture

This particular CNN network includes two successive convolutional layers (denoted by Conv 1 and Conv 2), followed by one maxPool layer, a third convolutional layer (Conv 3) and finally three fully connected layers, at the end of the chain. The corresponding parameters are the following:

- The kernel sizes of the three convolutional layers are of (18, 2), with 8, 16, and 8 channels respectively.
- The max-pooling layer is of size (4, 2).
- The two fully connected layers are of sizes 1000 and 100.
- The output layer is of size M_{stations}.

The dimensionality of the convolutional layers is calculated as:

$$O = \frac{W-K+2P}{S} + 1, \tag{5.4}$$

where:

- $\boldsymbol{O}$: output height/length,
- $\boldsymbol{W}$: input height/length,
- $\boldsymbol{K}$: filter size (kernel size),
- $\boldsymbol{S}$: stride
- $\boldsymbol{P}$: padding parameter, defined as:

$$\boldsymbol{P} = \frac{K-1}{2}\ , \tag{5.5}$$

Here are some architectural decisions that we have taken into account when designing this particular CNN architecture.

The specific kernel size was chosen because of the particular elongated shape of the image/data (iTDM matrix) used for the ODM data model. The inclusion of the fully connected layers at the end of the chain is a commonly used step, necessary to go down from the convolutional layer to the output decision layer. The padding parameter was set to 0 because the traffic information at the beginning and the end of the image is not essential for the prediction process.

The MSE measure was used as a loss function and the SGD (*Stochastic Gradient Descent*) algorithm (with a *learning rate* of 0.001) has been considered for the learning stage. The algorithm was trained on 105 epochs.

5.6.5. RNN (Recurrent Neural Network) and LSTM (Long-Short Term Memory)

RNNs and LSTMs are machine learning algorithms that have unique characteristics and capabilities that can be exploited for our purposes.

RNNs are class of ANNs designed to capture the sequential characteristics of times series and determine patterns to produce the predicted output. On the other hand, LSTMs, like the name suggests is a special kind of recurrent network, capable of taking into account long-term dependencies. Such networks were specifically designed to address long-term dependency problems. So, remembering the information for a longer period of time is their default setting. Both architectures use the output from the previous step that is fed as input to the current step, which makes them algorithms with internal memory.

Figure 5-14 proposes a deeper understanding of how the LSTM algorithm works.

In order to successfully train the LSTM algorithm, certain things need to be considered. Since this is a neural network that can address long-term dependencies, the implementation process is different than the other ANNs. Input data pass-forward (computed) though the network should be executed sequentially, as shown in Figure 5-14. In our particular case, the input data (iTDM) has a rectangular shape of size ($M_{stations}$, T) where $M_{stations}$ is the number of bus stop stations and T is the number of temporal measurements (400 in our case, corresponding to a simulation of 4000 seconds evenly samples at every 10 seconds). The resulting temporal sequence to be processed is thus composed of 400 instants, with the $M_{stations}$ values of the measurement stations.

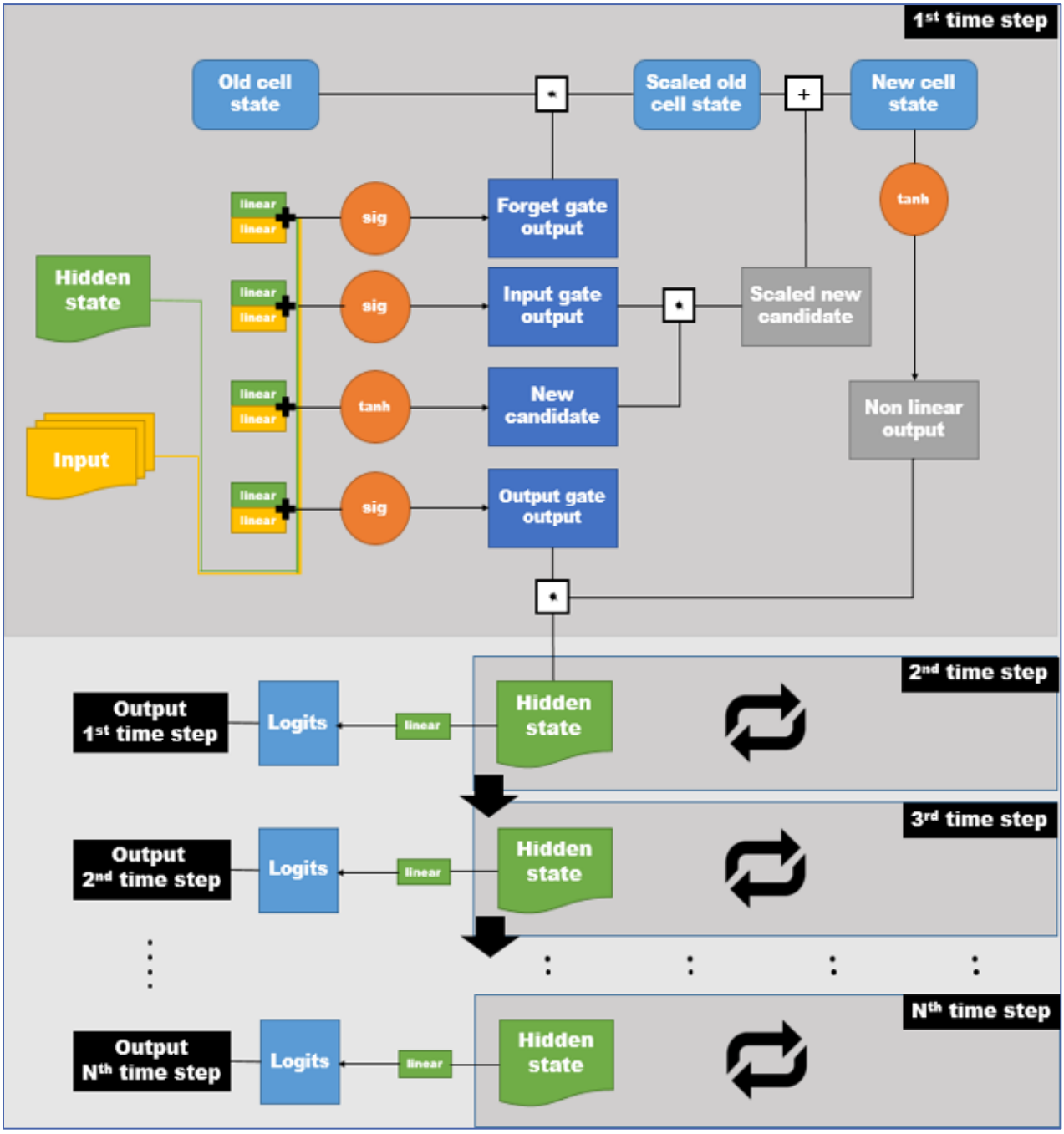


Figure 5-14. LSTM description

The process starts with the input data and the hidden state, where the hidden state is initialized with zeros. Then it passes them 4 times throw the linear layer. Next comes the addition of the output of the two linear layers, repeated 4 times, as shown. Here, the output of the linear layer of the hidden state is added to the output of the linear layer of the input. The next step includes non-linearity, and it can be divided into 4 groups, namely: forget gate, input gate, new candidate and output gate.

- Forget gate – uses a sigmoid function and determines how much we want to forget from the previous time step.
- Input gate – uses a sigmoid function and determines the amount of information coming into the system.
- New candidate – uses a tanh function, and represents the information that we can add to the new input.
- Output gate – uses sigmoid function and determines the output information to the new time step.

Elementwise multiplication between "*old cell state"* and "*forget gate"* will output "*scaled old output*". This determines what we want to forget from the old cell state. When the "*new candidate"* and "*input gate"* do elementwise multiplication the output is called "*scaled new candidate*" and determines what we want to add on top of the old information. The next step is elementwise multiplication between "*scaled new candidate*" and "*scaled old cell state*", this determines the "*new cell state*" and throw tanh function also determines the "*non-linear output*". The last step is the elementwise multiplication of "*non-linear output*" and the output gate, which will give the new "*hidden state"* of the next time step, this also provides the output of the first time step. The algorithm iteratively repeats this process for each time step.

The configuration of the RNN approach is presented in Figure 5-15.

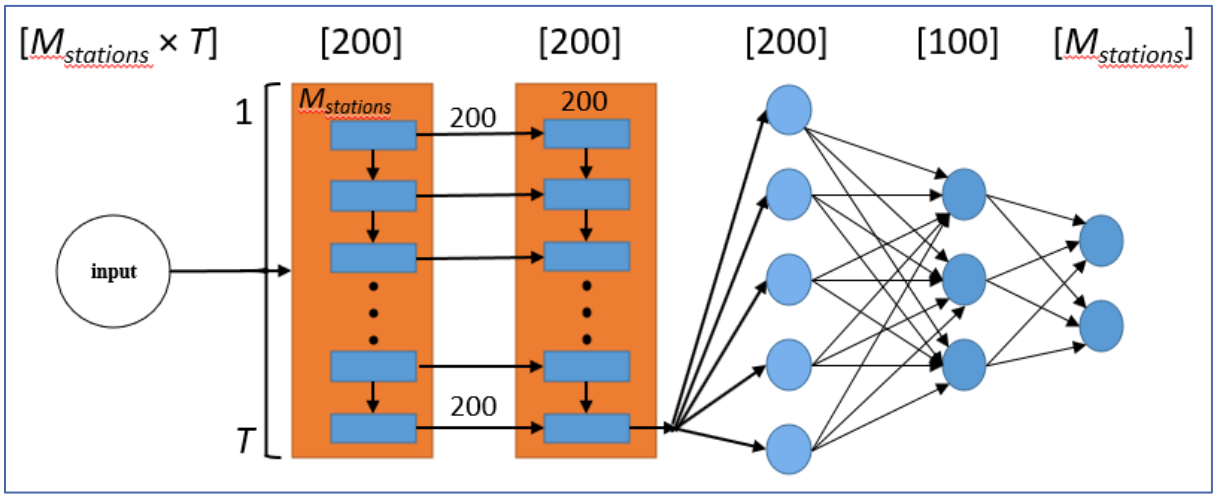


Figure 5-15. RNN architecture

The proposed RNN configuration includes 2 RNN layers of size 200 and 2 linear layers of size (200 - 100) with ReLU as an activation function, and output layer (36).

The SGD (*Stochastic Gradient Descent*) algorithm (with a *learning rate* of 0.001) has been considered for the learning stage. The number of trained epochs was 166 and the batch size was 10.

The adaptation of the LSTM architecture to our particular objectives is presented in Figure 5-16, Here, each LSTM layer corresponds to the representation presented in Figure 5-14.

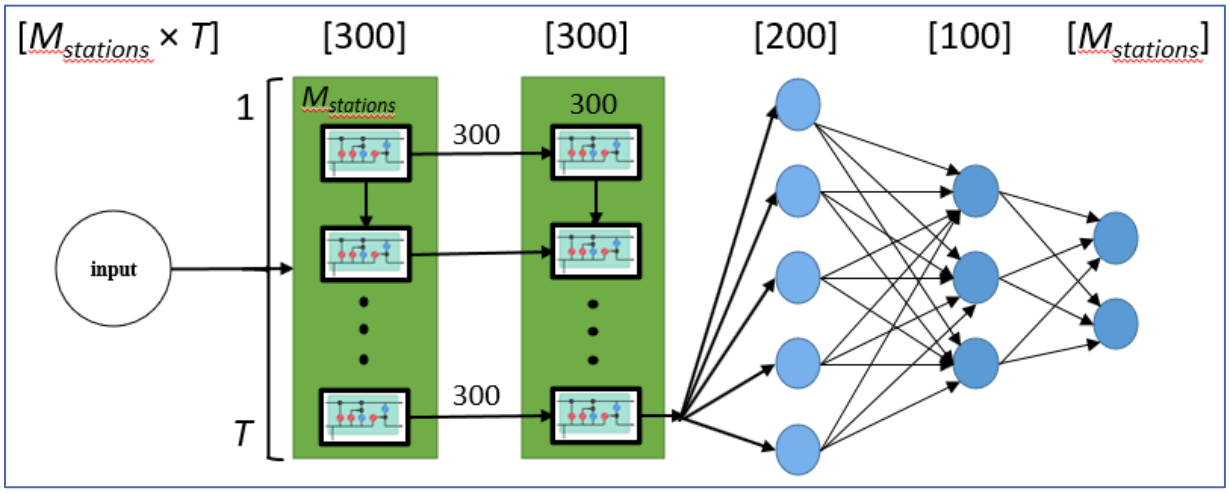


Figure 5-16. Proposed LSTM architecture

The proposed LSTM configuration includes 2 LSTM layers of size 300, 2 fully connected layers of size (200 - 100) and the output layer (of size $M_{stations}$).

The SGD (*Stochastic Gradient Descent*) algorithm (with a *learning rate* of 0.01) has been considered for the learning stage. The number of trained epochs was 125 and the batch size was 10.

The machine learning algorithms (Scikit-learn and PyTorch frameworks) were computed on the following PC configuration:

- Windows 10 Pro
- CPU – Intel Core i7 – 6800K @ 3,40GHz
- RAM memory - 32GB DDR4 (2400MHz)
- GPU – NVIDIA GeForce GTX 1080Ti (11GB)
- Storage – NVMe SSD Force MP500

Let us now detail the experimental results obtained.

5.7. Experimental results

Experimental results were computed over both ODM and CDM data models. In order to randomly and evenly sample the data, a data distribution per simulation with an appropriate train/test data split needs to be performed. The considered data partition is presented in Table 5-3.

TABLE 5-3. DISTRIBUTION OF SIMULATIONS WITH RESPECT TO THE NUMBER OF VEHICLES INSERTED IN THE SYSTEM

×1000	11 - 12	12 - 13	13 - 14	14 - 15	15 - 16	16 - 17	17 - 18	Total
Train data	434	465	456	458	439	465	482	3200
Test data	136	112	117	107	118	99	111	800
Total	570	577	573	565	557	564	593	4000

The total number of successfully computed simulations was 4000. Each simulation was computed with a random number of vehicles in the system, within the range [11000, 18000]. This is also illustrated in Figure 5-17, where the random distribution of the data between each range (11000 – 12000, …, 17000 - 18000) is presented. On the other hand, the randomness of the data split function was calculated with 10-fold cross-validation (80% train and 20% test), as previously explained in Section 5.5. Here, we may notice that the data split and the vehicle distribution functions have been made in a uniform manner. This is an important issue because the uniformity of the data is crucial for avoiding any potential biases.

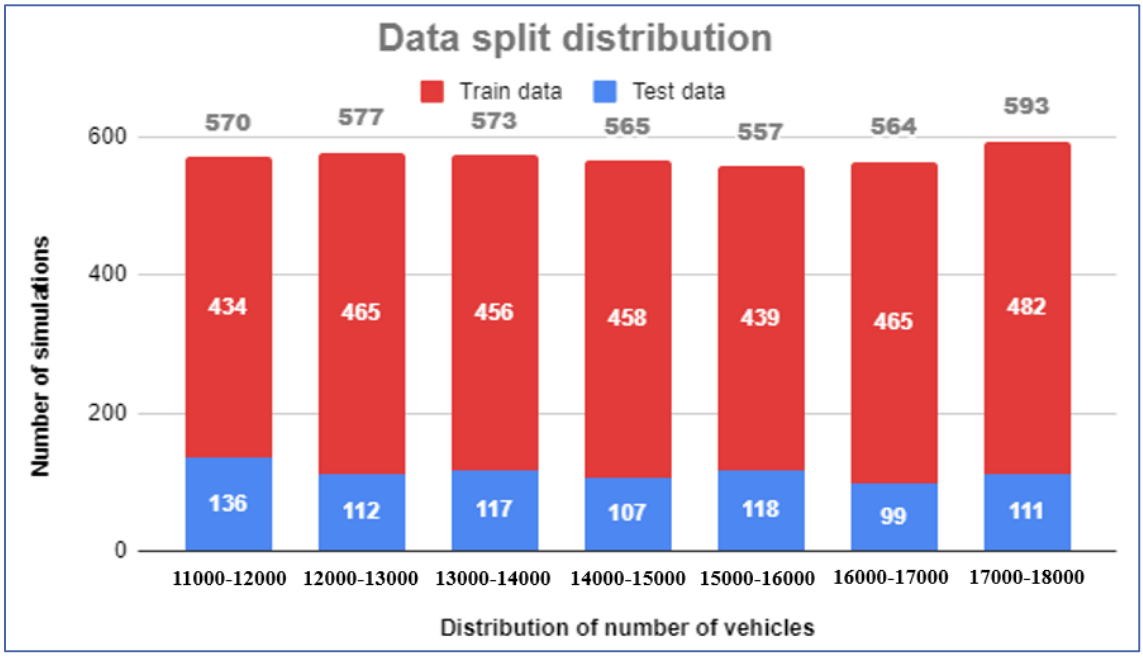


Figure 5-17. Train/test data split distribution

The resulting range of computed simulations is between 557 and 593 per group, from which 80% were used as training data (between 434 and 482) and 20% as test data (between 99 and 136).

5.7.1. Experimental results for the Operator Data Model (ODM)

In the first stage, we have considered for evaluation the OLS and SVR (with different kernels) approaches. The algorithms were used for predicting the bus arrival times at each bus stop station (long-term prediction) for both bus lines 79 and 89.

In a first example, Figure 5-18 plots the predicted bus arrival times per station, under various traffic conditions (which are controlled by global the number of vehicles inserted in the system), for both bus lines (79 and 89).

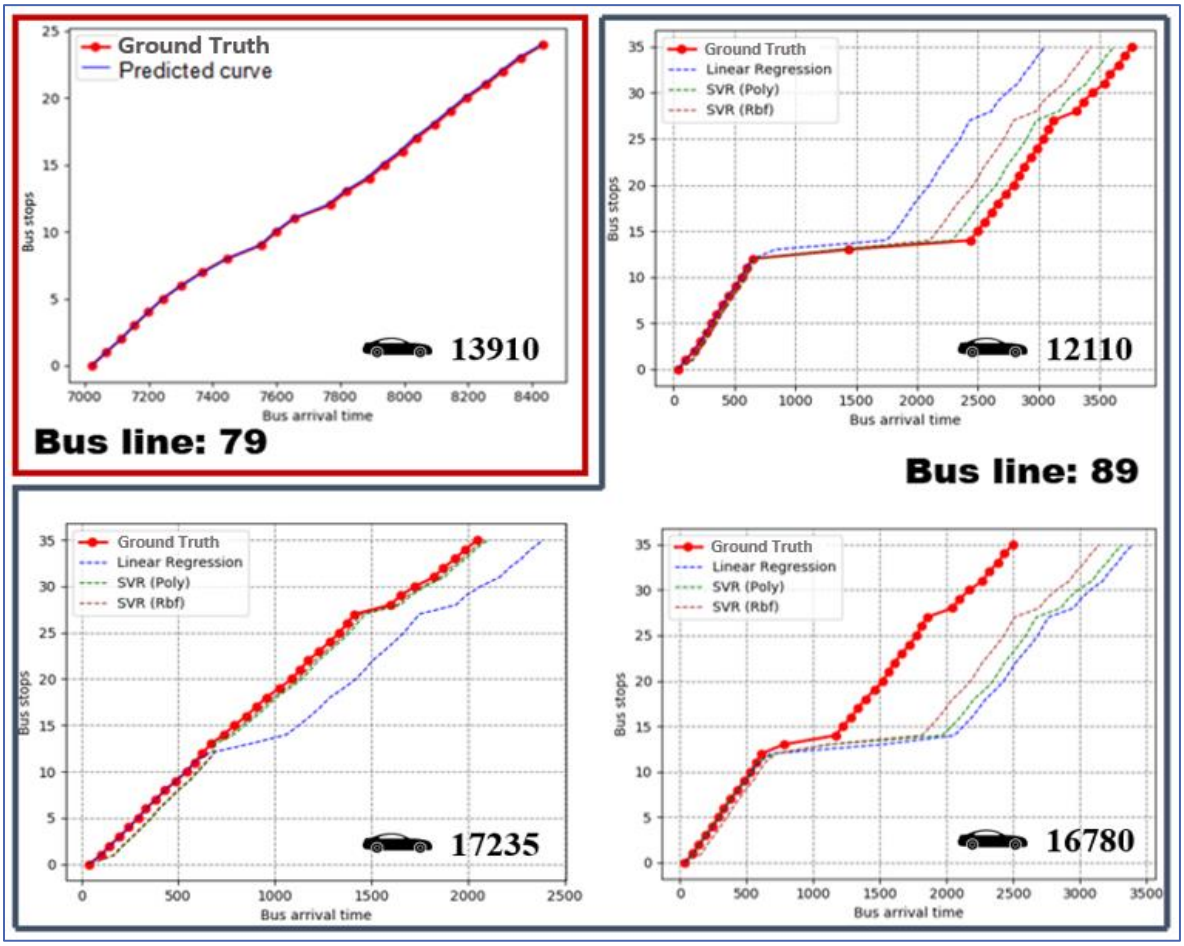


Figure 5-18. Predicted arrival times for bus line 79 (red boundary) with OLS versus predicted arrival times for bus line 89 (blue boundary) with OLS and SVR (polynomial and RBF)

The plot (red boundary box) in the upper-left corner represents bus line 79. In this case, the OLS approach is perfectly capable of predicting accurately the bus arrival time per station.

This behavior for bus 79 has been confirmed by all the other experiments that we have conducted. On this itinerary, no major traffic congestion zones (traffic jams), which may cause major problems, are encountered and a linear approach perfectly manages to predict correctly the arrival times.

TABLE 5-4. PREDICTION RESULTS FOR BUS LINES 79 AND 89 WITH OLS

OLS	79 Beau > Ovra	79 Ovra > Beau	89 Beau > Le Cr	89 Le Cr > Beau
MAE	4.96	3.18	134.18	133.2

To further confirm this claim, let us refer to Table 5-4, which summarizes the MAE scores, globalized over the entire test data set. In the case of bus 79, the obtained MAE scores range between 3 and 5 seconds, which corresponds to a highly accurate prediction.

However, the behavior is completely different in the case of bus line 89. As we can see from Figure 5-18 (plots in the blue boundary box), the predicted curve deviates from the ground true significantly (Table 5-4) with an MAE score of 133 and 134 seconds (for the two directions, respectively). A finer analysis allowed us to understand what happens in this case. Actually, some very punctual, localized traffic jams are occurring in some intersections and are responsible for the stair-wise character of the curves.

This problem is illustrated in Figure 5-19.

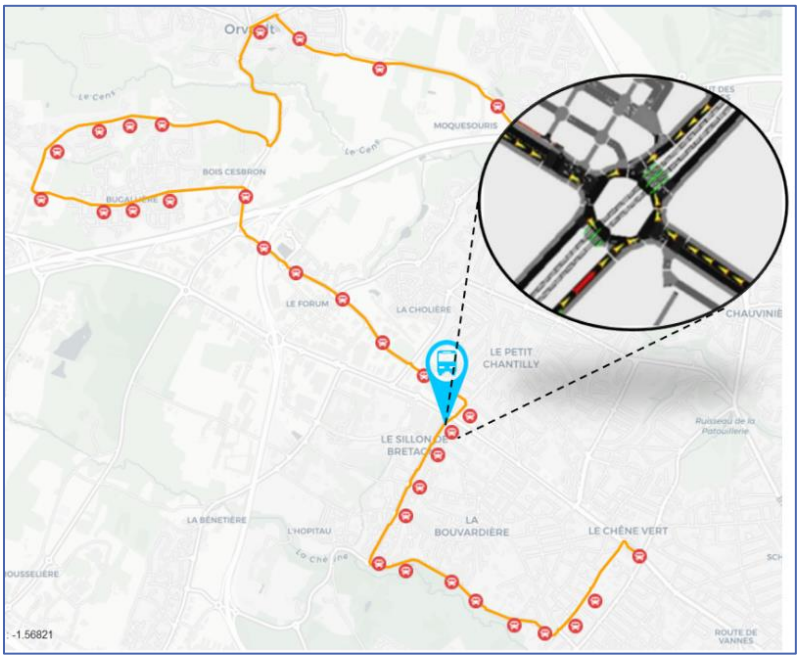


Figure 5-19. Traffic jam responsible for the prediction failure. Here, the bus is waisting 400 seconds.

Here, the bus remains stuck in the traffic jam occurring between bus stops 13 and 14 for 400 seconds.

The OLS approach is too elementary to take into account such complex, non-linear behavior. As a result, the prediction fails.

Let us now examine, for the same bus line 89, the behavior of the SVR approaches, when compared to the OLS technique. The obtained global MAE scores are reported in Table 5-5.

TABLE 5-5. MAE(OLS *VS.* SVR) OF THE PREDICTION RESULTS FOR BUS 89, DIRECTION BEAUSEJOUR > LE CARDO

	OLS	SVR Rbf	SVR Poly
MAE	134.186	71.062	72.063

The obtained results show that the MAE in the case of OLS is significantly higher than those obtained by the SVR approaches. The performance of SVR decreases the corresponding MAE down to 71 seconds, which by itself is impressive. This result represents almost a 47 % error decrease, with respect to OLS.

The analysis of these first results makes us think that more complex and highly non-linear approaches are necessary for taking into account such localized traffic jams events. In order to validate this intuition, let us analyze the results obtained by the various deep learning approaches (FNN, CNN, RNN and LSTM) considered. They are presented in Figure 5-20 and Figure 5-21, for different groups represented by a letter (from A to L). Such groups differ one from another by the total number of vehicles inserted in the system, indicated next to the small car sign that is present in the right corner of each plot.

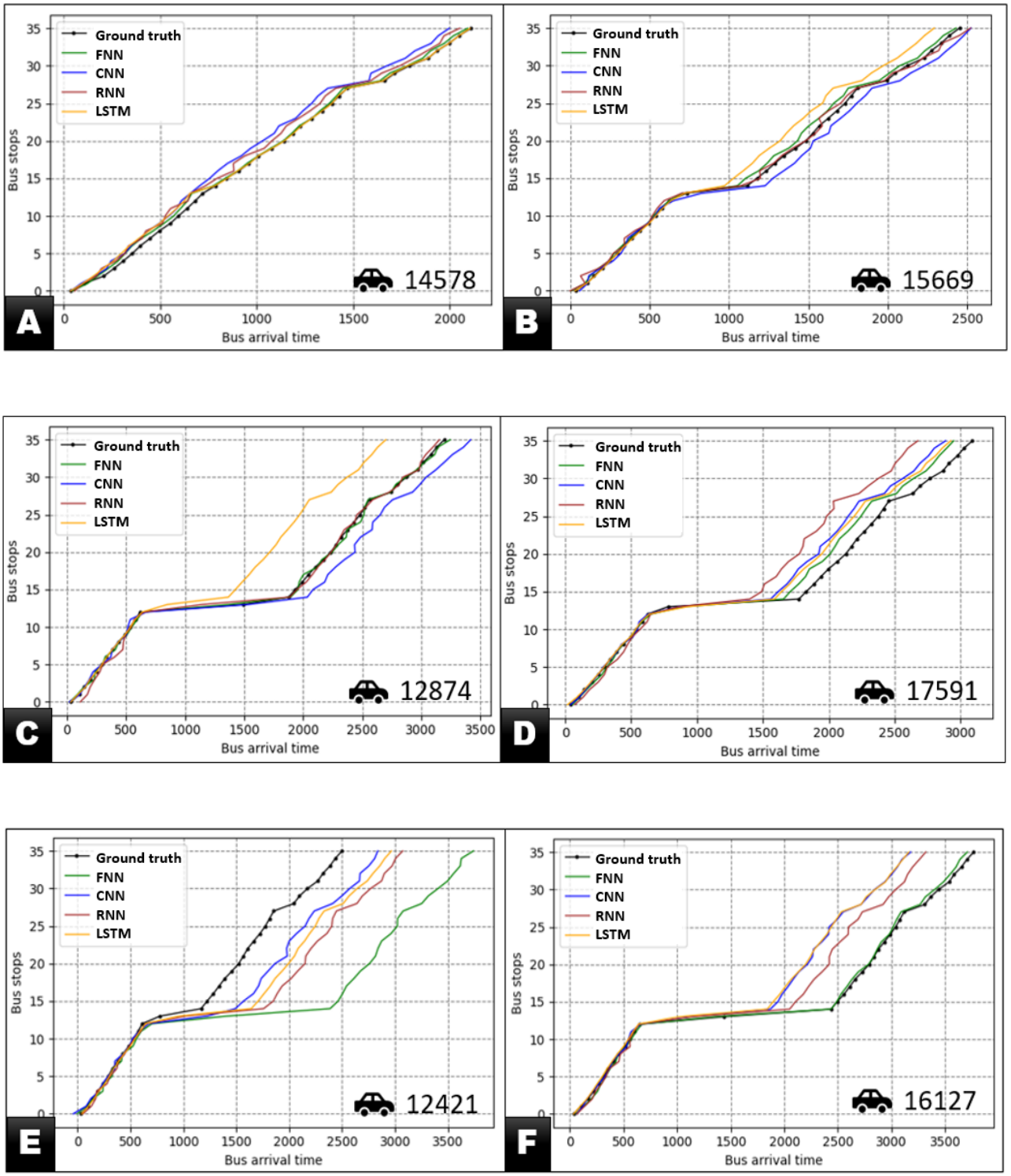


Figure 5-20. Deep learning prediction with FNN, CNN, RNN, and LSTM deep learning approaches (1st part)

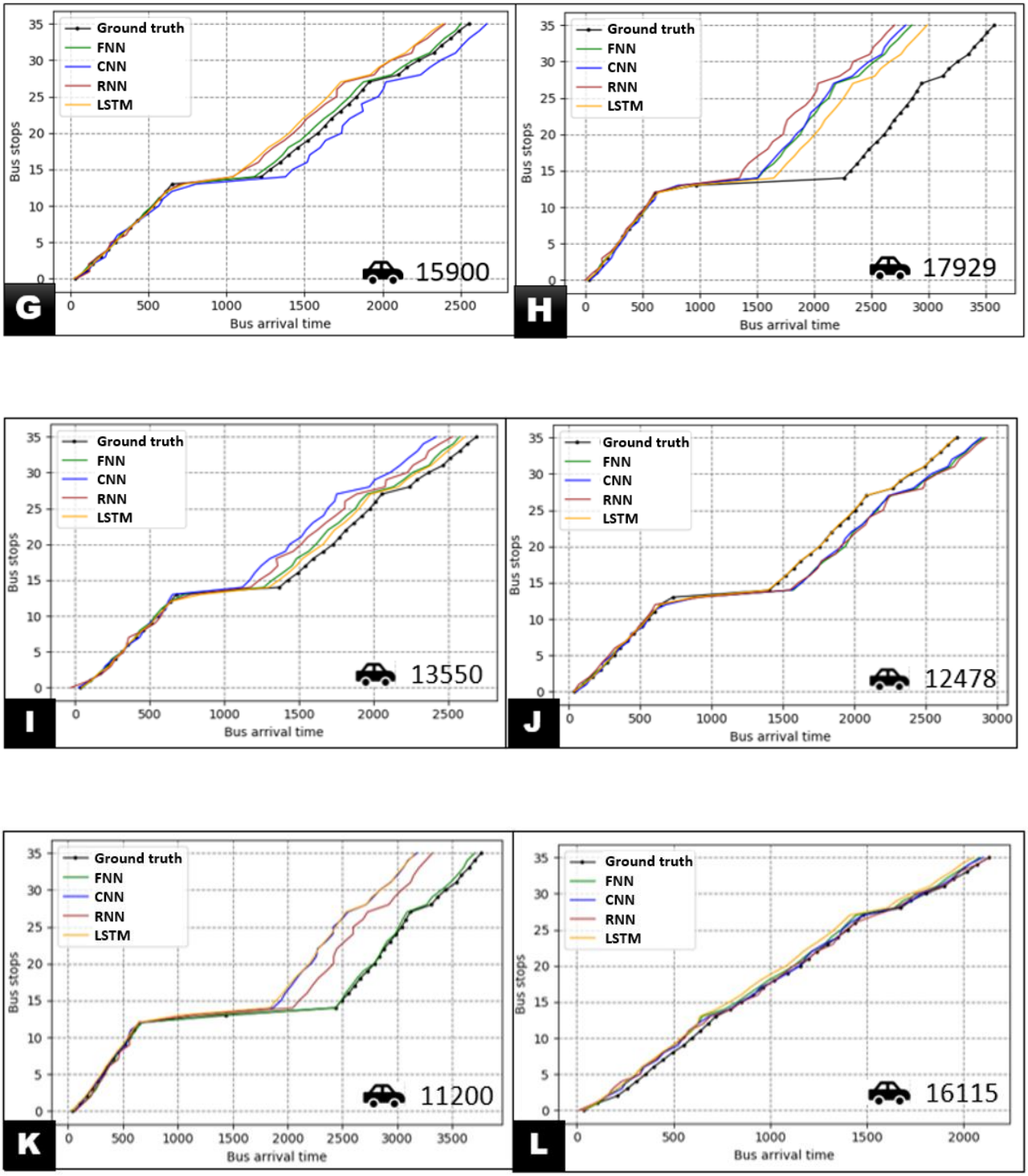


Figure 5-21. Deep learning prediction with FNN, CNN, RNN, and LSTM deep learning approaches (2nd part)

The predicted times are expressed in seconds and the considered bus line is number 89, direction Beausejour – Le Cardo. The obtained results show that, globally, all the considered approaches yield good prediction results. In most of the cases (Table 5-6), the FNN technique performs best and is closest to the ground truth. However, in some cases, like those illustrated in Figure 5-20(B) RNN shows better results than FNN. In Figure 5-21(H, J, I), the LSTM techniques show better performances than FNN. Finally, there are some cases, such as those illustrated in Figure 5-20(E) and Figure 5-21(L), where the best performer is CNN. This is also confirmed in Table 5-6. Here, techniques with their best and worst performance are presented.

TABLE 5-6. DL TECHNIQUES (BEST VS WORST) PERFORMANCE

DL technique	Color	Best performance	Worst performance
FNN	Green	A, C, D, F, G, K	E
CNN	Blue	E, L	A, K, I
RNN	Red	B	D, H
LSTM	Yellow	H, J, I	B, F, G, L

In order to objectively present and compare the results, in addition to the MAE score, we have considered the following set of supplementary criteria:

- minMAE presents the lowest MAE value that is achieved by the algorithm compared to the ground truth, while the maxMAE is the opposite (highest MAE value),
- The MEDIAN MAE value,
- The MAE standard deviation (STD). A low standard deviation indicates that the values tend to be close to the average of the set, while a high standard deviation indicates that the values are spread out over a wider range.
- cTime, denoting the computational time of the learning stage,
- EPOCH is the number of iterations that requires a full pass through the whole data set.

Table **5-7** summarizes in detail the performances of the various machine/deep learning techniques considered, according to this set of criteria.

The lowest MAE (44,6 seconds) is by far achieved by the FNN algorithm, while the highest by OLS (134,1 seconds). The FNN approach also leads to the lowest median MAE value, which is quite remarkable (16,2 seconds).

The lowest minMAE number is achieved by the LSTM technique (3,3 seconds) and the minimum maxMAE is achieved by the CNN approach (782,6 seconds). Concerning the standard deviations obtained, they are quite equivalent to all the algorithms considered, with the exception of the OLS approach, which is significantly higher.

TABLE 5-7. ML/DM PERFORMANCE

	OLS	SVR (Poly)	SVR (RBF)	FNN	CNN	RNN	LSTM
MAE	134,2	72,1	71,1	44,6	59,4	65,5	56,4
minMAE	12,3	5,9	5,7	3,4	5,5	10,7	3,3
maxMAE	1563	1024,8	1229,9	973,8	782,7	983,3	1084
MEDIAN	109,8	53,6	50,5	16,2	33,5	36,6	35,1
STD	109,8	67,5	70,7	73,6	73,1	75,2	73,9
cTIME	818	12193	12063	1500	1569	1654	1332
EPOCH	-	-	-	416	104	166	125

OLS – Ordinary Least Squares, SVR – Support Vector Regression, POLY - Polynomial, RBF – Radial Basis Function, FNN – Feedforward Fully Connected Neural Network, CNN – Convolutional Neural Network, RNN – Recurrent Neural Network, LSTM – Long-Short Term Memory, MAE - Mean Absolute Error, minMAE – Minimal Mean Absolute Error, maxMAE – Maximal Mean Absolute Error, MEDIAN – the Median, STD – Standard Deviation, cTime – Computational Time, EPOCH – Number of Epochs

In order to better illustrate these results, Figure 5-22 presents the cumulative MAE histogram obtained by the various approaches retained. The vertical drop (line) corresponds to the maximum MAE obtained for each of the methods involved.

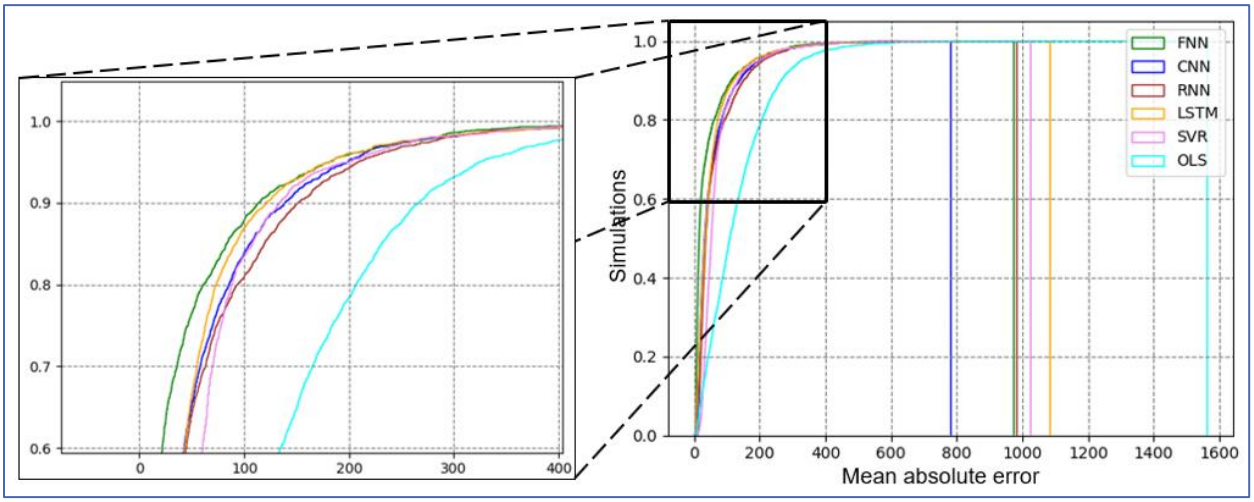


Figure 5-22. Cumulative MAE histogram

In addition, Figure 5-23 presents the MAE distributions obtained over the various simulations performed.

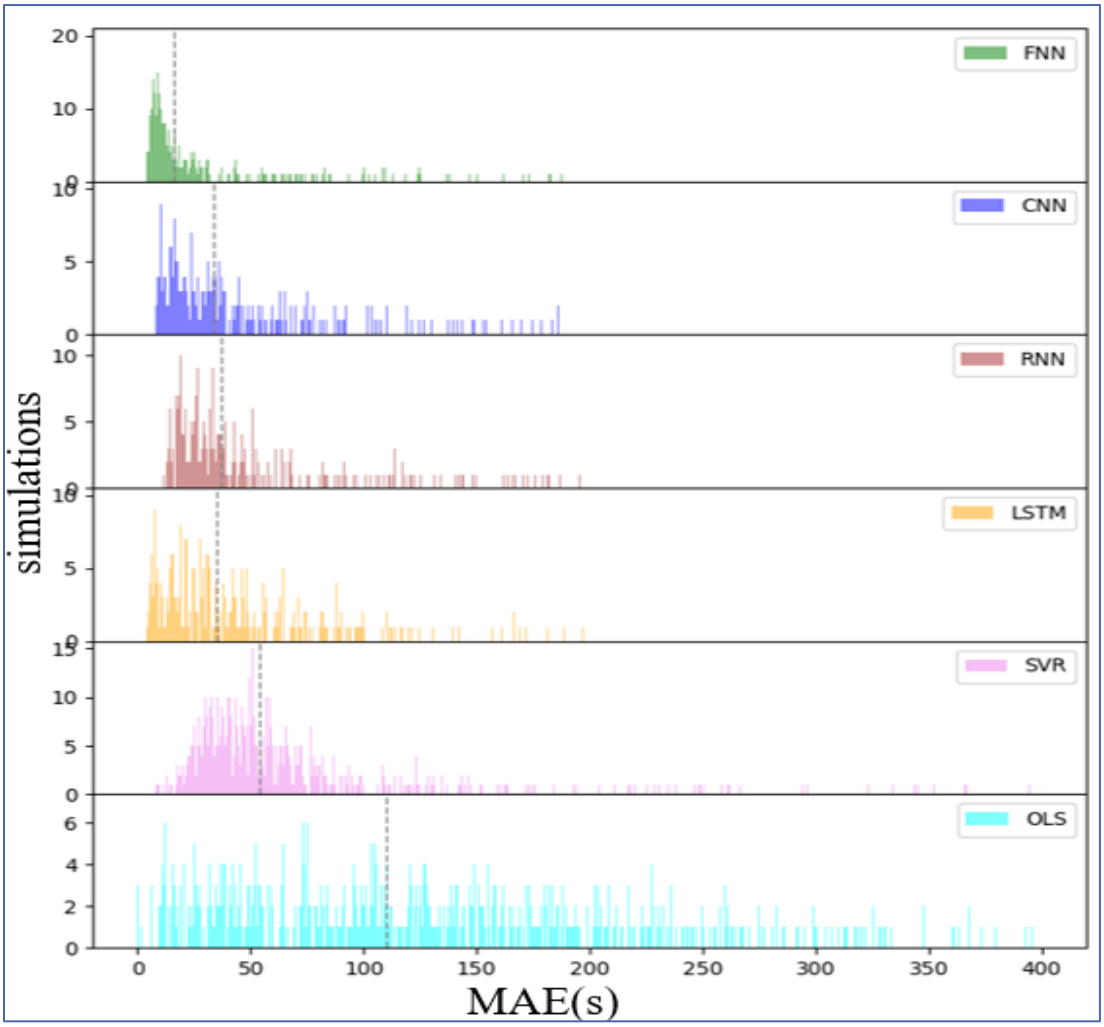


Figure 5-23. MAE distributions over the various simulations and corresponding median values (vertical dotted lines)

Here, the dotted vertical lines illustrate the median MAE values. The best median is achieved by FNN (16,2), which is an incredible result. This means that in 50% of the obtained results, the MAE is lower than 17 seconds.

To benchmark the performance of each algorithm in terms of computational time and effort, we have considered the cTime parameter, which provides the time requested for completing the learning stage. The best computational performance cTIME is achieved by OLS (with a high penalty of overall performance), with 818 seconds. The LSTM approach comes in the second position, with 1332 seconds. Let us also note that all the deep learning approaches considered are significantly more efficient in terms of computational effort (with cTIME values ranging in the [1332-1659 seconds] interval, than the SVR techniques (with cTIME values exceeding 12000 seconds).

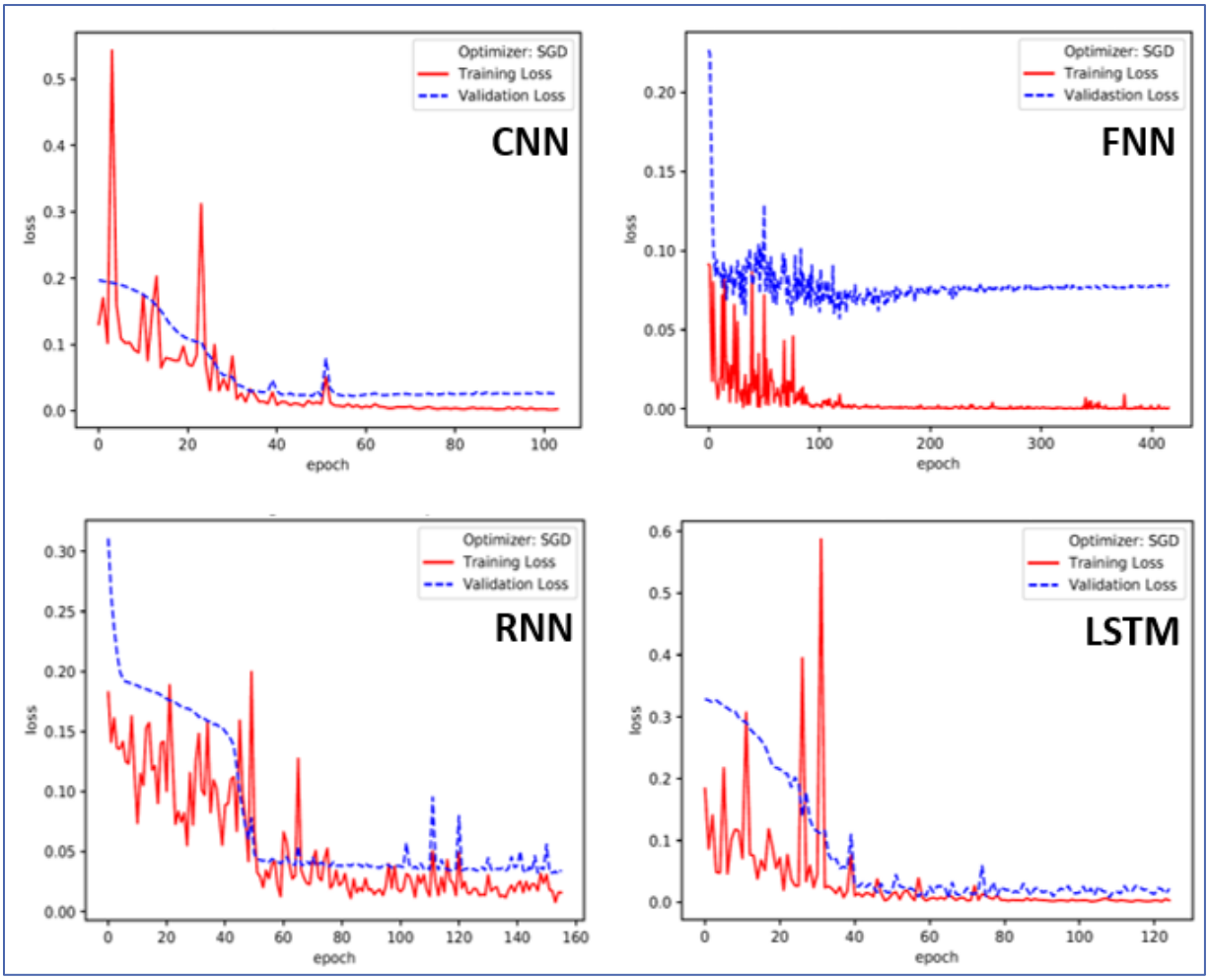


Figure 5-24. Deep learning curves: evolution of the loss function over the number of epochs

The EPOCH parameter presents the total number of times where the whole dataset was browsed during the learning stage. The smallest number of epochs to train successfully the model was CNN with 104 epochs in total, as illustrated in Figure 5-24. Here, the learning curves between different models and their performance are also presented. The training loss is presented with the red line and the validation loss is represented by the blue dotted line. The training curve presents the learning performance of the model, which is easy to observe (the convergence of the learning curve.). On the other hand, the validation curve is also important when a deep learning model is developed. It can serve to measure the evolution and performance of the model in real-time (very useful) and also can give an intuition if the model overfits the training data.

The deep learning curves presented in Figure 5-24, converge successfully for all the considered algorithms. The validation curve shows that there is no overfitting.

From all the presented results so far, a conclusion about the best machine learning algorithm is not obvious, since they present quite equivalent results. However, the best performer is the FNN approach, which offers the lowest MAE (44,6 seconds) and median MAE (16,2 seconds).

In order to further understand the performance of the algorithms, a comparative analysis with recently published algorithms is proposed (Table 5-8).

TABLE 5-8. COMPARATIVE PERFORMANCE WITH DIFFERENT SOA APPROACHES (LONG-TERM PREDICTION)

Article	Year	Algorithms ML/DL	MAE	Improvement in %
Proposed approach	2020	OLS	134.186	0
		SVR poly	72.063	46.3%
		SVR rbf	71.062	47.04%
		RNN	65.474	51.21%
		CNN	59.389	55.74%
		LSTM	56.383	57.98%
		FNN	44.629	66.74%
Zhang and Liu [114]	2019	HA	29.45	0
		FVDM	28.04	4.79%
		BPNN	78.11	-165.23%
		CK-means	19.27	34.57
He *et al.* [110]	2018	HA	4.220	0
		LR	3.684	12.7%
		SVR	3.601	14.67%
		FCNN	4.093	3.01%
		PCF	1.978	53.13%
Treethidtaphat *et al.* [123]	2017	OLS		0
		FCNN	None	55%
Yamaguchi *et al.* [122]	2018	LR		
		ANN	P values	
		RF	(T-test to MAE)	None
		SVR		(Best GBDT)
		GBDT		
Yu *et al.* [121]	2017	OLS	1.631	0
		RR	1.629	0.12%
		LASSO	1.445	11.4%
		FCNN	3.643	-123.36%
		LSTM	1.003	38.5%
		Mix LSTM	0.970	40.53%
Pan *et al.* [111]	2012	HA		0
		BPNN	None	5.7%

MAE - Mean Absolute Error, HA – Historical Average, LR – Linear Regression, RR – Ridge Regression, LASSO – Regression, OLS – Ordinary Least Squares RF – Random Forest, ARIMA - AutoRegressive Integrated Moving Average, SVR – Support Vector Regression, StoS – Sequence to sequence

learning, SMA – Simple Moving Average, FVDM – Front Vehicle Data Model, PCF – Partitioning and Combination Framework, GBRT – Gradient Boosting Regression Tree, SAE – Stacked AutoEncoder, GA – Genetic Algorithm, GBDT – Gradient Boosting Decision Tree, CK-means – K means Clustering, OPFA – Optimized Particle Filtering Algorithm, HM – Hybrid Model, ANN – Artificial Neural Network, DNN-BTF – Deep Neural Network Based Traffic Flow, GRU – Gated Recurrent Unit, FCNN – Fully Connected Neural Network, FNN – Feedforward Fully Connected Neural Network, BPNN – Back-Propagation Neural Network, LSTM – Long-Short Term Memory

Table 5-8 presents the results obtained by some recent research works for the long-term prediction of public transportation (buses). The approach proposed in this thesis is presented in the first row, in gray color.

The table is divided into 5 columns, including article citation, year of publishing, algorithms implemented, MAE values and achieved improvement. Each method has an improvement score expressed in percentages with respect to the reference baseline method (RBM) retained. In Table 5-8, the RBM techniques are indicated in red (and show a 0% improvement score over themselves).

A first observation is that the FNN proposed in this thesis performs objectively best, with an improved score of 66% over the baseline (in our case OLS). The other deep learning algorithms proposed perform similarly, with improvement over baseline around 55%. This result is approached solely by the FCNN-based method introduced by Treethidtaphat *et al*. in [123]. Another remark is that in some cases, such as those reported by Zhang and Liu [114] and Yu *et al*. [121], the deep learning techniques perform very poorly, and worse than the baseline approach. This can be due to some specific implementation or because of a lack of sufficient or diverse data.

The next section presents the results obtained for the CMD data model.

5.7.2. Experimental results for the Client Data Model (CDM)

This section presents the experimental results obtained for the CDM. The proposed algorithms were used for prediction of bus arrival time at a particular station (short-term prediction) rather than all bus stop stations. Let us underline that in this case, the prediction window is very narrow in the moment of the observation (real-time).

For comparison, three different machine learning algorithms have been considered: OLS (*Ordinary Least Squares*), SVR (*Support Vector Regression*) with 2nd-degree polynomial kernel and FNN (*Feedforward Fully Connected Neural Network*) (*cf*. architecture introduced in Section 5.6.3, Figure 5-11).

The obtained prediction results are summarized in Table 5-9 and illustrated in Figure 5-25. Let us recall that the data development scenario for the CDM data model is explained in greater detail in Section 5.5.3.

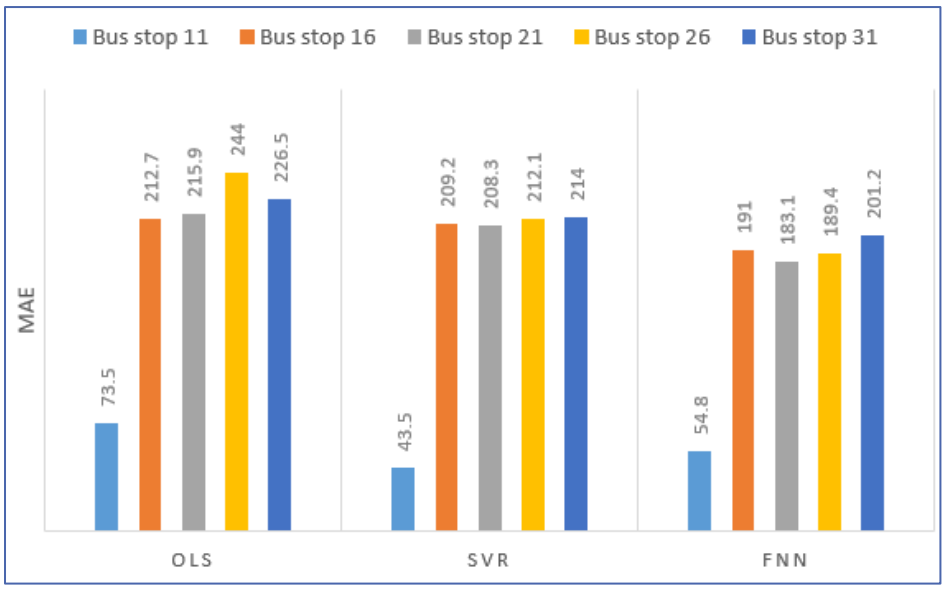


Figure 5-25. MAE per bus stop

Here, 5 different bus stop stations were considered for prediction, namely: 11th, 16th, 21st, 26th, and 31st,. The corresponding MAE scores are presented in seconds and represent the error time of the bus arriving at the desired station. On the other hand (Table 5-9), the global MAE (gMAE) score measures the average MAE over the whole set of 5 bus stop stations. In addition, the cTime parameter presents the time needed to compute each model during the learning process.

TABLE 5-9. CDM DATA MODEL PERFORMANCE

	11	16	21	26	31	gMAE	cTIME
OLS	73,5	212,7	215,9	244,0	226,5	194,52	43
SVR	43,5	209,2	208,3	212,1	214	177,42	2985
FNN	54,8	191	183,1	189,4	201,2	163,9	399

MAE - Mean Absolute Error, CDM – Client Data Model, OLS – Ordinary Least Squares, SVR – Support Vector Regression, FNN – Feedforward Fully Connected Neural Network, gMAE – global MAE, cTime – Computational Time.

Table 5-9 shows that the FNN approach yields the best short-term prediction results (with 15,76 % improvement over OLS and 7,62 % over SVR). The FNN performance is also better for all predicted bus stops, with the exception of only one instance which corresponds to the bus stop 11. Here, the SVR technique has an MAE of 43,5 seconds against 54,8 seconds for the FNN. This result can be explained by the fact that during the first bus stops, there is poorly probable to encounter traffic jam problems.

The best cTime needed for computing the algorithms during the learning process (time needed to learn and build the model from the data) is achieved by the OLS approach (with only 43 seconds). However, the OLS-related accuracy is the worst.

Globally, we can observe that here again, the non-linear solutions are more appropriate for performing accurate predictions.

This result is furthermore confirmed by the plot illustrated in Figure 5-26, which shows the prediction trend (MAE performance between the three considered methods) for the bus arrival time at the bus stop stations from 11th to 31st.

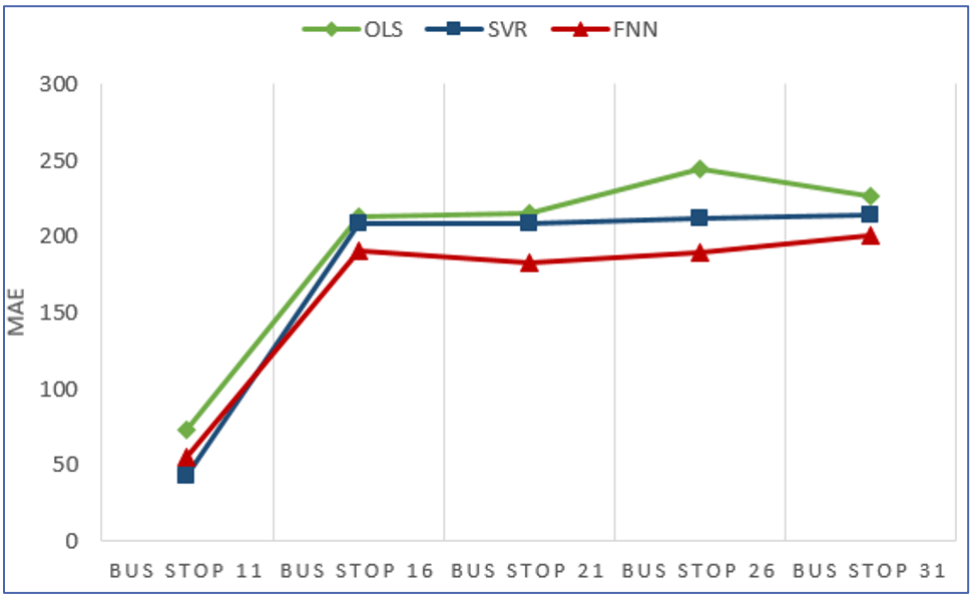


Figure 5-26. CDM prediction trend

Figure 5-27 analyzes the results obtained for a single bus stop station. Here, bus stop station No.26 has been considered and the predicted errors are presented under a form of a scatter chart for the 3 methods. The chart presents all the errors plotted with different colors (OLS with green, SVR with blue and FNN with red).

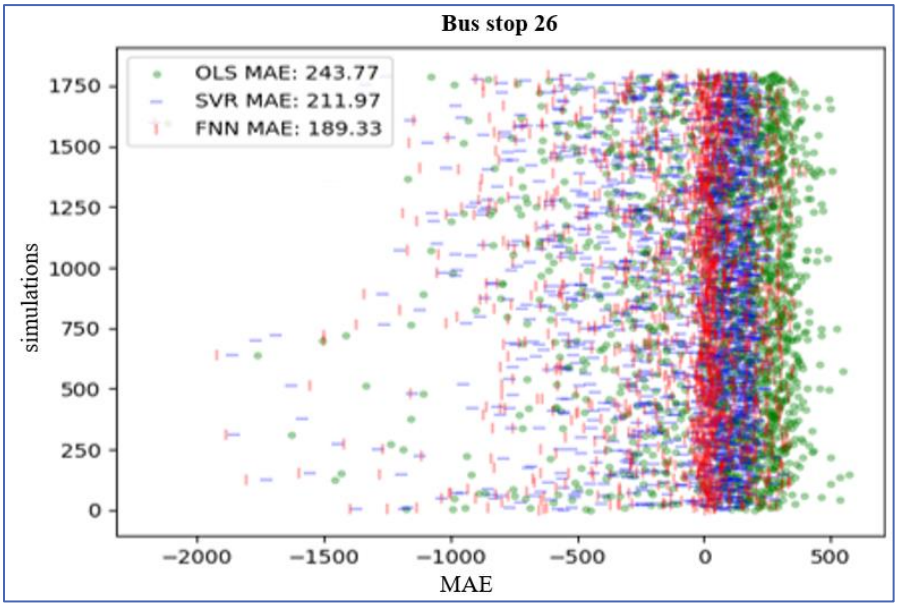


Figure 5-27. MAE for bus stop 26

Figure 5-28 presents the learning and validation curves of the FNN method for the CDM data model. Here, the algorithm was trained for only 25 epochs, which was sufficient to successfully obtain reasonable performance.

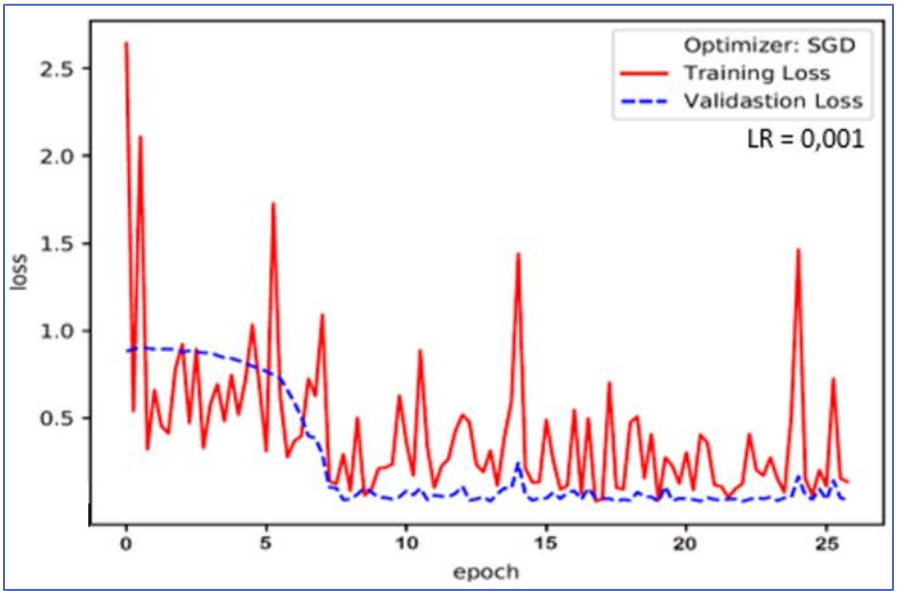


Figure 5-28. FNN learning curve for CDM

As in the case of the ODM data model presented in the previous section, in order to further understand the performance of the algorithms, we propose a comparison with the recent state of the art techniques (Table 5-10) that are targeting short-term prediction of public transportation (buses). The CDM proposed in this thesis (our approach) is presented in the first row with gray color. The reference baseline method considered by each approach is represented in red.

TABLE 5-10. COMPARISON WITH SOA APPROACHES (SHORT-TERM PREDICTION)

Article	Year	Algorithms ML/DL	MAE	Improvement In %
Our approach	2019	LR	194.52	0%
		SVR	177.42	8.79%
		FNN	163.9	15.74%
Liu and Xiao [115]	2019	HA	78.2	0%
		OPFA	71.67	8.35%
Bohan and Yun [116]	2019	LSTM	0.682	0%
		GRU	0.761	7.84%
Fu et al. [109]	2016	ARIMA	19.175	0%
		GRU	18.127	5.47%
		LSTM	17.211	10.24%
Wu and Tan [119]	2016	GBRT	22.52	0%
		FCNN	20.61	8.48%
		SAE	20.36	9.59%
		LSTM	21.53	4.4%
		CNN+LSTM	19.37	13.99%
Lam et al. [113]	2019	SMA		0%
		ANN	None	17%
		HM		None
Wu et al. [120]	2018	LASSO	29.586	0%
		StoS	27.562	6.96%
		SAE	27.173	8.16%
		BPNN	25.672	13.23%
		DNN-BTF	25.189	14.86
Huang et al. [78]	2019	HOD		0%
		GA	None	7.5%
		LSTM		14%

MAE - Mean Absolute Error, ML – Machine Learning, DL – Deep Learning, HA – Historical Average, LR – Linear Regression, RR – Ridge Regression, LASSO – Regression, OLS – Ordinary Least Squares RF – Random Forest, ARIMA - AutoRegressive Integrated Moving Average, SVR – Support Vector Regression, StoS – Sequence to sequence learning, SMA – Simple Moving Average, FVDM – Front Vehicle Data Model, PCF – Partitioning and Combination Framework, GBRT – Gradient Boosting Regression Tree, HOD – Historical Origin Destination, SAE – Stacked AutoEncoder, GA – Genetic Algorithm, GBDT – Gradient Boosting Decision Tree, CK-means – K means Clustering, OPFA – Optimized Particle Filtering Algorithm, HM – Hybrid Model, ANN – Artificial Neural Network, DNN-BTF – Deep Neural Network Based Traffic Flow, GRU – Gated Recurrent Unit, FCNN – Fully Connected Neural Network, BPNN – Back-Propagation Neural Network, LSTM – Long-Short Term Memory

Here, the performance of the proposed FNN deep learning technique shows a 15% improvement over the baseline. The highest achieved score was 17% improvement in Lam et al. [113], but this score was calculated against the SMA (*Simple Moving Average*) baseline method, which is not as powerful as OLS. The slight improvement of SVR and FNN over the baseline OLS was expected since this data model (CDM) is limited in terms of the robustness of the input data. In this case, the data contains less information when compared with the ODM data model, so the model can learn less.

5.7.3. ODM vs. CDM comparison

Even though the two data models are totally different in nature and in the prediction window (long-term *versus* short-term prediction), Table 5-11 summarizes the performance comparison, with MAE scores obtained, percentage of improvement over the corresponding baseline reference methods and cTime (learning time) involved.

TABLE 5-11. COMPARATIVE PERFORMANCE BETWEEN CDM AND ODM-BASED PREDICTIONS

	CDM data model			**ODM data model**		
	OLS	SVR	FNN	OLS	SVR	FNN
gMAE	194,52	177,42	163,9	134,18	71,23	44,63
% improvement	0% (baseline)	8,79	15,74	0% (baseline)	46,91	66,74
cTIME (seconds)	43	2985	399	818	12063	1500

In terms of MAE, the ODM-based algorithms perform significantly better and lead to lower MAE when compared with the CDM-based approaches. The relative gain in accuracy over the baseline reference methods is also noticeably higher in the case of ODM algorithms. On the counterpart, the cTime is lower for the CDM prediction. But this is not a surprise since the ODM data model is much more heavy and complex in comparison with CDM.

5.8. Discussions and conclusion

In this chapter, we have elaborated extensively the public transportation prediction. The objective was to predict the bus arrival times at the bus stop stations, under the framework of both long-term and short-term prediction.

All the proposed prediction methods are based on a novel concept, so-called TDM (*Traffic Density Matrix*) that was first introduced. The TDM technique introduces localized information about the traffic conditions in a given city area, with the help of a set of measurement stations, which capture the number of vehicles that are present in the vicinity of the considered measurement point. The resulting TDM data is presented under the form of an image-like structure that represents the evolving vehicle density over a period of time.

A real-life scenario has been conducted in a virtual environment, with the help of the SUMO traffic simulation platform. It concerns a sub-part of the city of Nantes, France for two real bus lines numbered by 79 and 89 (which leads to four itineraries in total). The total number of simulations was 4000 for both bus lines since the scenario was constructed in a way that allows us to simulate both lines in a single simulation.

In order to predict the bus arrival times in the stop stations, various techniques have been proposed and explored. Ordinary Least Square and Support Vector Regressors (with both Polynomial and RBF kernels) have been retained as baseline methods. Then, we have proposed a set of dedicated deep learning architectures, including a Feed-Forward Fully Connected Neural Network, a Convolutional Neural Network, a Recurrent Neural Network and a Long-Short Term memory.

The experimental results obtained showed that the best performances are achieved by the FNN approach, which yields the lowest MAE and median MAE scores among all of the considered techniques. In a general manner, all the deep learning techniques proposed outperform, in terms of prediction accuracy, the classical OLS and SVR machine learning approaches. This shows that increasing the degree of non-linearity of the methods makes it possible to obtain superior results, in particular in the case where local singularities that are caused by traffic jams occur.

For future analyzes and development, we may explore different machine learning techniques, as well as different bus lines where the possibilities of traffic jams are highly possible. One interesting future possibility may concern the change of the location of the measurement station in the bus itinerary, from the bus stop stations to specific road intersections. More generally, the optimization of the number and position of the considered measurement stations is a promising axis of research.

Another interesting axis of future research concerns the inclusion within the prediction process of various other parameters, that may concern the population density in given areas and the degree of occupancy of the buses, or moreover the introduction of unexpected events (infrastructure works, accidents).

Chapter 6. Visualization platform

Abstract

This Chapter presents in details the visualization platform proposed, so-called Web Interface 1.0. The main objective is to propose a scalable solution, that can help visualizing the traffic situation (with identification of potential problems that may cause delays) as well as the bus-related features (bus occupancy rate, estimated time of arrival at a considered bus station) at an arbitrary scale in the city. To this purpose, two dedicated interface are elaborated and proposed. A first one concerns the transportation operator. Here, a rich set of analysis features is integrated, that allow the expert user to monitor the bus situation at long-time and perform the necessary analysis. A simplified interface, dedicated to the end-user client that can be deployed on arbitrary devices is secondly proposed. The goal here is to offer a simple tool that can inform the user about the bus arrival times and conditions (e.g., occupancy bus rates).

6.1. Introduction

The objective of any visualization process concerns the presentation of the computed data in an ergonomic and visually understandable form that can facilitate its exploitation by human agents. In our case, the main goal is to present the public transportation (buses)-related issues in a way that can facilitate the planning process for both the end-user and the transportation operator. As the corresponding requirements are quite different, depending on the user's type, two distinct web interfaces have been developed:

- *Client interface* – this web interface presents the results on the client-side, and notably concerns the short-term predictions of the bus arrival times at a particular station that is requested by the user. The CDM data model needs to be considered and integrated here. The related constraints concern the real-time and on-demand response capabilities.
- *Operator interface* – this web interface platform is intended to present in a relevant manner the long-term prediction results at the operator side. Here, the amount of information that needs to be displayed is much more developed and detailed, in comparison to the client-side. The elements to be visualized are meant to be used by a company or agent that is responsible for planning, creating, or maintaining public transportation networks. The web interface was built in this case on top of the ODM data model results and shows a detailed version of the whole network.

In order to underline the necessity of disposing of such visualization tools, let us first analyze how the visualization is performed in the SUMO simulator.

6.2. SUMO GUI

The SUMO simulator offers a built-in graphical user interface (GUI) which is included in the SUMO suite framework. It is friendly and easy to set up but represents a simple visual interface with very limited real-life functionalities. Thus, it lacks of performant visualization facilities, clarity, complexity and robustness. In addition, it does not support any GPU computational acceleration. A simple zoomed-in visualization of two intersections with traffic lights and vehicles performed with the SUMO GUI is illustrated in Figure 6-1. Here, we can observe that the vehicles that are evolving in the considered region are visible.

Figure 6-1. Zoomed-in visualization with the SUMO GUI

However, if we want to display larger regions of a city, the approach suffers from a complete lack of scalability (Figure 6-2). Thus, all the important elements of information that need to be displayed are no longer visible. In addition, the time needed to perform zoom-in and zoom-out is highly penalizing. In order to present the whole scenario (part of a city or whole city) in a more scalable manner, a different visualization framework needs to be developed. Such a solution should have the ability to be scaled up and down, zoomed in or out without distorting the presented data. For this reason, a scalable web-based interface (web interface 1.0) has been developed.

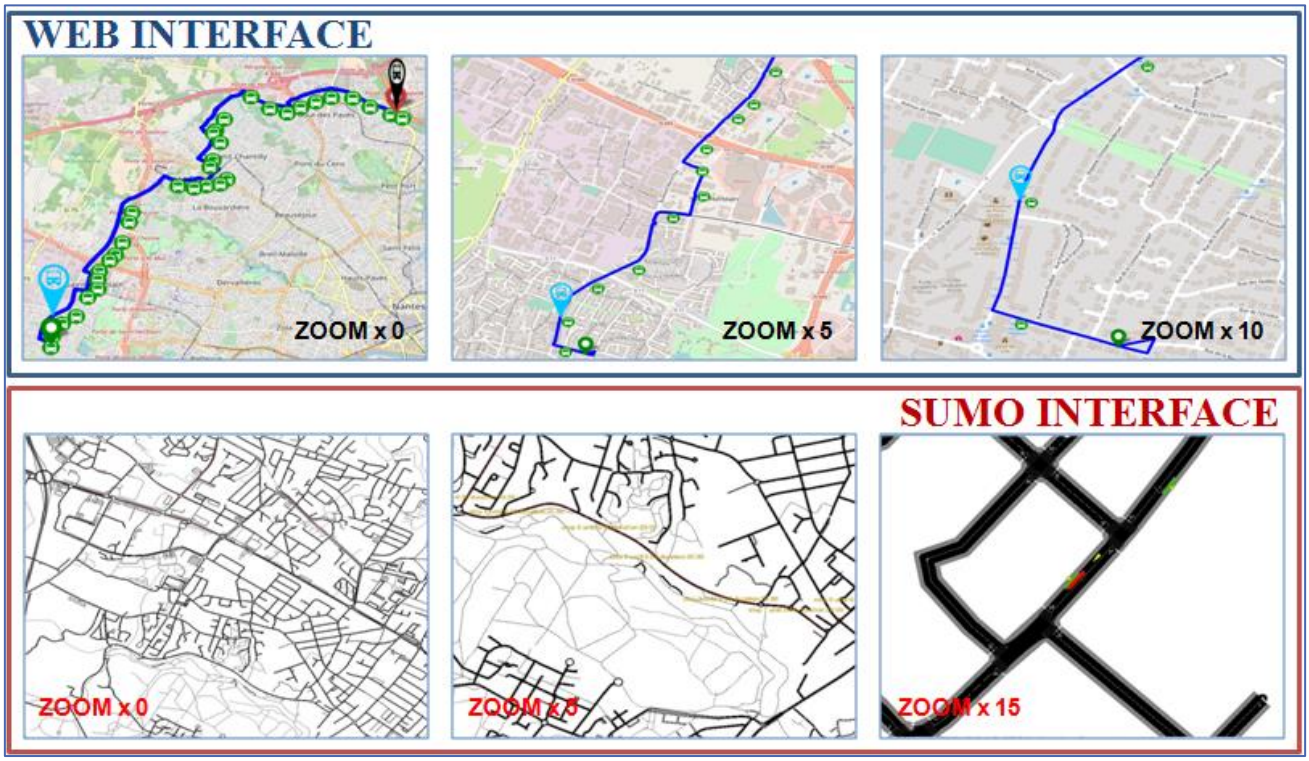


Figure 6-2. Web interface 1.0 *versus* SUMO GUI

Figure 6-2 shows the real difference between SUMO GUI and the proposed interface (so-called *web interface 1.0*). In our case, the images are much more detailed, the zoom in and out are smoothly performed and all the important details remain visible at any scale.

The SUMO GUI is meant to be an effective tool for simple, low-scale visualization purposes, but becomes useless when considering areas at the size of real cities.

Let us now detail the proposed Web interface 1.0.

6.3. Web interface 1.0

The web interface 1.0 is illustrated in Figure 6-3.

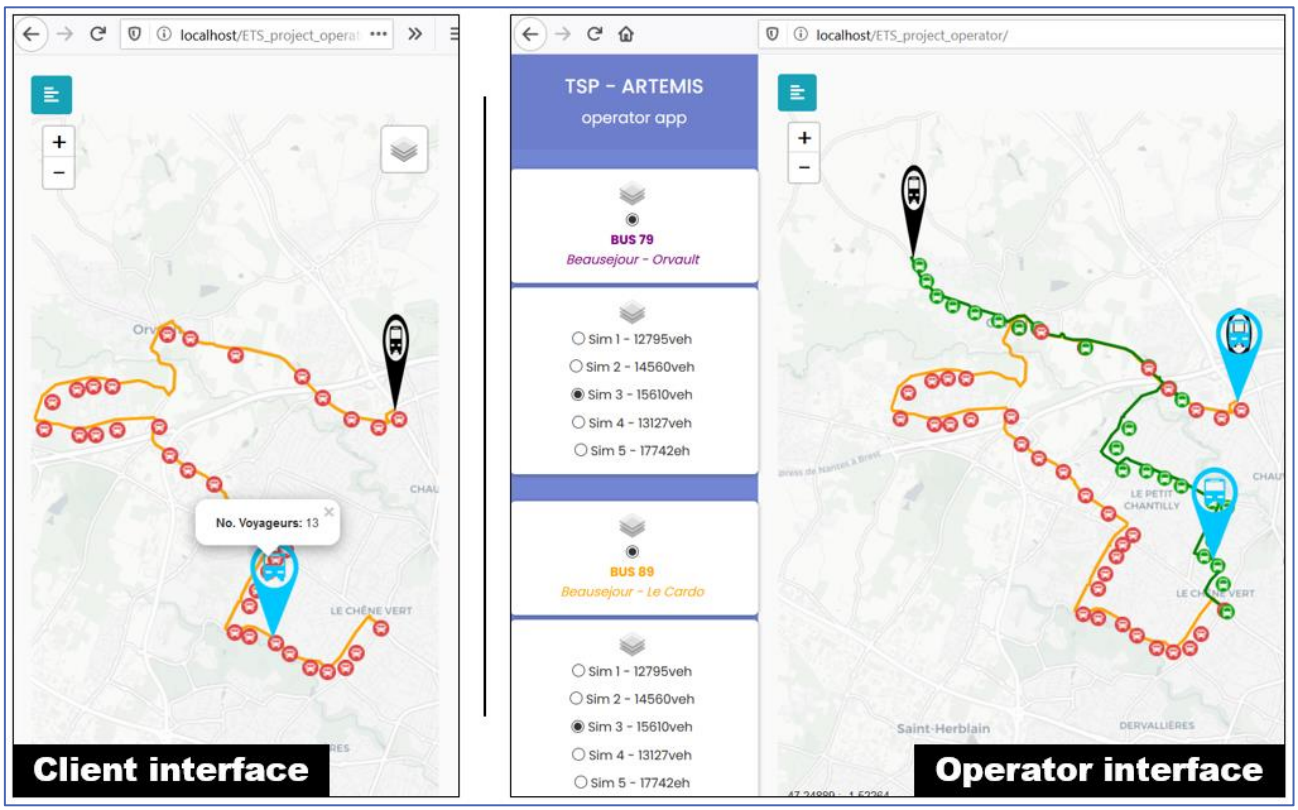


Figure 6-3. Web interface 1.0 (Client - Operator)

Here, the web interface 1.0 for both client and operator side is presented. A close-up illustration is also presented in Figure 6-4. The main functionalities of the proposed visualization framework system include:

- Representation of the full bus itinerary with all the bus stops,
- Display of the number of people per bus stop,
- Display of the number of people inside the bus,
- Interactive triggering of the desired itinerary,

- Traffic conditions analysis.

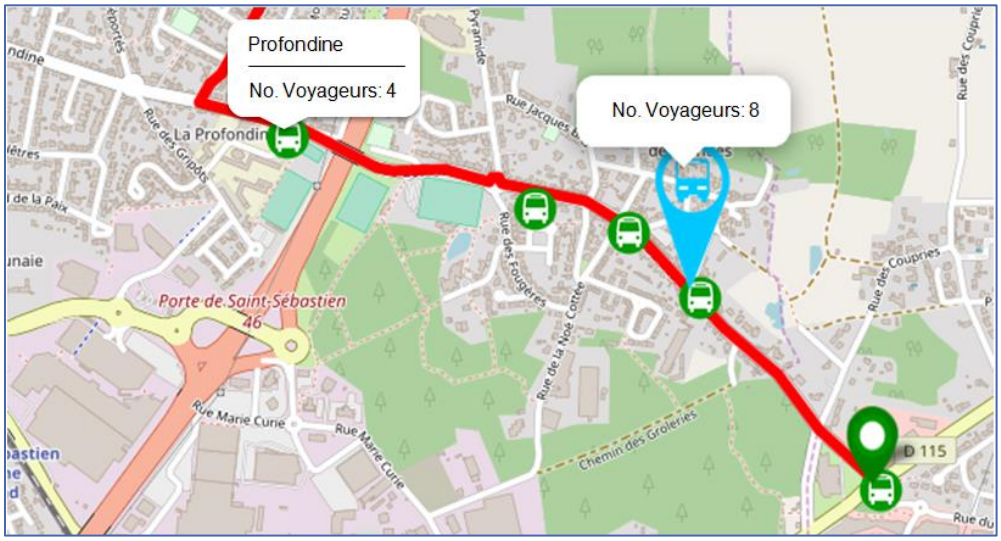


Figure 6-4. Web interface 1.0 (close-up)

In order to develop the web interface, the following tools/utilities have been considered: OpenStreetMap as cartographical representation, JavaScript as a computational library, PHP and Python for backend development and HTML with Bootstrap for scalable web representation.

The following two sub-sections explain in detail the Operator and Client web interfaces separately.

6.3.1. Operator web interface

The visualization platform is developed to be used by both the operator and the client. Let us first provide a more detailed description of the operator side. Let us underline that a transportation operator is a person who is trained to use the platform to make analysis and real-life changes to the whole network and infrastructure if such intervention is needed.

Here are some of the use cases and functionalities implemented.

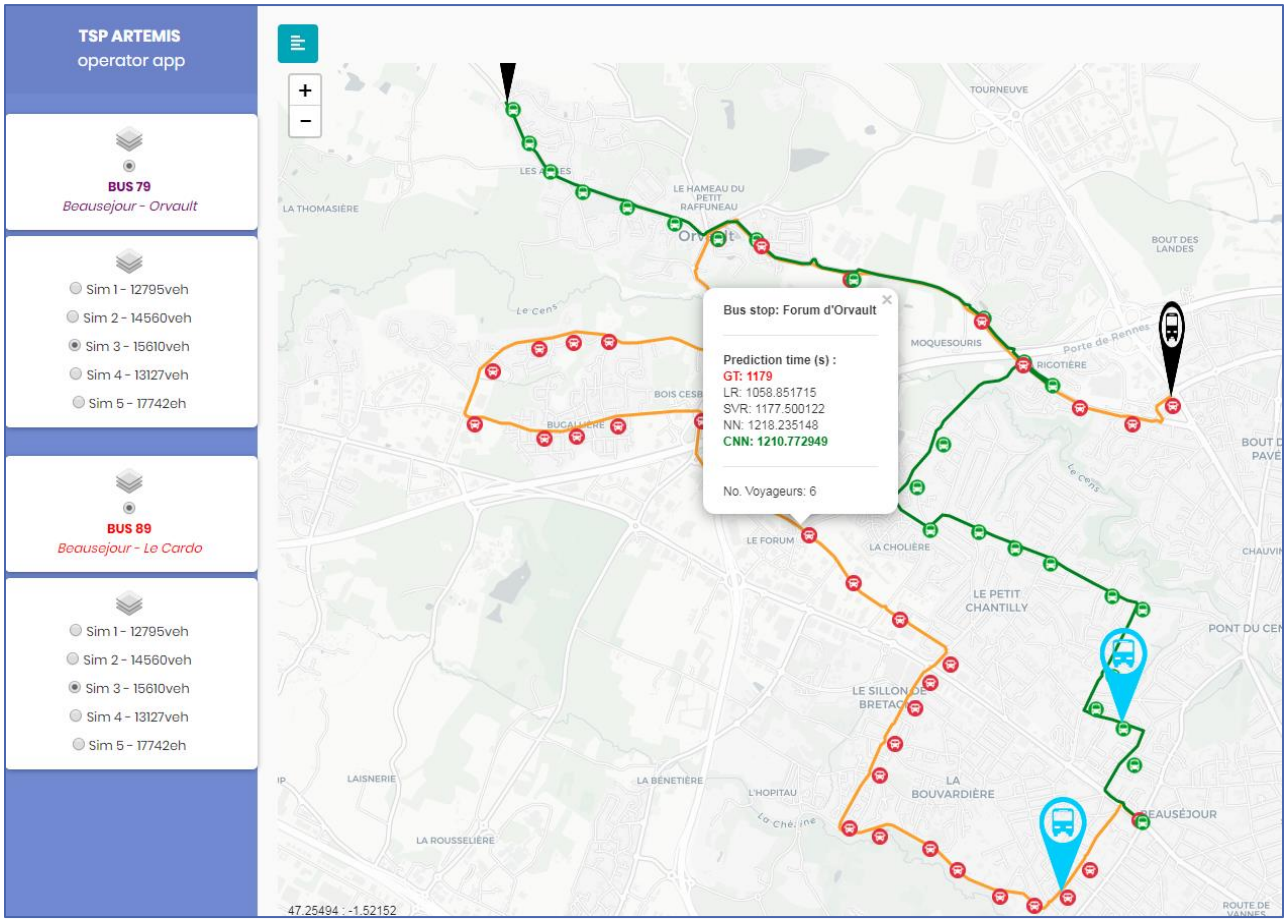


Figure 6-5. operator web interface – analysis window

On the left side of the interface (Figure 6-5) the operator can toggle between different bus lines. Here, the already considered bus lines 79 and 89 of the city of Nantes are available, but additional bus lines can be inserted.

In the main window (on the right) the map with the bus lines will appear. Next, the operator may choose a simulation (bus run) that he wishes to analyze. Clicking on each individual bus stop will pop-up the window with possible (OLS, SVR, FNN, CNN, LSTM) predictions of the bus arrival time at the station, plus the ground truth for comparison.

The pop-up window also shows the potential number of passengers waiting at the bus stop station for the bus to arrive. A click on the bus icon (blue icon), will launch another small pop-up window that indicates the number of passengers inside the bus (Figure 6-6).

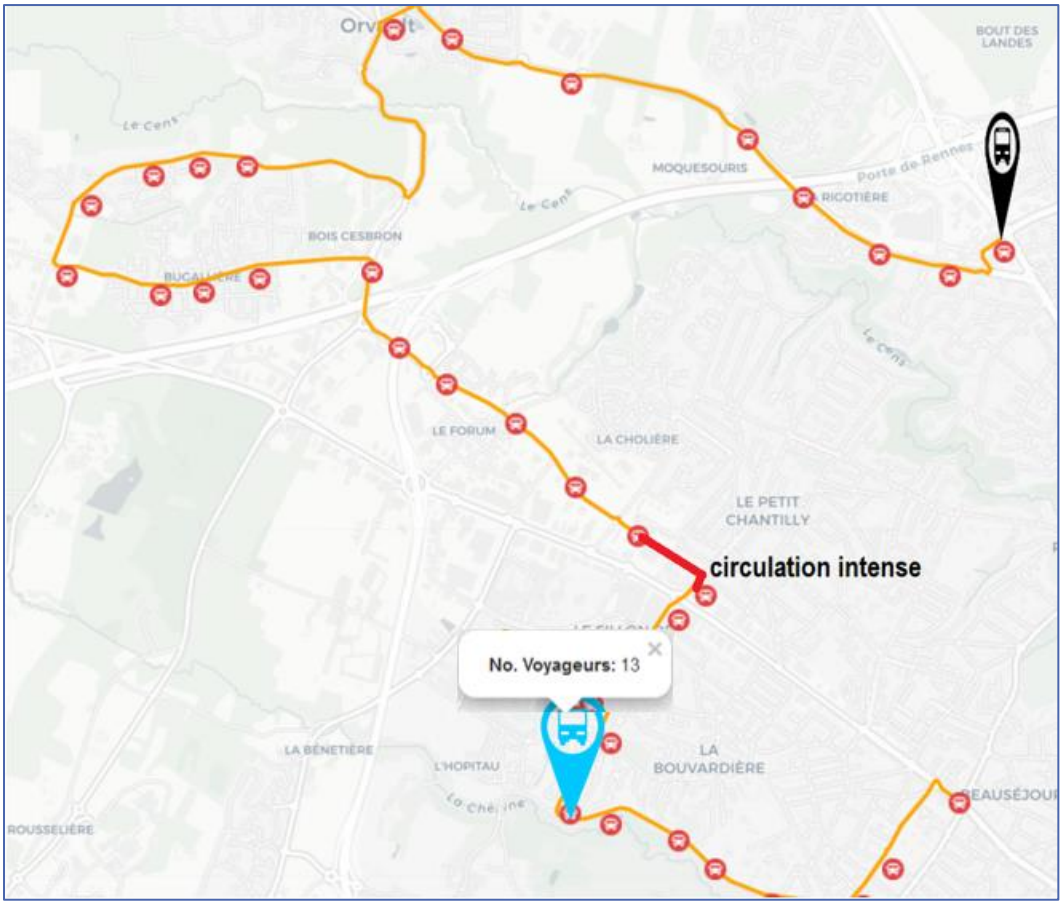


Figure 6-6. Web interface 1.0 – operator side (analysis window)

The final facility in the operator interface is the analysis result window. This window will be triggered after computing the whole simulation and displaying the predicted results. Each bus ride (in this case simulation) will be analyzed and a prompt window will appear as a result of the analyses, as shown in Figure 6-6, which clearly states the potential issue (problem) that arrived on the itinerary.

6.3.2. Client web interface

The client interface should be a scalable and suitable version of the web interface that can be used on varieties of devices, like, PC, Laptop, Tablet, and Smartphone. The user may choose the device that he or she wants to receive the needed information. The main objective is two-fold: (1) displaying the bus position in real-life on the itinerary; and (2) show the predicted bus arrival time at the station where the user is located or at any other (desired) station, as illustrated in Figure 6-7.

.

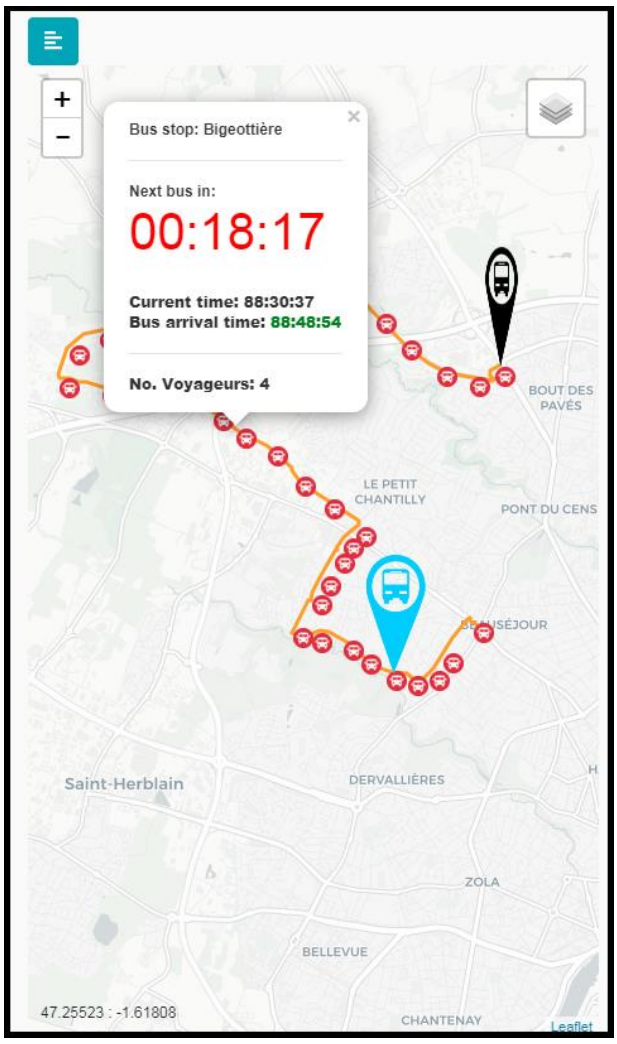


Figure 6-7. Web interface 1.0 – client-side

The visual functionalities of the client-side interface are the following: displaying the name of the bus stop station, bus time (how many minutes/seconds until arrival), predicted bus arrival time (local time), current time and the number of waiting passengers. The main window also displays the current position of the bus on the itinerary and the bus occupancy at that moment.

6.4. Discussion and perspectives

In this chapter, we have proposed a visualization platform dedicated to enhance and improve the final prediction results. This web interface 1.0, can be used as a platform that can host many bus lines in certain urban areas where it is needed. In our work, two bus lines were considered bus lines 79 and 89, so they are included and presented in the proposed visual platform.

The interface was divided into two sub-applications, with client and operator interfaces. This was necessary since their use-cases and target audiences were different. On the client-side, the accent is put on scalability, accessibility and mainly prediction (short-term) on demand. While on the operator-side more functionalities and information about the prediction and causes were introduced, these broader pieces of information can provide a better understanding of the real situation on the considered itineraries.

Different functionalities were developed and included in this visualization platform, namely: bus itineraries with all bus stop stations, the real-time display of the bus location, on-site passengers, bus occupancy rate, scalability on the interface and congestion analysis.

This web interface 1.0 is still in beta, this is more like a proof of concept rather than a full-fledged (finished) product that is ready to be deployed. It is built as a testing platform that may grow up as a real application.

For future functionalities and development, we may consider the following: real-life testing of the platform, introducing more bus lines and web analytics (real-time plots).

Chapter 7. Conclusions and future work

In this PH.D. thesis, we have proposed novel techniques that address a very important subject in today's modern life: improve public transportation and traffic flow management in urban areas by exploring different cost-effective solutions.

Because of the non-availability of real-life traffic data, all the data used in this work was produced in an artificial environment with a traffic simulation program. Thus, in order to develop a real-life scenario an appropriate software (simulation program) is needed. In our work, we have adopted the SUMO traffic simulator, extensively described in Chapter 2 with main advantages, limitations and proposed functionalities. Despite its several limitations, the SUMO platform proved to be well-suited and sufficiently adaptable for the task that we have needed, which concerns the creation of simulation data for public transportation (buses) in urban areas.

Ensuring a good traffic flow management is essential for every modern-day city. In order to achieve this goal, in Chapter 3, we have proposed a traffic lights cycle optimization algorithm, which aims at improving the fluidity of the overall traffic traffic flow. We have shown that optimizing the traffic lights cycle program using bio-inspired optimization techniques such as PSO (*Particle Swarm Optimization*) can directly influence the overall waiting time of the vehicles at intersections and also decrease the time needed by vehicles to cross a specific distance and to complete their journey. The experimental results obtained under various simulated traffic conditions have shown significant improvements, over the baseline STLS (*SUMO Traffic Lights Syste*m) approach, with gains of up to 21.53% and 38.67% in terms of average waiting time and time of completion, respectively.

Next, in Chapter 4 we have introduced the second paradigm that has been addressed in our work, which concerns the public transportation prediction with machine learning algorithms. An extensive overview of the state of the art was here discussed, with a comparison between the most representative and recent researches reported in the field. The analysis of the state of the art showed that deep learning techniques offer today the most promising performances in terms of prediction accuracy. The final part of this chapter was devoted to the introduction of the main theoretical machine learning concepts needed for our developments.

In Chapter 5, we have introduced our main contributions related to public transportation prediction. First, we have set up a simulation scenario close to real-life conditions, which concerns two bus lines (79 and 89) from the city of Nantes, in France. Various parameters and traffic conditions have been here considered. A first analysis of the simulation data has demonstrated that performing a global prediction approach, based on the total number of vehicles inserted in the system is not feasible: the localized traffic jams that can randomly occur in some parts of the city represent singularities that strongly affect the eventual correlations between the time of completion of itinerary and number of vehicles within the system. In order to overcome such a difficulty, the proposed approach is based on a novel concept, so-called Traffic Density Matrix (TDM). Based on a set of measurements performed

at the various bus stops, the TDM captures the situation of the traffic conditions in a local and over-time evolving manner. This structure is further exploited for performing both long-term and short-term prediction. To this purpose, two dedicated data models, co-called ODM (*Operator Data Model*) and CDM (*Client Data Model*) have been respectively constructed from the simulation data. The prediction techniques proposed include both traditional approaches (linear regression, support vector regression) and deep learning techniques, with hand-crafted, dedicated architectures and including fully connected (FNN), convolutional (CNN), recurrent (RNN) and LSTM neural networks. The experimental results obtained show that highly non-linear solutions based on neural networks significantly improve the prediction accuracy both short-term and long-term predictions, with gains in terms of MAE (*Mean Average Error*) scores of up to 15.74% and 66.74% respectively.

Finally, Chapter 6 introduced the visualization platform proposed. Designed to overcome the limitations of the basic SUMO GUI, the so-called Web Interface 1.0 presents dedicated interfaces for both transport operator and end-user (client) modes. The strong advantages of the proposed solution comes from the scalability features, which make it possible to visualize the traffic in the city at arbitrary scales (for the operator mode) and allows smooth integration on various devices (for the client mode). In addition, the various functionalities integrated in the operator mode permit a fine analysis of the events and eventual problems that can occur in the traffic.

The perspectives of future work are numerous and exciting.

First, let us observe that the TDM concept can be used in several different scenarios. Thus, applying the TDM solution at a specific intersection may lead to several possible benefits: real-time monitoring of the density of the vehicles, easy and reliable detection of traffic jams, building a temporal map for future analysis. Introducing TDM on live events like sports or concerts may facilitate crowd movement and discovering the bottlenecks. This may also serve as a good method for emergency exits in times of crisis. The ultimate goal will be to apply the TDM to the scale of the whole city. This may lead to creating detailed past images of the city, with all the entities like vehicles, pedestrians and public transportations. Utilizing this information properly can lead to a system that may learn the patterns and behaviors and propose appropriate responses to various problems. In different words, this can lead to building a global AI solution with the power to predict accidents, identify security risks areas and suggest possible improvements in a specific region.

Some other axes of future research concern, at short term:

- Testing different simulation software environments, creating supplementary scenarios and expanding the current one.
- Using different optimization techniques like GA (genetic algorithms) for traffic lights cycle optimization.
- Enhancing the visualization platform with more tools for analytics and real-time graphs.

At a longer-term, it would be interesting to investigate the possibilities of extension of both the TDM concept and of the related prediction approaches, in terms of additional data that can relate to

population (density of population at semi-localized area, social level…), level of pollution, weather conditions, inclusion of singular events (accidents, cultural or sports events), optimized measurement stations...

Dear reader, this is the end of our journey. My hope is that this manuscript sparks curiosity and provides you with usable knowledge and a better understanding of the topic addressed in this work.

I want to thank you for completing this journey with me.

List of publications

[1] Panovski, D. and Zaharia, T., 2016, November. Simulation-based vehicular traffic lights optimization. In 2016 12th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS) (pp. 258-265). IEEE.

[2] Panovski, D., Scurtu, V. and Zaharia, T., 2018, November. A Neural Network-based Approach for Public Transportation Prediction with Traffic Density Matrix. In 2018 7th European Workshop on Visual Information Processing (EUVIP) (pp. 1-6). IEEE.

[3] Panovski, D., Scurtu, V. and Zaharia, T., 2019, January. Simulation and Prediction of Public Transportation with Maps of Local Density Blobs. In 2019 IEEE International Conference on Consumer Electronics (ICCE) (pp. 1-4). IEEE.

[4] Panovski, D., & Zaharia, T. (2019, December). Public Transportation Prediction with Convolutional Neural Networks. In 3rd International Conference on Intelligent Transport Systems (pp. 150-161). Springer, Cham.

[5] Panovski, D. and Zaharia, T., 2020, January. Real-time Public Transportation Prediction with Machine Learning Algorithms. In 2020 IEEE International Conference on Consumer Electronics (ICCE)

[6] Panovski, D. and Zaharia, T., 2020, January. Machine Learning for Public Transportation Prediction with Traffic Density Matrix. Under submission to IEEE Access Journal

Posters

[1] Panovski, D. and Zaharia, T., 2017, September, November. Simulation, optimization and visualization of transportation data. In 2017 “journée doctorant Samovar”.

[2] Panovski, D., Scurtu, V. and Zaharia, T., 2018, November. A Neural Network-based Approach for Public Transportation Prediction with Traffic Density Matrix. In 2018 7th European Workshop on Visual Information Processing (EUVIP) (pp. 1-6). IEEE.

[3] Panovski, D. and Zaharia, T., 2020, January. Real-time Public Transportation Prediction with Machine Learning Algorithms. In 2020 IEEE International Conference on Consumer Electronics (ICCE)

Simulation, optimization and visualization of transportation data

Author
Dancho PANOVSKI

Ph.D. Director
Titus ZAHARIA

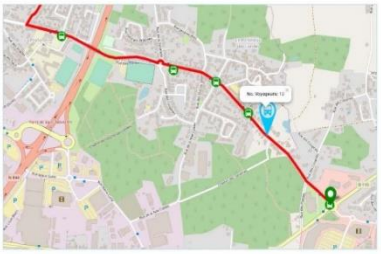
Partners

ETS
Electronic Ticketing System (FUI20 project)

THALES
zenbus

Context and objectives

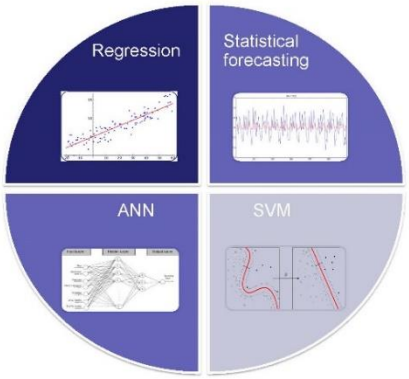
Optimize public transportation in saturated urban areas



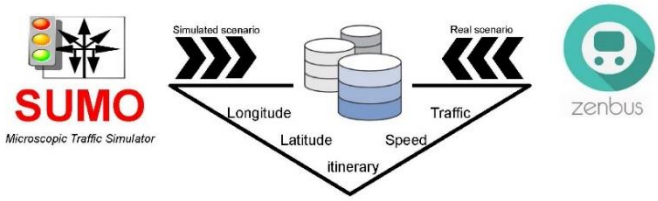
- Availability of passengers geo-localization information with smartphones/electronic ticketing systems
- Develop an *operator application* that can analyze historical data and help for decision
- Elaborate a *client application* integrating **real-time**, **multi-modal traffic forecast/prediction**

Methodological framework

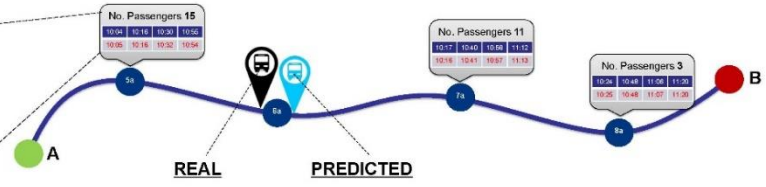
Statistical analysis and learning methods



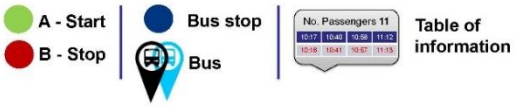
- Set up a generic transportation model that can aggregate both operator and user data
- Data/graph/time series mining for identifying correlations between occupancy rates, traffic conditions, densities of population
- Real-time prediction of occupancy rates and arrival times over sets of itineraries
- Dynamic optimization of traffic lights cycles
- Ensure scalability within a Big/Open Data framework
- Validate the approaches on both synthetic (simulated) and real data



Bus Stop ID	5a	6a	7a	8a
Real Time of Arrival (hh:mm)	10:04	10:09	10:17	10:24
Predicted Arrival Time (hh:mm)	10:05	10:08	10:16	10:25
Real No.Passengers	15	9	11	3
Predicted No.Passengers	16	9	12	5



Example: prediction and forecast



October 2017 Simulation, optimization and visualization of transportation data

Contact {dancho.panovski, titus.zaharia}@telecom-sudparis.eu **Site web**

A Neural Network-based Approach for Public Transportation Prediction with Traffic Density Matrix

Institut Mines-Télécom

Authors

Dancho PANOVSKI

Titus ZAHARIA

Partners

FUI 20 ETS

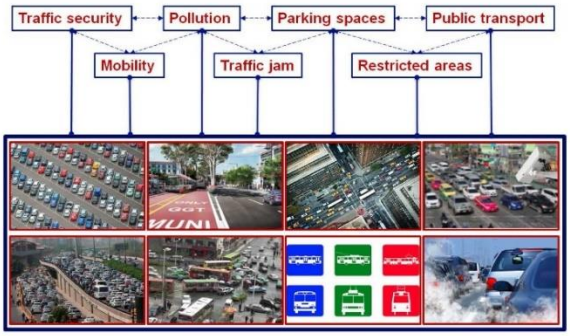
THALES

list

Cliris

zenbus

Context and objectives



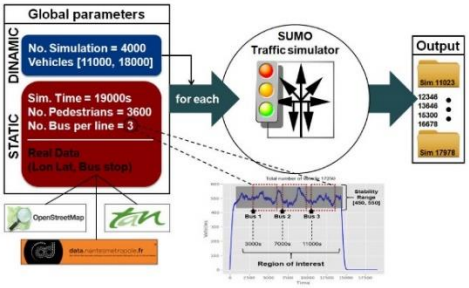
Two target Bus itineraries (city of Nantes)

BUS 79: Ovrault/Beausejour
26 bus stops, 9.64 km

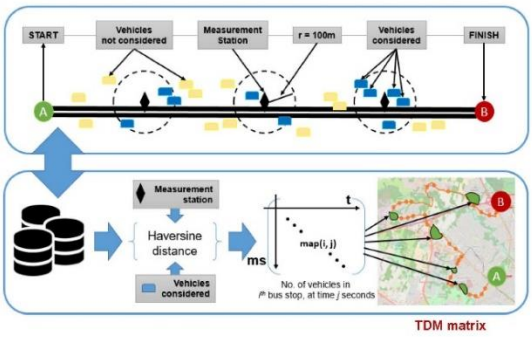
BUS 89: Le Cardo/Beausejour
36 bus stops, 13.4 km

➢ Predict the bus arrival time in bus stations given the global traffic conditions

Simulation and data development

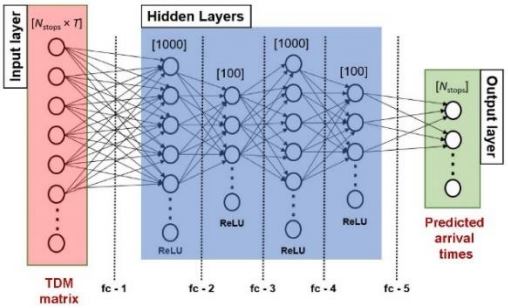


Concept: Traffic Density Matrix



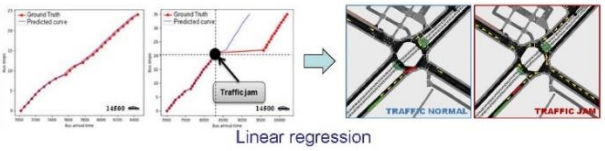
TDM matrix

Proposed approach: Neural Network

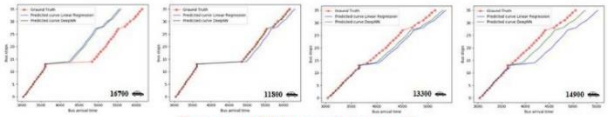


- ❑ 5 layer fully connected NN
- ❑ 4 hidden layers, size
- ❑ (1000 – 100 – 1000 – 100)
- ❑ Activation function: ReLU
- ❑ Loss function: MAE
- ❑ Objective function: SGD
- ❑ No. of training epoch 400

Prediction results



Linear regression



Proposed NN-based approach

	Linear Regression	Neural Network
MAE (seconds)	101.91	53.4

Contact dancho.panovski@telecom-sudparis.eu

Site web http://samovar.telecom-sudparis.eu/

Real-Time Transportation Prediction with Machine Learning Algorithms

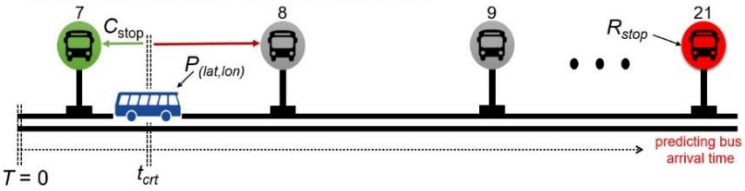
Auteurs

Dancho PANOVSKI
Titus ZAHARIA

Partenaires

FUI 20 ETS
THALES
list
Cliris
zenbus

CONTEXT AND OBJECTIVES

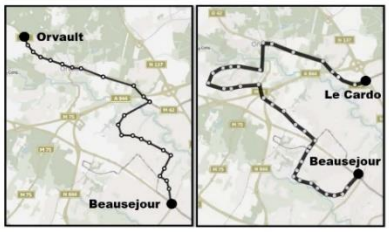


- C_{stop} – Closest bus stop
- $P_{(lat,lon)}$ – Bus position
- t_{crt} – Current time
- R_{stop} – Predicted bus stop
- – Bus stop

Objective:

Predict the bus arrival time in bus stations given the current bus position and traffic density

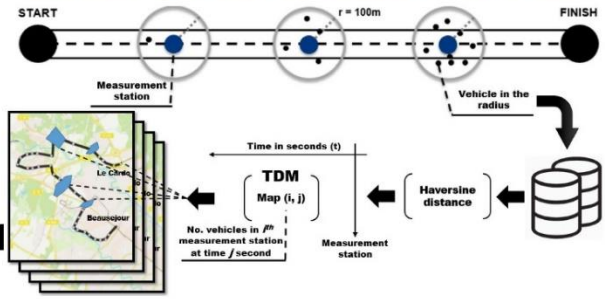
TWO BUS ITINERARIES (city of Nantes)



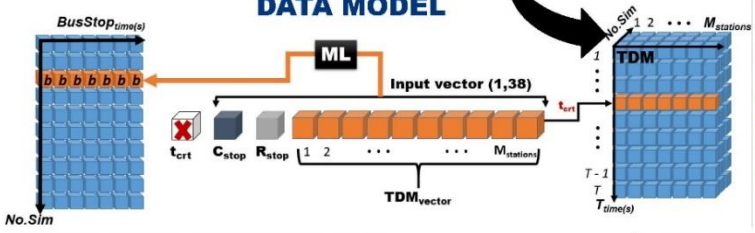
BUS LINE 79
Ovrault <> Beausejour
No. bus stops: 25
Distance: 9.64 km

BUS LINE 89
Le Cardo <> Beausejour
No. bus stops: 36
13.4 km

CONCEPT: TRAFFIC DENSITY MATRIX



DATA MODEL



SIMULATION AND DATA

$$12000_{\ sim} \times 5R_{stops} \times 10_{runs} = 600000_{inputs}$$

- 12 000 simulations
- 5 bus stops (R_{stop})
- 10 runs per bus stop
- 600 000 inputs
- Train dataset 80 %
- Test dataset 20 %
  (from witch 10 % validation)
- 10 – fold cross val.

PROPOSED APPROACHES

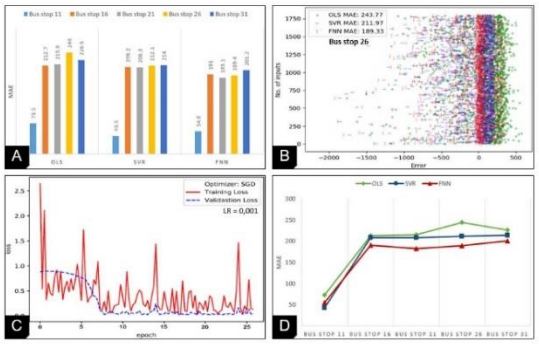
- OLS (*Ordinary Least Squares*)
- SVR (*Support Vector Repressors*)
- FNN (*Feedforward Fully Connected Neural Network*)

MACHINE LEARNING ALGORITHMS PERFORMANCE

	11	16	21	26	31	gMAE	cTIME
OLS	73,5	212,7	215,9	244,0	226,5	**194,52**	**43**
SVR	43,5	209,2	208,3	212,1	214	**177,42**	**2985**
FNN	54,8	191	183,1	189,4	201,2	**163,9**	**399**

gMAE = Global Mean Absolute Error, cTIME = Computational Time in (s), OLS = Ordinary Least Squares, SVR = Support Vector Regression, FNN = Feedforward Fully connected Neural Network, (11, 16, 21, 26, 31) = Bus Stop

RESULTS



Contact dancho.panovski@telecom-sudparis.eu **Site web** www.teleocm-sudparis.eu

INTSYS conference
December 2019

References

[1] “What is a Smart City? Definition from WhatIs.com.” [Online]. Available: https://internetofthingsagenda.techtarget.com/definition/smart-city. [Accessed: 20-Dec-2019].

[2] “The Return on Investment (ROI) of the UK Smart City Initiative.” [Online]. Available: https://advancecities.com/blog/how-do-smart-cities-in-the-uk-measure-their-return-on-investment-roi/. [Accessed: 22-Oct-2020].

[3] “What is a Smart City? - Definition from Techopedia.” [Online]. Available: https://www.techopedia.com/definition/31494/smart-city. [Accessed: 04-Jul-2020].

[4] O. Adedayo, “What should be the definition of Smart City?” 2015.

[5] S. An, B.-H. Lee, and D.-R. Shin, “A Survey of Intelligent Transportation Systems,” in *2011 Third International Conference on Computational Intelligence, Communication Systems and Networks*, 2011, pp. 332–337.

[6] “Intelligent Transportation System - Topic - Digital Journal.” [Online]. Available: http://www.digitaljournal.com/topic/Intelligent Transportation System. [Accessed: 22-Oct-2020].

[7] L. Qi, “Research on Intelligent Transportation System Technologies and Applications,” in *2008 Workshop on Power Electronics and Intelligent Transportation System*, 2008, pp. 529–531.

[8] N.-E. El Faouzi, “Data fusion in intelligent transportation systems: Progress and challenges – A survey,” *Inf. Fusion*, vol. 12, no. 1, pp. 4–10, Jan. 2011.

[9] R. Ghosh, R. Pragathi, S. Ullas, and S. Borra, “Intelligent transportation systems: A survey,” in *2017 International Conference on Circuits, Controls, and Communications (CCUBE)*, 2017, pp. 160–165.

[10] “I on the framework for the deployment of Intelligent Transport Systems in the field of road transport and for interfaces with other modes of transport (Text with EEA relevance),” 2010.

[11] SOeS, “mars 2015 Chiffres clés du transport.” [Online] Avalible: www.statistiques.developpement-durable.gouv.fr [Accessed: 28-Oct-2020].

[12] “Where Experiences are Engineered.” [Online]. Available: https://www.valtech.com/sv-se/?gclid=Cj0KCQjwreT8BRDTARIsAJLI0KKuRdZGLzwE6lzioSQDGMbPpNH-ulEtalcF5fdzAfzMFlW4375CLlwaAoMdEALw_wcB. [Accessed: 28-Oct-2020].

[13] “城市规划 Archives - 中外对话.” [Online]. Available: https://chinadialogue.net/zh/tag/urban-planning-zh/. [Accessed: 22-Oct-2020].

[14] H. Kallberg, “Traffic simulation,” no. Licentiate thesis, Helsinki University of Technology, Transportation Engineering. Espoo, 1971.

[15] J. Hueper, G. Dervisoglu, A. Muralidharan, G. Gomes, R. Horowitz, and P. Varaiya, “Macroscopic Modeling and Simulation of Freeway Traffic Flow,” *IFAC Proc. Vol.*, vol. 42, no. 15, pp. 112–116, Jan. 2009.

[16] M. Behrisch, L. Bieker, and J. Erdmann, “SUMO–simulation of urban mobility: an overview,” *Proc. SIMUL*, 2011.

[17] P. A. Lopez *et al.*, “Microscopic Traffic Simulation using SUMO,” in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2018, vol. 2018-November, pp. 2575–2582.

[18] M. Balmer and M. Rieser, “MATSim-T: Architecture and simulation times,” *Multi-Agent Syst. Traffic Transp. Eng.*, 2009.

[19] North, M.J., Tatara, E., Collier, N.T. and Ozik, J., 2007. Visual agent-based model development with repast simphony (pp. 1-20). Tech. rep., Argonne National Laboratory.

[20] J. Dallmeyer and I. Timm, “MAINSIM–MultimodAl INnercity SIMulation,” *35th Ger. Conf. Artif. Intell.*, 2012.

[21] Paramics, Q., 2009. The paramics manuals, version 6.6. 1. Quastone Paramics LTD, Edinburgh, Scotland, UK.

[22] J. Li and L. Yu, “A Visum-based method for determining road network capacity,” *Urban Transp. China*, 2006.

[23] M. Wei, F. Yang, and Z. Cao, “A Review of Development and Study on the Traffic Simulation,” *Acta Simulata Syst. Sin.*, 2003.

[24] M. Fellendorf and P. Vortisch, “Validation of the microscopic traffic flow model VISSIM in different real-world situations,” *Transp. Res. Board 80th Annu.*, 2001.

[25] J. Casas, J. Ferrer, D. Garcia, and J. Perarnau, “Traffic simulation with aimsun,” *Fundam. Traffic*, 2010.

[26] D. Van Vliet, “SATURN-a modern assignment model,” *Traffic Eng. Control*, 1982.

[27] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, “SUMO - Simulation of Urban MObility - an Overview,” *Proc. 3rd Int. Conf. Adv. Syst. Simul.*, no. c, pp. 63–68, 2011.

[28] “SUMO Documentation.” [Online]. Available: https://sumo.dlr.de/docs/index.html. [Accessed: 23-Jul-2020].

[29] D. Krajzewicz, G. Hertkorn, and C. Rössel, “SUMO (Simulation of Urban MObility)-an open-source traffic simulation,” *Proc. 4th*, 2002.

[30] “Simulation of Urban MObility - Browse /sumo at SourceForge.net.” [Online]. Available: https://sourceforge.net/projects/sumo/files/sumo/.

[31] D. Krajzewicz, J. Erdmann, and M. Behrisch, “Recent development and applications of SUMO–simulation of urban mobility,” *Int. J.*, 2012.

[32] M. Fellendorf and P. Vortisch, “Microscopic traffic flow simulator VISSIM,” *Fundam. traffic Simul.*, 2010.

[33] M. Dupuis, M. Strobl, and H. Grezlikowski, “OpenDRIVE 2010 and Beyond -Status and future of the de facto standard for the description of road networks.” 2010.

[34] J. Jokar Arsanjani, A. Zipf, P. Mooney, and M. Helbich, “An Introduction to OpenStreetMap in Geographic Information Science: Experiences, Research, and Applications,” 2015, pp. 1–15.

[35] E. W. Dijkstra, “A Note on Two Problems in Connexion with Graphs,” *Numer. Math.* ~, pp. 269–271, 1959.

[36] W. Zeng and R. L. Church, “Finding shortest paths on real road networks: the case for A*,” *Int. J. Geogr. Inf. Sci.*, vol. 23, no. 4, pp. 531–543, Apr. 2009.

[37] R. Geisberger, P. Sanders, and D. Schultes, “Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks,” 2008.

[38] “Simulation/Traffic Lights - Sumo.” [Online]. Available: http://sumo.dlr.de/wiki/Simulation/Traffic_Lights.

[39] D. Krajzewiczt, M. Bonertt, and P. Wagnert, “The Open Source Traffic Simulation Package SUMO.”

[40] A. Stevanovic, “Adaptive Traffic Control Systems: Domestic and Foreign State of Practice,” *Transportation (Amst).*, p. 114, 2010.

[41] D. Zhao, S. Member, Y. Dai, and Z. Zhang, “Computational Intelligence in Urban Traffic Signal Control : A Survey,” *IEEE Trans. Syst. Man, Cybern.*, vol. 42, no. 4, pp. 485–494, 2012.

[42] K. N. Hewage and J. Y. Ruwanpura, “Optimization of traffic signal light timing using simulation,” in *Proceedings - Winter Simulation Conference*, 2004, vol. 2, pp. 1428–1433.

[43] C. Tolba, D. Lefebvre, P. Thomas, and A. El Moudni, “Continuous and timed Petri nets for the macroscopic and microscopic traffic flow modelling,” *Simul. Model. Pract. Theory*, vol. 13, no. 5, pp. 407–436, 2005.

[44] P. Lowrie, “Scats, sydney co-ordinated adaptive traffic system: A traffic responsive method of controlling urban traffic,” Roads and Traffic Authority NSW, Darlinghurst, NSW Australia, 1990.

[45] P. Hunt, D. Robertson, R. Bretherton, and R. Winton, “SCOOT-a traffic responsive method of coordinating signals,” Transport and Road Research Laboratory (TRRL), 1981.

[46] W. Ming, Y. Qin, and J. Xu, “The Application of SCOOT in Modern Traffic Network,” *Manag. Eng.*, 2015.

[47] A. Pascale, T. Hoang, and R. Nair, “Characterization of network traffic processes under adaptive traffic control systems,” *Transp. Res. Part C Emerg.*, 2015.

[48] B. De Schutter and B. De Moor, “Optimal traffic light control for a single intersection,” *Eur. J. Control*, 1998.

[49] C. Karakuzu and O. Demirci, “Fuzzy logic based smart traffic light simulator design and hardware implementation,” *Appl. Soft Comput.*, 2010.

[50] M. Wiering, J. Vreeken, and J. Van Veenen, “Simulation and optimization of traffic in a city,” *Veh. Symp. …*, 2004.

[51] M. Wiering, J. Van Veenen, J. Vreeken, and A. Koopman, “Intelligent Traffic Light Control,” 2004.

[52] L. Kuyer, S. Whiteson, B. Bakker, and N. Vlassis, “Multiagent reinforcement learning for urban traffic control using coordination graphs,” *Jt. Eur. Conf.*, 2008.

[53] J. Sanchez, M. Galan, and E. Rubio, “Genetic algorithms and cellular automata: A new architecture for traffic light cycles optimization,” *Comput. 2004. CEC2004. …*, 2004.

[54] S. Wolfram, *Theory and applications of cellular automata*. 1986.

[55] S. Haykin and N. Network, “A comprehensive foundation,” *Neural Networks*, 2004.

[56] P. Ramesh Sharda, S. Voß, and M. Brown, “Genetic Algorithms: Principles and Perspectives A Guide to GA Theory,” *Impact Emerg. Technol. Comput. Sci. Oper. Res. Barth / Interfaces Comput. Sci. Oper. Res. Adv. Metaheuristics*.

[57] K. Teo, W. Kow, and Y. Chin, “Optimization of traffic flow within an urban traffic light intersection with genetic algorithm,” *2010 Second Int.*, 2010.

[58] J. Sánchez-Medina and M. Galán-Moreno, “Traffic signal optimization in ‘La Almozara’ district in Saragossa under congestion conditions, using genetic algorithms, traffic microsimulation, and cluster computing,” *IEEE Trans.*, 2010.

[59] H. Kwasnicka and M. Stanek, “Genetic Approach to Optimize Traffic Flow by Timing Plan Manipulation,” *Sixth Int. Conf.*, 2006.

[60] L. Singh, S. Tripathi, and H. Arora, “Time optimization for traffic signal control using genetic algorithm,” *Int. J. Recent Trends*, 2009.

[61] A. Turky, M. Ahmad, and M. Yusoff, “Using genetic algorithm for traffic light control system with a pedestrian crossing,” *Conf. Rough …*, 2009.

[62] J. Kennedy, J. Kennedy, R. Eberhart, and Y. Shi, *Swarm intelligence*. 2001.

[63] R. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” *sixth Int. Symp. …*, 1995.

[64] Y. Shi and R. Eberhart, “A modified particle swarm optimizer,” *Proceedings, 1998. IEEE World Congr. …*, 1998.

[65] J. Garcia-Nieto, A. C. A. Olivera, and E. Alba, “Optimal cycle program of traffic lights with particle swarm optimization,” *IEEE Trans. Evol. Comput.*, vol. 17, no. 6, pp. 823–839, Dec. 2013.

[66] N. Baba, “Convergence of a random optimization method for constrained optimization problems,” *J. Optim. Theory Appl.*, 1981.

[67] K. Price, R. Storn, and J. Lampinen, *Differential evolution: a practical approach to global optimization*. 2006.

[68] I. Wijaya and K. Uchimura, “Traffic light signal parameters optimization using particle swarm optimization,” *Intell. Technol.*, 2015.

[69] R. Abushehab and B. Abdalhaq, “Genetic vs. particle swarm optimization techniques for traffic light signals timing,” *Comput. Sci.*, 2014.

[70] J. Barros, M. Araujo, and R. J. F. Rossetti, “Short-term real-time traffic prediction methods: A survey,” in *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, 2015, pp. 132–139.

[71] F. Su, H. Dong, L. Jia, Y. Qin, and Z. Tian, “Long-term forecasting oriented to urban expressway traffic situation,” *Adv. Mech. Eng.*, vol. 8, no. 1, p. 168781401662839,

Jan. 2016.

[72] M. Ben-Akiva, M. Bierlaire, H. Koutsopoulos, and R. Mishalani, “DynaMIT: a simulation-based system for traffic prediction,” 1998.

[73] T. Schwerdtfeger, “DYNEMO : A MODEL FOR THE SIMULATION OF TRAFFIC FLOW IN MOTORWAY NETWORKS,” 1984.

[74] M. Rickert and K. Nagel, “Dynamic traffic assignment on parallel computers in TRANSIMS,” *Futur. Gener. Comput. Syst.*, vol. 17, no. 5, pp. 637–648, Mar. 2001.

[75] A. K. Ziliaskopoulos and S. T. Waller, “An Internet-based geographic information system that integrates data, models and users for transportation applications,” *Transp. Res. Part C Emerg. Technol.*, vol. 8, no. 1–6, pp. 427–444, Feb. 2000.

[76] L. Mannini, S. Carrese, E. Cipriani, and U. Crisalli, “On the Short-term Prediction of Traffic State: An Application on Urban Freeways in ROME,” *Transp. Res. Procedia*, vol. 10, pp. 176–185, Jan. 2015.

[77] W. Zheng, D.-H. Lee, and Q. Shi, “Short-Term Freeway Traffic Flow Prediction: Bayesian Combined Neural Network Approach,” *J. Transp. Eng.*, vol. 132, no. 2, pp. 114–121, Feb. 2006.

[78] Z. Huang, Q. Li, F. Li, and J. Xia, “A Novel Bus-Dispatching Model Based on Passenger Flow and Arrival Time Prediction,” *IEEE Access*, vol. 7, pp. 106453–106465, 2019.

[79] K. Friso, L. J. J. Wismans, and M. B. Tijink, “Scalable data-driven short-term traffic prediction,” in *2017 5th IEEE International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, 2017, pp. 687–692.

[80] D. Solomatine, L. M. See, and R. J. Abrahart, “Data-Driven Modelling: Concepts, Approaches and Experiences,” in *Practical Hydroinformatics*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 17–30.

[81] B. Ghosh, B. Basu, and M. O’Mahony, “Multivariate Short-Term Traffic Flow Forecasting Using Time-Series Analysis,” *IEEE Trans. Intell. Transp. Syst.*, vol. 10,

no. 2, pp. 246–254, Jun. 2009.

[82] J. Guo and B. Williams, “Real-Time Short-Term Traffic Speed Level Forecasting and Uncertainty Quantification Using Layered Kalman Filters,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 2175, pp. 28–37, Dec. 2010.

[83] M. Karimpour, A. Karimpour, K. Kompany, and A. Karimpour, “Online Traffic Prediction Using Time Series: A Case study,” in *Integral Methods in Science and Engineering, Volume 2*, Cham: Springer International Publishing, 2017, pp. 147–156.

[84] K. Y. Chan, T. S. Dillon, J. Singh, and E. Chang, “Traffic flow forecasting neural networks based on exponential smoothing method,” in *2011 6th IEEE Conference on Industrial Electronics and Applications*, 2011, pp. 376–381.

[85] Z. H. Mir and F. Filali, “An adaptive Kalman filter based traffic prediction algorithm for urban road network,” in *2016 12th International Conference on Innovations in Information Technology (IIT)*, 2016, pp. 1–6.

[86] N. Zhang, Y. Zhang, and H. Lu, “Seasonal Autoregressive Integrated Moving Average and Support Vector Machine Models,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 2215, pp. 85–92, Dec. 2011.

[87] D. Panovski, V. Scurtu, and T. Zaharia, “A Neural Network-based Approach for Public Transportation Prediction with Traffic Density Matrix,” in *2018 7th European Workshop on Visual Information Processing (EUVIP)*, 2018, pp. 1–6.

[88] C. P. VAN HINSBERGEN, J. W. VAN LINT, and F. M. SANDERS, “Short Term Traffic Prediction Models,” *Proc. 14TH WORLD Congr. Intell. Transp. Syst. (ITS), HELD BEIJING, Oct. 2007*, 2007.

[89] J. XUE and Z. SHI, “Short-Time Traffic Flow Prediction Based on Chaos Time Series Theory,” *J. Transp. Syst. Eng. Inf. Technol.*, vol. 8, no. 5, pp. 68–72, Oct. 2008.

[90] H. Sun, H. Liu, H. Xiao, R. He, and B. Ran, “Use of Local Linear Regression Model for Short-Term Traffic Forecasting,” *Transp. Res. Rec. J. Transp. Res. Board*, vol. 1836, pp. 143–150, Jan. 2003.

[91] S. V. Kumar and L. Vanajakshi, “Short-term traffic flow prediction using seasonal ARIMA model with limited input data,” *Eur. Transp. Res. Rev.*, vol. 7, no. 3, p. 21, Sep. 2015.

[92] L. Zhu, F. R. Yu, Y. Wang, B. Ning, and T. Tang, “Big Data Analytics in Intelligent Transportation Systems: A Survey,” *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 1, pp. 383–398, Jan. 2019.

[93] Z. M. Fadlullah *et al.*, “State-of-the-Art Deep Learning: Evolving Machine Intelligence Toward Tomorrow’s Intelligent Network Traffic Control Systems,” *IEEE Commun. Surv. Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.

[94] Y. Meng and X. Liu, “Application of K-means Algorithm Based on Ant Clustering Algorithm in Macroscopic Planning of Highway Transportation Hub,” in *2007 First IEEE International Symposium on Information Technologies and Applications in Education*, 2007, pp. 483–488.

[95] Li, Li, Yisheng Lv, and Fei-Yue Wang. "Traffic signal timing via deep reinforcement learning." IEEE/CAA Journal of Automatica Sinica 3, no. 3 (2016): 247-254.

[96] G. A. F. (George A. F. Seber, A. J. Lee, and Wiley InterScience (Online service), *Linear regression analysis*. Wiley-Interscience, 2003.

[97] Welling, Max. "Support vector regression." Department of Computer Science, University of Toronto, Toronto (Kanada) (2004).

[98] M. Yang, C. Chen, L. Wang, X. Yan, and L. Zhou, “BUS ARRIVAL TIME PREDICTION USING SUPPORT VECTOR MACHINE WITH GENETIC ALGORITHM.”

[99] T. Yin, G. Zhong, J. Zhang, S. He, and B. Ran, “ScienceDirect A prediction model of bus arrival time at stops with multi-routes-review under responsibility of WORLD CONFERENCE ON TRANSPORT RESEARCH SOCIETY,” *Transp. Res. Procedia*, vol. 25, pp. 10–15, 2017.

[100] B. Yu, Y. L. Jiang, B. Yu, and Z. Z. Yang, “Application of support vector machines in bus travel time prediction,” *Dalian Haishi Daxue Xuebao/Journal Dalian Marit. Univ.*, vol. 34, no. 4, pp. 158–160, 2008.

[101] X. Ma, Z. Dai, Z. He, J. Na, Y. Wang, and Y. Wang, “Learning Traffic as Images: A Deep Convolutional Neural Network for Large-Scale Transportation Network Speed Prediction,” Jan. 2017.

[102] B. Yu, H. Wang, W. Shan, and B. Yao, “Prediction of Bus Travel Time Using Random Forests Based on Near Neighbors,” *Comput. Civ. Infrastruct. Eng.*, vol. 33, no. 4, pp. 333–350, Apr. 2018.

[103] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015.

[104] P. Wang, G. Zhao, and X. Yao, “Applying back-propagation neural network to predict bus traffic,” in *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 2016, pp. 752–756.

[105] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang, “Traffic Flow Prediction With Big Data: A Deep Learning Approach,” *IEEE Trans. Intell. Transp. Syst.*, pp. 1–9, 2014.

[106] L. M. V and S. K. Devi J, “AN ARCHITECTURE OF DEEP LEARNING METHOD TO PREDICT TRAFFIC FLOW IN BIG DATA,” 2016.

[107] R. Yu, Y. Li, C. Shahabi, U. Demiryurek, and Y. Liu, “Deep learning: A generic approach for extreme condition traffic forecasting,” in *Proceedings of the 17th SIAM International Conference on Data Mining, SDM 2017*, 2017, pp. 777–785.

[108] Y. Li, R. Yu, C. Shahabi, and Y. Liu, “Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting,” Jul. 2017.

[109] R. Fu, Z. Zhang, and L. Li, “Using LSTM and GRU neural network methods for traffic flow prediction,” in *Proceedings - 2016 31st Youth Academic Annual Conference of Chinese Association of Automation, YAC 2016*, 2017, pp. 324–328.

[110] P. He, G. Jiang, S. K. Lam, and D. Tang, “Travel-Time Prediction of Bus Journey with Multiple Bus Trips,” *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 11, pp. 4192–4205, Nov. 2019.

[111] J. Pan, X. Dai, X. Xu, and Y. Li, “A Self-learning algorithm for predicting bus arrival

time based on historical data model," in *Proceedings - 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, IEEE CCIS 2012*, 2013, vol. 3, pp. 1112–1116.

[112] W. Treethidtaphat, W. Pattara-Atikom, and S. Khaimook, "Bus arrival time prediction at any distance of bus route using deep neural network model," in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2018, vol. 2018-March, pp. 988–992.

[113] C. T. Lam, B. Ng, and S. H. Leong, "Prediction of Bus Arrival Time Using Real-Time on-Line Bus Locations," in *International Conference on Communication Technology Proceedings, ICCT*, 2019, pp. 473–478.

[114] X. Zhang and Z. Liu, "Prediction of bus arrival time based on gps data: Taking no. 6 bus in huangdao district of qingdao city as an example," in *Chinese Control Conference, CCC*, 2019, vol. 2019-July, pp. 8789–8794.

[115] J. Liu and G. Xiao, "Efficient bus arrival time prediction based on spark streaming platform," in *Proceedings of the 2019 IEEE 23rd International Conference on Computer Supported Cooperative Work in Design, CSCWD 2019*, 2019, pp. 416–421.

[116] H. Bohan and B. Yun, "Short-term traffic flow prediction based on bus floating car," in *ICEIEC 2019 - Proceedings of 2019 IEEE 9th International Conference on Electronics Information and Emergency Communication*, 2019, pp. 303–306.

[117] C. Heghedus, A. Chakravorty, and C. Rong, "Neural network frameworks. Comparison on public transportation prediction," in *Proceedings - 2019 IEEE 33rd International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2019*, 2019, pp. 842–849.

[118] Y. Wang, D. Zhang, Y. Liu, B. Dai, and L. H. Lee, "Enhancing transportation systems via deep learning: A survey," *Transportation Research Part C: Emerging Technologies*, vol. 99. Elsevier Ltd, pp. 144–163, 01-Feb-2019.

[119] Y. Wu and H. Tan, "Short-term traffic flow forecasting with spatial-temporal correlation in a hybrid deep learning framework," Dec. 2016.

[120] Y. Wu, H. Tan, L. Qin, B. Ran, and Z. Jiang, “A hybrid deep learning based traffic flow prediction method and its understanding,” *Transp. Res. Part C Emerg. Technol.*, vol. 90, pp. 166–180, May 2018.

[121] R. Yu, Y. Li, C. Shahabi, U. Demiryurek, and Y. Liu, *Deep Learning: A Generic Approach for Extreme Condition Traffic Forecasting*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2017, pp. 777–785.

[122] T. Yamaguchi, M. As, and T. Mine, “Prediction of Bus Delay over Intervals on Various Kinds of Routes Using Bus Probe Data,” in *Proceedings - 5th IEEE/ACM International Conference on Big Data Computing, Applications and Technologies, BDCAT 2018*, 2019, pp. 97–106.

[123] W. Treethidtaphat, W. Pattara-Atikom, and S. Khaimook, “Bus arrival time prediction at any distance of bus route using deep neural network model,” in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 988–992.

[124] “Google Maps.” [Online]. Available: https://www.google.com/maps. [Accessed: 27-Feb-2019].

[125] “Bing Maps - Directions, trip planning, traffic cameras & more.” [Online]. Available: https://www.bing.com/maps. [Accessed: 23-Jul-2020].

[126] “Citymapper - The Ultimate Transport App.” [Online]. Available: https://citymapper.com/paris. [Accessed: 27-Feb-2019].

[127] F. Schanzenbacher, R. Chevrier, and N. Farhi, “Fluidification du trafic Transilien : approche prédictive et optimisation quadratique,” p. 2p, Feb. 2016.

[128] “Munich Transport and Tariff Association | MVV.” [Online]. Available: https://www.mvv-muenchen.de/. [Accessed: 28-Feb-2019].

[129] D. O. T - Federal Transit Administration, “Las Vegas Metropolitan Area Express,” 2005.

[130] N. Ketkar, “Introduction to PyTorch,” in *Deep Learning with Python*, Berkeley, CA:

Apress, 2017, pp. 195–208.

[131] S. Ruder, “An overview of gradient descent optimization algorithms,” Sep. 2016.

[132] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[133] S. Hochreiter and J. J. Urgen Schmidhuber, “LONG SHORT-TERM MEMORY,” 1997.

[134] “Tan - Ma vie sans arrêt - tan.fr.” [Online] Avalible: https://www.tan.fr/fr/horaires [Accessed: 28-Oct-2020]

[135] “OSM Transport Editor.” [Online]. Available: http://makinacorpus.github.io/osm-transport-editor/#/1710482. [Accessed: 22-May-2018].

[136] G. Van Brummelen, *Heavenly mathematics : the forgotten art of spherical trigonometry*. Princeton University Press, 2013.

[137] Travers Ching and D. Eddelbuettel, “RcppMsgPack: MessagePack Headers and Interface Functions for R,” 2018.

[138] Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

[139] A. Paszke *et al.*, *Automatic differentiation in PyTorch*. 2017.