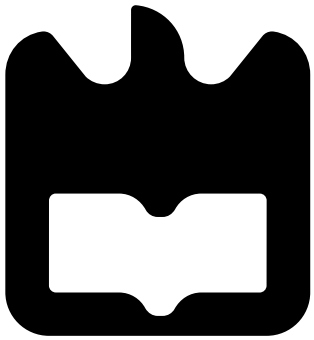
Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2017

Leandro Ricardo

Sistema de Suporte à Decisão para Transportes Públicos
Decision Support System for City Public Transportation



Leandro Ricardo

Sistema de Suporte à Decisão para Transportes Públicos
Decision Support System for City Public Transportation

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica da Professora Doutora Susana Sargento, Professora Associada com Agregação do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e co-orientação científica do Professor Ílidio Oliveira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor José Manuel Matos Moreira
Professor Auxiliar do Departamento de Eletrónica Telecomunicaões e Informática da Universidade de Aveiro

vogais / examiners committee

Professor Doutor Pedro Miguel Alves Brandão
Professor Auxiliar do Departamento de Ciência de Computadores da Faculdade de Ciências da Universidade do Porto

Professora Doutora Susana Isabel Barreto de Miranda Sargento
Professora Associada com Agregação do Departamento de Eletrónica Telecomunicaões e Informática da Universidade de Aveiro (orientadora)

agradecimentos /
acknowledgements

No culminar do meu percurso académico não me poderia esquecer de agradecer a todas as pessoas que contribuiram para o meu sucesso.

Em primeiro lugar, agradeço aos meus pais Jorge e Elisabete e aos meus irmãos Rita e Francisco, por todo o amor, carinho e apoio incondicional.

À minha namorada, Maria António, por todas as horas de paciência e carinho.

Aos meus amigos, Nuno Henriques, Marco Silva, Cristina Silva, Rui Pedro, David Silva, Eduardo Sousa, Mafalda Rodrigues, Aléxio Simões, Daniela Sousa, João Simões e José Moreira pelo companheirismo e amizade que concerteza permanecerão durante muito mais anos.

À minha orientadora, a incansável Susana Sargento, que além de me ter dado a oportunidade de colaborar num grupo de investigação relevante e me ter ajudado com todos os problemas recorrentes do desenvolvimento deste trabalho, me incentivou e inspirou a pensar e a desenvolver trabalho com impacto.

Ao meu co-orientador, Ilídio Oliveira, pelas suas sugestões, opiniões e visão, ajundando-me a melhor reflectir sobre os problemas deste trabalho.

Ao professor José Maria Fernandes, pelas suas sugestões e infindáveis ideias.

Ao Jorge Pereira e Bruno Areias colegas do grupo de investigação que apesar de terem também os seus problemas, arranjaram sempre um tempinho para me ajudar com os meus.

Aos investigadores postdoc do grupo de investigação, em particular ao Miguel Luís, pelas suas críticas extremamente assertivas e construtivas, ao Carlos Senna pela sua partilha de experiência e ao Lucas Guardalben.

Resumo

Hoje em dia existe tecnologia para tornar as cidades inteligentes. As cidades inteligentes são capazes de sentir, analizar e reagir: sentir através dos variados sensores espalhados em torno da cidade, sensores estes que podem ser fixos (sensores para a monitorização do estado ambiental) ou móveis (por exemplo, os cidadãos, graças aos seus smartphones). Um caso notável é o da cidade do Porto, que incorpora uma rede em malha com mais de 600 veículos (autocarros, taxis e camiões do lixo) que comunicam entre si, habilitando os passageiros dos autocarros da maior operadora da cidade a navegar na internet gratuitamente, enquanto viajam.

O maior impacto de uma rede como esta é a mobilidade; e uma das preocupações das instituições governamentais locais é como elas podem melhorar a mobilidade.

É por isso crucial analisar o que pode ser feito para melhorar a mobilidade de uma cidade. Utilizando os dados gerados pelo movimento dos autocarros é possível fornecer um conjunto de novas utilidades práticas que podem ser úteis ao quotidiano dos cidadãos e dos gestores de frota. Na perspetiva dos passageiros pode ser introduzido o conceito de *smart schedule* que consiste em fornecer o tempo estimado de chegada de um autocarro que se vai adaptando ao longo do tempo, de acordo com a dinâmica da cidade, que pode ser acedido diretamente a partir do seu smartphone. Na perspetiva dos gestores de frota é possível fornecer introespeções sobre o comportamento habitual das linhas de autocarros, dando abertura a que estes sejam capazes de melhor reagir a novas ou anormais dinâmicas dos transportes públicos da cidade.

Esta dissertação apresenta uma abordagem para analisar os dados provenientes da rede veicular e de como usá-los para tornar as ideias previamente esclarecidas, possíveis. Devido à inexistência da identificação do traço GPS a uma linha de autocarro, um algoritmo de map-matching foi implementado. Isso torna a computação de estimações e predições sobre o tempo de passagem dos autocarros possível. No que toca à predição, foram testados três algoritmos diferentes de aprendizagem automática em conjunto para a construção de modelos preditivos. Por fim, foram implementadas aplicações como prova de conceito que demonstram a aplicabilidade no mundo real, ajudando os passageiros dos autocarros e os gestores de frota a reagir aos diferentes eventos do seu quotidiano.

Os resultados demonstram que o algoritmo de map-matching apresenta uma boa qualidade. Também demonstram que o melhor algoritmo de aprendizagem automática, considerando o erro de predição, é o *Bagging* utilizando como estimador base *Support Vector Regressor*. Por fim, os perfis obtidos pelo painel de controlo permitem distinguir linhas de autocarro com um funcionamento ótimo daquelas em que o funcionamento é insatisfatório.

Abstract

Nowadays, the technology to turn cities smart already exists. Smart Cities, as they are called, are capable to sense, analyze and react: sense through the set of sensors displaced along the city, as they are sensors either fixed (for environmental monitoring) or moving (for instance, citizens with their smartphones). A notable case is Porto, which incorporates a mesh network with more than 600 vehicles (buses, taxis and garbage trucks), communicating in-between and enabling the passengers of the buses of the city major bus carrier to access freely to the Internet while commuting.

A vehicular network like this has huge positive impact in the city mobility, which is one of the biggest concerns of the governmental institutions.

Therefore, it is crucial to understand what can be done to improve mobility. By analyzing the data generated by the movement of the buses, it is possible to deliver a new set of tools that might be useful for the everyday life of the bus passengers and bus fleet managers. From the passengers perspective, the utility can be brought by the introduction of smart schedules, which consists on delivering estimated time of arrival that is adapting itself to the city dynamics, through the evolution of the time, and that can be accessed directly from their smartphones. From the perspective of the bus fleet managers, it is possible to deliver insights about the usual behaviour of their bus lines, giving openness for them to react to the new or abnormal city public transportation dynamics.

This dissertation presents an approach for analyzing the data descendent from the vehicular network and how to use it to answer the previously addressed problems. Regarding the missing link between the GPS trace from the bus and the bus line that they are doing, a map-matching algorithm is implemented. That turns possible the computation of estimations and predictions of the bus' passing times. In what concerns prediction, three machine learning ensemble algorithms have been tested. Finally, proof-of-concept applications are implemented to demonstrate the real-life applicability, by helping the bus passengers and bus fleet managers to react to the different events of their quotidian.

The results show that the map-matching algorithm presents a good quality. Also, they demonstrate that the best machine learning algorithm, considering the prediction error, is *Bagging* using *Support Vector Regressor* as the base estimator. Finally, the profiles obtained in the performance dashboard enable distinction between optimal and non-optimal bus lines.

Contents

Contents i

List of Figures v

List of Tables vii

Acronyms ix

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Objectives . . . 2
- 1.3 Contributions . . . 3
- 1.4 Acknowledgments . . . 3
- 1.5 Document outline . . . 3

2 Concepts 5
- 2.1 Introduction . . . 5
- 2.2 Smart Cities . . . 5
  - 2.2.1 Definition . . . 5
  - 2.2.2 Challenges on mobility . . . 7
- 2.3 Vehicular Networks . . . 7
  - 2.3.1 Definition . . . 7
  - 2.3.2 Importance of VANET under the context of this work . . . 9
- 2.4 Fundamentals of mapping . . . 10
  - 2.4.1 Introduction . . . 10
  - 2.4.2 The shape of the earth . . . 10
  - 2.4.3 Datums and projections . . . 11
  - 2.4.4 Issues regarding spatial data handling . . . 13
- 2.5 Data processing fundamentals . . . 13
  - 2.5.1 Introduction . . . 13
  - 2.5.2 Concurrent, parallel and distributed computing . . . 13
  - 2.5.3 Big Data definition and characteristics paradigms . . . 14
    - Big Data processing Paradigms . . . 15
- 2.6 Machine Learning . . . 16
  - 2.6.1 Definition . . . 16
  - 2.6.2 Types of Machine Learning . . . 17
    - Supervised Learning . . . 17

Unsupervised Learning . . . 18
Reinforcement Learning . . . 18
2.6.3 Fundamental concepts . . . 19
Machine Learning Workflow . . . 19
Generalization, Overfitting and Underfitting . . . 20
Model Selection and Cross-Validation . . . 20
Improving models performance with Ensemble Learning . . . 21
2.7 Summary . . . 22

3 Related Work **25**
3.1 Introduction . . . 25
3.2 Bus Trajectory Identification by Map-Matching . . . 25
3.3 Comparing state-of-the-art regression methods for long term time prediction . . . 26
3.4 Empirical Study of Travel Time Variability Using Bus Probe Data . . . 27
3.5 Using Bus Probe Data for Analysis of Travel Time Variability . . . 28
3.6 A review of travel time estimation and forecasting for Advanced Traveller Information Systems . . . 30
3.7 Real-time Trip Planner in Urban Public Transport . . . 32
3.8 Summary . . . 33

4 Matching lines with GPS logs and Building Performance Indicators **35**
4.1 Introduction . . . 35
4.2 Problem Setting . . . 35
4.3 Available Context . . . 36
4.3.1 Data Sources . . . 36
4.3.2 Insights on the log database . . . 36
Database Tables . . . 36
A detailed view on `node_data`: attributes, granularity, quality and quantity . . . 37
4.3.3 STCP Website as data source . . . 38
4.4 Exploring, visualizing and choosing data . . . 38
4.4.1 Position Log Data . . . 39
4.4.2 Bus network data description . . . 39
4.5 Performance Indicators . . . 41
4.5.1 Definitions . . . 41
4.5.2 Main restritions . . . 42
4.6 Summary . . . 42

5 Architecture and Technical Design **43**
5.1 Introduction . . . 43
5.2 Requirements . . . 43
5.2.1 Functional requirements . . . 43
5.2.2 Non-functional requirements . . . 44
5.3 Architecture . . . 44
5.4 Technical Design . . . 46
5.4.1 Overview . . . 46
5.4.2 Exploiting parallelism using the Worker Design Pattern . . . 47

5.4.3 Algorithm Design . . . 49
Making spatial searches . . . 50
Using and choosing a detection radius . . . 50
Detecting line starts . . . 52
Finding a solution . . . 52
5.4.4 Building performance indicators . . . 57
A deeper overview on `completeness` metric . . . 57
Estimating arrival times . . . 57
Predicting arrival times . . . 59
5.5 Summary . . . 60

6 System Implementation **61**
6.1 Introduction . . . 61
6.2 Bus Network Information Retrieval . . . 62
6.2.1 Script to retrieve base data implementation details . . . 64
6.2.2 Script to transform base data implementation details . . . 65
6.3 Matching Unit . . . 66
6.3.1 Development History . . . 66
6.3.2 Overview . . . 67
6.3.3 Modules . . . 67
6.3.4 The `entities` module . . . 68
`Worker` implementation overview . . . 68
`Dispatcher` implementation overview . . . 69
`Logger` implementation overview . . . 69
6.3.5 The `pipeline` module . . . 69
Algorithm to detect when a line starts . . . 70
Algorithm for finding solutions from line starts . . . 70
6.3.6 The `tools` module . . . 73
6.4 Matches Database . . . 73
6.4.1 Overview . . . 73
6.4.2 Database schema description . . . 74
6.4.3 Routines . . . 75
6.4.4 Functions . . . 75
6.5 Estimation Database . . . 76
6.5.1 Overview . . . 76
6.5.2 Development history . . . 76
6.5.3 Database diagram . . . 77
6.5.4 Database schema description . . . 77
6.5.5 Routines . . . 78
Functions . . . 78
Triggers . . . 80
6.5.6 How estimates are calculated . . . 80
6.6 Synchronization Script . . . 80
6.7 Prediction Module . . . 81
6.7.1 Overview . . . 81
6.7.2 Algortihms . . . 81
6.7.3 Data Collection and Preparation . . . 82

Requirements . . . 82
Dataset characterization . . . 83
6.7.4 Model Selection and Cross Validation . . . 83
6.7.5 Evaluation . . . 84
6.7.6 Deployment . . . 85
6.8 Database Wrappers Module . . . 85
6.9 Integration APIs . . . 85
6.9.1 Bus Network Information API . . . 86
6.9.2 Match API . . . 87
6.9.3 Estimation API . . . 87
6.9.4 Prediction API . . . 87
6.10 Applications . . . 87
6.10.1 Bus Line Performance Dashboard . . . 88
6.10.2 Bus Passenger Mobile Application . . . 89
6.11 Summary . . . 90

7 Deployment and Results **93**
7.1 Introduction . . . 93
7.2 Deployment . . . 93
7.2.1 Hardware . . . 93
7.2.2 Software . . . 93
7.3 Results . . . 95
7.3.1 Matching GPS traces with bus lines . . . 95
Context . . . 95
Processed Data in numbers . . . 95
Results presentation . . . 95
Analysis and Validation . . . 96
7.3.2 Delay metrics for bus lines and estimated times of arrival . . . 98
Context . . . 98
Result presentation 1.1: Delay plot of a bus line match . . . 98
Result presentation 1.2: Detecting problematic lines . . . 102
Result presentation 2: Estimated time of arrival given a previously defined static value from the STCP time tables . . . 103
Result presentation 3: Estimated time of arrival given a dynamically chosen value (user given or inferred by the delay plot analysis) 104
Validation . . . 104
7.4 Prediction Module Results . . . 104
7.4.1 Regression Metrics Comparison . . . 105
7.5 Summary . . . 107

8 Conclusion and Future Work **109**
8.1 Conclusion . . . 109
8.2 Lessons Learned . . . 109
8.3 Future Work . . . 112

Bibliography **113**

List of Figures

1.1 A figurative image of a smart city 1

2.1 A vehicular network and its interactions 8
2.2 Vehicular Network Architecture (figure from Andre Cardote) 9
2.3 The earth Geoid GOCE (ESA/HPF/DLR) 11
2.5 A standard Lambda Architecture, its modules and methodologies, presented by the company MapR [23] 16
2.6 An example of classification task distinguishing cats from dogs [26] 17
2.7 An example of a clustering task, which distinguished 3 different groups [26] . 18
2.8 The typical machine learning workflow [26] 19
2.9 Presentation of the relation between the model complexity and the model prediction error (taken from [14]) 21
2.10 Majority voting used in a machine learning ensemble method (taken from [33]) 22

3.1 Comparison of the different time periods in terms of stability [6] 28
3.2 Data transformation steps described by Uno et al. [45] 29
3.3 Traffic data sources. (a) Point detectors, (b) probe vehicles and (c) Interval detectors (figure from the review [28]) 31
3.4 *"Difference between time travel estimation and prediction"*[28] 32
3.5 NextBus Architecture (figure from [3]) 33

4.1 An all day long position Log from a bus from the first day of March 40
4.2 Bus stops of Porto 40
4.3 Bus lines of Porto 41

5.1 Architecture components and the data flow 45
5.2 Data Processing Pipeline 46
5.3 Matching Unit Architecture 47
5.4 Worker Design Pattern Design 48
5.5 A non-ideal and an ideal radius 51
5.6 The stop S has two lines starting in its position. If the bus is passing near that position S, there is a probability that one of the lines starting there is the solution. 52
5.7 Detect line starts algorithm 53
5.8 Algorithm phases 54
5.9 Matching Algorithm Flowchart 56
5.10 Example of a distribution of time points for calculating a estimation 58

6.1 Line Details Page from STCP Website . . . 62
6.2 Capture on Wireshark showing the filter and the HTTP GET Requests to the STCP API . . . 63
6.3 Matching Unit Modules . . . 68
6.4 Matches Database Diagram . . . 74
6.5 Data Mart Diagram . . . 76
6.6 Data Mart Diagram (not definitive image) . . . 77
6.7 `TimeSeriesSplit` iterations over data . . . 82
6.8 User Interface of Matches API, generated by Swagger . . . 86
6.9 This is the match browser view, where the bus fleet manager can analyze each one of the matched bus journeys. . . . 89
6.10 A view over the delay plot . . . 89
6.11 Two different views of the mobile application . . . 91

7.1 The match from the `node_id` 2764 from March, 1 St, from 07h00m to 07h55m, with 100% completeness. . . . 96
7.2 The match from the `node_id` 2474 from March, 15 St, from 14h32m to 15h19m, with 100% completeness . . . 96
7.3 The match from the `node_id` 2801 from March, 8 St, from 18h10m to 19h17m, with 91% completeness due to gaps on the input records from the position log database, outlined with stroked circles. . . . 97
7.4 A line completion with uncertainty . . . 98
7.5 Line 204 Hospital de S. João from `07:01:33` to `07:53:48` . . . 100
7.6 Line 204 Hospital de S. João from `07:57:50` until `09:02:20` . . . 100
7.7 Line 204 Hospital de S. João from `12:01:05` until `12:56:51` . . . 100
7.8 Line 204 Hospital de S. João from `15:03:36` until `15:59:51` . . . 101
7.9 Line 204 Hospital de S. João from `17:26:37` until `18:44:52` . . . 101
7.10 200 Castelo do Queijo, 3 months analysis . . . 102
7.11 204 Hospital de S. João, 3 months analysis . . . 102
7.12 902 Boavista, 3 months analysis . . . 103
7.13 Bagging (using support vector regressor) . . . 106
7.14 Random Forrest . . . 106
7.15 Gradient Boosting . . . 107

8.1 Development phases . . . 110

List of Tables

3.1 Time Periods from this study[6] 28

4.1 Description of the attributes of the table node_data 37

7.1 Statistics for a time query associated to a bus line and bus stop 103
7.2 Statistics for a time query associated to a bus line and bus stop 104
7.3 Resulting Evaluation Metrics 105

Acronyms

API Application Programming Interface. 45, 61–63, 83, 85–90, 94, 104, 107

CPU Central Processing Unit. 44, 47, 66

ESA European Space Agency. 11

GOCE Gravity field and steady-state Ocean Circulation Explorer. 11

GPS Global Position System. 2, 3, 7, 11, 12, 23, 25, 28–30, 33, 35, 37, 39, 49–51, 54, 55, 57, 104

HTTP Hypertext Transfer Protocol. 63, 65

ICT Information and Communication Technologies. 23

JSON JavaScript Object Notation. 64, 65, 67, 87

LMA Local Mobility Anchor. 9

MAG Mobile Access Gateway. 9

MIT Massachusetts Institute of Technology. 33

mMAG Mobile MAG. 10

MNN Mobile Network Node. 10

NoSQL Not-Only SQL. 15, 111

OBU On-Board Unit. 10, 37

RAM Random Access Memory. 44

SMS Short Message Service. 2

SQL Structured Query Language. 73, 75, 79, 85

SSD Solid-State Disk. 44

STCP Sociedade de Transportes Colectivos do Porto. 2, 9, 27, 38–40, 44, 46, 57, 62–64, 90, 94, 95, 98, 99, 103, 104, 108

SVM Support Vector Machine. 20

TETRA Terrestrial Trunked Radio. 27

TTL Time To Live. 70, 71

TTP Travel Time Prediction. 26

URL Uniform Resource Locator. 63

VANET Vehicular Ad-hoc Networks. 2, 7–9

WGS World Geodetic System. 11

WLAN Wireless Local Area Network. 7

Chapter 1

Introduction

1.1 Motivation

In the last several years, the term *Smart City* has been acknowledged, first by the researchers, then by the governments, and finally by the people.

As a multidimensional concept, *Smart City* covers a wide range of equally *smart* topics: *green city* (related to *smart* profiteering of the environmental resources), the *smart growth* (related to a human-aware and sustainable urban development), the *smart grid* (related to efficient electrical energy delivering), etc.

These topics are intersected between them, and they have a thing in common: they look to enhance the people's life through the use of the technology. The Figure 1.1 depicts this relation.

Figure 1.1: A figurative image of a smart city

Mobility is also a concern for the governmental institutions. Citizens are moving around

the city and commuting is, for some, a significant part of their everyday life. Therefore, the concept of *Smart Transportation* becomes apparent and answering to the question "how can cities improve their public transportation system" becomes a top priority.

Regarding mobility and the *smart transportation* topic, an accessible case-study can be further explored. Porto, the second largest city in Portugal after Lisbon, is a *Smart City*, and it deploys a world-level pilot project which implements a Vehicular Ad-hoc Networks (VANET) connecting buses, garbage trucks and taxis. This mesh network is capable of delivering free wireless internet access to bus passengers, enhancing their commuting experience. One may ask what can be done for enhancing further the commuting experience.

Thinking about the impact of the existing applications in everyone daily life, Sociedade de Transportes Colectivos do Porto (STCP), which is the major Porto's bus carrier, provides a service called *SMSBUS*. It delivers estimated times of arrival for a given line or bus stop, recurring to the sending of a Short Message Service (SMS). Unfortunately, it is a paid service[1] and it depends on specific and expensive requirements (like using antenna triangulation).

Regarding the existing framework (the vehicular network), it may be possible to produce mobile applications with the objective of delivering bus line schedules, routes and others for free. Also, it would be interesting if it was possible to provide customer oriented services enabling bus passengers to subscribe a bus stop and querying for delays, which could be dynamic, depending on the past history (estimated time of arrival) and on the current traffic (prediction of the time of arrival).

Predicting the bus route behavior in terms of delay is relevant for the bus carrier managers, in the scope of understanding if there are problems due to road works, bus route congestion, and other events that may interfere, even despite of the existence of proprietary systems, in production for years.

Therefore, the answer may live on optimizing the way the passengers commute, potentially by increasing the knowledge about the behaviour of the buses. This may be possible through the development of a mobile application concerning a map of bus stops, bus lines, and schedules which could be estimated, using the past history. On the other hand, bus carriers are constantly looking for optimizing the way they manage their resources, being absolutely essential for them to understand how is their bus carrier performing in a particular bus line.

Given the data resulting from the operation of the network, like the movement of the nodes in space (GPS position, velocity, etc), it is possible to tackle these specific problems. The motivation of this work is to make use of the network operation data for optimizing the way bus passengers commute, and optimize also, the way bus carrier managers perceive the bus network behavior.

1.2 Objectives

Given the previously exposed problems, the main purpose of this dissertation is to:

[1] SMS BUS tax (`http://www.stcp.pt/smsBusMicroSite/quantocusta.html`)

- Create a solution for obtaining the bus estimated times of arrival in a given line and stop (or a set of stops), based on the past history.
- Regarding the first objective, solve the missing link between the position log data and the bus network infrastructure because it is not possible to know which bus was completing a given line. This will require matching the Global Position System (GPS) trace of the buses with a bus line.
- Using machine-learning techniques on historical data, predict the bus behavior, in terms of delay, in the days to follow.
- Deliver proof-of-concept applications as they are an important step on extracting some results and on generating real life examples and tests.

1.3 Contributions

Following this study development, two articles have been submitted, with the same title *"Decision Support System for City Public Transportation"*.

The first submission was targeted to INForum 2017, in October 12th and 13th, as a communication. It was accepted and presented via oral presentation and poster.

The second submission was targeted to VEHITS 2018, in October 30th and it is waiting for approval.

1.4 Acknowledgments

This work was supported in part by National Funds through FCT - Fundação para a Ciência e a Tecnologia under the project UID/EEA/50008/2013, in part by the IT Internal Project SmartCityMules and in part by the CMU-Portugal Program through S2MovingCity: Sensing and Serving a Moving City under Grant CMUP-ERI/TIC/0010/2014.

1.5 Document outline

This document is organized as follow:

- **Chapter 1** is the introduction for the work.
- **Chapter 2** focuses on explaining some fundamental concepts which are useful on understanding some decisions.
- **Chapter 3** presents some of the most relevant works and their fundamental ideas.
- **Chapter 4** explains the problem setting in further detail.
- **Chapter 5** presents the architecture and the theory behind the implementation.
- **Chapter 6** presents the implementation of the components comprising the architectural specification.

- **Chapter 7** presents how can this system be deployed and discusses the results of this work.
- **Chapter 8** presents the conclusion and the future work.

Chapter 2

Concepts

2.1 Introduction

This chapter introduces the main concepts which are meaningful for this work, presenting definitions which provide ground for understanding the importance of enclosing paradigms and the most common terminologies.

We start by explaining what is a **smart city** and presenting its challenges on mobility. Then, due to the fact that this study is based on data collected from a **vehicular network**, this topic is addressed too.

The data descending from the vehicular network is spatial-temporal and it has considerable volume. Regarding this data nature, **the fundamentals of mapping** are presented as means for drawing attention to the impact of this data type into the information systems. On the other side, because the volume of the data raises some concerns on the processing times, the different **data processing paradigms** are presented, first by explaining the concepts of the concurrent, parallel and distributed computing and then, by presenting the data processing paradigms.

Finally, it is presented the fundamental concepts of **machine learning**, which is a way of extracting knowledge from data.

2.2 Smart Cities

2.2.1 Definition

"There is neither a single template of framing smart city nor a one-size-fits-all definition of smart city. ... The label smart city is a fuzzy concept and is used in ways that are not always consistent." [29]

To better define what is a *Smart City*, one could understand it as a multidimensional representation of a generic city, with three main fronts [29][11]:

- Technology dimension.
- Human dimension.
- Institutional dimension.

The **Technology Dimension** borrows most of its meaning from the concept of *Digital*, *Ubiquitous* and *Information City*, and thus, it is objectively focused on the infrastructure.

The concept of *Digital City* focuses on the communication infrastructure and on understanding how flexible, service-oriented, based on open standards and capable should it be for delivering innovative services for citizens, businesses and to the government.

The *Ubiquitous* is referred to the set of ubiquitous devices which can be available on the urban elements, either active or passive urban particles (people, public or private transportation, or buildings and infrastructure) and generate information which results from the state or interaction between these devices.

Finally, *Information City* refers to the characteristic of the city on being capable of exploring and delivering information from local communities and systems, through the use of the Internet (not necessarily from web portals but also from other closer alternatives like mobile applications).

The **Human Dimension** focuses its meaning in child concepts like *Creative*, *Learning* and *Knowledge City*, which are subsequently related on human relations and on how can they improve a city as a whole.

Creative City concerns the view of the city as friendly environment for the development of the human and social infrastructure. *Human infrastructure* defines the set of human organizations where people are engaged due to work or other activity (for example, through an association, a volunteer group, etc) while *Social Infrastructure* is about the people, their relationship and how they generate benefit from social capital *i.e..*, how human relations can *"generate benefits that flow from the trust, reciprocity, information, and cooperation associated with social networks"* [18].

Last but not least important, the **Institutional Dimension** is an umbrella definition which relies on the *Smart Community* and *Smart Growth* concepts to highlight *governance among stakeholders and institutional factors for governance* [29].

One strong definition for *Smart Community "was coined by a blue-ribbon panel of experts, created by the Canadian government in 1998, to provide advice on a potential national Smart Communities programme"* [47]. They defined *Smart Community* as community with common or shared interests in which its members, organizations and/or governing institutions work with information technologies to transform their daily life in a positive and significant way, regardless of the community size [24].

From another perspective, *Smart Growth* and *Green City* are two related concepts. The first one, *Smart Growth*, is focused on strategies which promote the development and conservation of the citizens health and environment. The second defines city as green when it puts effort on designing itself to generate the lowest impact possible to the environment, minimizing its resource consumption requirements (food, water, energy, etc) and greatly reducing the production of pollution (air, heat, gas emissions, noise, etc).

Thus, the concept of *Smart City* could be expressed as an urban development vision which tries to bring the best out from the cooperation between people (citizens, organizations and the government), infrastructures (telecommunications, transports, general purpose building like schools, bridges, etc) and policies (either environmental, either urban development ones) through the use of Information and Communication Technologies, that connects and empowers both.

2.2.2 Challenges on mobility

The previous subsection supports the idea that the*"Smart City will be the future trend of urban development"*[42]. Generally, the development of a smart city encompasses the last three dimensions, which can have a huge impact in the real life.

The umbrella term for the smart initiatives concerning the mobility is called "Smart Mobility". It is not only concerned on increasing the commuting speed of the people around the city, but also, with reducing the costs of commuting, improving people safety, reducing pollution (either by the reduction of the emissions, either by the reduction of the noise) and reducing traffic congestion. This definition was first employed by Benovolo et al [8].

These objectives are overlapped in-between, being part of the scope of the smart mobility concept, and also, under the scope of the *Smart Transportation*.

Smart Transportation is all about ubiquity. Taking *"good advantage of sensor network, the Internet of Things and other technical means"* [42] a city can increase its intelligence (*i.e.,* knowledge) about the way urban particles – citizens, buildings, transportation and communication – interact.

It is almost classic to think of this as mean for understanding more about traffic and public transportation in such a way that it is possible to establish a smart traffic management system, or a dynamic public transportation performance tracker which could provide performance metrics and dynamic schedules. Therefore, the concept of *Smart Urban Management* arises.

Viewing citizens as urban particles is also a current vision. For example, *SenseMyCity* [36] points out some observations which make clear why citizens are good candidates as "moving sensors".

Citing the same work, it is observed that *"people treat smartphones as a second skin, having them around nearly 24/7 and constantly interacting with them"* and also that they (smartphones) *"are equipped with a wide range of embedded sensors, like GPS for location, magnetometer, accelerometer, gyroscope"*.

If one takes advantage of this huge sensor network, it is possible to go further and understand how the mobility (or the lack of it) affects citizens and how they interact with the existing infrastructures.

In sum, it would be possible to deliver a myriad of solutions capable of delivering strong support for the integration between the different smart city areas of development (the urban planning, construction, management and operations) and providing a deep understanding how the smart urban ecosystem works.

2.3 Vehicular Networks

2.3.1 Definition

In the wide topic of the intelligent transport systems, vehicular networks have been one of the trending areas in the last several years. Vehicular Ad-hoc Networks (VANET) is a network development paradigm that makes use of *"inexpensive wireless local area network (WLAN) technology that connects notebook computers to each other and the Internet, and, with a few tweaks, install it on vehicles"*[15].

Figure 2.1: A vehicular network and its interactions

The concept of VANET is more *"similar to the one applied on ad-hoc networks"*[25], meaning that there is a dynamic/spontaneous creation of a wireless mesh network.

The effort of bringing such technologies to vehicles results in a unique environment which raises new opportunities, challenges and requirements:

- Vehicles communicating between each other directly and with the infrastructure, raise **opportunities** for developing a more safe and aware road network and for building a set of applications in diverse areas, like safety, marketing, etc. For example, one of the earliest applications of vehicular networks was delivering internet access.

- Current vehicles are able to reach high speeds and work in highly dynamic environments, which can differ a lot in terms of connectivity, being reasons that can raise some **challenges** on modelling the communication infrastructure and the communication protocols.

- Several issues regarding the government concerns on privacy and security raise new **requirements** [15]. Also, the new applications can raise new demands regarding higher packet delivery rates and lower packet latency.

2.3.2 Importance of VANET under the context of this work

Porto, the second largest city in Portugal after Lisbon, is a living lab. Thanks to an alliance between *IT*[1], *UA*[2], *UP*[3], *VENIAM*[4], *Porto Digital* [5] and STCP[6], it was possible to deploy a large dimension *mesh network* using the buses, taxis and garbage collection trucks for providing free WIFI access to bus passengers.

This mesh network is, objectively, a Vehicular Ah-hoc Network capable of exchanging big amounts of information. It is also capable of generating big amounts of heterogeneous data (for example, time-series or spatial data) related to the buses position and velocity, opening opportunities for the creation of new applications not only related with safety, but also, with mobility and other important smart city issues.

Regarding that, being familiar with the meaning of the components of the architecture and its terminologies is very important for extracting meaning from *wrangling*[7] the data.

The figure 2.2 shows the Vehicular Network Architecture, as deployed on Porto.

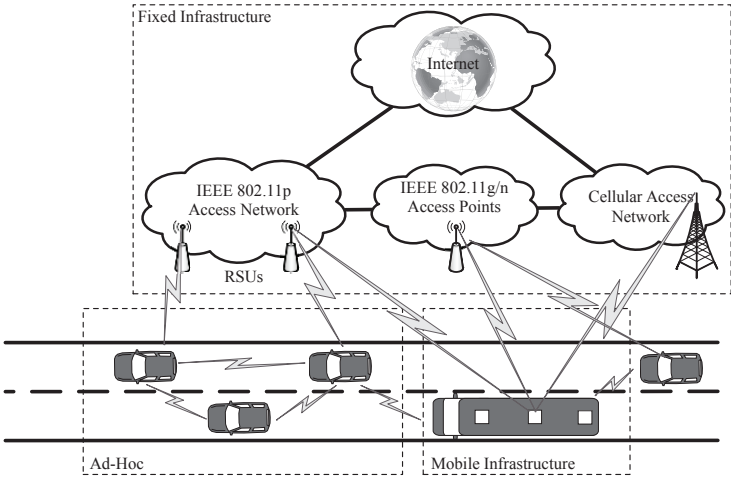


Figure 2.2: Vehicular Network Architecture (figure from Andre Cardote)

The architecture of the VANET deployed *i.e.* Porto can be divided in four major components:

- **Local Mobility Anchor (LMA)**: is the central component of the architecture that manages the IP mobility. It is located in one of the network machines (it could be either on a server, either on the cloud).
- **Mobile Access Gateway (MAG)**: *"is a fixed infrastructure access point"* [25]. It connects the mobile agents to the core of the network. A MAG can be a road side unit,

[1]Instituto de Telecomunicações
[2]Universidade de Aveiro
[3]Universidade do Porto
[4]A vehicular networks company (https://veniam.com/)
[5]Porto Digital is a private association which promotes ICT projects within the context of Porto City and Porto's metropolitan area (https://portodigital.pt/index.php?artigo=19)
[6]Porto's major bus carrier (http://www.stcp.pt/en/travel/)
[7]Data wrangling is the process of transforming raw data into a more valuable and meaningful format, for a variety of purposes such as classification, analytics, etc.

a specific kinf of stationary units which are distributed strategically in space, along roads for example, Wi-Fi hotspots, etc.

- **Mobile MAG (mMAG)** is a mobile access point inside the vehicles, which acts like the access layer of this network architecture, it enables the connection of end-devices. The main architecture component is called On-Board Unit (OBU) and it is composed by multiple network interfaces *"such as Wi-Fi (IEEE 802.11a/b/g/n), WAVE (IEEE 802.11p) or LTE (4G)"*[25], enabling connected vehicles for *"sharing contents or spreading messages"*[25] between them.

- **Mobile Network Node (MNN)** corresponds to the devices from the end-users (notebooks, smartphones, tablets, etc).

This Figure 2.2 shows the strong interest on implementing the *Always Best Connected* paradigm, *"which refers to the target of keeping always the best connection available for the user while performing all the horizontal/vertical handovers without impacting on the running services"* [25].

2.4 Fundamentals of mapping

2.4.1 Introduction

Geodesy (also known as geodetics) is the field of study in area of the applied mathematics, which focuses on studying the representation of Earth, including the shape, the gravitational field and the exact position of geographical distributed points.

When working with spatial data, some attention is required, regarding the spatial reference system being used. If we consider two different maps, with the same scale, and we look up for two random locations on both, we may find it disturbing that overlapping them, they may not intersect. That happens because they are not using the same spatial reference system.

The main objective of this subsection is to introduce some hints about the fundamental of mapping, by explaining and introducing some terms and definitions.

2.4.2 The shape of the earth

Understanding the shape of the earth is a big step on understanding how geodetic models work.

The photography *Earthrise*, taken by the astronaut *William Anders*, under the Apollo 8 mission in 1968, left no apparent clues about the earth shape: it is blue and round like a marble.

For us, living on earth, that is very far from true. *"Earth is a very misshapen object."*[22] *"The surface of the earth with all its nooks and crannies resembles a slightly charred English muffin much more than a lustrous marble."*[31] Even the idea of the earth being spherical like a marble is not accurate, because the earth is flattened in the poles, meaning that the theoretical circumference along the equator is bigger than the one passing through any of the meridians.

The earth true shape can be known using an omnipresent phenomena: the gravity. At the school, we were being told that gravity is a constant – 9.8 m/s^2. In the reality, this standard value assumes that the earth has a fixed radius (or in other words, that is a sphere).

Regarding this and other reasons [13], the value of gravity is not constant and slightly varies between 9.78 m/s^2 and 9.82 m/s^2.

From the measurement of the gravity, the definition of **geoid** emerges. A geoid is a very complex surface which results from the measurement of gravity in the different positions of the earth.

The figure 2.3 shows a geoid caputred by the European Space Agency (ESA) GOCE mission [8].

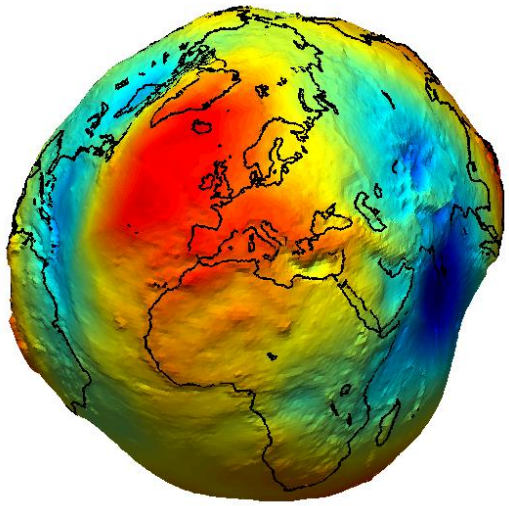


Figure 2.3: The earth Geoid GOCE (ESA/HPF/DLR)

As previously said, the geoid is a very complex mathematical surface. Regarding this reason, geodesists make use of a different and more spherical surface to model the overall shape of the earth: an **ellipsoid**.

An ellipsoid is a *"closed surface of which all plane cross sections are either ellipses or circles"*[35], being symmetrical in the mutually perpendicular three-dimensional axes.

In the first model of the earth, *"the blue marble"*, the earth was seen as a sphere, that is an ellipsoid where the sections are circles. Because of the earth shape is flatten in the poles, a better approximation is using an ellipsoid where the plane cross sections are ellipses.

The figure 2.4a and 2.4b shows the two types of referred ellipsoids.

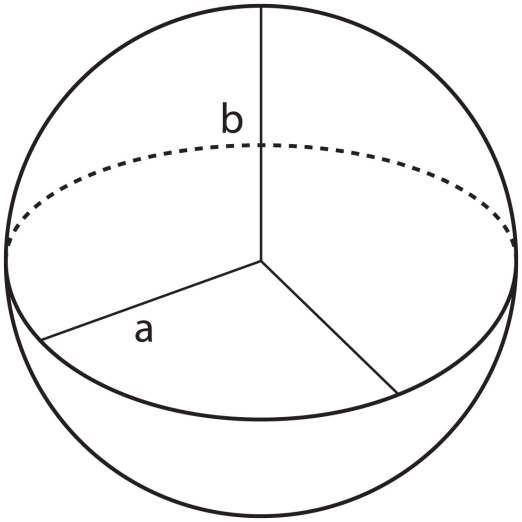
Datums define the ellipsoid shape. For example, the World Geodetic System (WGS), which is the standard used on GPS, defines an oblate spheroid. For understanding more about datums, refer to the next subsection.

2.4.3 Datums and projections

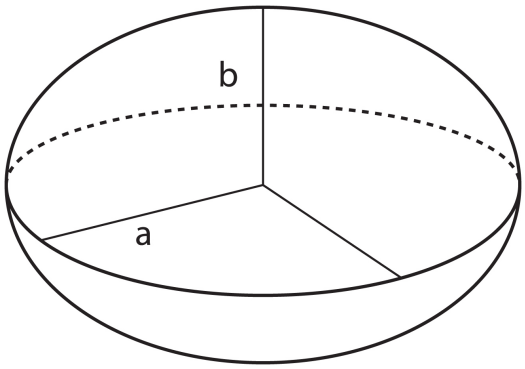
It is often referred that *"the ellipsoid models the overall shape of the earth."*[31] Being noted as global, it means that it is non-optimal too. With respect to this situation, geodesists choose the elipsoid that best fits their regional area geoid, when studying a particular area.

A **datum** is a standard point of reference, a set of points or surface from which survey measurements are based. It can be seen as system of coordinates that results from this

[8]ESA GOCE (http://m.esa.int/Our_Activities/Observing_the_Earth/GOCE/Introducing_GOCE)



(a) The earth as a Sphere[22]



(b) The Earth as an Ellipsoid [22]

approximation to earth surface, using an ellipsoid which is anchored to a given location (local datum) or using an ellipsoid that approximates globally to the earth surface (global datum). A local datum from Europe would fit poorly in another place in the world, with different geographical attributes (like the local datum of Canada).

It was a recurrent practice to divide a datum in two types: an horizontal and a vertical datum. Horizontal datums allow to measure distances on Earth surface. To do that, they often define two zero levels references: one delimited by the equator and other delimited by the Greenwhich meridian. These two references set the base coordinate reference system for locating objects in space: latitude and longitude. Vertical datums are used for measuring the Earth elevation relatively to reference point (for example, the mean sea level) – this characteristic is called elevation.

Thanks to the creation of the global navigation satellite systems such as *GPS*, GLONASS and *Galileu*, global datums had to be created. They are based on the idea of an ellipsoid that almost fits the earth surface, being the center of the ellipsoid concentric with the Earth's center of mass.

Latitude and longitude are coordinates of spherical surface, but, they are used also as coordinates for locating positions in planar surfaces, like it happens in a map. A map is a flat surface which, regarding its differences with a spherical one, needs to have their key components (e.g. shapes) transformed. This transformation with mathematical roots is called **projection**.

A projection morphs the ellipsoid into a flat surface. There are lot of different ways of doing it, and some techniques are better than others. The most popular projection is called *Mercator* and it is the one that is seen usually on maps, and taught on schools. It is *"good for maintaining shape and direction and span the globe"*[31], but not so good for making measurements as regions that are nearest to the poles become exaggeratedly stretched, raising some misconceptions regarding its observation.

For example, using the *Mercator* projection, Greenland seems as bigger as Africa. In the

reality, Africa is 14 times[9] bigger than Greenland. This interesting website [10] is focused on showing this misconception regarding the bad measurements.

Examples of datums are:

- ED50[11]
- Datum 73[12]
- NAD83[13]

2.4.4 Issues regarding spatial data handling

All these particular concepts have the objective of making evidence that spatial data is a very sensitive data type, requiring a special handling.

First of all, due to the existence of several datums and projections, one must ensure that the chosen reference system is the most adequate for the existing data. If one deals mostly with *"regional data, say for a country or state, then it's generally best to stick with one of the national grid or State Planes systems"* [31]. They provide a good measurement accuracy and look fairly well on a map.

Then, regarding the fact that coordinate reference system is three-dimensional (because latitude and longitude are units measured on spherical surface), measuring the distance between two points in the Earth surface is not a matter of applying the distance between two points in a Cartesian plane, but rather, a matter of using different approximations like the Vincenty Formulae [44]. Ignoring this fact will lead to errors while making, for instance, proximity queries.

2.5 Data processing fundamentals

2.5.1 Introduction

Dealing with moderate or high amounts of data, requires some knowledge about the techniques which can enable a faster processing. Along time, computers have been evolving towards the direction of doing more and more faster. For doing more in less time, there are three techniques that can be employed. Over this section, we present the concepts of concurrency, parallelism and distributed computing, which are progressive methodologies enabling this purpose.

2.5.2 Concurrent, parallel and distributed computing

The further developments in computer systems have been driven by the increasingly need of doing more in less time. The term **concurrency** refers *"to the general concept of a system with multiple, simultaneous activities"*[32] and the term **parallelism** is *"the use of concurrency to make a system run faster"*[32]. Modern processors are known as multi-programming

[9]Ratio between Africa and Greenland, by Wolfram Alpha (`https://www.wolframalpha.com/input/?i=africa+area+vs+greenland+area+ratio`)

[10]The True Size Of (`http://thetruesize.com/`)

[11]European Datum 1950 (`https://epsg.io/6230-datum`)

[12]Portugal Mainland Datum (`https://epsg.io/4274`)

[13]North American Datum (`https://epsg.io/6140-datum`)

systems, meaning that, if a computer system has a uni-processor and it is running, apparently, many programs at the same time, none of them is truly executing in the same and exact time. But, if the system has a multi-processor, two or more programs can run exactly at the same time. Regarding this, the concept of parallelism can be divided in two different types [5]:

- **pseudo-parallelism** is related to the illusion of running a set of tasks at the same time.
- **true parallelism** or **hardware parallelism** is the kind of parallelism that is exploited by the use the multiple cores of the CPU.

Concurrent and parallel programming are two programming paradigms related with the concept of modular programming, regarding the fact of big task being divided in a set of smaller tasks that can be done, respectively, concurrently and in parallel.

These paradigms must be applied under different circumstances. Concurrent programming is useful for dealing with slow I/O device access, human interaction, servicing multiple network clients, etc. Parallel programming is more convenient when the main objective is to complete a set of tasks as fast as possible.

Sometimes, requirements demand for a faster processing which can not be achieved using only one system, even when it is already using concurrent and parallel computing techniques. A third paradigm that goes beyond these two concepts is called **distributed computing**, and its model assumes software components which are common but distributed across different computers, cooperating between them.

These three computing paradigms are always present in different tiers of the software architecture, being fundamental concepts on the understanding of the purpose of some solutions and also, crucial for tackling and optimize computational intensive tasks.

2.5.3 Big Data definition and characteristics paradigms

"In the past decade the amount of data being created has skyrocketed. More than 30,000 gigabytes of data are generated every second, and the rate of data creation is only accelerating"[30]. Such amount of data is generically referred as *big data*.

From a high-level point-of-view, some authors defend that *"big data is all about seeing and understanding the relations within and among pieces of information that, until very recently, we struggled to fully grasp"* [46].

In a more technical and low-level fashion, **Big Data** is defined as a massive volume of data, that may be structured or unstructured, being so extensive that it outsizes the available capacity for storing, processing, analyzing and understanding it. This means that traditional software and databases do not provide enough power for delivering results, being needed different and more innovating techniques for tackling this problem. It is important to notice that data may be classified as big data depending on the context of institution, and not regarding quantities. For example, in the context of our research group, big data is about 5TB[14]. But for other business company, like eBay, that reaches over 90 PB![15].

The literature often presents four essential problems of big data, first introduced by IBM [20]. They are:

[14]Having into account the existing computational resources, like computers and storage

[15]Inside eBay's 90PB data warehouse (`https://www.itnews.com.au/news/inside-ebays-90pb-data-warehouse-342615`)

- **Volume** is the characteristic of the data which is related to its scale (size).
- **Variety** defines the different forms of data.
- **Velocity** is a definition regarding the data regeneration rate.
- **Veracity** is a characteristic which reflects the uncertainty about the quality of the data.

These characteristics reflect the current state of the of technology after the development of the latest years, particularly since the early 2000's.

First, the web evolved towards a maturing state characterized by the rising of blogs and the social media - the Web 2.0. Early since the rise of Web 2.0, new platforms enabled people to generate information in the form of text, images, audio and video (variety). For example, Google reported that it *"is receiving 400 hours of video uploaded to YouTube every minute"*[17].

At the same time, there was the development of the cloud storage and computing, delivering an always available data (velocity). Later, we have been witnessing an increasingly bet on the Internet of Things and the proliferation of always connected mobile devices like smart phones and tablets, which augment the consumption of media in daily basis, on a worldwide level (also related with variety and volume).

Big Data appeared has a constant developing paradigm, adapting to evolutionary needs. New technologies and paradigms surged. For example, in terms of databases, Not-Only SQL (NoSQL), which appeared in the late 60s, turned out to be very relevant on tackling the variety problem, by enabling a data modeling that goes beyond the classic tabular model, being adopted by the early Web 2.0 adopters like *Facebook*, *Amazon* and Google. The three more relevant paradigms which developed during time were **batch processing**, **real-time processing** and **hybrid processing**.

Big Data processing Paradigms

Batch Processing has been employed as a technique for tackling the new problem of data volume. It is focused on processing big amounts of data in group of similar objects, for processing them sequentially, as fast as possible and without human intervention. This processing paradigm is scalable, meaning that, it is able of maintaining *"performance in the face of increasing data or load by adding resources to the system"*[30]. Also, regarding the large amount of data, this type of processing is, generally, fault tolerant, so it is possible to resume it after being interrupted. The major problem of using the batch processing paradigm is the fact of having a high latency.

After the popularization of batch processing, dealing with velocity was the top priority for some business companies because of their specific need of faster response time from the intelligence systems. **Real-time Processing** plays a significant role, by reducing significantly the latency (when comparing it to the batch processing) by processing data continuously (for example, by using data streams).

Finally, in the last years, a new paradigm has emerged, with the objective of combining the best characteristics of the previously described paradigms. This paradigm is called **Hybrid**

Processing and its most popular architecture is called *Lambda Architecture* (as shown on the figure 2.5. *Lambda Architecture* is designed to handle massive amounts of data by combining batching and real-time processing. It is divided on three layers: the batch layer, the serving layer and the speed layer. *"Each layer satisfies a subset of the properties and builds upon the functionality provided by the layers beneath it"*[30]. For a better description about the function of each layer, please, refer to the presentation [40] or to the book [30].

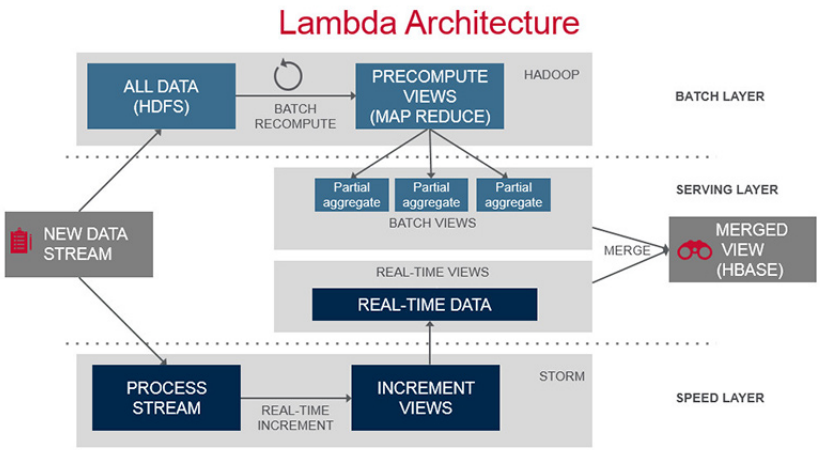


Figure 2.5: A standard Lambda Architecture, its modules and methodologies, presented by the company MapR [23]

2.6 Machine Learning

2.6.1 Definition

Machine Learning (also known as predictive analytics or statistical learning) is *"a research field at the intersection of statistics, artificial intelligence, and computer science"* [4] which focuses on extracting knowledge from data.

This methodology evolved as a branch of artificial intelligence dedicated to the development of self-learning algorithms. With machine learning, humans are not required to derive rules and build models for analyzing big amounts of data, but rather to offer *"a more efficient alternative for capturing the knowledge in data to gradually improve the performance of predictive models, and make data-driven decisions"*[33].

Machine learning is not a popular methodology that is only used on research. It is *"already being used in your daily lives"* [16] even though we may be not aware of it. Examples of common applications are:

- Getting a set of pictures by keyword in a photo gallery (as it happens in Google Photos[16].
- Email spam filters.

[16]Suggested Sharing, Shared Libraries, and photo books in Google Photos utilize machine learning to group photos together (`http://www.zdnet.com/article/google-weaves-machine-learning-into-new-google-photos-features/`)

- Content moderation filters, as seen, for instance, on the web search engines to detect and discard graphic images.
- Automatic generation of music playlists for a specific user based on his musical tastes (as seen, for instance on Spotify[17]).

The literature often refers three types of machine learning: supervised, unsupervised and reinforcement learning. They are presented on the subsections to follow.

2.6.2 Types of Machine Learning

Supervised Learning

Supervised learning is a type of machine learning applied when we want to *"make predictions about the unseen or future data"*[33] from a model that has been learning from labeled training data, a type of data that is previously characterized, either by a class, either by a set of attributes. In simpler words, machine learning *"is learning from examples"* [33].

The two major types of supervised machine learning are:

- **Classification** consists in predicting a *class label* from *"a choice of predefined list of possibilities"* [4] (labels), having into account a set of features (a list of known attributes). Those classes act like a group membership, being unordered [33]. For instance, we could classify a car as being a sedan, a minivan, a pickup or a sports-car giving the height, width, depth, number of doors, cylinder capacity.
- **Regression**, consists in predicting a continuous value giving, also, a set of features. One example is predicting a price of a house or apartment given its location, number of rooms, area, etc.

Figure 2.6: An example of classification task distinguishing cats from dogs [26]

[17]Spotify's Discover Weekly: How machine learning finds your new music (`https://hackernoon.com/spotifys-discover-weekly-how-machine-learning-finds-your-new-music-19a41ab76efe`)

Unsupervised Learning

Unsupervised learning is a machine learning technique concerned on finding meaningful information from data that is unlabeled or its structure is not known. In simpler words, the goal of unsupervised learning is to discover unknown patterns on data.

The two most common tasks of unsupervised learning are:

- **Clustering**, which is a data exploratory analysis technique [33] that allows us to relate information in groups without having any prior knowledge about possible relations that can exist in-between. One example of a task that can be accomplished using this technique is the discovery of new market segments given a non-identified set of customers.
- **Dimensionality reduction**, which consists in removing dimensions (individual characteristics, also known as features) while retaining most of the relevant information. It is commonly employed when the high dimensionality of the features degrades the performance of the computational system or hardens the data visualization. It is also applied to remove noisy data that is capable to *"degrade the predictive performance of certain algorithms"* [33].

Figure 2.7: An example of a clustering task, which distinguished 3 different groups [26]

Reinforcement Learning

Reinforcement learning is the type of machine learning which is more closely related with AI. This type of machine learning is about learning *"what to do"*[43] and *"how to map situations to actions"* [43].

Regarding that, its goal is to build an agent that progressively learns better, based on successive *"interactions with the environment"* [33]. At each interaction, an *action* is dispoleted and a *reward signal* is given to the agent. The main goal of the agent is to learn a set of actions that are capable to maximize the returned reward, either, by trial-and-error, either by deliberative planning [33].

Some examples of reinforcement learning include:

- A chess game agent, whose objective is to win a game.
- A robot that learns how to jump between platforms.
- A drone flying in autonomous mode that decides if it will continuing to fly or if it has to go back before the battery ends.

2.6.3 Fundamental concepts

Machine Learning Workflow

There is an almost standard way of using machine learning. The steps are the following:

- Data Collection Phase.
- Data Preparation Phase.
- Data Splitting Phase.
- Training Phase.
- Testing and Validation Phase.
- Deploying Phase.

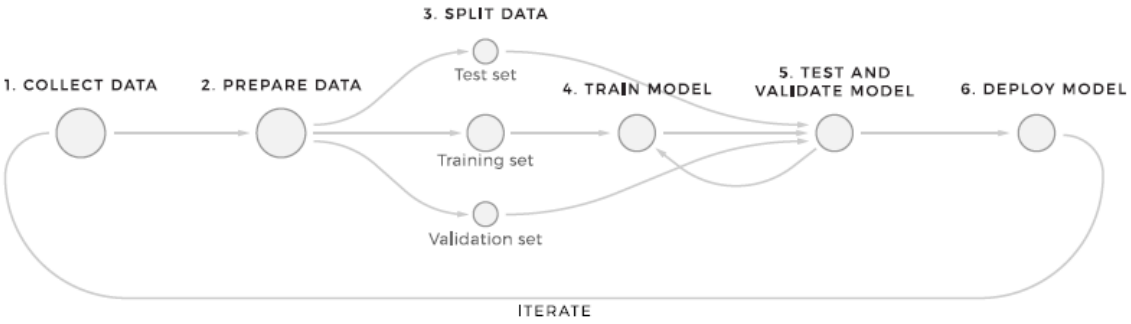


Figure 2.8: The typical machine learning workflow [26]

The **Data Collection Phase** consists on gathering data from a raw data source (for instance, in a database) or, from an already processed source of data (like UCI Machine Learning Repository[18]).

Then, **Data Preparation Phase** is used for preparing the data for processing. This preparation can be achieved by normalizing features (for example, normalizing class labels that vary their description in case letters, removing special symbols) or scaling the data for making the data representation more suitable for some algorithms, that are very sensible to the scaling of data (like SVM) [33].

[18]UCI Machine Learning Repository (`http://archive.ics.uci.edu/ml/index.php`)

After having prepared data, the algorithm must be fed for learning from data. This process is called training. But before, the data must be split. This is accomplished under the **Data Splitting Phase**, where the original dataset is divided in two parts: the *train* and the *test* dataset. The training dataset is a dataset that is used as example for feeding the algorithm that is building the mathematical model in training phase, while the testing dataset is used to evaluate the quality of the built model and its consequent predictions (under the **Testing and Validation Phase**).

Finally, under the **Deploying Phase**, the model is prepared and optimized for use on production applications.

Generalization, Overfitting and Underfitting

In machine learning, an algorithm fits a model to data. This model is, in fact, like a mathematical function and its goal is to predict the actual class or continuous value resulting from the input of a given data object.

If we overtrain our model, or, in other words, if we train the model with the same data all and over again, it will make a prediction with almost 100% accuracy over already seen data but, it will perform poorly when unseen data is presented. This phenomena is called **overfitting** and happens when a model is incapable of providing a good **generalization** to unseen data. Otherwise, when we undertrain our model, it becomes pessimistic: it is incapable of predicting the class of either the training data, either unseen data. This phenomena is called **underfitting**.

The figure 2.9 presents the relation between the model complexity and the model prediction error. The top left plot presents a case of underfitting, highlighting that, despite of being a simple model, it has a high training and prediction error.

In the top right, the plot presents a case of overfitting. The model becomes too optimistic because it is overtrained, meaning that the training error is low, but the prediction error is high regarding its difficulty to predict value from unseen data.

Finally, at the middle top, it is shown how a good generalization looks like and presents the best trade-off between the model complexity and the model prediction error.

Model Selection and Cross-Validation

As described previously, finding a good generalization is the objective of machine learning. Achieving it requires a more careful examination, for understanding which generalization is the best.

The definition of Model Selection embraces its full meaning here. It is a methodology that consists on *"tuning and comparing different parameter settings to further improve the performance for making predictions on unseen data"*[33]. These tuning parameters are also known as *hyperparameters*.

Choosing those parameters carefully is not enough by itself. There must be means for obtaining performance metrics. The two most common means are the **Holdout Cross-Validation** and **K-Fold Cross-Validation**.

The **Holdout Cross-Validation** is the most simple and popular way used for evaluating a given model. It consists on spliting the initial dataset in two parts: the first for training and the second for assessing its performance (for testing). Unfortunately, this method is discouraged because it only makes use of a single test iteration and if more than one is done,

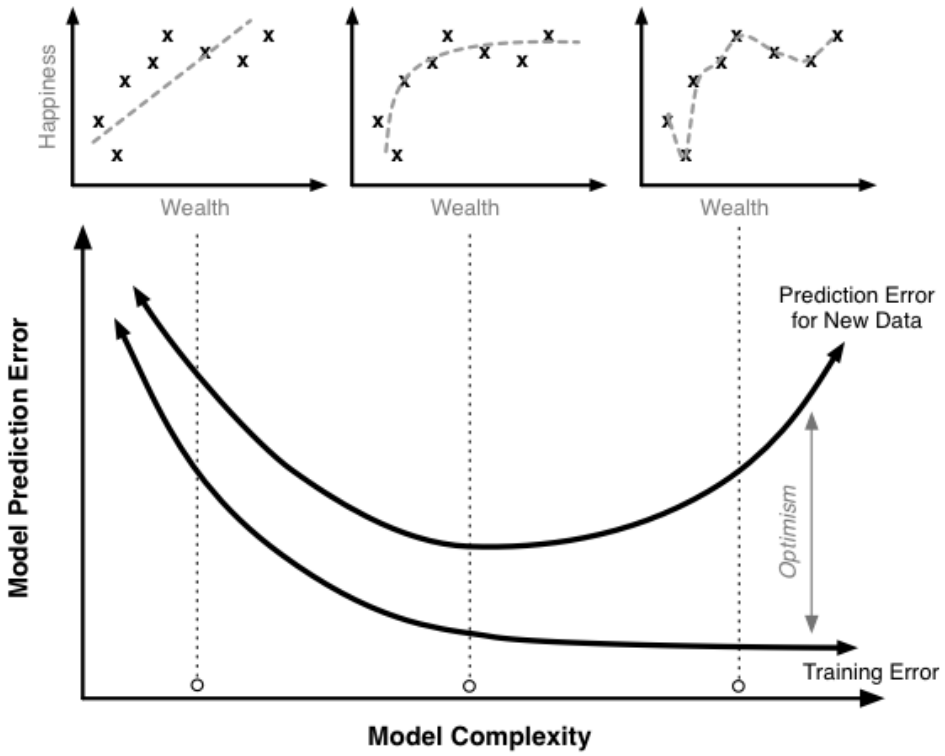


Figure 2.9: Presentation of the relation between the model complexity and the model prediction error (taken from [14])

---

the same dataset for training the model more than once, making it optimistic (or in other words, making it overfit).

The other technique, called **K-Fold Cross-validation**, consists in splitting randomly the initial data into k folds (without repetition) where $k-1$ folds are used for model training and the remaining one is used for testing. This process is subject to be repeated exactly k times in such a way that it is possible to obtain k models and k performance assessments, calculating then the average of the estimates of each one of the groups, being a *"less sensitive"* estimate than the one provided by the holdout [33].

Improving models performance with Ensemble Learning

Parameter tuning and cross-validation help on adjusting the model bias. However, there are also *"methods that combine multiple machine learning models to create more powerful models"* [4]. These methodologies are called **ensemble methods**.

We may perceive an ensemble as a set of experts from which we gather a value (a prediction), allowing us *"to strategically"* combine them [33].

Ensemble methods are commonly separated in two different categories:

- **Averaging Methods** are methods compounded by several independent estimators, which use the average of the predicted value of each one of them, as result.

- **Boosting Methods** are methods that combine weak and inaccurate estimators for creating a more accurate one.

An example of an averaging method is a *random forest*. A random forest is, essentially, an assortment of slightly different decision trees. As decision trees tend to overfit [4], this methodology tends to reduce overfitting by averaging the results. On the other hand, an example of a boosting method is AdaBoost [38].

The figure 2.10 shows how ensemble works, using an approach called *majority voting*.

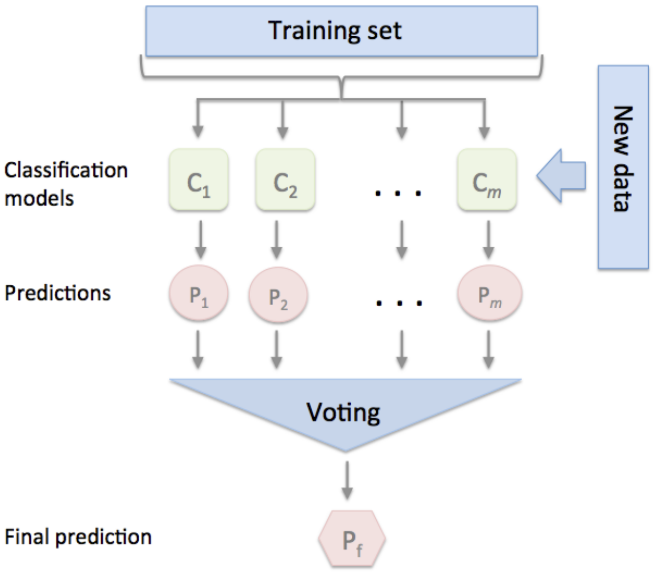


Figure 2.10: Majority voting used in a machine learning ensemble method (taken from [33])

For a more insightful explanation about how ensemble learning works, please refer to the references [16], [33], [4] and [39].

2.7 Summary

This chapter started by presenting a set of different but related concepts.

A **Smart City** is presented as multi-dimensional definition focused more on the human and urban development, where ICT works as a mean to an end. One of the main smart city concerns is the mobility, and it will be subject of further development in the following years to come.

Vehicular Ad-hoc Networks are reaching maturity, opening opportunities for further development on the same topics covered by *Smart Cities*, such as *smart transportation*.

On the other hand, we learned that the **earth shape** is irregular, requiring several approximations for warranting spatial information usability, either in analog way (maps), either in the digital ones (GPS, databases, etc). Operations concerning this type of data must be done carefully. One example of operation that must be carefully handled is the distance between two points.

Data processing is a very important topic for the development of this work. Under this topic, we explained some important notions such as concurrency, parallelism and distributed computing. More complex processing paradigms like batch, real-time and hybrid processing are presented, as a manner for introducing the readers to the current data processing paradigms. The development of the data processing paradigms evidence the urge of methodologies and technologies which appeared as an answer for the development of technology, particularly, to the Web 2.0, Cloud Computing and Internet of Things.

Finally, **Machine Learning** is presented as a methodology for extracting knowledge from data. It can be useful in a wide-range of areas, solving problems like classification, regression, pattern detection and more. Machine Learning has an almost standard workflow that consists on 6 phases: data collection, data preparation, data splitting, training, testing, validation and deploying. The phases requiring the most scientific effort are the training and testing phases because the main goal of machine learning is not finding the model delivering the best precision, but rather chosing the one that offers the best generalization. Cross-Validation is a possible solution for minimizing the effect of overfitting, finding the most general model that fits the data. Finally, we present a brief definition of the ensemble methods, which are a type of machine learning estimators that conjugate multiple estimators that, together, perform better than a single one.

Chapter 3

Related Work

3.1 Introduction

This chapter presents approximations done by the scientific community as an effort for tackling several problems which in part, intersect the interest of this work. These themes include map matching, travel time estimation, prediction, travel time variability, among others.

3.2 Bus Trajectory Identification by Map-Matching

Raymond and Imamichi [34] solve the problem of identifying bus trajectories from a geospatial-temporal dataset, by applying a *simple and robust* technique which results from the combination of map-matching, a variation of the bag-of-words heuristic and a dimensionality reduction.

The authors focus on three important topics:

- The importance of spatial-temporal datasets.
- The noisy and sparsity characteristics of GPS data.
- The notion and the problem setting.

Despite of the many advanced map-matching techniques claims about being capable of achieving a high accuracy, only few public datasets trajectories exist, supporting them. Therefore, *such datasets are highly valuable for map-matching.* In the oposite direction of some studies which do not use real world data, this one makes use of a real world dataset which belongs to Rio de Janeiro city hall open data initiative[1].

Several problems arise from the usage of GPS trajectories from buses. *GPS datasets are noisy and sparse*, what means that the overall shape of the traversed routes are not easily obtained. Also, buses may behave abnormally, drifting away from their predefined routes. This is subject to happen due to special events like festive events or unexpected traffic situations like congestion or road closures.

Regarding this, the authors determination in defining the problem setting delivers some answers which help understanding not only this problem but also similar types of problems too.

[1]Rio de Janeiro City Hall Open Data Portal (`http://data.rio/`)

This work is modeled to handle bus trajectories, so it is needed to have two types of data: data produced by the position variation from each bus (latitude, longitude, time stamp) and information about the bus network, regarding a set of predefined routes.

Two different classification problems arise, from actual perspective:

- The identification of the predefined route for each bus GPS position log.
- The identification of the predefined route of each bus where predefined routes are not reliable.

On each of the cases, several assumptions are done. In the case of the first one, the authors assume that *"all buses strictly follow their assigned routes"*, while in the second, they rely on the observation that *"there should be a group of buses serving the same route, and therefore"*, that it is possible to predict a trajectory of an unknown bus *"by comparing its trajectory to those of buses with known routes"*.

Several insights about the techniques are extensively done and can be consulted on [34]. The most important ones refer to the map-matching of GPS sequences, and to route similarity and comparison. The two topics have a big relevance on the understanding of general operation, on the type of applied machine learning (unsupervised vs supervised learning) , and on the heuristics borrowed from text mining (like *bag-of-words*, which is a role model under the document classification problem).

As concluding remarks, the authors found that the map-matching based on the Hidden Markov Models was proven as the most significant, regarding its effectiveness on the noisy and sparse GPS data, challenging potential new out comers to use different types of map matching in the same data, and test their accuracy on the same dataset.

3.3 Comparing state-of-the-art regression methods for long term time prediction

Sometimes the literature is very vague on expressing the true dependencies and urging necessities of the public transport agencies. Also, most of them, ignore the operation planning of a public transportation agency.

The work in [27] makes an import remark on understanding how the public transportation agencies planning works, how operational systems operate and on *how* and *which* metrics have relevance. This work also makes an important overview on how can one handle the data from those operational systems with the objective of applying different regression methods for obtaining the best approximation for a long term time prediction.

Long-term Travel Time Prediction (TTP) is a travel time prediction method based on historical data retrieved in a long-term base, in conjunction with actual measures. The authors defend that long term TTP is not used due to the *"implementation of dispatch systems that allow monitoring of the fleet"* [27] and also, due to the *"lack of knowledge"* [27] on how to handle this type of of data in order to *"reduce costs and/or increase clients satisfaction"*[27].

Regarding this, the paper later focus on making the data and experimental description. For curiosity sake, the experiments described in this study, use data from the same bus carrier from where the data used on this dissertation is generated, but from different sources. Their dataset, which is more complete than ours, comes from SAEI (a bus dispatching system) which makes use of *"different reckoning sensors supplemented by differential GPS technology"* [27], *"voice and data communication using Terrestrial Trunked Radio (TETRA)"* [27] and others.

SAEI is designed for control, but despite of that, it is the main data source for an existing data warehouse owned by STCP, which is very relevant for management purposes. This data warehouse is very detailed, *"allowing a very detailed level of analysis"* [27]. This study uses data from SAEI system, for predicting travel time of trips of the routes, in a end-to-end fashion, *i.e.* only taking into account the beginning and end times. The data warehouse provides information until the *"bus stop level"* [27], and bus stops passing time is available but this information is discarded in the scope of this paper.

Other hints are provided, regarding the spatial-temporal nature of the data. In particular, they refer the usage of a 30-day time stamp sliding window regarding the expectation of the last days giving more information about what can happen in the three days to follow.

Finally, a deep overview is done, with the purpose of choosing the regression methods. To do that, they define which characteristics should the algorithms have. The chosen algorithms were SVM[2], PPR[3] and RF[4] - without any special justification from the authors. Furthermore, the study regards tests and considerations that result from the comparison between the algorithms operation and quality of their results. For more details, please refer to this reference [27].

3.4 Empirical Study of Travel Time Variability Using Bus Probe Data

As the name implies, this work [6] is focused on exploring techniques to classify the usual behaviour of a bus network. First, they starts by defending that one of the most valuable resources that can be given to the bus passengers is the estimated time of arrival. Their argument is that, by enabling bus passengers to know *"their departure time and arrival time at the destination"*[6] they can *"reduce their waiting time at the bus stop"*[6]. It also presents a solution for better understanding the time variability of a bus line, because the bus line *"reliability is closely linked to the unpredictable variations"*[6], meaning that bus passengers are unwilling to use bus lines which behave irregularly. Also, this variation makes any possible prediction about the estimated time of arrival or about the trip time less accurate.

To achieve such result, they developed a method which consists in dividing the day in eight different time periods: early morning, morning peak, late morning, mid-day, early afternoon, afternoon peak, evening and late night. The time periods can be observed in further detail, in table 3.1. Then, the average travel time between each of the bus stops is calculated and accumulated accordingly (each interval with another interval in a different time, but in the same departure and arrival bus stop), for calculating the standard deviation and the average of the time.

The used probe data was provided from a Japanese company (NISHITETSU), referring to approximately one month (from November 21st to December 20th, 2013). It contains information such as "time and position of buses, route number, a day of a week, numbers of bus stops, travel direction and bus performance history"[6].

This work resulted in a detailed investigation and classification of the time variability of the routes, having successfully distinguished the stable bus lines (bus lines which usually

[2]Support Vector Machines (`http://scikit-learn.org/stable/modules/svm.html`)
[3]Project Pursuit Regression
[4]Random Forest

Table 3.1: Time Periods from this study[6]

No	*Time*	*Time Period*	*Symbol*
1	05:00:00 - 07:29:00	Early Morning	EM
2	07:30:00 - 09:29:59	Morning Peak	MP
3	09:30:00 - 11:59:59	Late Morning	LM
4	12:00:00 - 12:59:59	Mid-Day	MD
5	13:00:00 - 15:29:59	Early Afternoon	EA
6	15:30:00 - 17:29:59	Afternoon Peak	AP
7	17:30:00 - 19:29:59	Evening	E
8	19:30:00 - Later	Late Night	LN

perform similarly) from the unstable ones (those bus lines with hard to predict behaviours), as presented in the figure 3.1.

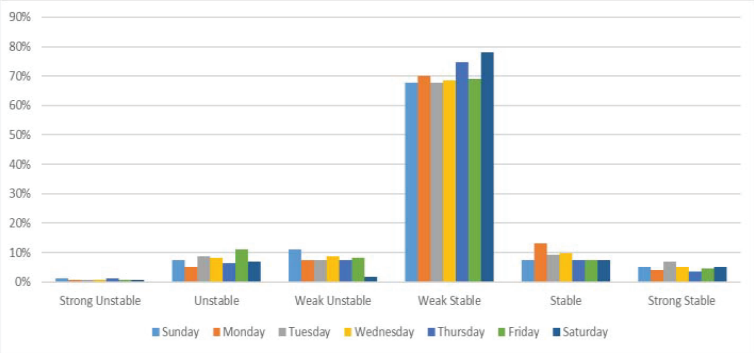


Figure 3.1: Comparison of the different time periods in terms of stability [6]

3.5 Using Bus Probe Data for Analysis of Travel Time Variability

Regarding the fast progress of the information technology, the authors of the work [45] agree that new insights about the traffic phenomena can be solved. Also, they refer that the identification of particles moving through the city is one of the key areas of application on the traffic and transportation study areas. This work focuses first , on presenting a methodology for using GPS data with the aim of turning it meaningful to transportation analysis. It summarizes also methodologies and transformation techniques that can be applied to the GPS data. The final aim of this study is to propose an approach for evaluating the bus lines quality of service, from the point of view of the travel time stability and reliability.

The used methodologies are very broad. The second chapter presents a detailed perspective on how GPS data can be used for traffic analysis. In other perspective, the third chapter focuses on solving the study aim.

In what concerns the use and transformation of GPS data, a general procedure is presented, consisting in four steps: map matching, data reduction, data processing and data reporting. Map matching is the act of *"matching the GPS data to a digital map"* [45]. Data

reduction consists, as the name implies, into reducing the data by transforming it into links (or paths). Data processing is the act of operating individual vehicle data. Finally, data reporting consists in *"generating indicators easily understood by the users"*[45]. All the reasons and considerations about why is this general procedure is needed, are explained in [45], in further detail.

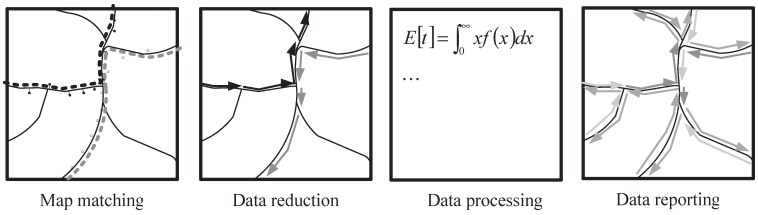


Figure 3.2: Data transformation steps described by Uno et al. [45]

For evaluating the travel time reliability, a study was conducted over 18 bus lines. Some important information is outlined. For example, the Ministry of Land, Infrastructure and Transport (of Japan) is collecting bus probe data in Kansai district since 2001, with a capture granularity of 1 second. Their dataset does not consider any information about the bus network (bus line, bus stops, etc). Therefore, a supplemental survey was needed. It consisted in assigning each researcher *"to each bus route"*[45] for observing *"the operational modes"*[45] while capturing the location of the bus location with a GPS unit.

Through the same chapter, some hints are given regarding the bus probe data:

- The bus probe data is reviewed, with the objective of extracting the trajectories from them. The researchers making the survey make annotations about the times where the bus stopped in a bus stop, where the operational modes of the bus, verifying if it stopped in a bus stop, in a traffic light or sign, etc.

- Then, they correct the travel time by detecting the amount of time where the bus was stopped.

Later in the same chapter, it is also explained how they have developed a procedure for estimating the travel time distribution of a bus line, explaining also the used mathematical functions and a case-study, showing the level of service of a line.

The level of service is a metric focused on both efficiency and stability of the road transport conditions. The optimal road transport condition happens when the bus passengers are able to *"reach their destinations in rapid and reliable manner"*[45]. The level of service can be divided separately for evaluating different dimensions of the bus network:

- The coefficient of variation (COV) of a bus line gives hints about the reliability of the network.

- The average travel time for one kilometre gives hints about the efficiency of the network.

Having these into account, they managed to implement a methodology which is capable of evaluating the metrics generated by the already existing infrastructure.

3.6 A review of travel time estimation and forecasting for Advanced Traveller Information Systems

This academical work [28], presents an overview on the literature about the travel time modelling and introduces the fundamental concepts and methodologies associated with it. Most of the attention of this review is focused on the concepts of **time estimation** and **time prediction**, which are the *"two of the most relevant challenges in travel time modelling"*.

In the introduction, the challenges and base concepts are introduced. Regarding the fact that the traffic volume and the congestion increased in the last years, several problems arosed. The technological advancement enabled the creation and diffusion of the traffic information. Regarding that, two types of systems have been developed over the years:

- **Advanced Traffic Management Systems** (ATMS), which are used, usually, *"by traffic engineers and administrators in order to enhance mobility and obtain a more efficient and safe traffic in road networks"*[28].
- **Advanced Traveller Information Systems** (ATIS), which are focused on providing to the commuters, *"the necessary traffic information and tools to enable decision-making"*[28].

Concerning the previous systems, three definitions are presented. The first is called **travel time**, which is *"defined as the total time for a vehicle to travel from one point to another over a specified route, taking into account the stops, queuing delay and intersection delay"*[28]. **Time estimation** and **time prediction** are also introduced, as main issues of the previous definition:

- **Travel time estimation** results from calculating *"travel times of trajectories that have already ended, using data captured during the trip"*[28].
- **Travel time prediction** makes use of *"the current and past data to forecast the travel time in future time intervals"*[28].

This review claims that estimation and prediction are very useful on the traveler information systems, and because of that, that these themes deserve focus on this study.

Before advancing into studying the travel time estimation and prediction in further detail, the authors have chosen to describe aspects related with the data sources, as they differ a lot and must be treated differently. Given the devices used, they classified the data sources in two different types:

- **Point detectors**, which are devices set on different and fixed points of the road, capturing traffic variables.
- **Interval detectors**, which *"capture data that enables the direct calculation of travel time between two points, as opposed to point detectors that are only able to describe a single point of the road"*[28].

The GPS enabled vehicles are classified under the second point. However, it is made a distinction between GPS enabled devices as well. The first type defines **floating vehicles**, which are vehicles *"specifically employed for data collection purposes"*[28]. On the other hand,

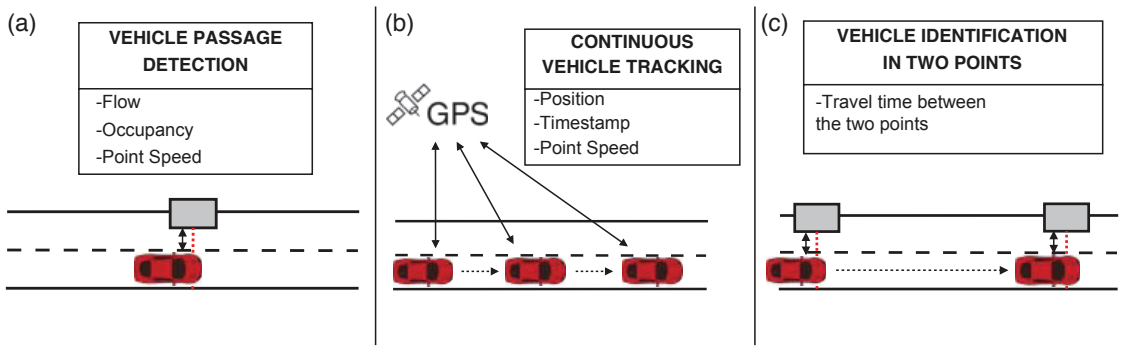


Figure 3.3: Traffic data sources. (a) Point detectors, (b) probe vehicles and (c) Interval detectors (figure from the review [28])

---

probe vehicles *"are passive vehicles that travel in the road network for other reasons"*[28] (as seen in the Figure 3.3).

Looking further to the third chapter, the definitions and differences between the travel time estimation and prediction are explained. As a rephrasing of the previous definition of travel time estimation, the authors clarify that the *"travel time estimation consists in reconstructing travel times of trips completed in the past based on data collected during the trip"*[28]. The figure 3.4 shows graphically, the difference between these two concepts.

The chapters 4 and 5 of this work are focused on explaining travel time estimation and predictions models, respectively, in a high-level of detail. For more information regarding those methods, please, refer to this study [28].

The review ends after a evaluation and a discussion. The discussion makes a critic to the state of the art of the time travel modelling methodologies and results.

The first appreciation is that the method choice depends very closely on the *"typology, quantity and quality of the available data."* [28].

Then, it is also referred that most of the authors of reviewed studies *"attempt to use real data to validate their proposals, especially in the case of prediction"*[28]. Despite of being increasingly easy to acquire real data, obtaining data from *"specific traffic contexts such as incidents, road works, congestion"*[28] can be more difficult. Regarding that, in [28] the authors defend that *"simulated data acquire more relevance in estimation models because of the difficulty in obtaining ground truth travel time measurements for validation"*[28].

An analysis regarding the popularity, accuracy and the efficiency of the methods is also available.

It is clarified that the *"methods based on traffic flow theory and simulation are more popular for estimation tasks than for prediction"*[28], a characteristic that was expected due to the fact that prediction models are *"estimation models that become predictive"*[28].

In terms of accuracy, the authors of this review defend that such results *"should be handled and interpreted with caution"* because there is no evidence about the data provided by the studies and *"no complete comparative studies"*[28] have been done.

Finally, with scope on the efficiency, [28] refers that despite of this being an import factor to have in account when comparing techniques, it is *"completely ignored by the authors and is almost never mentioned in the publications"*[28], making it harder to make any conclusions.

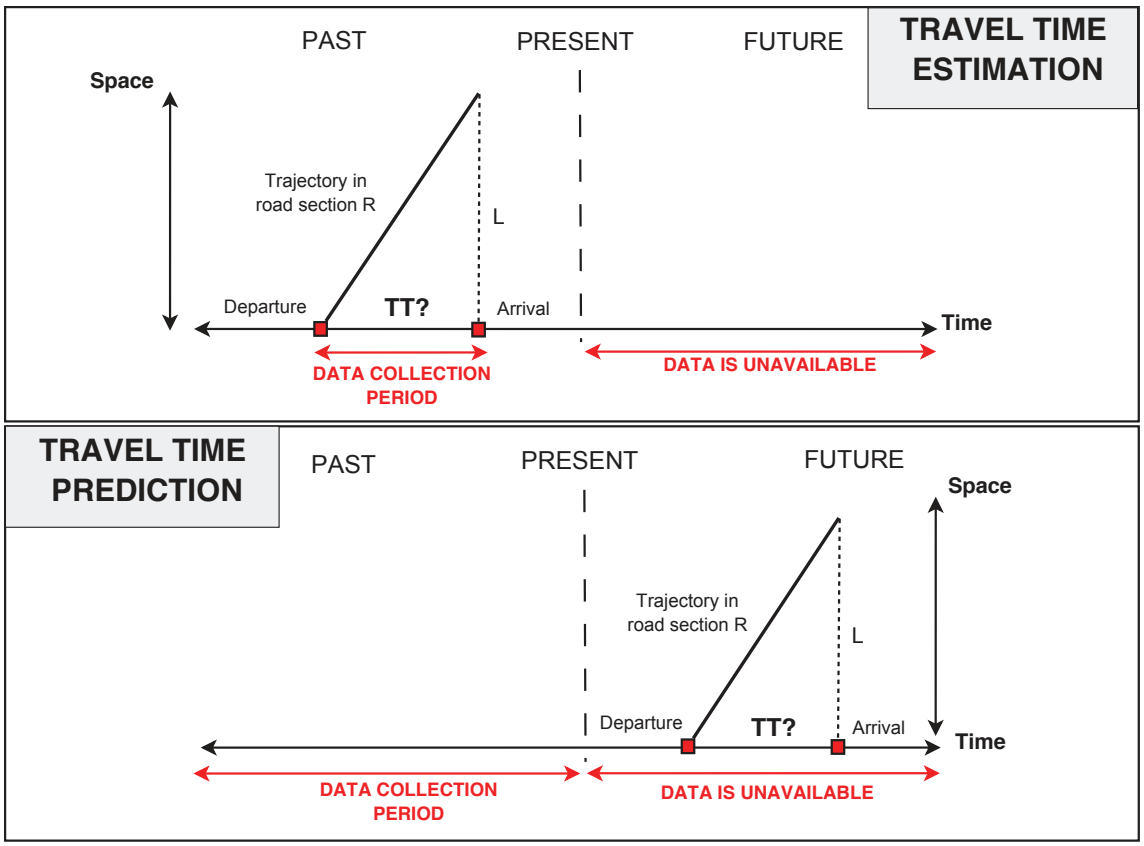


Figure 3.4: *"Difference between time travel estimation and prediction"*[28]

3.7 Real-time Trip Planner in Urban Public Transport

The study in [3] was driven in the framework of the master thesis dissertation of David Manuel de Oliveira Alves, a student of the Master in Civil Engineering from Instituto Superior Técnico, in October of 2011.

The objective of his work was to create a real-time trip-planer for Carris, the major public transportation carrier in Lisbon, Portugal. Despite of being a different project, it aims for the same type of results, namely, getting the estimated times of arrival.

Before starting to solve the purposed problem, David studied the state of the practice in terms of real-time information systems on public transportation. The document summarizes the current devices and mechanisms like *dynamic message signs* and internet and mobile devices. In particular, it is demonstrated one of the operating systems in that time, **NextBus**, and its architecture, which differs a lot from the one presented by this dissertation. The author also presents real world examples of systems deployed world-wide, in cities like London, California and Singapore. A summary table makes evidence of the lack of a real-time travel time forecasts in a list of 25 cities.

Then, the study presents the log data used for his study and explains how the data mining techniques were applied to the logfiles which were obtained through a data availability protocol signed between EFACEC (which produced the system and owns the system operation) and MIT Portugal Program, under the project CityMotion [21]. The main difference between

Figure 3.5: NextBus Architecture (figure from [3])

those log files and the position log database that we have access (as seen later in the chapter 4.3.1) is that their attributes consider the stop identification, a bus line identification, a vehicle identification and other bus network specific attributes. Therefore, the problem of matching GPS positions with a given line did not existed and thus, the classification problem was already solved.

Regarding this, most of the problem is driven in a more straight fashion. First, he developed a set of procedures to tackle some intermediary problems. Then, a data mining pipeline was created, and a clustering algorithm was chosen. The results were evaluated later, for understanding if the prediction model was or not right, and finally, a model for simulating *"the bus network operation and travel time prediction was developed within the framework of Agent-Based Simulation"*.

More details about this work may be found in [3].

3.8 Summary

This chapter presented a set of related academic works. They were not chosen randomly and each one reveal something interesting: an observation concerning the data, a methodology, an exploration technique, and even a global comparison of the most popular methodologies.

The first thing to notice is the assertiveness of the related work. For instance, *Bus Trajectory Identification by Map-Matching*, by Raymond and Imamichi presents a clarification about importance of real data sets, highlighting the importance of validating and comparing new methods.

Then, *"A review of travel time estimation and forecasting for Advanced Traveller Information Systems"* not only confirms the claims of the previous study, as it also presents significant terminologies (like the difference between time estimation and prediction) while presenting data gathering techniques and their impact on the data treatment.

The other works are also significant, presenting relevant insights about the problem setting and about the development of the bus infrastructures in different places (Porto, Lisbon, Japan) and revealing approaches that can be followed on the development of our solution. One example is in [27] that highlights the performance of the different machine learning methods.

Chapter 4

Matching lines with GPS logs and Building Performance Indicators

4.1 Introduction

The objective of this chapter is to explain what are the problems that need to be solved and what information is available for helping solving it:

- The section **Problem Setting** explains what are the main objectives and challenges of this work.
- The section **Available Context** presents what and how are the existing data sources.
- The section **Exploring, visualizing and choosing data** presents some interpretations about the data, its shape, and how it can be chosen and visualized, for a specific purpose (such as debugging).

4.2 Problem Setting

As previously seen, Porto has a Vehicular Ad-hoc Network deployed over the city. This vehicular network connect buses, taxis and garbage trucks, being capable of delivering Internet connection to the bus passengers. The information generated by the movement of these nodes is stored into a central database (described in the next section), and it can be used for developing new mobility applications. There are 4 major problems to be solved:

- The first, refers to the missing link between the log position database and the bus network information: there is no information, in the central database, about the route being completed by a given node. This gap in the knowledge makes it impossible to process the estimated times of arrival. It is required to *match* a GPS position log with a given line, before going further.
- Then, an extensive study must be done, in order to understand what data will be used for producing such time estimations (passing times, times between bus stops, etc).
- Time estimations are based in the past. A more sensible approach would be using past and current data to deliver a more accurate result, process that is commonly known as prediction. This is possible using a data science technique like machine learning.

- Finally, having data is not useful *per se*. Applications must be developed for delivering the results produced by the newly created system.

The following section, explains what information is available, so it is possible to reach such results.

4.3 Available Context

This section objective is to present the available context, in terms of what data is available and discuss its quality.

4.3.1 Data Sources

There are two major data sources:

- *vanetV3*, a *MySQL* database which has stored logs from the nodes (buses), in terms of its position and speed.
- *STCP website*, which provides a big collection of information about the bus carrier infrastructure.

4.3.2 Insights on the log database

The positions log database is a central point of raw data storage. In the following subsections, it is presented a brief description about its tables, attributes and data quality attributes.

Database Tables

vanetV3 is a MySQL database, containing five different and *relationshipless* tables:

- `lma_data`, whose purpose is to store the LMA information about the amount of internet traffic data, such what traffic comes from cellular or using the mesh network. This is the data that traverses the central server and represents the uplink and downlink users data.
- `node_data`, whose purpose is to store information about the nodes (buses with OBUs), in terms location (latitude, longitude, altitude), speed, number of hops, traffic, etc.
- `rsu_data`, whose purpose is to store information about the RSUs in terms of the amount of internet traffic.
- `sessions` and `sessions_data`, which are meant for storing information about the WIFI sessions of the mobile clients: MAC address, incoming and outcoming traffic, location, etc.

Attribute	Description
node_id	Node identifier
system_time	The node's OBU OS time
GPS_time	The time of the GPS chapture
server_time	The time when the server received the record
latitude	GPS captured latitude
longitude	GPS captured longitude
speed	GPS captured instant speed
heading	GPS captured heading
hdop	GPS dilution of precision
accel_x	Accelerometer capture for the x axis
accel_y	Accelerometer capture for the y axis
accel_z	Accelerometer capture for the z axis

Table 4.1: Description of the attributes of the table node_data

---

A detailed view on `node_data`: attributes, granularity, quality and quantity

For the purpose of this work, the focus must be on the tables which present valuable data. The table `node_data` delivers most value as it contains *floating car data*.

The following table presents the **attributes** representing most of the interest.

The attribute `gps_time` identifies the exact time when the spatial attributes (`latitude`, `longitude`, `altitude`, `heading`, `hdop`) were captured. They enable the calculation of the average speed of a given node.

The **granularity** is also important and can be discussed in terms of time and GPS deriving attributes. The time between the insertion of two records in the database is 15 seconds. Latitude and longitude granularity is good, compressing at the least six decimal cases. It means that the precision range is about thirty meters - which may arise some problems but it is enough for most applications.

In terms of **quality**, the present data can deliver most of the answers. The only real problem with it, is that it does not hold any information about the type of node (if it is either a taxi, a truck or a bus) and its current state. Having into account this data, it is trivial to understand if some particle is moving or not, due to the fact that position and time stamp are being logged. In contrast, it is not easy to find what line is the bus doing if it is, indeed, a bus. Thus, this last problem has to be solved.

Another existing problem occurs due to the bad synchronization of this replica. It is being synchronized from the original that is controlled by VENIAM. When they change the original database (a table, a type, etc) or when there exist a power failure or other third-party problem, the replica is not able to receive new data.

Regarding this, there are consecutive days (reaching months) of synchronization with errors. This data is not suitable to be processed and therefore, a different time frame has to be chosen.

Finally, the **quantity**. Depending on the months and on health of the synchronization status of the database, the number of records vary, day to day. Looking to the most stable months, each day produces between 1.2 to 3 millions of SQL records - at the least, 36 million of records per month. As consequence, strategies for processing this data in parallel or even,

in distributed manner should be applied, depending on the target deploy system and speed requirements.

4.3.3 STCP Website as data source

While *vanetV3* delivers floating car data, *STCP Website* provides static information about the bus supporting infrastructure and environment, specifically:

- Bus stops (code, name, address and zone);
- Lines (code, name, direction).
- Stops within lines (which bus stops belongs to a given line).
- And more recently, the full path of the line (not available in the time of the implementation).

Their website also explains:

- The line naming conventions. The most important aspect is the existence of *night lines* that, as the name implies, are lines operated only by the dawn. These lines are often very similar to daily lines and may cause route matching problems.
- Service modifications, which can be useful to understand if a given algorithm classification is well done or not.

The extraction of this information into a computational friendly representation can be performed in several ways: one is scrapping the website, work that usually requires creating a program, recurring to some HTML or XML library (like LXML in the Python Language); other is to check the page source code for API calls; and finally, the third is using packet inspection tool (like Wireshark) to detect API calls while querying the site (from client-side).

Despite of the mostly static nature of this information, it may change over time when lines change, thus requiring it to be extracted again.

Finally, due to the spatial nature of this data, using spatial data structures might prove convenient for making spatial queries (querying a point for proximity queries, for example) is a foreseeable possibility.

Regarding all this, the extracted information should be stored into a file (or a set of files) with the objective of being later loaded by the applications. The spatial data structures can be built once and loaded when the applications need, or they can be built every single time that the application loads.

4.4 Exploring, visualizing and choosing data

The previous sections explained that two main data sources exist and that these data can be extracted in convenient way to be used computationally.

Regarding the spatial nature of the information, it is useful for the analyst to visualize the extracted data for having notion of its shape and distribution through the space, so he is able to become familiar with dataset and more aware of potential anomalies which are not visible at first.

QGIS[1] is extensively used for visualization. It is also a powerful tool that can be used for debugging results from the algorithm to be built and making queries.

In the following subsections, it is presented a small discussion supported by figures showing the data shape and distribution.

4.4.1 Position Log Data

The position log data is considered to be *raw* data, meaning that is data which has not been processed yet, being exactly the same as the one that exits from the vehicular network sensors.

There are various advantages that outcome from visualizing it graphically:

- Visualizing the position log data gives instant hints about the spatial distribution of the bus positions. This is useful for understanding potential issues on the design of the algorithm or in the bus line completion development.
- It enables the recognition of patterns in the movement of these bus, enabling the analyst on identifying the existence of the line visually. Important patterns include detecting lunch hours, returns to the bus station and bus line chaining (doing the same line in the opposite direction).
- It helps choosing fixed datasets for developing the study and implementing the algorithm, by helping on identifying captures which are not suitable for this study (like the ones from buses which are stopped all day long in the bus carrier parking lots).

One must also concern about the data health, a characteristic that refers to the balance between quantity, quality and the granularity.

The first indicator for noticing the health state of the data is to verify how many database records are stored in a day. Daily, there are stored 1.2 to 3 million records[2]. If the value is exaggeratedly lower than the normal quantity, there might be an error.

The second indicator that evidences a potential issue on data health are the gaps in the granularity. This issue can be easily visualized, either on a table form, either on QGIS, because the interval between records is much higher than the usual (up to three minutes).

The figure 4.1 shows a position log, from one bus, in March 1st. The red dots represent the bus GPS position variation along the day.

4.4.2 Bus network data description

The bus network data, which is available on STCP website, regards bus stops and bus lines of the bus carrier. Visualizing this data is very important. For instance, it is a way of understanding the shape of the bus lines, a characteristic that may be important on the development of the algorithm. Regarding this, the following paragraphs and figures describe the bus stops and lines distribution and shape.

Figure 4.2 shows the stop density over Porto city. The yellow dots represent bus stops, which are more dense in Porto's downtown. The bus stop presence is less dense once one exits Porto main area: less dense from Matosinhos and upper areas, below Porto in the direction

[1] QGIS Website (http://www.qgis.org/en/site/)

[2] This value can be obtained by consulting the source database

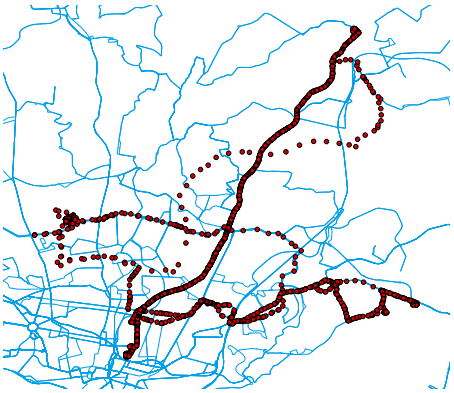


Figure 4.1: An all day long position Log from a bus from the first day of March

---

of Douro river (Gaia) and also less dense in the center east after Campanhã, in the direction of Valongo.

Then, Figure 4.3 shows the bus line map of STCP in Porto. The line coverage is so high that it is easy to recognize the city given the path of the bus lines. The density of the lines is as equally distributed as the bus stops because each line has a set of bus stops ie., they are related.

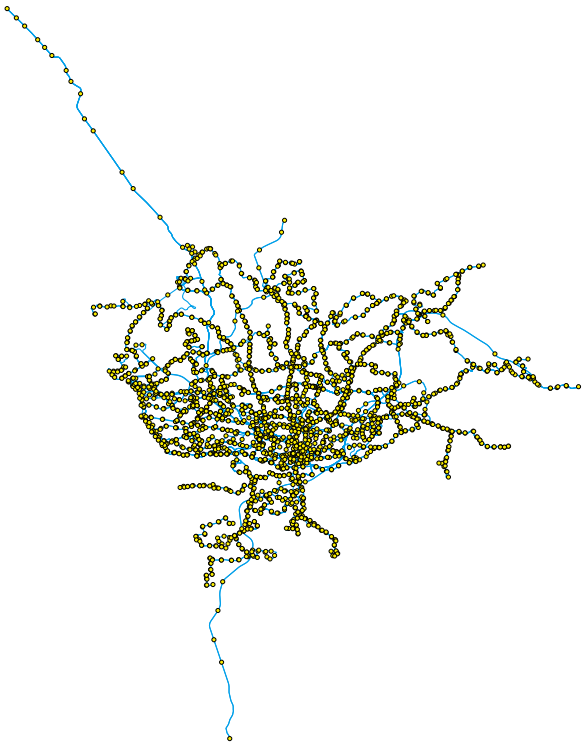


Figure 4.2: Bus stops of Porto

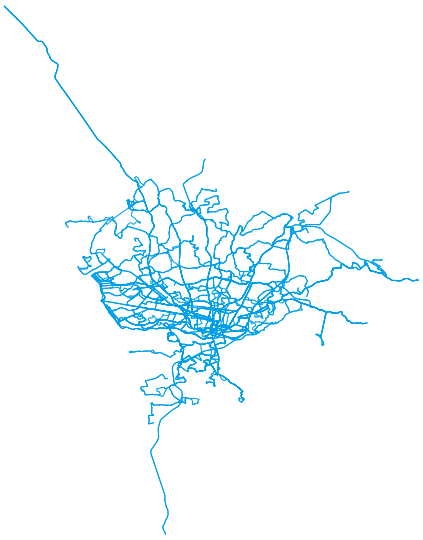


Figure 4.3: Bus lines of Porto

4.5 Performance Indicators

4.5.1 Definitions

There are several factors that can be used for evaluating the performance of a bus line. Some of the factors that may be explored are:

- The **completion rate**, which measures the number of bus stops completed out of the total bus stops that define a bus line.
- The **time between bus stops**, which measures the elapsed time between two bus stops. This can be seen as the period time understood between two states of inactivity.
- The **stay time**, which measures how much time takes a bus on each of the bus stops.
- The **passing time**, which consists on the moment that a bus was first detected in a bus stop. This metric can be perceived as the *"estimated time of arrival"*.
- The **total completion time**, that measures the total time that a bus took to complete a bus line.

Each one of these factors raise the concern about their relative importance on the context. Understanding which of them is best suitable for our use-case and if it is possible to complement each one of them for building a more reliable metric are natural questions.

The **completion rate** helps on understanding empirically if the line is suffering or not modifications, and on understanding if the suppressions are arising due to delays or because of external factors (like road words). Alone, the completion rate does not provide any more answers.

The **time between stops** enables a fine and detailed perspective of elapsed times between the bus line pairs that, together build the concept of bus line. The time between stops analysis

enables to understand if a segment is generally faster or slower than its counterparts, being a particulary useful metric for the city planers and bus fleet managers.

In a similar way, it is possible to measure the amount of time that a bus is stopped in a bus stop. The *stay time* measures that time and it may be useful for inferring the occupancy of a vehicle, given the total amount of stoped time or its popularity.

Then, the **stay time** captures the approximate time that a bus passes by a given bus stop. This metric is important because of its simplicity, and because it is easy to compare with the fixed time tables, known by the carrier. Also, they enable the inferring of the schedules by taking a simple statistical value like the average or the median from its values.

Finally, the **total completion time** may be useful for understanding what is the usual rhythm of conclusion of a bus line, and to study how actually is a bus line impacted during different periods of the day.

4.5.2 Main restritions

Despite of those methods being useful, they depend on the data granularity for providing a good accuracy. For instance, the **time between stops** and the *stay time* are hard to compute with a high accuracy on the system that we want to develop because of the capture of our granularity being too low, *i.e.*, too spaced in time.

Regarding this, only the completion rate and the passing time will be explored, despite of being easily implemented after matching the information.

4.6 Summary

This chapter started by highlighting the 4 fundamental problems that need to be addressed. Then, it is presented the overall context, for better understanding what knowledge can be extracted from this raw data.

The terminologies present inside the data sources have a clear inheritance from the vehicular network. Persons willing to explore this database should be aware of the terminologies, which are explained in the section 2.3.

Then, some focus is given to the process of choosing and visualizing data, as a manner for being aware of the layout of the data and its density, something that will certainly be useful when implementing the solution.

Finally, the performance indicators are presented for the knowledge of this project development, and its exploitability is discussed regarding this study's limitations.

Chapter 5

Architecture and Technical Design

5.1 Introduction

This chapter's objective is to present a theoretical foundation for the overall solution. Its outline consists on:

- The section **Requirements** which define the expected behaviour and function of the system we want to develop.
- The section **Architecture** presents the main architecture of the system and also its components and flows.
- The section **Technical Design** describes the main components of the architecture and discusses the most important design options.

5.2 Requirements

Having present the objectives and the available context, it becomes clear what the requirements are for designing and implementing an architecture.

Making a separation between the requirements which refer to the operation of the system and for those which refer to its behavior and function, the following subsection present the functional and non-functional requirements.

5.2.1 Functional requirements

The functional requirements are:

1. There must be a mobile application where a bus passenger is able to verify the estimated time of arrival of a bus.
2. There must be a dashboard for monitoring the line performance in terms of delay.
3. Due to the non-existence of a relation between the bus network information and the log position data enabling the construction of delay metrics, this data set must be matched first. The system should find what line was a bus doing and output a matching result identifying the node, the date of the capture and the identification of the matched bus stops with the corresponding detection time stamp.

4. In behalf of delivering metrics for analyzing the performance of a line, it is needed to understand if a bus stops, in a given line, is or not delayed respecting to a defined time-frame like *the past week*, *the past two weeks*, *the past month*, *the three later months*. (which resulted from the previous matching).

5. Regarding the amount of data, the system must be capable of processing the data differential between two days of capture in the maximum of three hours.

6. There must be an abstract way of delivering the different types of data to the different applications, giving opportunity to other applications being developed too.

5.2.2 Non-functional requirements

The non-functional requirements define that:

- In terms of resource constraints, only one computer is available. The target system has 8 CPU-cores, 32 GB of RAM memory, 512 GB of SSD memory storage.
- It should be easily deployed, regarding the research group potential out-comers and partners.
- It should be fault tolerant.
- It should be easy to scale, either vertically, either horizontally.

5.3 Architecture

With respect to these requirements, an architecture is implemented, with the objective of satisfying each of them.

The proposed system architecture, represented in the figure 5.1, includes several modules and data sources to support the processing pipeline.

From the bottom of the diagram to the top, please find the description of each element:

- **vanetV3 and the STCP website** are the main raw data sources. They make part of the architecture because, without them, it wouldn't be possible to match the records with a bus line due to the nonexistence of data to be processed or context, respectively.
- The **Extraction Scripts** transform the raw data, available on the website, in a set of well-known and well structured files containing all the context information such as the bus lines, bus stops, etc.
- The **Matching Unit** consists in a Python program which is able to match the log position data with the bus network data, producing matches: a set of objects regarding the node identifier, the date of the capture and the identification of the matched bus stops with the corresponding detection timestamp.
- The **Matches Database** is responsible for holding the Matching Unit results (match identification, matched bus stops, etc).
- The **Data Mart** helps to gather performance metrics from bus delays.

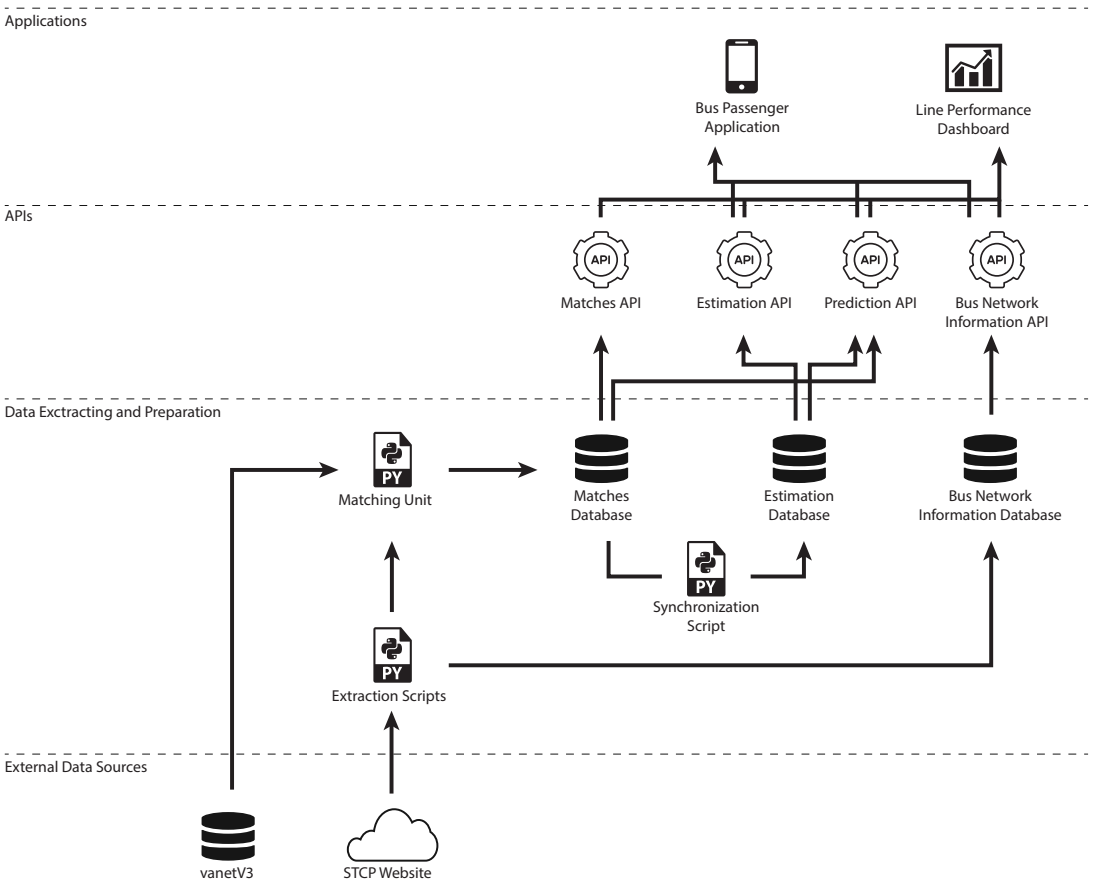


Figure 5.1: Architecture components and the data flow

- The **Synchronization Script** updates the Data Mart. It performs the synchronization between the Matches Database and the Data Mart, being aware of the differences since the last synchronization. It is meant to be executed every night.
- The **Bus Network Information Database** holds information about the carrier infrastructure, such as line, bus stops and their relation.
- The **Matches API** is an API for consulting the output that resulted from the Matching Unit and stored in the Matches Database.
- The **Estimation API** is meant to provide an interface for querying estimated times of arrival. It makes queries to the Data Mart and it should be used by both mobile and web applications.
- The **Prediction API** provides a service for making time prediction on-demand.
- The **Bus Network Information API** provides a front-end for the same named database and it is used to deliver information about the bus carrier infrastructure for both web and mobile applications.

- The **Line Performance Dashboard** is a dashboard for the bus carrier manager to consult the performance of the carrier's lines. It is a decision support dashboard because it provides hints about the health of the bus transportation system.
- The **Bus Passenger Application** is a mobile application meant to provide for the carrier's end-users estimated times of arrival for a given line and bus stop.

5.4 Technical Design

5.4.1 Overview

The Matching Unit is the central piece that drives the overall architecture. It implements an algorithm that matches the data from *vanetV3* with a bus line, given the context information existing in the STCP website. It also grants parallelism, with the goal of taking a better advantage of the computational resources.

To exploit parallelism, one can divide processing flow into smaller pieces and then, make use of entities which are able of processing each piece independently. Those entities implement a data processing pipeline, having a well-defined set of tasks, as seen in the figure 5.2.

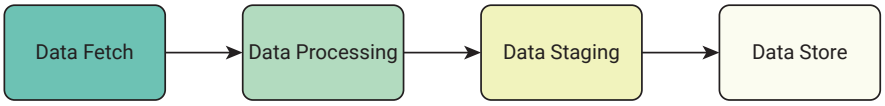


Figure 5.2: Data Processing Pipeline

The first task is called **Data Fetch**. As the name implies, it is the moment in the pipeline where the data is retrieved from the log database. As soon as the data is available, it is delivered to the next stage.

The **Data Processing** task consists in the algorithm implementation. It makes use of a heuristic and a validation procedure, as described in the subsections 5.4.3 and 5.4.3.

Once processing is done, the **Data Staging** task prepares that data to be stored into the Matches Database.

Finally, the **Data Store** task is responsible for storing the staged results into the Matches Database, for further analysis and use.

These pipeline stages have a temporal dependency but, all of them except the first can be parallel, or in other words, can be done at the same time. The Matching Unit implements an architectural style called *Worker Design Pattern*, that consists in having a set of entities where one is "master", controlling the process, and the others are "slaves", processing a subset of the problem. If the master entity is responsible for creating the jobs and the workers implement the other three-stages, two types of parallel execution can be achieved: data-level parallelism and task-level parallelism. The explanation regarding the entities and how they collaborate are explained in further detail in the section 5.4.2.

The following subsections explain:

- How *Worker Design Pattern* is designed and how it can be made effective on taking better advantage of the computational resources.
- How the algorithm is designed and what are the main principles and techniques which are crucial for solving this problem.

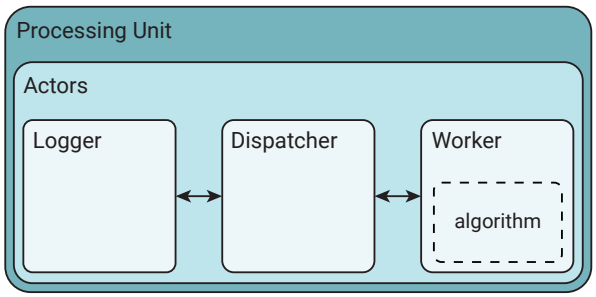


Figure 5.3: Matching Unit Architecture

5.4.2 Exploiting parallelism using the Worker Design Pattern

Despite of the matching algorithm being a central part of this work, additional measures need to be implemented in order to increase the throughput of the system, mainly, through the use of parallelism.

As clarified in the subsection 4.3.2, 1.2 million of records are, at the least, stored in the database per day. Although the data dimension is not in the "big data" level, it can be partitioned and processed in parallel, taking better advantage of the computational resources.

Generally speaking, software has been written for serial computation [7]: one single processor, with only one instruction to be executed at any moment in time. But today, major part of the processors are called multi-processors. This means that multiple instructions can be done at the same time. If no measure is taken for making a software parallel, it will work in a sequential way, and potentially, taking more time than its parallel version.

One should not confuse concurrency with parallelism. The objective of concurrency is to manage the share of the state between different threads, while, the objective of parallelism is to use multiple CPU cores for achieving the maximum performance, reducing the processing time. Those entities should be running on different processors. Under these circumstances, the algorithm is implemented on top of an architecture based on the *Worker Design Pattern*.

The are three types of entities:

- The **dispatcher** retrieves the information from the position log database, keep tracking the work being done and it is responsible for managing the program flow.
- The **logger** is responsible for logging what jobs are already done and what is the progress of the processing.
- The **worker** is the unit which receives a small subset of the data, implements the algorithm and stores the results into the Matches Database. The number of Workers should be approximately, the number of CPU cores.

The figure 5.4 presents how these entities interact, sequentially:

1. The `Dispatcher` is launched and starts to fetch data from the Position Log Database. The data is retrieved day by day and then it is filtered and divided by node identifier. Each one of this data portions, with the given identifiers, is called `Job`. As soon as a job is ready, it is inserted on the `Job Queue`. When it ends dispatching all its jobs, it inserts termination jobs in the queue, so `Workers` can turn itselves off. This work is done in the Data Fetch phase of the architecture.

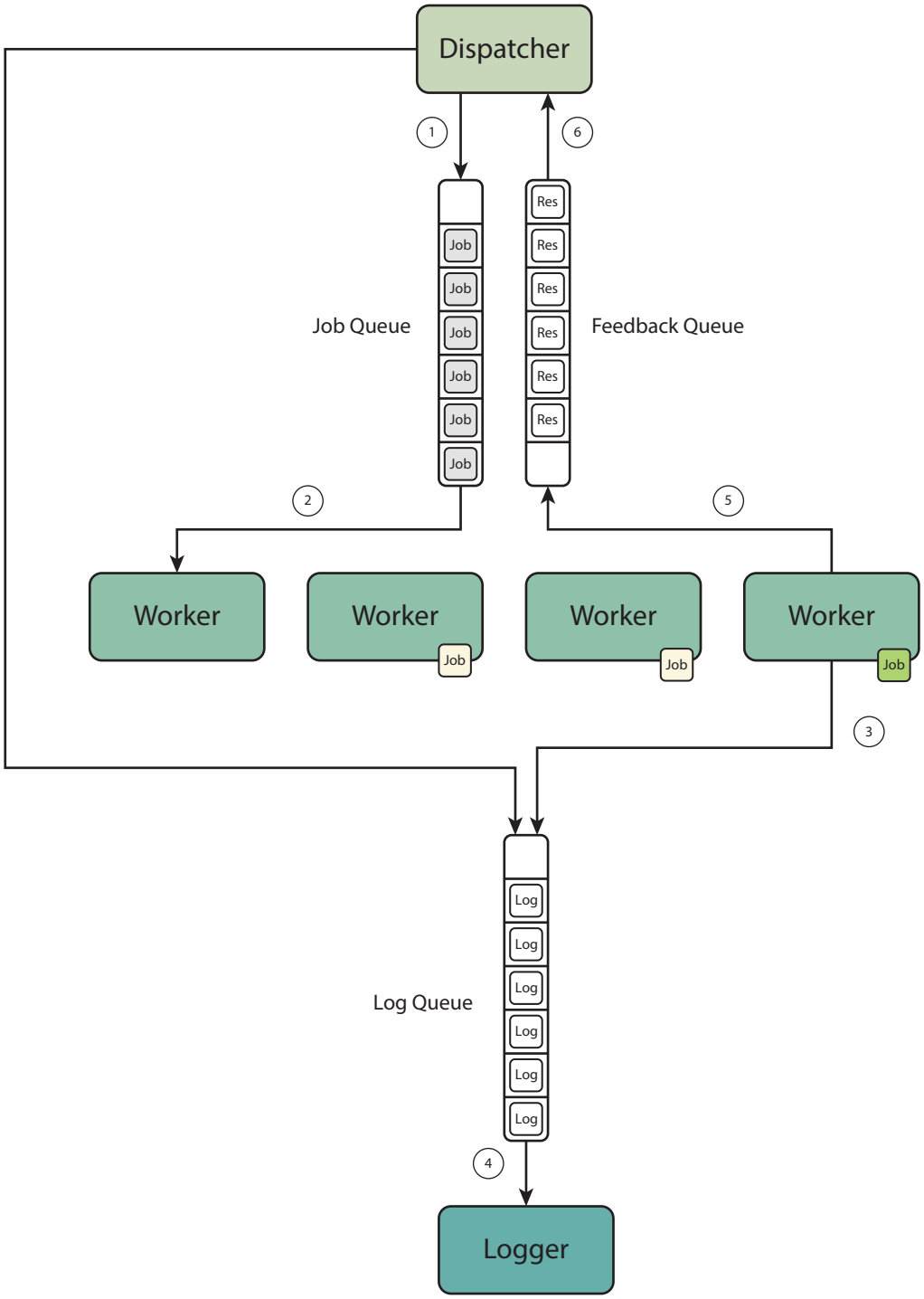


Figure 5.4: Worker Design Pattern Design

2. When a `Worker` is launched or ends processing a job, it tries to retrieve a Job from the queue. If it is unable to, it blocks until one is available. Otherwise, it starts to process a job (as figured by the two middle `Workers` of the figure). The Worker implements the other three stages of the pipeline architecture.

3. After a `Worker` finishes its job, it puts a log in the log queue with the objective of informing the `Logger` that the job is completed, about the time that it took, etc.

4. The `Logger` blocks until a log message arrives in the log queue. When one arrives, it parses and writes it to disk.

5. The `Worker` also informs the dispatcher that it completed its jobs, so it can also keep tracking of the current number of completed jobs, and repeats the process described on the second step.

6. The `Dispatcher`, which dispatches jobs and keeps tracking of the process is notified, through the `Feedback Queue`, that a `Worker` completed a job or that it is completed. When all the `Workers` terminate, the Dispatcher sends a termination job to the `Logger`, through the logger queue.

This relationship between entities enables an efficient division of tasks and it is a way of exploiting processing parallelism, since there can be multiple workers processing different data in the processing pool. Otherwise, only one unit of processing would be done, at time. The purpose of using *pooling* is to limit the amount of workers that can be executing work in parallel.

5.4.3 Algorithm Design

The initial available data, described in the section 4.3.2 does not provide the identification of the bus route being done. Therefore, there is the need of matching those positions with the bus line being done by a given node.

Addressing this problem, a custom algorithm is implemented. It has, as main development vision, the simulation of the bus route. To understand the concept behind the algorithm, we will use a metaphor.

Let us consider a person called John. John is french, having strong difficulty to talk other language like English or German. He travelled to Germany and has felled asleep inside a bus. Later, he wakes up and doesn't remember anything, even, the bus he decided to catch. No person is in the bus, only the driver and he is not able to communicate with him. John finds a flyer in a seat, with the location of the bus stops and the lines. Then he tries to identify the bus destination.

The first thing John can do is to memorize all the bus stops in which the bus is stopping (thankfully marked in the bus interior display). Then, given the stops and their relative order, he can check the flyer for lines which meet those characteristics. If he finds a line that meets them, his problem is solved: he knows the bus destination.

Having this metaphor in mind, it is easy to transport it to this real world use-case. First, the log database contains the position of all the buses. We can retrieve the GPS trace of the bus and process it. Then, iterating all *(latitude, longitude)* pairs one can find the lines which start between the geographical space. This requires making proximity queries for finding which bus stops are nearby, and then, selecting which bus stop is the first stop of a line.

Then, using a strategy that borrows foundation from the exhaustive and greedy approaches, an algorithm will try and test each of the starting lines that were detected and, iteratively, it tries to reach for a solution meeting a criteria: completeness rate, detected stops, detection order, line conclusion.

In the following subsections, we explain in more detail the set of smaller problems that need to be solved in order to make the algorithm effective, like:

- How it is possible to make spatial searches.
- How it is possible to make more specific spatial searches, using a capture radius of fixed or dynamic length.
- How can starts of a line can be detected and why finding them is a good heuristic for minimizing the number of candidate solutions.
- How it is possible to find a solution, using proximity queries and matching bus stop detection occurrences.

Making spatial searches

Finding the nearest bus stops to a given GPS position requires some attention. There are some questions that need to be answered. The first is to find which bus top is near of a given point in the geographical space.

Spatial data structures enable a kind of sorting given the spatial attributes of their objects – like latitude and longitude.

Our fixed data set includes the bus stops and the lines. Bus stops have various properties, including latitude and longitude. They will be subject of search and thus, inserting them into a spatial data structure would be a good idea, because spatial data structures are able to sort its objects relatively to a position or by the relation in terms of distance between them.

R-tree is a notable example of these type of data structures. It is ideal for indexing the spatial objects of this problems – namely, bus stops. R-tree helps out finding the *K-nearest neighbours* spatially but it does not cover the problem of finding the *K-nearest neighbours inside a radius of X meters*. This must be calculated aside: first by using the proximity (using R-tree) and then, by filtering points which are above a given radius. Two questions remain: how to calculate the distance between two points and which radius should be used.

Using and choosing a detection radius

As seen on the chapter 2, the Global Position System uses the Global Position System 1984 which views the earth as an ellipsoid.

Regarding this, measuring the distance between two geographical points can not be achieved in the same way as measuring the distance between two points in a Cartesian plane. The Cartesian plane is flat, in contrast to the ellipsoid plane. Thus, ignoring this fact leads to increase the error in the calculation of those distances; this aspect should not be put aside.

Several approximations for calculating the distante between two *(latitude, logitude)* points have been done, like *Haversine* or *Vincenty's* formulas. Given that it is needed to minimize the error, using these formulas to compute the distance between two geographical points must be done, either by implementing these formulas, either by using third-party libraries.

The behaviour of smaller and larger radius is known. If the radius is very small, bus stops are likely to be defectively detected, either by GPS jitter, either by the speed between two captures.

In contrast, if the radius is too big, there will be a huge amount of bus stops, and thereby, more candidate lines and, consequently, more processing.

Figure 5.5: A non-ideal and an ideal radius

The figure 5.5 illustrates what can happen when the radius is smaller than the ideal. The red markers represent bus stops, while the blue dots represent the GPS logs from one bus, which is moving down the road (as the grey arrow indicates).

Because of the speed that the bus travel and given the periodicity of insertion between two consecutive GPS positions, the distance between two captures can be greater than twice of the radius, and so, such radius would be insufficient to detect stops between those two points. The figure on the right depicts what happens when the radius is optimally placed and thus, all the bus stops are successfully detected.

The approach for solving this problem is choosing the radius which is equivalent to half of the maximum distance between two points when the bus is driving at the maximum speed. In Portugal the maximum speed limit inside municipalities is 50 kilometers per hour. If we consider punctual events resulting from the driver behavior, like speeding, a margin of 10% could be added and so, 55 kilometers per hour is a very reasonable max value for the maximum speed, 15.2 meters per second.

Regarding the capture's periodicity being from 15 to 15 seconds, the maximum travelled distance translates, theoretically, to 15 $seconds \times 15.2$ $meters\ per\ second$, that is 228 meters, and therefore, resulting in a radius with length of half that distance – 114 meters.

Despite of these theoretical values, it may be reasonable to increase the radius, for compatibility with slightly higher bus speeds. If it is a priority to have a dynamic radius, which would change over time depending on the current speed of a bus, it can be implemented by analyzing the GPS log pairs for finding the speed between those two point and calculating a radius from it. It is very important to have in mind the limitation of GPS and that jitter exists, being subject to increase with trees, buildings, bridges, etc – so radius lower than 30

meters are not recommended.

Detecting line starts

After understanding that the bus stops objects can be indexed in a spatial data structure, and that, using a helper method, spatial proximity searches with a radius become possible and applying the problem heuristic becomes easier.

Detecting line starts is a heuristic divided in two steps:

- Detecting bus stops nearby while iterating over the retrieved positions log.
- For each bus stop in range, verify if it is the very first stop of a line.

If, in the second step, the detected bus stop turns out to be the first bus stop of a line, that line becomes a candidate for solution.

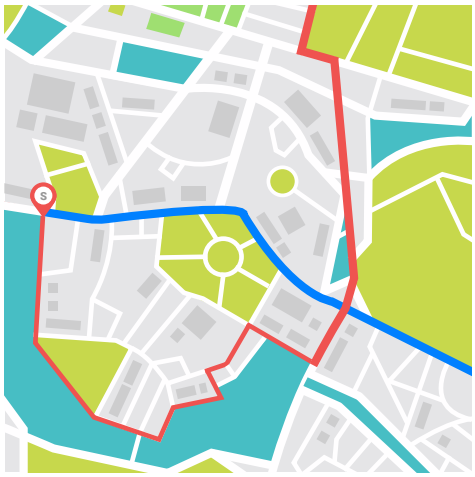


Figure 5.6: The stop S has two lines starting in its position. If the bus is passing near that position S, there is a probability that one of the lines starting there is the solution.

The main objective in detecting line starts is not to find the final and best solution, but rather to, reduce the dimension of the problem. Instead of having a total of N lines per record in the worst case, that number of lines turn to be much lower as they will be related geographically to the positions log.

Figure 5.7 presents a flowchart representing this algorithm logic.

Finding a solution

While finding line starts is an exhaustive search algorithm which reduces the problem dimension, the algorithm for finding solutions is greedy. An algorithm is said to be greedy when it *"always takes the best immediate, or local, solution while finding an answer."*[9]

For finding a solution, this algorithm iterates over the detected line starts and, for each one, it evaluates whether or not it is the best immediate solution.

Evaluating a candidate line as solution implies iterating the positions log with the hope of finding enough evidence that such candidate is the solution. This iteration can be seen as kind of simulation where a context is built, iteration over iteration. That context collects all

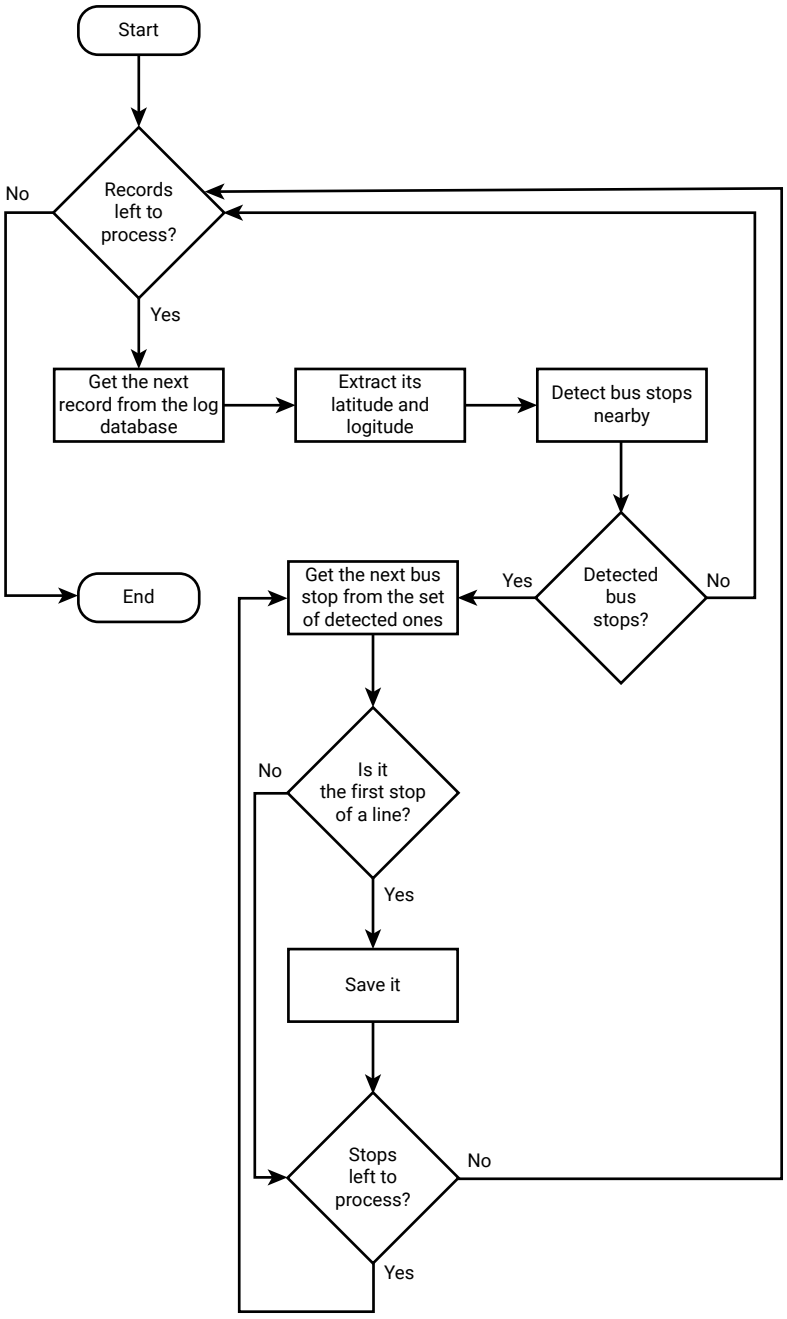


Figure 5.7: Detect line starts algorithm

the detected bus stops of the candidate line and the detection timestamp. This simulation ends when no more records are left for processing.

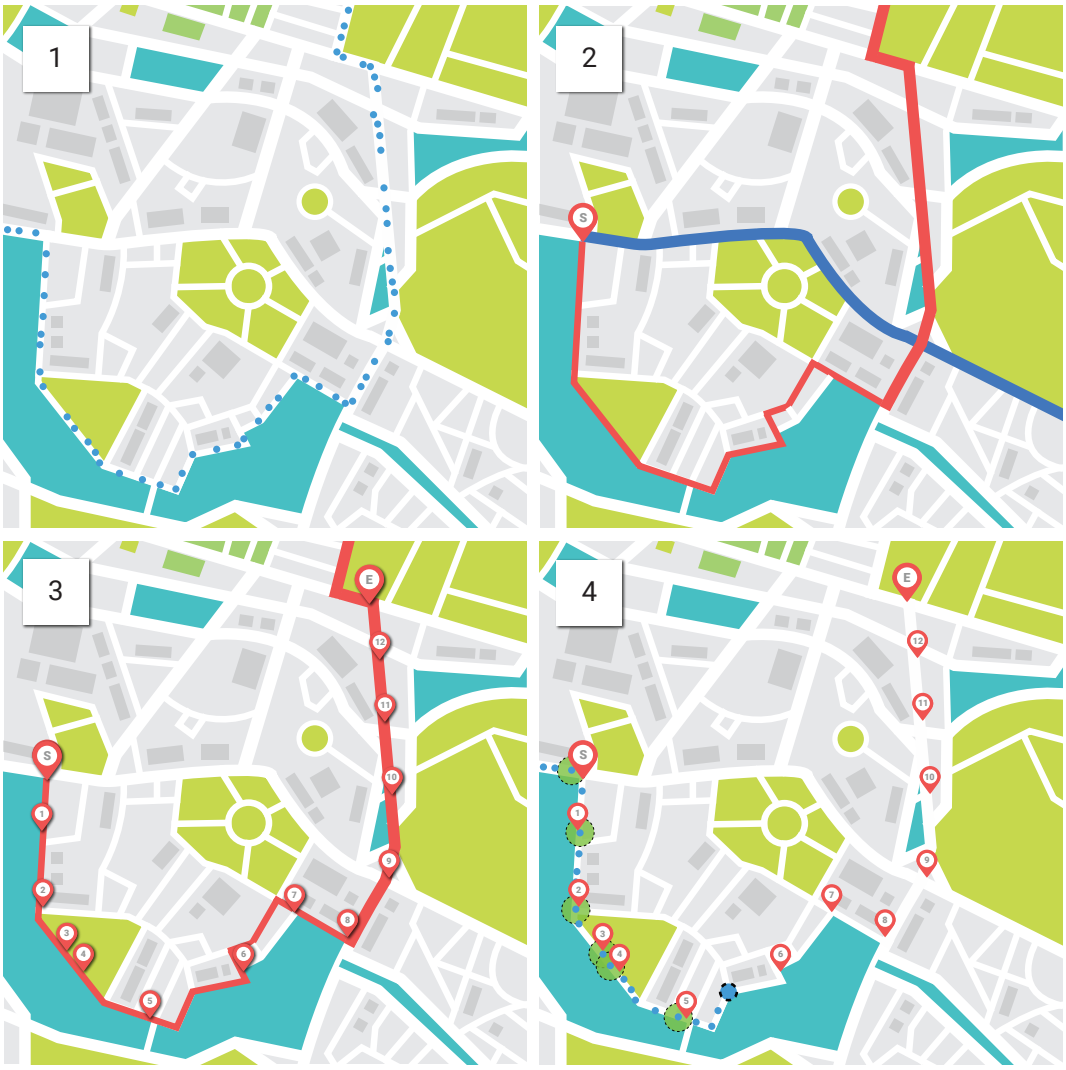


Figure 5.8: Algorithm phases

The figure 5.8 shows the execution steps of this algorithm. First, it is shown the position log, being the blue dots the GPS positions. Then it proceeds to the line start detection, described in the subsection before. In this case, it detected two.

Then, it starts to test each of the line starts. The line is outlined in red, as markers do. Each of the markers illustrates a bus stop.

Finally, the fourth square shows an in-going iteration where the finding solutions algorithm tries to find a bus stop that matches the candidate line. The detection radius are marked in green, around the bus stop markers. The bigger dashed blue dot is the position of current record being iterated (with no bus stops nearby).

A candidate becomes a solution when:

- The last stop of the candidate line is detected.
- The number of detected bus stops of that line is above a certain threshold, that needs to be calculated.

These two conditions are purely probabilistic. One another consists in analyzing the bus stop order monotony. If the bus stop sequence appears to not be increasingly monotonic, or other words, it appears to be out of order, that candidate line is not a solution.

Often, the movement of a bus is not related with a line. It can happen due to a lot of reasons, including:

- Not moving at all, for an extended period of time (e.g.: when a bus is stopped in the bus station).
- Moving to the bus carrier platform (e.g.: when there is an end of shift).
- Cross the city for starting a line (e.g.: when there is a start of a shift).

These reasons are significant to increase the amount of false positives. If the candidate line that is being analyzed is not a solution and, at the same time, none of the evidenced conditions are able to declassify it as solution, the simulation can go further and naturally, waste more processing time than expected. To address this problem, a timeout can be used.

A timeout is a time that defines a deadline for something to happen. Two types of timeouts can be implemented. The first is more permissive and consists in establishing that a line should be completed within a given time. This value can be set statically as 2 hours, for example. The second and smartest consists in defining that the next suitable bus stop must be found in 20 minutes. These values are examples, and must be studied and tested for an ideal suitability.

One thing that must be noticed is that, generally, a node's GPS position log from a given day has several hours of capture and then, there is almost always more than one line to be matched. Thus, when a line is found, all the records together with starting lines between the first and the last bus stop of the current solution must be discarded for processing, and the algorithm will continue to iterate the starting lines from the index corresponding to the last bus stop from the last solution.

After processing all the data, the core algorithm may or not present results. This is as expected. First, because no *profiling* is being done to the data, which means that, before processing, no information exists about the spatial variance of it, and thus, even if a bus was stopped everyday and it was storing data into the database, there is no way of knowing it.

But, otherwise, the core algorithm produces results. They are the following:

- The line code and direction.
- The bus stops of the line which were detected and their order.
- The percentage of stops which were detected by the node in that path.

These results are enough for taking the next step – extracting performance metrics.
In figure 5.9, it is placed the flowchart of this algorithm.

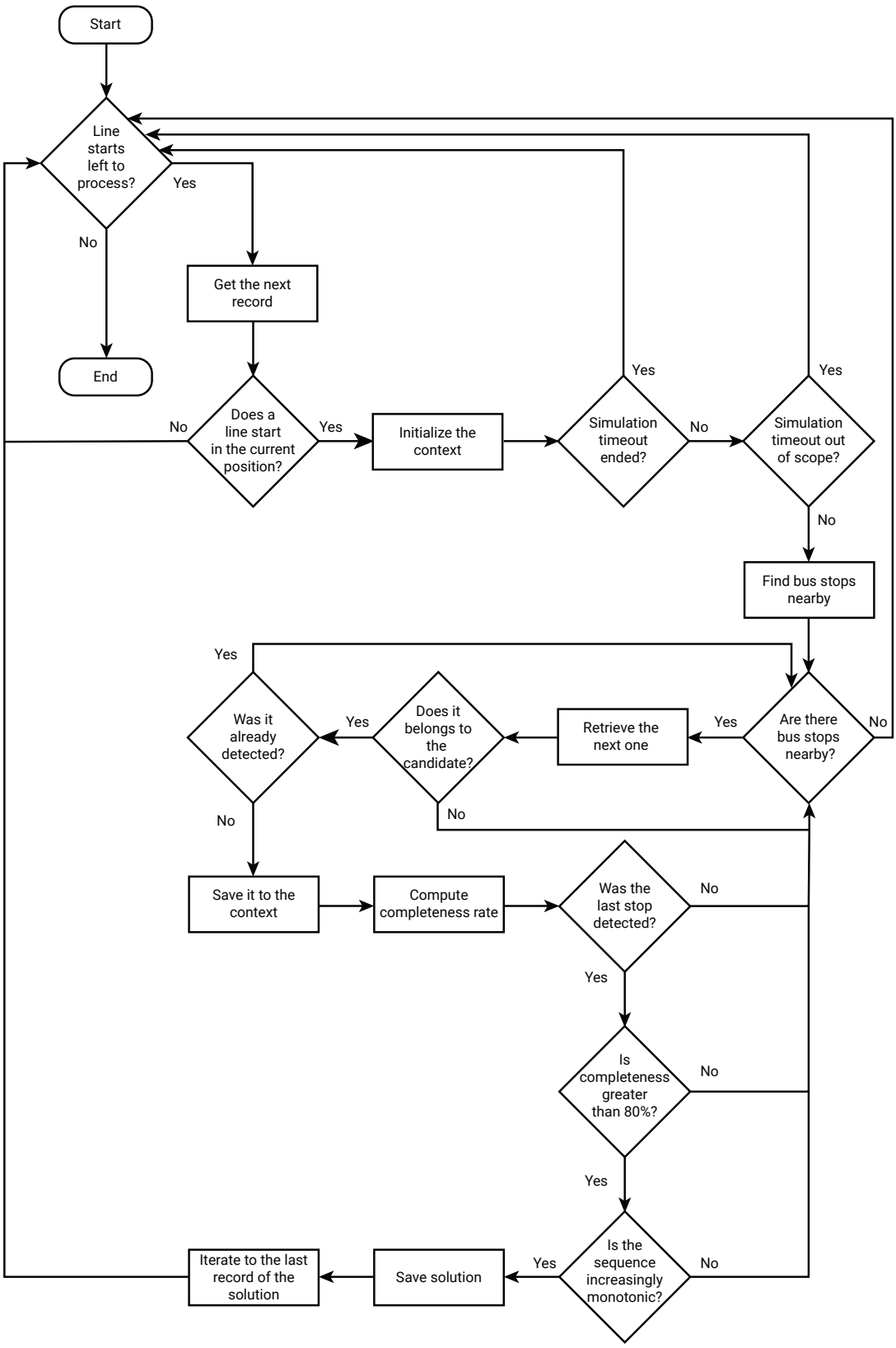


Figure 5.9: Matching Algorithm Flowchart

5.4.4 Building performance indicators

After matching GPS coordinates with the context information, it becomes possible to build statistics about the performance of the lines being matched. By performance we mean the set of the characteristics that evidence the good quality of the bus line completion. Those are clarified upon the chapter 4.5.1. The logic behind the interpretation and calculation of them is discussed below.

A deeper overview on `completeness` metric

The concept of completeness is trivial to understand:

- If a line has one-hundred percent completeness, it means that all bus stops of the line were detected.
- If a line has zero percent completeness, it means that none of the bus stops were matched.
- Otherwise, if the values are placed between those previous marks, it means that bus stops were missing.

Completeness can be seen as a mean for classifying a match as valid or invalid. Most of the times, due to line modifications and other unexpected behaviours like bad matching events, gaps between records and others, the completeness metrics might not be 100%.

One way to calculate a static but yet reasonable completeness rate is to calculate the median number of bus stops of the available data set. The median number of bus stops per line is a characteristic of the context information in use, and it depends on the bus carrier and on the city that the study is being employed.

Regarding the contextual information of this case-study, the median length of a line in terms of bus stops is 36. The chosen statistical attribute is the median as it measures the most popular value, being more accurate than the average.

Then, it is needed to understand how many suppressions may a line suffer at the same time.

Taking into account that typical service modifications impact this metric negatively, it is important to think about where they happen and look for real life evidences, such as lines subject to change temporarily to a parallel street due to road works or lines subject to have more than one line modification. It can be observed, on the STCP website, lines missing up to 6 stops due to those reasons.

It is fair to discuss this value, considering that one must ensure that it is enough to provide a good metric. The analyst can choose it based on the quality of results he wants. If the goal is to obtain a more permissive system, delivering more results but with less quality, the completeness rate should be decreased. On the opposite, if the goal is to obtain a more rigorous system that gives less but better results, the completeness rate should be increased.

Given those reasons, the adopted completeness rate is 80%, which is equivalent to not detecting 7 in 36 bus stops. Values below 80% are not valid as solution. This threshold value might and should be adjusted to the other cities bus networks.

Estimating arrival times

After the system is being capable of identifying which bus line has a bus completed, it is possible to use the data from the source database to build statistics about the bus line arrival

times. Unlike most of the literature, which makes use of times between stops to calculate time estimations, our study uses passing times, mainly because they require less computation power and because the data granularity is very low (15 seconds).

The process of estimating arrival times consists in the following steps:

1. Choosing a bus line and a bus stop.
2. Finding a reference time (for instance, a timestamp chosen by the user or a timestamp gathered from a fixed time table of the carrier).
3. Then, over all the found matches in a period of time (for instance, one month), find the matches that fit the previous captured timestamp in a given reference window (for instance, 5 minutes around the reference time).
4. Obtain those corresponding timestamps.
5. Compute the average or the median of the gathered timestamps.

The figure 5.10 highlights the concepts which serve as core for making estimations.

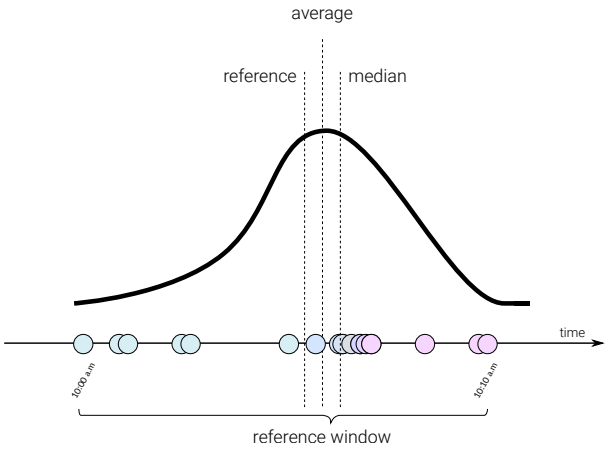


Figure 5.10: This figure presents the main components used for calculating an estimation. First a *reference* time is set and all the history points inside a limit (for instance, 5 minutes) are loaded (*reference window*). Finally, we chose to calculate the median because it is a better representative of the distribution than the average. The first and third quartiles, despite of not being represented, can be used for representing the dispersion of the times around the median.

Building time estimations is important because:

- They can be seen as a **performance metric**, expressing the **execution rate** of a bus line.
- They are, in their natural form, **an arrival time recommendation** meaning that they can be delivered to the bus passengers.

The **execution rate** is a **performance metric** related to the speed that a bus takes between bus stops. It may perform faster or slower, thus impacting directly the time of arrival

to the next target bus stop. The **estimated time**, by itself, expresses a timestamp that is computed based on experience.

For calculating both of these metrics, we may use **passing times**, as in 4.5.1. They can be easily manipulated for calculating the average, the median, and other statistical artifacts. This methodology enables the computation of an estimated time of arrival, but there are two main problems that need to be solved. The first is finding a reference, and the later is choosing a time period.

Finding a reference timestamp may be easily solved by obtaining the system time of a device, by user input or by using a static asset. For the use-case of the bus carrier, using values from a static assets (like the existing schedules) provides the best reference about the delay that may or not exist because they are the promise established between the bus carrier and its passengers. That knowledge can be used to introduce line modifications regarding consistently abnormal values.

On the other hand, the bus passenger use-case takes a better benefit in using history-based values because it removes the dependency of having such static references while providing a more and more accurate estimation about the arrival time, given the increasing convergence along the development of the time.

This convergence results from the bus lines behaviour that tends to be repeated over time, reflecting a pattern. For example, the data shows that a particular day of the week is more similar with the same day of the week, one week later than looking for the 3 last days before. This pattern impacts the schedules like a butterfly effect: a bus is as fast or as slow as the number of persons entering or exiting of the bus and also, as fast or as slow as the city traffic enables. These happenings are subject of being, also, repeated over time, for example, on the rush hours, being a reflection of the city mobility dynamics. If one collects timestamps of bus stop detections making statistics concerning the time variation, it is possible to deliver estimated times of arrival that, together, build a *smart schedule*: a schedule that changes over time and is able to adapt itself to punctual situations (like road works, city events, etc).

Regarding those reasons, it is fair to admit that a bus line has a known behaviour that becomes more and more evident over time, and that behavior can be expressed using statistics. For instance, the observation of the percentiles respecting the captured passing times is a way of obtaining recommendation values. The median is represented by the maximum value of the 50% percentile and corresponds to the most popular time value, while values lower than the 25% and higher than the 75% percentile are the less popular and are said to represent abnormal discrepancies. These values can be delivered as recommendation for the users.

Predicting arrival times

While the arrival time estimation makes use of the past history for providing a time estimate, the arrival time prediction goes on the opposite direction by predicting the future, by using old data for building a mathematical prediction model.

The essential steps for making arrival time predictions are:

1. Choose a bus line.

2. Find a reference (for instance, a timestamp chosen by the user or a timestamp gathered from a fixed time table of the carrier).

3. Then, over all the found matches in a period of time (for instance, one month), find the matches containing that reference time for that bus line.

4. Apply a machine learning workflow to the found matches, using a regression technique.

5. Deploy the machine learning model.

The gathered matches should be applied in a patterned way (choosing, for instance, the last 5 wednesdays), like explained in 5.4.4.

Then, the machine learning workflow requires a different data shape (it requires all the bus stops matches instead of all the matches regarding a bus stop). The employed machine learning techinique must be a regression because:

- It is a supervised learning technique (history target values are fed into the machine learning algorithm).
- The goal is to predict a continuous value (and not to classify a data object in a group).

Finally, after training, testing and validating, the machine learning model should be deployed for better performance (if possible).

5.5 Summary

First, the functional and non-functional requirements are presented. They define some ground rules, making distinction between an acceptable and non acceptable system behaviour.

Then, the architecture is presented, showing not only the components but also the data flow through all the components.

Under the Technical Design, it is done a detailed overview about the Matching Unit and their key elements. Its processing pipeline and its parallel computing solution helps bringing the best performance out of the computational system. Also, the main ideas that leaded to the development of the matching algorithm are explained along with its heuristics.

Finally, the foundation for building performance indicators is explained, presenting the fundamental concerns and methodologies.

Chapter 6

System Implementation

6.1 Introduction

Once the problem is known and a strategy as been defined for solving it, the implementation proceeds. The objective of this chapter is to present the practical methodologies which are crucial for the implementation of the architecture defined previously on this work. The chapter outline is the following:

- The section **Bus Network Information Retrieval** present methodologies for gathering bus network information from the bus carrier website.
- The section **Matching Unit** describes the development of the Matching Unit, describing its development history, the implementation of its modules, the algorithms implementation, and more.
- The section **Matches Database** describes the Matches Database, which is the data output point for the Matching algorithm.
- The section **Estimation Database** describes the database supporting the estimation calculation, which is the location on the architecture responsible for delivering the delay metrics.
- The section **Synchronization Script** presents the implementation details about the synchronization script, which is responsible for synchronizing the Matches Database with the Estimation Database.
- The section **Prediction Module** presents the used workflow for implementing the arrival times prediction.
- The section **Database Wrappers Module** describes the Database Wrappers Module, which is a high-level programmatic interface written in python for the applications interact with the databases.
- The section **Integration APIs** presents, briefly, the integration APIs, which serve as bridge between the database and the applications.
- Finally, the **Applications** presents the applications which make use of the information generated by the deployed system.

6.2 Bus Network Information Retrieval

Bus Network Information is the information that builds a foundation for understanding and processing the positions log data. This data can be obtained from the STCP public website, using data extraction routines.

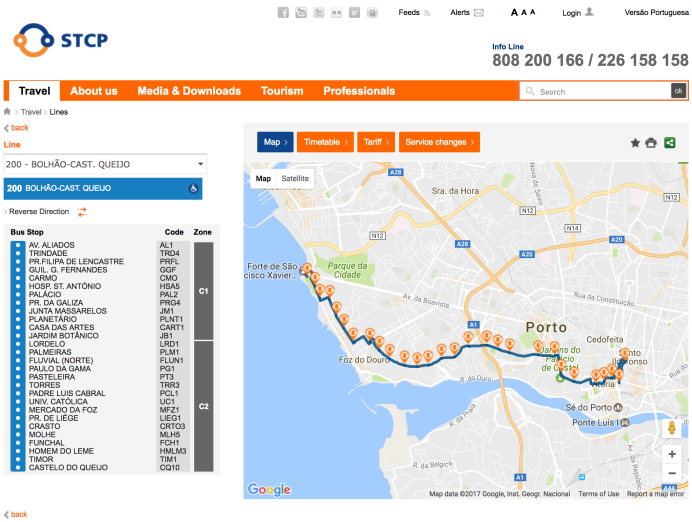


Figure 6.1: Line Details Page from STCP Website

The page issued on the figure 6.1 presents information in wide variety of forms: lists, tables, maps, etc.

There are two pages that worth being analyzed:

- The lines page[1], which contains a dropdown list of bus lines and after selecting one line, it shows its bus stops and a map representation of the line and stops.
- The journey planner, called *itenerarium.net*[2], contains information about bus stops, lines, route and journey plans.

To extract the information about the bus lines and bus stops from the STCP public website, we need to inspect its implementation, either by looking for its source code, either by using network and packet analysis tools.

For monitoring network requests, we can use one of the most popular web browsers, such as `Google Chrome` or `Mozilla Firefox`, which are bundled with network analyzing tools on their inspector. For making packet inspection, `Wireshark` is an excelent opensource tool. Despite of a web browser being capable of monitoring network requests, *Wireshark* is used because it provides a cleaner way for filtering and studying the requests that are being done.

With further investigation, it is easy to understand that this public web site relies on APIs, primarily because of clues left on the JavaScript source code and then, due to client side calls which can be detected using the packet inspection tool. The packet inspection helps understanding the existing web methods and their responses, exposing the URLs and their input attributes.

[1]STCP Lines Page (http://www.stcp.pt/en/travel/lines/)
[2]Iternerarium (http://www.stcp.pt/en/itinerarium/)

For obtaining these URLs, one must:

- Open a web browser and navigate to one of the pages that worth being analyzed.
- Open Wireshark and then, start capturing, with the following filters: `ip.addr == XXX.XXX.XXX.XXX && http.request.method == "GET"`. `XXX.XXX.XXX.XXX` to the address of the server hosting STCP website and `http.request.method == GET` limits the capture for matching only HTTP GET request methods.
- Finally, it is required to analyze the captures with the objective of finding possible endpoints.

After following these guidelines, one should be able to observe the APIs endpoint URLs, as seen below.

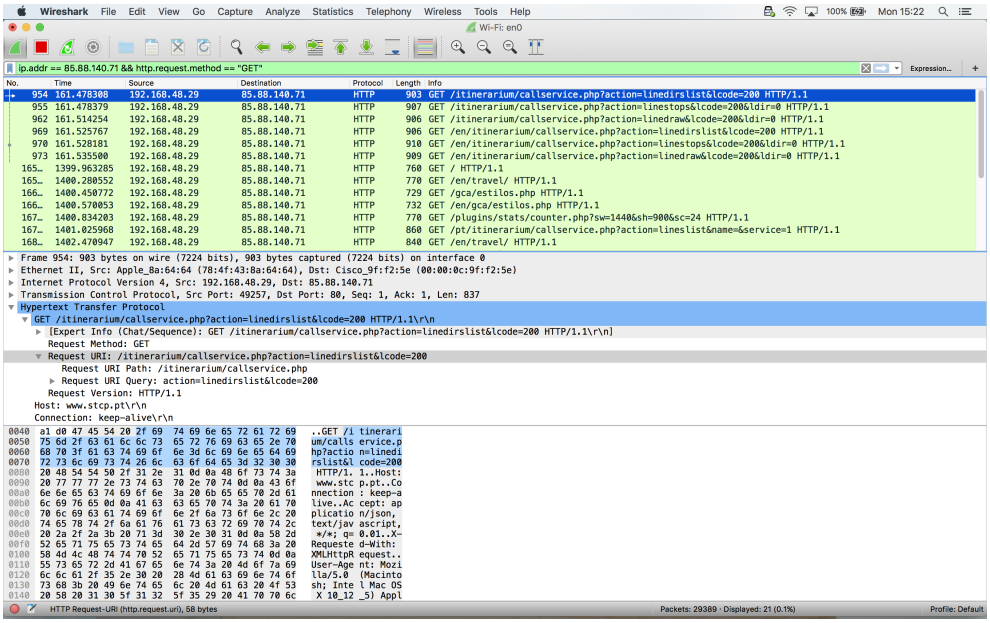


Figure 6.2: Capture on Wireshark showing the filter and the HTTP GET Requests to the STCP API

Analyzing them, we can further conclude that the URL `/itinerarium/callservice.php` acts like a router to several API endpoints. These endpoints are:

- `linedirslist`, delivers the list of bus line directions, with the attributes `name` and `code`, for a given line code and direction.
- `linestops`, contains the list of bus stops of a line, with their attributes: `zone`, `code`, `name`, `address` and `sequence`.
- `linedraw`, provides the route and the list of bus stops, with their spatial attributes, namely the `LineString or MultiLineString` and the set of `Point` geometries. This endpoint was not available at the time of the development of the algorithm.

- `linelist`, provides the list of lines with their attributes: `accessability`, `code` and `description`.

The information available on those APIs should be fetched in a regular basis and cached locally, because it will be needed later. This process is a performance and secure wise choice because the program will not have to rely on the network and on an external service to make its job.

The bus network data retrieving scripts are written in `Python`. They extract, transform and store the data into files that can be used in other programs. The information is stored in a portable and structured way, using the JSON (and GeoJSON[3]) format.

Regarding this, two scripts are implemented:

- `retrieve_base_data_from_rest.py` generates six JSON files: `lines.json` (bus lines), `stops.json` (bus stops), `line_stops.json` (stops of a line), `stop_lines.json` (lines of a stop), and `stop_coordinates.json` (bus stops coordinates).
- `transform_base_data_to_geojson.py` converts the extracted data into *GeoJSON*.

These files can be shared, later, among other applications with the Matching Unit itself. The GeoJSON version is very compliant with most of the spatial software like `ogr2ogr` and `QGIS`. It can be converted to most of the formats and visualized directly, including, in JavaScript web applications[4].

The following subsections explain how are these scripts implemented and what libraries are used.

6.2.1 Script to retrieve base data implementation details

This script generates the six files previously described. To achieve that, it is divided in three main parts:

- Initialization, which consists in defining the endpoints URLs.
- Procedure body, is composed by multiple but very similar procedures, responsible for retrieving different subsets of the data.
- Script main procedure, which arranges the call for the procedures and writes the result to storage.

The main procedures which retrieve the information from STCP website, are the following:

- `get_line_codes` retrieves the bus line codes.
- `get_lines` retrieves the bus lines and their properties.
- `get_stops_codes` retrieves the bus stop codes.
- `get_stops` retrieves the bus stops and their properties (except GPS coordinates).
- `get_stops_coordinates` retrieves the bus stop GPS coordinates.

[3]RFC 7946 - The GeoJSON Format (https://tools.ietf.org/html/rfc7946)
[4]Using GeoJSON with LeafletJS (http://leafletjs.com/examples/geojson/)

- `get_stops_of_line` retrieves the list of stops which belongs to a given line.
- `get_lines_of_stop` retrieves the list of lines from which a stop belongs.
- `get_line_geometries` retrieves the full path of a line.

These procedures are very similar on the implementation:

- They start by generating a HTTP GET request targeting the REST API, using the method `Request.get` from the library `requests`[5].
- The information is handled, and all that information that is repeated or not meaningful is removed.
- Finally, they read the result from the previous request in the JSON format. This is possible using the `.json()` method from the `Request.response` object which is returned by the GET request. The JSON map is returned for being used in the main procedure.

All this procedures return in-memory objects which are handled in the main procedure. Thanks to the helper function `write_map_to_file`, the objects are written into files, for later use in the `Matching Unit` and for being converted to `GeoJSON`, as described in the next subsection.

6.2.2 Script to transform base data implementation details

This script transforms the data stored by the script `retrieve_base_data_from_rest.py` and converts it to GeoJSON. Two `Feature Collection` objects are built and stored in two different files: `lines.geojson` and `stops.geojson`. Those files provide less knowledge than the original ones but they are more portable and compatible.

This script is composed by two procedures, `get_stops_geojson` and `get_lines_geojson`, each one with responsibility of generating a map object compliant with GeoJSON.

This program shares the same structure as the previous script. On the procedure body, its two procedures are implemented. They are, also, very similar.

First, it is created a placeholder object which will store all the GeoJSON `Feature` objects. This is the upper-level dictionary with `type` attribute `FeatureCollection`.

Then, it is created a string template of a GeoJSON feature. This is called a template because it is a string object which is later converted into a map. Having placeholders which are replaced with its own types, it provides a cleaner way of building such structure.

After having a placeholder and a template for a feature, the context information is loaded from the files. Depending in the procedure, it can load the lines, the stops coordinates, the lines stops and the lines route, etc.

The object set is iterated, and a string-like GeoJSON `Feature` objects are built from the template. Using Python's `ast`[6], this string holding a map structure is converted to a Python dictionary.

[5]Requests: HTTP for Humans (http://docs.python-requests.org/)

[6]Python's Abstract Syntax Trees Module Documentation (`https://docs.python.org/3.5/library/ast.html`)

Finally, the placeholder, holding a list of GeoJSON Feature objects is stored into a file, thanks to the same function `write_map_to_file`, for storing the `FeatureCollection` objects to disk.

For validating the generated files as GeoJSON compliant, one should use a validation program like `geojson-validation`[7], which is available through NPM[8].

6.3 Matching Unit

6.3.1 Development History

The development of this unit considered three approaches, that evolved during time:

1. A `non-parallel` version.
2. A `thread-level parallel` version.
3. A `process-level parallel` version.

The `non-parallel` version is still being used for testing the algorithm and the pipeline procedures. It does not implement any type of parallelism and does not consider the architecture previously defined for the Matching Unit. Regarding this, it is not suitable for production, either because it would be very slow.

The `thread-level parallel` version implements the previously described architecture strictly. It makes use of `Pykka`[9], which implements the Actor Model in the Python Language. `Pykka` was chosen because it seemed to be a well designed implementation of the actor model, which is a vision that abstracts from the low-level thread primitives, serving *"as mental scaffolding for building the software implementation"*[2].

For implementing parallelism, there are two Python modules that can be used. One is `threading`, which implements thread-level parallelism and `multiprocessing`, which implements process-based parallelism. Regarding the development progress, it was noticed that `Pykka` default implementation makes use of the Python `threading` module too or alternative `gevent`, which does not make use of multi-core processing. They are, in fact, concurrent solutions rather than parallel (only using a single processor core rather than using multiple cores in parallel), not being the best approaches to this problem. Having in consideration that the implemented algorithm is computational intensive, a parallel solution should have been implemented, regarding the objective of making use of the different CPU cores for processing more data per time unit. Despite of `Pykka` being an interesting solution, another had to be re-thinked for using multiple CPU-cores.

The `process-level parallel` version implements a variation of the architecture, based on the Python standard library `multiprocessing`. This implementation takes the best out of the computer performance because it makes use of the multiple processor cores. As consequence, it is significantly faster than the previous solutions.

The next subsections explain how is the Matching Unit implemented on its third and final version.

[7] geojson-validation (`https://www.npmjs.com/package/geojson-validation`)
[8] Node Package Manager (`https://www.npmjs.com/`)
[9] Pykka, *A Python implementation of the actor model* (`https://www.pykka.org/en/latest/`)

6.3.2 Overview

The use of the `multiprocessing` module required a slight modification to the base architecture. The major modifications are the following:

- The `Dispatcher` no longer launches *N workers* given *N jobs*. The number of `Workers` is equal to the number of CPU cores. This value can be obtained using the method `cpu_count` from Python `os` standard library.
- The entities are launched by the main program, one followed by the others. The first entity to be launched is the `Logger`, followed by the `Workers` then followed by the `Dispatcher`. This also means that the `Dispatcher` no longer launches the other entities.
- The communication between entities is done using synchronized queues, implemented by the Python standard library. These queues act like synchronization devices because entities life-cycle is driven through the exchange of specific objects (messages).
- Job units are queued by the `Dispatcher` and then consumed by one of the `Workers`, as soon as one of them becomes available.

They exchange messages between them, recurring to queues. Those messages are from 4 types:

- `info` messages are sent for informing generic stuff, like state changes.
- `command` messages are sent for sending commands, for instance, telling an entity to shutdown itself.
- `value` messages are sent for sending values regarding the program behavior.
- `log` messages are log specific messages.

Each one of the messages are sent in the JSON format, using standard Python dictionaries, and have their format.

The figure 6.3 shows the main Python modules which implement the problem solution.

6.3.3 Modules

The matching unit consists in three main modules:

- **entities** is a module implementing the entities `Dispatcher`, `Worker`, and `Logger`.
- **pipeline** is a module implementing a set of procedures that are used by the entities, along the process.
- **utilities** is a module which gathers the database wrappers, tool for querying data spatially and utilities for reading files in different formats, for example.

Regarding this, these main modules are explained in further detail in subsections to follow.

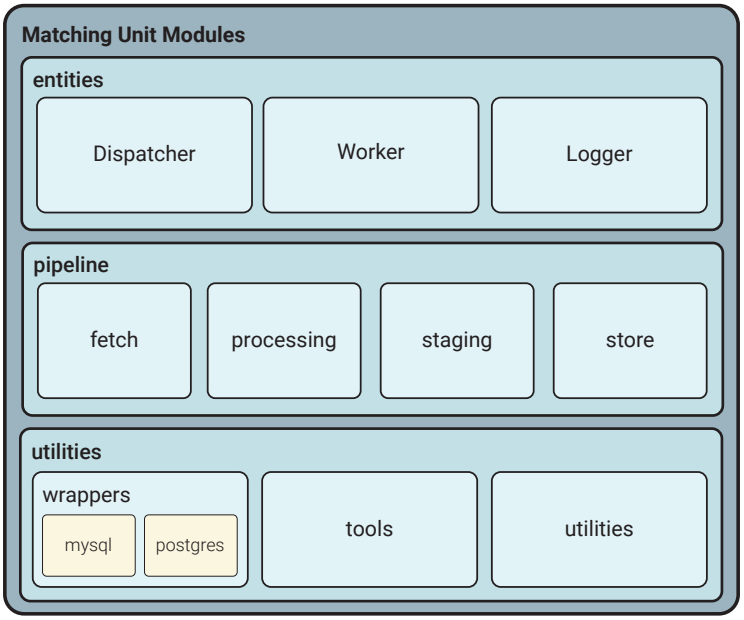


Figure 6.3: Matching Unit Modules

6.3.4 The `entities` module

This module implements `Dispatcher` and `Worker`, which extend `Process` from the module `multiprocessing`. They communicate between each other using two `Queue` objects, from the same module, which implements a synchronized queue.

Their life-cycle is described in the following subsections.

`Worker` implementation overview

`Workers` are the first entities to be launched. First, they are initialized with the following attributes:

- `tool` is `QueryTool` object.
- `output_db` is a database wrapper for storing the computation value to a database.
- `job_queue` is a synchronized queue from which the entity retrieves jobs.
- `log_queue` is a synchronized queue for notifying the `Dispatcher` that a given job is completed.

After that, upon the call of the inherited `start` method, it reaches the method `run`, which implements its life-cycle, consisting in a loop composed by four different moments:

1. First, it calls `task_queue.get()` for retrieving a job. If no job is present in the queue, it blocks until one appear.
2. If the retrieved job contains the flag `no_more_jobs`, the worker ends its life-cycle. Otherwise, it continues to the next step.

3. The job is processed and its results are stored, implementing the `processing` and `store` phases of the pipeline.

4. Finally, the log is written to disk, identifying what was the processing start time, end time, its `node_id` and the processed data date

`Dispatcher` implementation overview

This entity, based on a set of dates, creates job units: collections of different data which is processed by the workers.

Its life-cycle is divided in four phases:

- First, given a set of dates, it creates a set o job units. A job unit consists in a map-like structure holding the identification of the node from which the data belongs, the date of the capture and the records, regarding the previous attributes.
- After that, job terminators are inserted into the queue, so, when the jobs end, the Worker knows it needs to shutdown itself.
- Then, it tells to the logger that it has asserted the total number of jobs, sending a message from the type `value`. This happens because the data fetch from the log position database happens day-by-day, and so, the job units starts to increase until the fetch ends. For logging, the best way is to mark it down as soon as the information is available.
- Finally, it waits for the `Workers` to die.

`Logger` implementation overview

Despite of its importance, this is the simpler entity. It depends only on two attributes: the `log_queue` which is a unidirectional point of log information retrieval and the flag `verbose`, which is set to `True` by default and defines whether or not it prints the logs to the console.

Its life-cycle considers only three moments:

- The first consists in starting its context and creating a file.
- Then, it loops, waiting for a message to arrive in the queue. When a log message arrives, it writes it to a file and prints it to the console, only if, the verbose option is set to be `True`.
- Finally, after receiving a `command` type message from the `Dispatcher`, it closes its log file and ends its activity.

Otherwise, if no message is in the queue, it blocks and waits until another message comes.

6.3.5 The `pipeline` module

This module is composed by another four sub-modules which implement the data processing pipeline described on the subsection 5.4.1. They are named similarly: `data_fetch`, `data_processing`, `data_staging` and `data_store`.

The sub-module `data_fetch` contains procedures for loading the bus network data. They are used, in particular, by the `Dispatcher`.

Then, each `Worker` entity implements the remaining pipeline modules. The module `data_processing` is the most important one, because it implements the two procedures which together are capable of matching the data: `detect_line_starts` and `find_solutions`. The first implements the heuristic for reducing the number of solutions, as described in the section 5.4.3, while the other implements a greedy algorithm for finding which one of the starting lines are really a solution, as explained in the subsection 5.4.3.

The later `data_staging` and `data_store` contain, respectively, procedures for preparing the information to be stored into the database.

Regarding this, only the two procedures of `data_processing` module are explained in further detail, since they implement the solution for the matching problem.

Algorithm to detect when a line starts

The implementation of the algorithm developed to detect when a bus line start, follows the present direction:

1. Taking the records and local query tool as argument, a variable is initialized as an empty list, called `start_profile`. This list will hold all the lines codes of the detected lines from the GPS trace.

2. Then, all the records are iterated: first the position is extracted from the record and then, using the local query tool, the program detects all the line stops in the radius.

3. Finally, for each stop, is retrieved a set of lines for starting with it. Those lines are stored into the list, and then processed as candidates on the next step.

The following algorithm presents the pseudo-code for this solution.

Algorithm 1: Detect Line Starts Algorithm

Input: QueryTool: an object for making spatial queries
Input: Records: a set of records from a given node, in a given day
Output: *start_profile*: a list of the line start detections for each one of the records

```
1 start_profile ← ∅
2 foreach record ∈ Records do
3 |   record_position ← record.extractLatitudeLongitude()
4 |   nearest_stops ← QueryTool.getNearestBusStops(record_position)
5 |   lines_of_the_record ← ∅
6 |   foreach bus_stop ∈ nearest_stops do
7 |   |   line_of_bus_stop ← QueryTool.getLinesStartingWith(bus_stop)
8 |   |   lines_of_the_record.append(lines_of_bus_stop)
9 |   start_profile.append(lines_of_the_record)
```

Algorithm for finding solutions from line starts

This procedure implements a greedy strategy for finding a solution from a set of candidate ones, following strictly the design described on the section 5.4.3. Regarding the pseudo-code present on the Algorithm 2, this section describes its operation.

As input, it take four elements: a QueryTool object for making spatial queries, the position log database records to be matched, the set of starting lines for each one of the records, and finally, the base TTL, later explained.

The algorithm starts by creating an iteration variable i and initializing it to zero. Then, it initializes `solutions`, which is a list data structure for holding a set of solution objects (lines 1 and 2).

Iterating all over the records, the algorithm starts by verifying if there are or not a set of starting lines for the currently iterated record. If there are no starting lines, the algorithm proceeds to the next record. Otherwise, it continues (line 3).

Starting Lines is a list of map like-structures. Inside each one of these structures, there is also a list of lines starting in the position of the present record. The next step, is to verify if one of them is a solution. Regarding this, a simulation is done for each of the starting lines (line 5).

Before starting the simulation, a context must be built. The variable j tracks the current progress of the simulation. The stops of the the current line are loaded and the remaining variables are initialized. `matches` consists in a list of bus stops which will be matched; `indexes` is a variable for saving the time of that match, in terms of its record index. Then, `solution_found` controls the simulation, by interrupting the simulation cycle when a solution is found. Finally, the `target_order` holds the value of the order of the next bus stop that the algorithm will try to detect (line 6 to 14). The initial target order is one, regarding that for solving the problem of the detection of the circular lines (lines starting and ending in the same place), we transform the circular lines in normal lines by ignoring the first bus stop. This technique works because of the used heuristic, which detects line starts (not losing, therefore, the first bus stop detection).

The implemented timeout strategy consists in setting a metric called `time-to-live` (or simply TTL), as described in the section 5.4.3. With this metric, the algorithm is able to give up on the current simulation, if it is unable to find the target bus stop in a certain amount of time. The default timeout is 15 minutes because, based on the experience, it is the value that delivers best results. `TTL` will decrease along the simulation. If the simulation is not being successful, the cycle will go on until it reaches its termination, trying another candidate (line 15).

Then, the simulation starts, and it needs to verify if the simulation iterator reached out the data length. If that is true, the algorithm exits (lines 16-19). `TTL` decreases.

After that, the current GPS position is extracted, and the nearest bus stops are retrieved (lines 20-21). For each of the bus stops, it is verified if they belong to the line and if they are not already matched. If they are not matched, the bus stop order is extracted and it is made a verification for asserting that the bus stop is expected in a near future. This verification prevents bad matching of circular lines (lines 20-23).

If the order of the detected bus stop fits this criteria, both `matches` and `indexes` are updated, as is the `target_order` updated too. The variable `TTL` is put to its default (lines 24-29).

After all the bus stops finish the evaluation, the completeness rate is computed. If the last stop was detected and the completeness is greater than 80%, the solution is saved into a data structure and added to the solutions. The lines 32 and 33 handle the detection of the first stop, which is the first to be detected but the last to be tracked. This is also implied as technique for detecting circular lines since, if the completeness metric is targeted and the last stop is detected, only one bus stops will be missed (the first, regarding the initial target order). The iteration variable `i` is updated, making the main control cycle ignore the analysis of all the intermediate records which are covered by the found solution (lines 32-41).

Algorithm 2: Find Solutions Algorithm

Input: QueryTool: an object for making spatial queries
Input: Records: a set of records from a given node, in a given day
Input: StartingLines: a start profile of the the records, output from the procedure implementing the heuristic `detect_line_starts`
Input: BaseTTL: the timeout for finding a next bus stop (the default is 15 minutes)
Output: *solutions*: a list of matches

```
1  i ← 0;
2  solutions ← ∅
3  while i <Records.length() do
4      if StartingLines.index(i) is not empty then
5          foreach candidate ∈ StartingLines.index(i) do
6              j = i;
7              line_stops ← QueryTool.getStopsFromLine(candidate);
8              first_stop ← line_stops.getFirstElement();
9              line_stops.setElement(0, None);
10             last_stop ← line_stops.getLastElement();
11             matches ← ∅;
12             indexes ← ∅;
13             solution_found ← False;
14             target_order ← 1;
15             ttl ← BaseTTL;
16             while ttl >= 0 do
17                 if j >= Records.length() then
18                     return solutions
19                 ttl ← ttl − 1;
20                 current_position ← Records.index(j).getLatitudeLongitude();
21                 nearest_bus_stops ← QueryTool.getNearestBusStop(current_position);
22                 foreach nearest_bus_stop ∈ nearest_bus_stops do
23                     if nearest_bus_stop ∈ line_stops and nearest_bus_stop ∉ matches then
24                         stop_order ← line_stops.index(nearest_bus_stop);
25                         if stop_order ∈ [target_order, target_order + 5] then
26                             matches.append(nearest_bus_stop);
27                             indexes.append(j);
28                             target_order ← stop_order + 1;
29                             ttl ← BaseTTL;
30                 completeness ← (matches.size() + 1)/line_stops.size();
31                 if last_stop ∈ matches and completeness >= 0.8 then
32                     matches.insert(0, first_stop);
33                     matches.insert(0, i);
34                     solution ← initializeMap();
35                     solution.map('line', candidate);
36                     solution.map('stops', matches);
37                     solution.map('indexes', indexes);
38                     solution.map('completeness', completeness);
39                     solutions.append(solution);
40                     i ← j;
41                     solution_found ← True;
42                 j ← j + 1;
43                 if solution_found is True then
44                     break;
45     i ← i + 1
```

6.3.6 The `tools` module

This tools module implements a spatial querying tool, through the object `QueryTool`. This spatial querying tool enables a wide-range of spatial and bus carrier intelligence queries, such as:

- Get the nearest bus stops from a given GPS point.
- Get all the lines from which a bus stop belongs.
- Get all the stops belonging to a bus stops.
- And more.

To implement it, two main third-party libraries are used. One is `Rtree`[10], which provides spatial indexing for Python. Its functions enable operations like the nearest neighbour search, an essential feature for this program. Also, it serves as an inexpensive spatial in-memory database. Having into account that queries are exclusively done to bus stops, those stored attributes can be bus stops or bus stop codes. Because of the inevitable overhead and easiness of operation, a full GeoJSON feature representing each one of the bus stops is stored.

Unfortunately, `Rtree` is incapable to answer for proximity queries given a radius. For implementing this feature, this software makes use of `geopy`[11], for calculating distances between two GPS points, using *Vicenty's* formula.

6.4 Matches Database

6.4.1 Overview

The Matching Unit generates new data, which reflects an association between the data which is initially available from the different data sources. This relation, which is called by *match*, is stored in a database, defined in the architecture as the Matches Database.

This database stores the output of the algorithm 2. The main algorithm output is the match, which answers to the questions "*what*", "*when*", and "*which*": "what" node produced such result, "when" it happened, and "which" line it matched. Also, the algorithm 2 stores more detailed information about that match, like "what bus stops were matched" and "how much time passed between those bus stops". This information is correlated with the contextual information, as can be seen on the following database diagram in figure 6.4 and detailed description.

Regarding this, the following database modeling schema is implemented, using the SQL database management system `PostgresSQL`. Several functions are implemented, increasing the usability and security on the interaction with this database.

One of the most important functions is `get_matches`, as it is used on the `Synchronization Script`. In the fact, that stored procedure, from itself, extracts and transforms the data in the exact format needed to be loaded into the Estimation Database.

[10] Rtree website (http://toblerity.org/rtree/)
[11] Geopy website (https://github.com/geopy/geopy)

6.4.2 Database schema description

The figure 6.4 presents the database diagram, which depicts the modeling Given the specified requirements, the following database is designed:

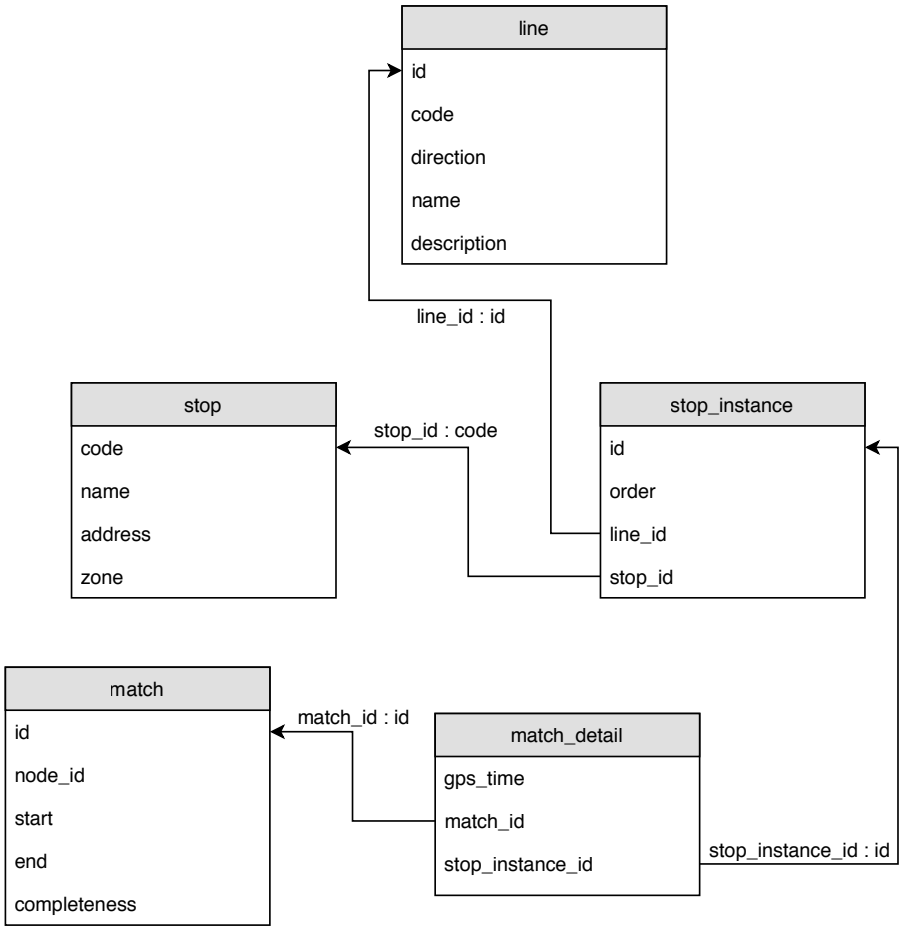


Figure 6.4: Matches Database Diagram

The table `match` stores all the found solutions. It has six attributes: `node_id` (identifies which node produced such result), `start` and `end` (which are timestamps delimiting the starting and ending time of the found match) and `completeness` (the percentage of detected stops).

The table `match_detail` stores, as the name concerns, details about a match. Its attributes are `match_id` (identifies the match), `stop_id` (identifies the `stop_instance`) and `gps_time` (a timestamp for this match occurrence).

Then, the table `stop` stores the bus stops. Bus stops are part of contextual information of the system and thus, this table is populated upon creation. A `stop` has the following attributes: a `code` (which serves as slug), a `name` (a human understandable identifier), a `address` (the address of the bus stop) and the `zone` (which relates to the carrier bus zone classification).

In a similar way, `line` stores the bus lines. Regarding the fact that line is also part of the contextual information, it is also populated upon creation. A `line` has several attributes,

namely, an `id` (unique identifier), a `code` and a `direction` (both can be a identifier too because the `code` relates to the set of stops and the `direction` refers to the iteration order).

Finally, the table `stop_instance` maps the relation between the table `stops` and the table `line`, regarding the many-to-many relationship between those two. The attribute `order` is the order of the bus stop (referred by the foreign key `stop_id`) in that line (referred by the foreign key `line_id`).

6.4.3 Routines

Regarding the interaction with this database, several routines are available. SQL Rotines are *"are routines that have logic implemented with only SQL statements"*[19]. They are used by the Database Wrapper Module, to later use on the APIs, and are named as follow:

6.4.4 Functions

- `get_line_id`
  - **Description**: Obtains the `line` object unique identifier, from the database, given the bus line code and direction.
  - **Entry Parameters**: line code, line direction.
  - **Return Value**: An unique identifier of the `line` object stored in the database.
- `get_stop_instance_id`
  - **Description**: Obtains the `stop_instance` object unique identifier from the database. , given the bus line code and direction.
  - **Entry Parameters**: stop code, line code, line direction.
  - **Return Value**: An unique identifier of the `stop_instance` object stored in the database.
- `get_match_count`
  - **Description**: Get the total amount of matches.
  - **Entry Parameters**: None.
  - **Return Value**: The amount of matches stored in database.
- `get_match_details`
  - **Description**: Gather set of matched bus stops of a given match.
  - **Entry Parameters**: Unique identifier of the `match` object
  - **Return Values**: A table with the timestamp of the matched bus stop, the bus stop code, direction, and line code and its direction.
- `insert_match`
  - **Description**: Inserts a match in the database.
  - **Entry Parameters**: Unique identifier of the node which did the matches bus line, timestamp of the first capture, timestamp of the last capture, completeness rate.

– **Return Value**: Unique identifier of the newly inserted match.

- `insert_match_detail`
  - **Description**: Inserts a matched bust stop in the database.
  - **Entry Parameters**: Timestamp of the matched bus stop, unique identifier of the match, stop code, bus line code and direction
  - **Return Value**: A unique identifier of the newly inserted match detail.

6.5 Estimation Database

6.5.1 Overview

The statistics database is one of the major components of this implementation. It provides a mean for calculating efficiently the target metrics (the estimated times of arrival and the time between bus stops).

6.5.2 Development history

The first design of the statistic database is an implementation of a data mart, in a star schema configuration [37]. In a start schema, the tables on the center are called *fact tables*, and the border tables, connected to the central ones, are called *dimension tables*. The diagram of this data mart is presented in the figure 6.5.

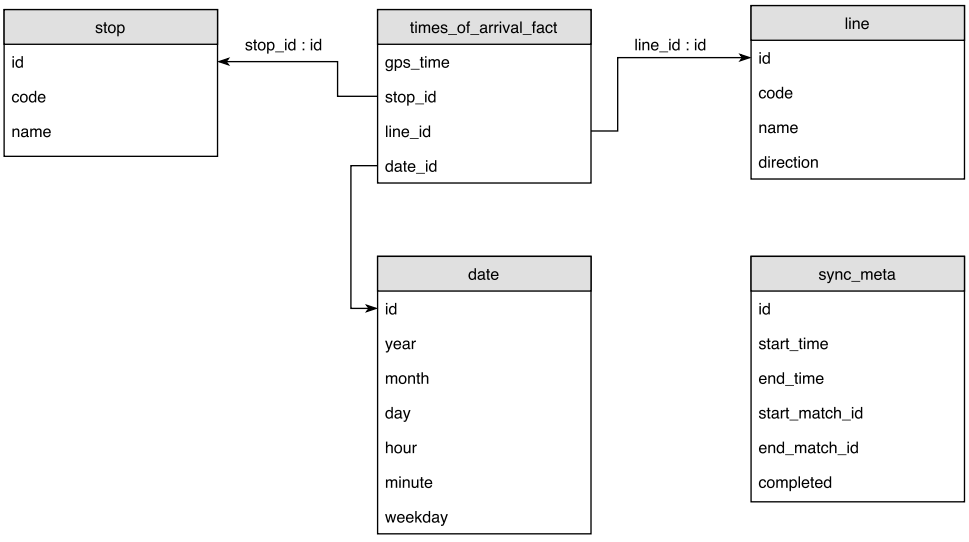


Figure 6.5: Data Mart Diagram

Later, after running the matching unit several times, the data mart started to showing some performance problems, mainly because of queries taking too much time (18 seconds). As a result, the issue was studied and several approaches have been placed.

First, the database columns data types and indexes have been verified. The indexes have been also rebuilt. The type of the attribute `gps_time` was `TIMESTAMP` and the stored procedures that were built for computing the median were not costly in terms of performance.

After refactoring them by converting the `TIMESTAMP` to `FLOAT` (relative to the start of the epoch), the footprint was reduced from 18 seconds to 1 second.

Despite of targeting 1 second of query time, it was not enough for providing querying times directly to the applications, through the APIs. For example, if a bus line has 30 bus stops, it would take 30 seconds to load the information. Thus, further improvements were required.

The first approach was to test a compatible schema using InfluxDB[12]. Influx is a time-series database, meaning that it is optimized to handle time series data. Unfortunately, InfluxDB does not provide means for measuring accurately its query performance, being that a reason to not use it as solution.

The second and final approach consists into using PostgreSQL[13], designing a database with only two relationship-less tables: one for the metrics, condensing the most important attributes of the previous dimensions existing in the data mart into one single table, and another for metadata. Multicolumn indexes are also built-in.

This final query time is between 20 and 45 ms being, by far, much more acceptable. Details regarding this later version are presented in the following subsections. The most noticeable side-effect is the fact that this schema occupies more more space in disk.

6.5.3 Database diagram

The figure 6.6 presents the estimation database diagram.

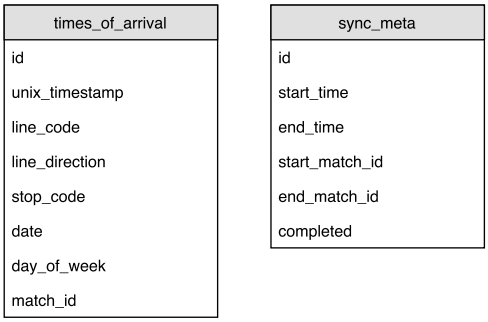


Figure 6.6: Data Mart Diagram (not definitive image)

6.5.4 Database schema description

This database schema is composed by two tables:

- `times_of_arrival` is the table where the performance metric is stored (`unix_time`).
- `sync_meta` is the table used by the synchronization script for understanding if there were or not synchronization errors.

The goal of this database is to operate `unix_time`, which is the GPS time of the match in a form of a UNIX epoch, for getting its most popular values in a set of percentiles (for instance, 25%, 50%, 75%). The other attributes (columns) are exclusively used as filters:

[12]InfluxDB (`https://www.influxdata.com/time-series-platform/influxdb/`)
[13]PostgreSQL: The world's most advanced open source database (`https://www.postgresql.org/`)

- `line_code` and `line_direction` identify the bus line of the match.
- `stop_code` identifies the bus stop of the match.
- `date` identifies the date of the match.
- `day_of_week` identifies the weekday.
- `match_id` identifies the match.

Finally, updating the Estimation Database is possible using the Synchronization Script, explained in the following section. To understand if there were or not synchronization errors, a meta-data table called `sync_meta` is included.

This table contains 6 columns:

- `id` is a unique identifier for a synchronization.
- `start_time` is the timestamp referring to the time when the synchronization started.
- `end_time` is the timestamp referring to the time when the synchronization ended.
- `start_match_id` refers to the unique identifier of the first match that was fetched under the synchronization process.
- `end_match_id` refers to the unique identifier of the last match that was fetched under the synchronization process.
- `completed` defines whether or not the last match was successfully synchronized.

More details about the synchronization process are described in the subsection 6.6.

6.5.5 Routines

Below, it is presented the functions and the triggers developed for building this system.

Functions

- `get_time_estimations`
  - **Description**: Gather the time estimations without any time constraint (*i.e.*, without distinguishing workdays from weekdays).
  - **Entry parameters**: query time, an initial and a end date for delimiting a period for calculating the estimations, a tolerance to be used with conjunction with the query time, for accepting values for the estimation and finally, the bus line code, direction and bus stop code.
  - **Return values**: The value of the percentiles, as previously cited in 6.5.4.
- `get_time_estimations_for_weekends`
  - **Description**: Gather the time estimations for timestamps in the weekend.

  - **Entry Parameters** and **Returns Values**: are the same as the ones from the function `get_time_estimations`. The difference is that time estimate computation only considers the days of the weekend.

- `get_time_estimations_for_working_days`
  - **Description**: Gather the time estimations for timestamps in the working days.
  - **Entry Parameters** and **Returns Values**: are the same as the ones from the function `get_time_estimations`. The difference is that time estimate computation only considers the work days.

- `to_epoch_dirty`
  - **Description**: This function converts a SQL timestamp to a UNIX epoch (float), in 1970. This conversion is done because it is not possible to calculate the average or the median of a UNIX epoch belonging to a different day. Therefore, this function sets a timestamp as being in `1970-01-01 HH:MM:SS`, where `HH:MM:SS` is the input timestamp.
  - **Entry Parameters**: A timestamp.
  - **Return Values**: A float representing the time in seconds, as the UNIX epoch, from 1970.

- `get_matches_corresponding`
  - **Description**: Returns a set of match detail objects corresponding to a given bus line and bus stop, in a certain day of the week.
  - **Entry parameters**: line code, direction, stop code, reference timestamp, day of week.
  - **Return Values**: a table comprising a list of date and matches corresponding to the input values.

- `get_matches_corresponding_for_weekends`
  - **Description**: Returns a set of match detail objects corresponding to a given bus line and bus stop, for the days of the weekend.
  - **Entry parameters**: line code, direction, stop code, reference timestamp
  - **Return Values**: a table comprising a list of date and matches corresponding to the input values.

- `get_matches_corresponding_for_workdays`
  - **Description**: Returns a set of match detail objects corresponding to a given bus line and bus stop, for the work days.
  - **Entry parameters**: line code, direction, stop code, reference timestamp
  - **Return Values**: a table comprising a list of date and matches corresponding to the input values.

- `insert_time`

- **Description**: Function for inserting a match detail in `times_of_arrival`.
- **Entry parameters**: time of the bus stop capture, bus line code, direction, bus stop code, date, unique identifier of the match from which the inserted match detail belongs.
- **Return Values**: an unique identifier of the newly inserted record.

Triggers

- `update_day_of_week`
  - **Description**: 'After insert' trigger which updates the column `day_of_week`, by extracting the day of week from the newly inserted timestamp.

6.5.6 How estimates are calculated

The calculation of an estimate is performed, according with the following workflow:

1. Select all of the matches.
2. Filter them using the bus stop code.
3. Filter them using the bus line code and direction.
4. Filter them by restricting them to a date interval.
5. Filter them by restricting them to a time interval.
6. Optionally filter them by restricting them to a day of week.
7. Apply the percentile calculation to the remaining dataset.

This algorithm is implemented by the database function family `get_time_estimations` and based on the workflow described in 5.4.4, which are later used for providing functionality to the dashboard and to the mobile application. The dashboard makes use of a date interval of one month and does not apply the restriction by day of week, but rather, by using working or weekend days depending on the reference date.

6.6 Synchronization Script

Given the system's architecture, it is fair to admit that the Matches Database is part of the operational system, because the Matching Unit generates data and interacts directly with the database, modifying its content, by adding more data.

The data must be loaded into the Estimation Database from other data source, potentially requiring a staging phase for filtering it and transforming it to the wanted format. In this case, the design of the two different databases is very similar, resulting from the well-defined and directed architecture. Thus, the data filtering is not needed, being the synchronization a matter of extracting the data from one side (Matches Database) and loading it in the other side (Estimation Database).

Therefore, a synchronization script is implemented, having the following flow:

- First, it analyzes the clean state of the Estimation Database, by verifying if the last synchronization was or not successful (using the meta-data table `sync_meta`). A bad synchronization consists on having more data in the estimation database than data which is already logged in the meta-data table. Therefore, it cleans all the insertions which are not tracked.

- Then, it verifies how many matches are out of sync when comparing the source database with the target, and builds a list of jobs (matches) to be uploaded.

- Finally, it starts transferring each one of the matches to the Estimation Database. More precisely, it uploads the result from a procedure called `get_matches()` which outputs a slight modification of the attributes of the Matches Database table `match_details` with the line code, line direction and stop code, rather than their identifiers. If an error occurs, the `try-except` clause makes sure that the synchronization log is written, setting the attribute `completed` to `FALSE`. It keeps track of the last match identifier, requiring that only that one must be synchronized again. If the problem source is from the outside (network, reset container, etc), there is no problem: the synchronization's first stage will erase everything which is not marked as done, and synchronize every single untracked match again. Otherwise, if everything goes right, the log is written, setting the attribute `completed` to `TRUE`.

6.7 Prediction Module

6.7.1 Overview

The goal of the prediction module is to deliver predictions of bus arrival times at each bus stop of a bus line. Thus, this module can be integrated in a service that can be accessed by bus passengers, in real-time.

Using machine learning is a natural approach for implementing this module. This approach can be classified as a supervised learning task because training values are provided (the timestamps). Since the target value is not a class but rather a continuous value (a float expressing a time in the form of UNIX epoch), the type of machine learning algorithms that need to be tested are regressions.

For developing this module, the machine learning workflow presented in 2.6.3 is applied, using the Python language and the machine learning framework `scikit-learn`[14].

A detailed description concerning the crucial steps are explained in the following subsections.

6.7.2 Algortihms

The three tested algorithms are the following:

- **Random Forrest Regressor** [15]

[14] `scikit-learn`: Machine Learning in Python (`http://scikit-learn.org/stable/index.html`)

[15] Random Forrest Regressor on `scikit-learn` documentation (`http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html`)

- **Gradient Boosting Regressor** [16]
- **Bagging Regressor (using decision trees or Support Vector Regression)**[17]

These machine learning algorithms are ensemble methods, as seen in 2.6.3. Their choice is based upon recommendation of the literature [27], in order to obtain a better accuracy.

The objective of using three different algorithm is to find which of them has the best overall performance.

6.7.3 Data Collection and Preparation

For feeding the machine learning algorithm, a procedure is called for fetching at the least one month of matches, counting from the reference date. Those matches must be, either, from working days or from the weekends, depending on the reference date day of the week.

Requirements

There are two requirements that must be fulfilled before proceeding:

- The first concerns the minimum amount of matches needed to make a prediction work, that is 2, regarding the used cross-validation method (`TimeSeriesSplit`, see [41]). `TimeSeriesSplit` is based on K-Fold [41] but does not shuffle the data randomly because time-series data is continuous, so disorganizing it would mean losing information (see figure 6.7).
- The second requirement is focused on the cardinality of each match. Since the gathered matches may not have the same completeness rate, the cardinality of each of the matches can be different, making the cross-validation method generate folds with different sizes and making the prediction impossible.

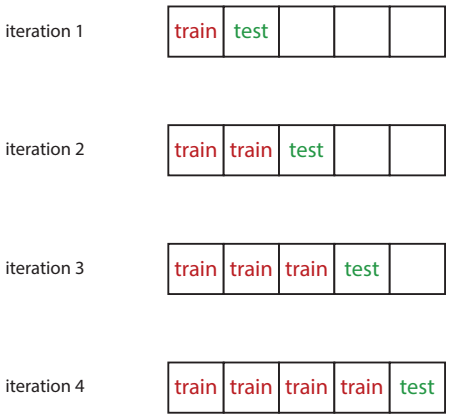


Figure 6.7: `TimeSeriesSplit` iterations over data

[16] Gradient Boosting Regressor on `scikit-learn` documentation (`http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#examples-using-sklearn-ensemble-gradientboostingregressor`)

[17] Bagging Regressor on `scikit-learn` documentation (`http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html`)

For solving both problems, the the data is normalized: if a match is missing a bus stop, its value is populated with the average of the value of the remaining matches. If all of the matches are missing that bus stop timestamp value, it is set to zero, signalling a miss. In this way, the machine learning algorithm can be trained, tested and validated using a normal machine learning workflow or using an utility class for choosing the best model given the hyper-parameters.

Dataset characterization

The dataset gathered for this study concerns the hypothetical bus passenger query for finding the time that a bus is predicted to arrive to bus stop `PRG4`, of the bus line `200 Castelo do Queijo` at the `15:00` (smartphone system time) of March 24th of 2017.

To gather the information, the function `get_matches_corresponding` is used. It considers a limit of 5 minutes below and above the reference time, and a date window of 1 month, with the objective of the machine learning model being more sensitive to newer input data. This results in 10 bus line completions, each one with 28 samples, being used on the final regression 9 bus for training and the remaining one for testing.

6.7.4 Model Selection and Cross Validation

Choosing a model is not an easy task. Tuning the hyper-parameters manually is an option but, better than trying to guess the values that make the model fits best the data it is important to use a scientific approach for obtaining them.

Regarding this, the hyper-parameter tuning is done with the help of `GridSearchCV`. It takes three main parameters:

- `base_estimator`: an estimator implementing the Estimator API (in this case, any of the ensemble learning regression algorithms).
- `param_grid`: dictionary with parameters as keys and lists of values to try as value.
- `cv`: a cross-validation generator (in this case `TimeSeriesSplit` is mandatory).

Three similar `Python` scripts have been developed for testing `GridSearchCV` with each one of the algorithms. The parameters which obtained the best settings were the following:

- **Bagging, using decision trees**:
  - `max_samples` (number of samples to use from the train dataset, to train each of the estimator): 0.9
  - `n_estimators` (number of boosting stages to perform): 1000
- **Bagging, using SVR**:
  - `base_estimator: SVR`
    - `C` (penalty parameter): 1e4
    - `Kernel` (kernel type for being used within the algorithm): 'rbf'
  - `max_samples` (number of samples to use from the train dataset, to train each of the estimator): 0.7

  - `n_estimators` (number of boosting stages to perform): 10

- **Gradient boosting**:
  - `learning_rate` (contribution of each estimator): 0.05
  - `loss` (loss function that needs to be optimized): 'lad'
  - `max_depth` (maximum depth of the individual estimators): 3
  - `n_estimators` (number of boosting stages to perform): 1000

- `Random Forrest`:
  - `max_features` (the number of features to consider when seaching for the best split): 'sqrt'
  - `n_estimators` (number of boosting stages to perform): 100

6.7.5 Evaluation

`GridSearchCV` also provides regression metrics to evaluate the model with newly found hyper-parameters.

Those metrics are:

- **Explained variance score** measures how well the mathematical model takes into account the dispersion of the input data. The possible values are between 0 and 1.0, being higher values better.

- **Mean absolute error** measures the average of vertical distance between the true point and the predicted.

- **Mean squared error** measures the square of the errors of the deviation. It is an indicator for the quality of the estimator. Values closer to zero are better.

- **Mean squared logarithmic error**[18] measures the square of the errors of the deviation. It is an indicator for the quality of the estimator. Values closer to zero are better. This metric is best employed when the target values of the regression has a exponential growth.

- **Median absolute error** measures the median *"median of all absolute differences between the target and the prediction"*[1]. Lower values are better.

- R^2 **score (coefficient of determination)** measures *"how well future samples are likely to be predicted by the model"*[1]. Values between 0 and 10, being 1.0 the best.

The metrics comparison are presented in the section 7.4, under **Results**.

[18] Mean Squared logarithmic error (`http://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-logarithmic-error`)

6.7.6 Deployment

It is, often, desirable to persist the machine learning model to avoid retraining the model again. In this study, the model is not persisted because the data would change for any variation of time and location. This pre-computation would require an effort that is out of the domain of this dissertation, requiring a totally different approach to the architecture – using a lambda architecture.

Therefore, the machine learning model is trained every time that a request is done, as seen in Prediction API (see 6.9.4), which is not a recommended way of deploying but serves well as proof-of-concept.

6.8 Database Wrappers Module

Despite of the creation of database functions for interacting with the databases, they still need a high-level way of interacting with the databases.

From the programmer point of view, calling those functions using a database adapter requires a certain amount of overhead when writing code: it requires opening a connection, creating a SQL statement, sending it, retrieving the result and finally, closing the connection.

To decrease this overhead, a set of database wrappers is implemented, making it easier to manipulate data from the database without having to write SQL code, but simply by using the methods of `Python` custom-made object.

This module makes use of `psycopg` (for the PostgreSQL databases) and `pymysql` (for the MySQL database), being reused in several parts of the architecture: on the Matching Unit, on the Synchronization Script and on the APIs.

6.9 Integration APIs

Delivering information in a interchangeable format must be a concern on applications like the one in this thesis. Regarding that, a set of very small REST-full services with a very specific purpose are implemented using the `Python` language and `connexion`[19], which provides a mean for presenting the API documentation automatically from a `swagger`[20] YAML[21] specification file.

Four separate APIs are implemented:

- **Bus Network Information API** delivers information about the bus carrier infrastructure (the lines, stops, the relation between them).
- **Matches API** delivers information about the matches, data which is stored in the *Matches Database*.
- **Estimation API** delivers only one endpoint for gathering the estimated times of arrival given a stop, a line and a date, data which is stored in the *estimation database*.

[19] Connexion, a Swagger/OpenAPI First framework for Python on top of Flask with automatic endpoint validation OAuth2 support (`https://github.com/zalando/connexion`)

[20] Swagger, The world's most popular API tooling (`https://swagger.io/`)

[21] YAML Ain't Markup Language (`http://yaml.org/`)

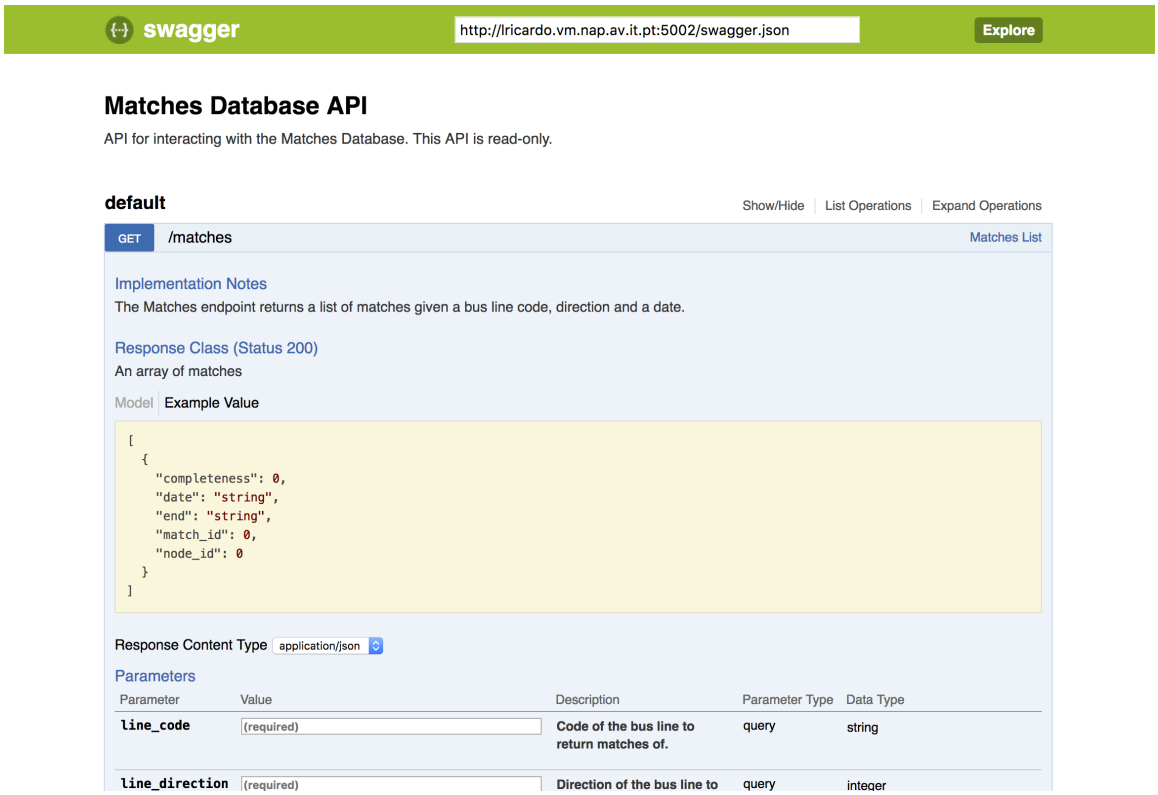


Figure 6.8: User Interface of Matches API, generated by Swagger

- **Prediction API** delivers an endpoint for making predictions, using the prediction module.

This is projected with two objectives in mind:

- To deliver information in an agnostic and inter-operable way for this project applications.
- To serve as an open API for future projects.

The APIs deliver information in the JSON and GeoJSON formats. The endpoints are explained in more detail on the following subsections.

6.9.1 Bus Network Information API

This API is composed by two endpoints:

- `/lines` provides the information about the carrier lines. It has two types of view: a list view and a detailed view. The list view lists all the lines and the detailed view displays only one line in detail. It is possible to expand the detailed view for obtaining the line's stops in a list view too.
- `/stops` provides information concerning the carrier bus stops. It also compresses a list view and a detailed view for delivering a list of stops and the details about a single one, respectively.

6.9.2 Match API

As the name implies, this API delivers information about the matches, being compelled by two endpoints:

- `/matches` presents a list view of the matches of given line a and date.
- `/matches/match_id` presents match details: a list of the matched bus stops from a given match, identified by a `match_id`.

In the first endpoint, the match presents some of the attributes of the table `match`, referred in the subsection 6.4.2: the `match_id`, `node_id`, `start`, `end` and `completeness`.

In the detailed view, it is shown a list like structure with three attributes: the `code` of the matched bus stop, the `order` of the stop in that matched line, and `gps_time`, which is the timestamp that is presented on the record which brought this match.

The Match API is used more consistently on the Line Performance Dashboard.

6.9.3 Estimation API

The Estimation API delivers time estimations for a particular bus stop of a bus line, in a date and time. These estimations are performed by calculating the values on the 0, 25, 50, 75, 100 percentiles. The value at the 0 percentile is referent to the time when the bus arrived sooner. On the opposite, the value at the 100 percentile is referent to the time when the bus arrived later.

There is a single endpoint – `/estimate`. It takes 7 arguments identifying the bus line, the bus stop and the time frame to gather the estimation from, and returns a single JSON object with the value of each of the percentiles.

Its main usage is done on the dashboard (described next) for displaying the evolution of the delays across a day, for example. This endpoint is also very relevant in the mobile application, where it is possible to a bus user to query the estimated time of a bus, in a given place.

6.9.4 Prediction API

This API is different than the ones previously stated. Despite of using the same technology (`Python` and `connexion`), it is a proof-of-concept that trains the `scikit-learn` machine learning algorithm described in 6.7, for predicting the estimated time of arrival of a bus in a given stop and line given a timestamp.

6.10 Applications

Two main applications have been developed as an effort to illustrate the outputs of the system architecture. One is a line performance dashboard, meant of bus carrier managers and the other is a mobile application for bus passenger. The following subsections explain each of the applications.

6.10.1 Bus Line Performance Dashboard

This dashboard makes use of the estimated times of arrival to deliver statistics based on the previous matches, and thus, based on past history.

The ideal solution for presenting this plot would be to gather the fixed time tables of the bus carrier, using them for feeding the estimation database and querying it for results. Unfortunately, that information is not available, and so, the plot compares the current match with the usual behavior in last month.

The first page of the dashboard consists on a list view, displaying the matches for a chosen line. The figure 6.9 presents this view, for the line *200 Bolhao*.

Then, after choosing a match, a detail view regarding the match detail for that line at that hour is shown. The plot, presented in the figure 6.10, presents a delay plot. Each point in the horizontal axis corresponds to a bus stop. The vertical axis measures the delay (in minutes). The zero value corresponds to the current measurement. The darker color track represent the values between the first and third quartile of the values.

To understand this plot, there are several characteristics that one must to be aware. The first is the stroke of the darker central area. If it is thicker, it means that the times of arrival have a high variance. However, if it is thin, the variance of the delay is lower, meaning that the line usually performs well (as the figure 6.10 shows).

In terms of technologies, `Angular`[22] is used. It is very important on the application organization and driving flow, given its capabilities for routing between pages, while dynamically fetching content, placing it in the web page seamlessly to the user.

While `Angular` provides a good foundation for building a web application, `ng-charts`[23] provides an interface for the flexible `ChartJS`[24] charting tool. For the other visual aids, `Semantic UI`[25] is used and for installing, managing and keep tracking of these libraries, `NPM`[26] is used.

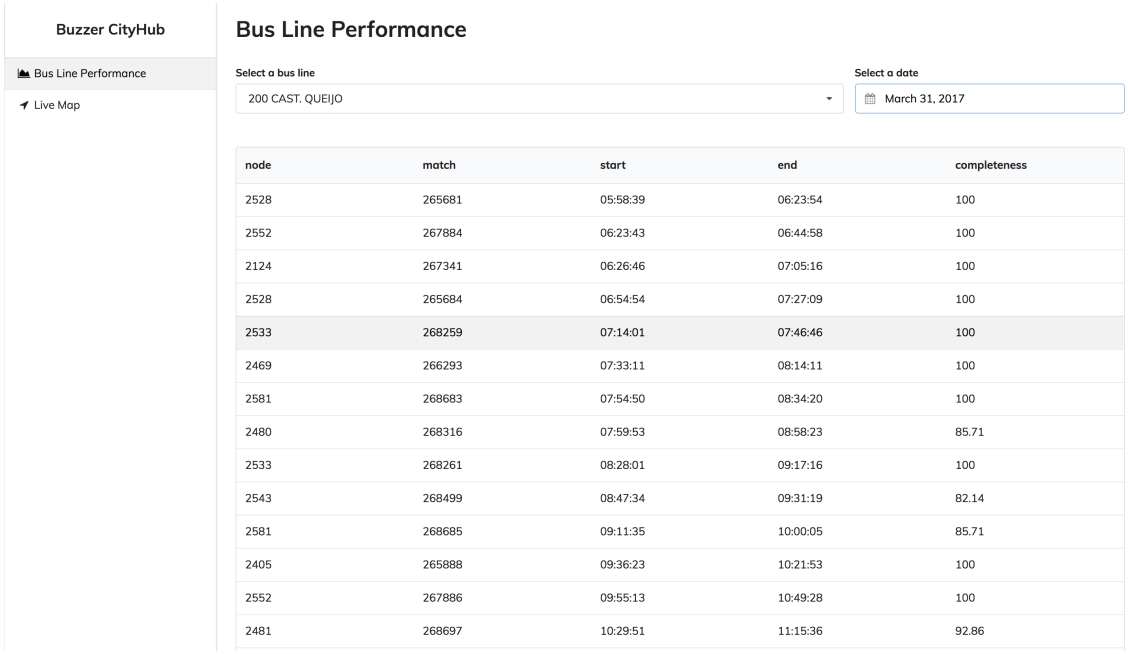
[22] Angular (`https://angular.io/`)

[23] ng-charts: Beautiful charts for Angular2 based on Chart.js (`https://github.com/valor-software/ng2-charts`)

[24] ChartJS: Simple yet flexible JavaScript charting for designers developers (`http://www.chartjs.org/`)

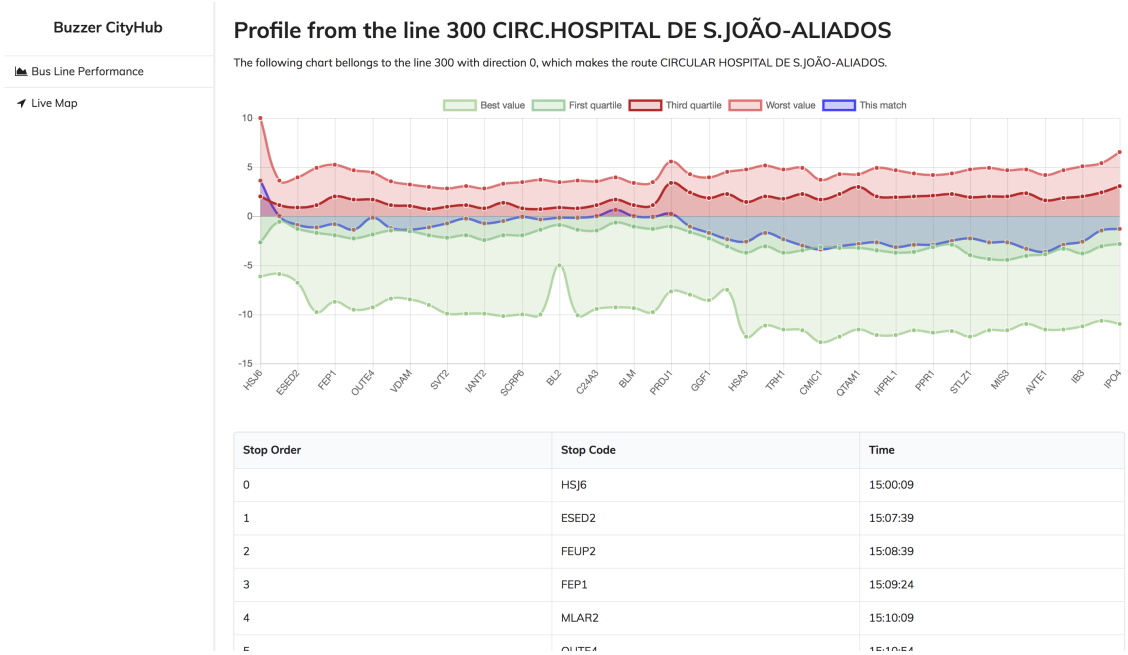
[25] SemanticUI: Design Beautiful Websites Quicker (`https://semantic-ui.com/`)

[26] Node Package Manager (`https://www.npmjs.com/`)



node	match	start	end	completeness
2528	265681	05:58:39	06:23:54	100
2552	267884	06:23:43	06:44:58	100
2124	267341	06:26:46	07:05:16	100
2528	265684	06:54:54	07:27:09	100
2533	268259	07:14:01	07:46:46	100
2469	266293	07:33:11	08:14:11	100
2581	268683	07:54:50	08:34:20	100
2480	268316	07:59:53	08:58:23	85.71
2533	268261	08:28:01	09:17:16	100
2543	268499	08:47:34	09:31:19	82.14
2581	268685	09:11:35	10:00:05	85.71
2405	265888	09:36:23	10:21:53	100
2552	267886	09:55:13	10:49:28	100
2481	268697	10:29:51	11:15:36	92.86

Figure 6.9: This is the match browser view, where the bus fleet manager can analyze each one of the matched bus journeys.



Stop Order	Stop Code	Time
0	HSJ6	15:00:09
1	ESED2	15:07:39
2	FEUP2	15:08:39
3	FEP1	15:09:24
4	MLAR2	15:10:09
5	OUTE4	15:10:54

Figure 6.10: A view over the delay plot

6.10.2 Bus Passenger Mobile Application

Enhancing public transportation can be done through various ways. One could be through the provisioning of open APIs, capable to deliver information about schedules, the actual

position of a bus, etc. Another way is by building applications whose purpose is to increase the mobility of public transportation users.

Regarding this, bringing applications for the bus passenger is a more appealing way of showing how the results can be an advantage for the topic of *Smart Transportation*.

This application is implemented using `Ionic`[27], which is a *"free and open source mobile SDK for developing native and progressive web apps with ease"*[10]. Being *progressive* means that it uses the last and trending technologies. By *web app*, it means that it uses web technologies like regular pages use, while it may appear to be looking like a normal mobile application. It is a cross-platform application too.

For showing the map, it is used `LeafletJS` with the plugin `Leaflet.markercluster`, which is a marker clustering plugin, because there are 2521 bus stops from STCP in Porto, and showing them in a mobile device in such litle area is computationally very intensive. A curiosity regarding this is that this implementation is much more faster than the Android native counter-part.

Other details of the implementation regard `Ionic` internals which are based on `Apache Cordova`[28], for the location (because the application is location aware) and 2D acceleration (used for better visual performance).

This application consists in three views:

- The **map view**, where the user may find a bus stop or query for its location and find bus stops nearby.
- The **detail view**, which enables a user to know from which bus lines the selected bus stop bellongs, as well as analyze the estimated time of arrival.

The figures 6.11a and 6.11b display the mobile app.

6.11 Summary

This chapter covered the most important implementation details of the project, starting from the data acquisition, going through the end-users applications.

One of the most important aspects of the implementation focuses on the matching unit, regarding the different approaches and conclusions, which resulted almost as direct consequence of the chosen development language (Python).

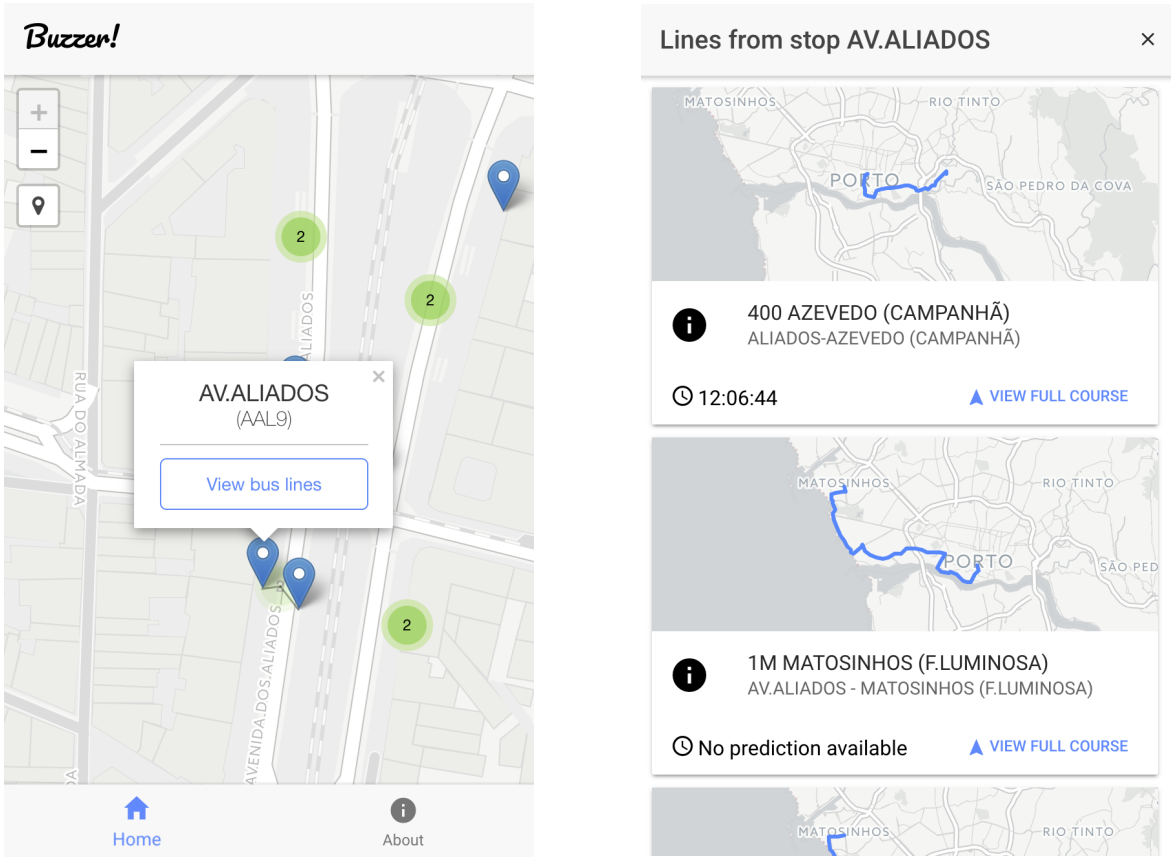
Also, the matching algorithm and its heuristic are explained, and the pseudo-code is provided.

The machine learning problem is also presented, clarifying the main methodologies adopted for computing its models.

Then, an overview was done concerning the database modelling and their routines. As mean for making applications interacting, APIs were also constructed.

Finally, the applications are presented as immediate results and proofs-of-concept of this implemented system.

[27] The top open source framework for building amazing mobile apps." (`https://ionicframework.com/`)
[28] Apache Cordova (https://cordova.apache.org/)



(a) The application map view

(b) The application detail view

Figure 6.11: Two different views of the mobile application

Chapter 7

Deployment and Results

7.1 Introduction

After the system implementation, two additional steps are required to prove its functionality. One is the act of putting it in production. The other is to analyze and discuss the quality of the results. This last step is required for validating this work.

This chapter presents:

- The **Deployment** is presented, describing the available hardware and software.
- The **Results** are presented and discussed, either for the case of the matching algorithm, either for the case of the travel time estimation and prediction.

7.2 Deployment

7.2.1 Hardware

The system is deployed on two different machines, with different specifications and availability. One is a virtual machine which, despite of high availability, is not powerful enough to deploy the full architecture. The other is a desktop computer, used development machine, that, regardless of not being equipped with power supply redundancy and being subject to network modifications, it is the only available machine capable of supporting the implemented system in terms of performance.

Therefore, target deploy system in use is the development machine, having the following characteristics:

- Intel Core i7-7700, with a base clock of 3.60GHz, with 4-cores and 8 threads.
- 32 GB of RAM memory.
- 512 GB of disk storage (SSD).

7.2.2 Software

The deployment, in terms of software, can be seem in two different layers:

- The operative system.

- The deployment tools (for orchestration and containerization utilities).

This software runs on top of a Linux-based operative system. It does not depends on a specific distribution, but it is required that the distribution supports the containerization system in use – `Docker`.

`Docker` is a top leading container platform which eliminates integration problems because, as a container platform, everything that is required to make the software to run is packed inside containers, built with all the libraries and settings which are strictly required to make the software run. Regarding this, the software will always run, independently of the used operative system.

Eight containers support the needed infrastructure:

- `matches_database` is based on the official `Postgres` image from Docker Hub and supports the Matches Database.
- `matches_api` is based on the official `Python` image from Docker Hub. It backs the Matches API, described on the section 6.9.2.
- `estimation_database` is based on the official `Postgres` image from Docker Hub and backs the project Estimation Database, where the times of arrival are stored for later computation.
- `estimation_api` is based on the official `Python` image from Docker Hub. It backs the Estimation API, described on the section 6.9.3.
- `busnetinfo_database` is based on an user image denominated `mdillon/Postigs`, from Docker Hub, providing a `Postgres` instance with `Postgis` extension. It provides STCP bus network information: the bus lines, stops and their relation. `Postgis` enables the making of spatial queries.
- `busnetinfo_api` is based on the official `Python` image from Docker Hub. It backs the Bus Network Information API, described on the section 6.9.1.
- `algorithm` is based on the official `Python` base image, and provides a base environment for running the Matching Unit.
- `sync` is also based on the official `Python` base image, and provides a base environment for running the synchronization script.

This arrangement is easily obtained thanks to `Docker Compose`, which *"is a tool for defining and running multi-container Docker applications"*[12]. It simplifies the management of the container's network, forwarding ports from the containers to the host, and also, organize the orchestration of services. All the containers are able to communicate between them, using their container name.

Considering the container's orchestration, a dependency chain is mapped between some of the containers. For example, the containers which are deploying APIs will launch the respective database container upon start. The container in which the Matching Unit runs depends on the containers which run both databases (Matches Database and Estimation Database).

The system implementation produces two types of results: the line matching and the delay metrics. The objective of this section is to discuss their quality and meaning.

7.3 Results

7.3.1 Matching GPS traces with bus lines

Context

Regarding the objective of obtaining delay metrics, part of this work focuses on solving the missing link between the position log database and the bus network information.

First, numbers are presented for explaining how many matches were actually done, and what the indicators tell about the matching processing.

It is essential to understand whether or not the match is acceptable or accurate, because a non-ideal match will generate inaccurate delay metrics.

Processed Data in numbers

A natural question, at this stage, is how many bus lines were matched in the period of this study (since the start of January until the end of March). Below, we present some numbers:

- **Number of SQL records processed**: 94 328 959
- **Average processing time (depends on the dataset)**: 35h14m per month.
- **Number of detected bus lines**: 268 702, totaling 2985 bus line completions per day.
- **Number of detected bus stops**: 9 460 737, totaling 105 119 bus stop detections per day.
- **Average of the completeness**: 97.199(89) %.

The three months of captures result in more than 94 million SQL records with the location of the buses. With this technique, 268 thousand matches were found, adding value to `vanetV3`.

The average completeness, despite of not being a totally reliable metric, shows that in the average, all of the bus lines detected have 97.2% of its bus stops detected, being a strong indicator that the algorithm works mostly well.

The inclusion of more variables to assess the accuracy of this algorithm will be a good topic for the future work.

Results presentation

Given the bus lines diversity in terms of shape, it is needed to test the effectiveness of the algorithm. The following figures present examples of different matched lines, with different shapes, on different dates. The figures have two main visual elements: the red dots represent the GPS log of the match being shown, while the blue line is the matched line.

Circular lines are the trickiest ones because they start and end in the same place. Regarding that, a major effort is done for ensuring its correct detection, as referred on the section 6.3.5.

The figure 7.1 shows the line `300 - CIRCULAR HOSPITAL DE S.JOAO-ALIADOS`, that is one of the most popular STCP Lines.

Most of the lines are **one-way straight** i.e. not making any cycle, as the figure 7.2 shows.

The algorithm is also able to detect line gaps, thus, detecting the correct line, as seen in the figure 7.3.

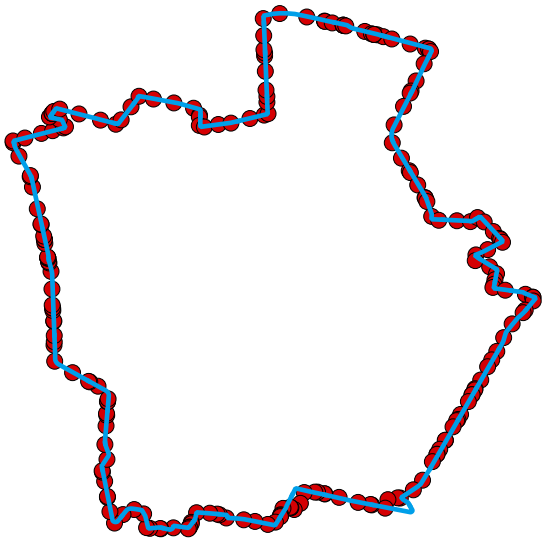


Figure 7.1: The match from the `node_id` 2764 from March, 1 St, from 07h00m to 07h55m, with 100% completeness.

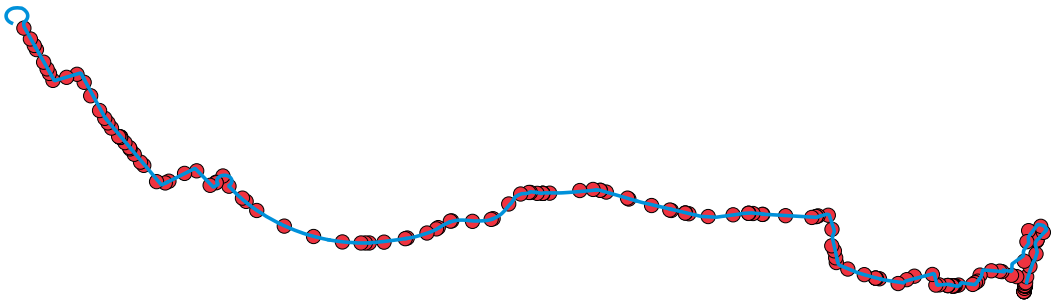


Figure 7.2: The match from the `node_id` 2474 from March, 15 St, from 14h32m to 15h19m, with 100% completeness

---

Analysis and Validation

Evaluating weather or not a line is being correctly matched is not an easy job. There is no way of validating this solution accurately in an automatic fashion.

Therefore, it is done a visual validation. From the visual validation, it can be observed that the match is done correctly, but some can left respectable clues.

From the analysis of the existing matches, the following type of errors can be observed:

- Errors that result from gaps existing on the input records which are bigger than the TTL, or gaps that suppress the completion of a line; the gaps on the input records can be observed on the figure 7.3.

- Errors that result from a bus starting the reverse line that it completed previously, taking more time than the TTL to reach the first target stop of the new starting line.

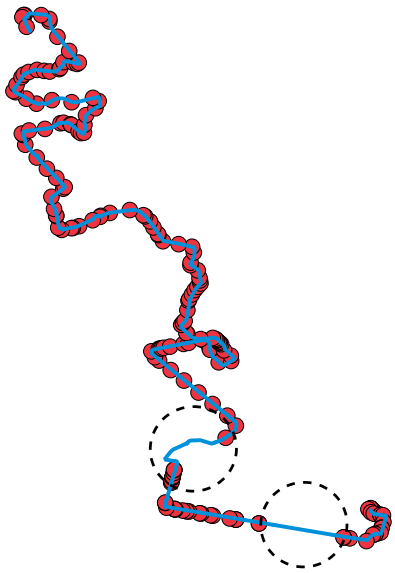


Figure 7.3: The match from the `node_id` 2801 from March, 8 St, from 18h10m to 19h17m, with 91% completeness due to gaps on the input records from the position log database, outlined with stroked circles.

There are other types of situations that do not fall in the error category, but yet, reveal uncertainty about the match being done. The line `1 - INFANTE`, shown on the figure 7.4, can be matched with 100% completeness, and yet, be a classification mismatch.

Looking deeper to its match details, it is possible to understand what can happen.

Describing the figure 7.4:

- In t_0, the bus arrives after completing the line `500 Praça da Liberdade`. Then, it goes around.
- The algorithm tests the lines starting in t_1. The reverse line `500 Matosinhos` starts there. Unfortunately, the algorithm is not able to find the line ending because, despite of the bus going along the route and passing the position marked by the label t_3, there is not any GPS information after the last red dot (in the left), for a long period of time (almost 30 minutes).
- In t_2 it is detected a line that overlaps the previous one. It is tested and it is declared as a solution on t_3.

Regarding this, there are three situations which are plausible of causing problems, decreasing the level of confidence in the match, which are:

- There was an error with the log gathering.

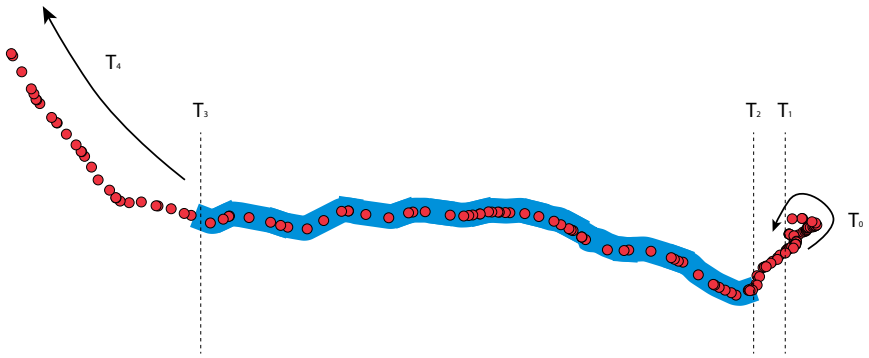


Figure 7.4: A line completion with uncertainty

- It was a strategical stop, for making the reverse line `1 - PASSEIO ALEGRE`.
- It was the bus driver launch time (this occurred circa twelve o'clock).

Therefore, no accurate validation can be made, except if the bus carrier is involved.

7.3.2 Delay metrics for bus lines and estimated times of arrival

Context

Under the context of this study, the bus carrier network website provides information about the bus lines and bus stops. The STCP website also considers the bus schedules (also known as time tables) for a small subset of the bus stops of a bus line[1].

Given the previous processing and considering the static timetables available, we can consider the following results:

- The delay plot of a bus line match.
- The estimated time of arrival in a stop, given a previously defined static value from the STCP time tables.
- The estimated time of arrival in a stop, given a dynamically chosen value (user given or inferred by the delay plot analysis).

The following sections present results regarding each one of those three situations.

Result presentation 1.1: Delay plot of a bus line match

The delay plot of a bus line match is presented on the Bus Line Performance Dashboard, described on the section 6.10.1. This dashboard focuses on displaying statistics for understanding the evolution of a given line completion, mainly through the use of a *delay plot*, which is a plot enabling the comparison between each timestamp of the matched bus stops with the usual behaviour, that is based on historic values.

[1]STCP Timetable Page (http://www.stcp.pt/en/travel/lines/?linha=200sentido=0t=horariostd=2)

Five values are presented: the current match represents the current bus line completion that is being compared, the inner red line represents the value of the third quartile (75%) and the inner green represents the first quartile (25%). These metrics are represented towards the median value (second quartile, with value 50%), defined as zero. The outter values sample the edges of the distribution. Values higher than zero are said to be late in relation to the distribution.

The plot x axis corresponds to the bus stops and the labels identify the stop codes. The y axis corresponds to the delay, in minutes.

This plot is an important tool because it enables:

- Understanding if the current match does behave normally, when comparing it to its past history.
- Understanding if the line completion being analyzed on the plot is completed uniformly, or, if it varies locally (next to a bus stop or a set of bus stops).
- To watch the difference in the behaviour between the line completion in the rush and non-rush hours.
- Performing long-term time estimations (for instance, in 3 months).

Regarding these characteristics, we have chosen some of the bus lines of STCP that seem to reveal traffic patterns or mobility problems.

The bus line `204 - HOSPITAL DE SAO JOÃO` reveals very well the traffic patterns. It comprises some popular points of interest, such as *Faculdade de Engenharia da Universidade do Porto*, *Escola Superior de Educação* and *Hospital de São João*.

Having such important destinations, it is a line that is worth studying. Either students, teachers and other faculty members must arrive in time to their destination. Topographically, it traverses the whole city, including the center, being potentially impacted by traffic.

Below, we present the variation of the bus line profiles during February 20, using the period from January 20th until February 20th of 2017 using 5 distinct periods:

- From `07:01:33` until `07:53:48`, which considers the start of the morning rush hour.
- From `07:57:50` until `09:02:20`, which is located during the rush hour.
- From `12:01:05` until `12:56:51`, which is the lunch time, out of the rush hour period.
- From `15:03:36` until `15:59:51`, which is out of the rush hour period but may reveal some patterns of traffic regarding the city daily life (for instance, end of the school time).
- From `17:26:37` until `18:44:52`, which considers the start of the late hours rush hour.

The plots regarding each one of these time periods are presented on the figures 7.5, 7.6, 7.7, 7.8 and 7.9. The *x-axis* represents the bus stop codes and the *y-axis* represents the time in minutes.

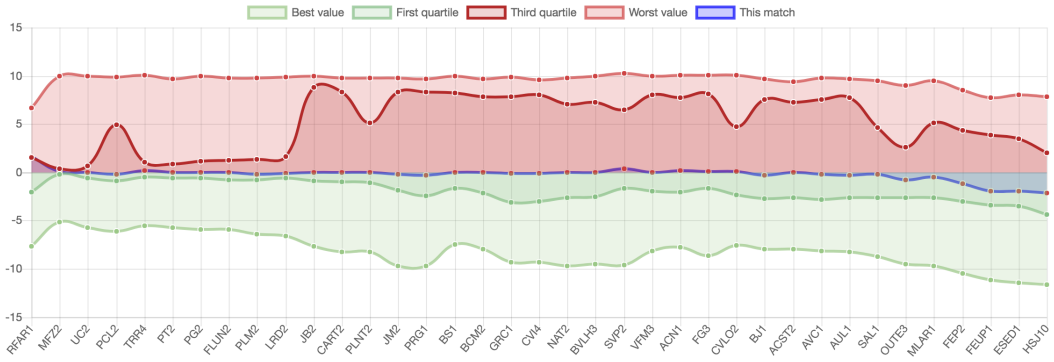


Figure 7.5: Line 204 Hospital de S. João from `07:01:33` to `07:53:48`

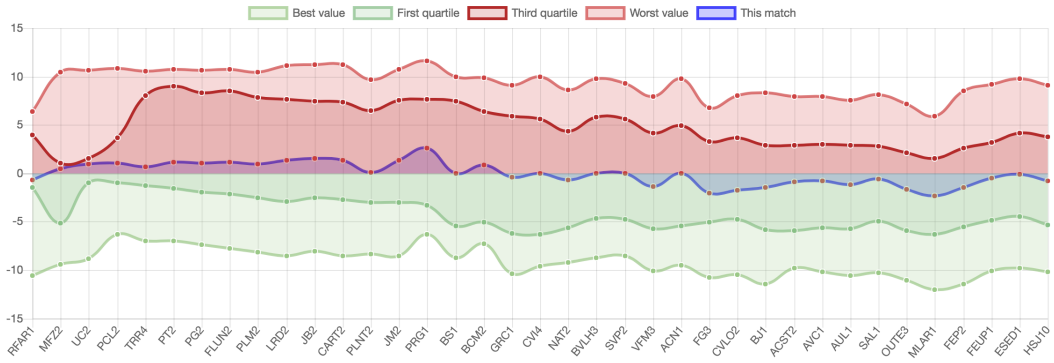


Figure 7.6: Line 204 Hospital de S. João from `07:57:50` until `09:02:20`

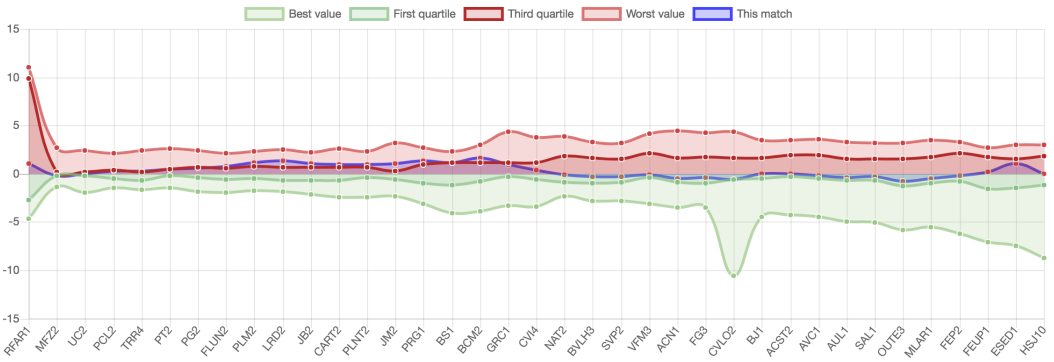


Figure 7.7: Line 204 Hospital de S. João from `12:01:05` until `12:56:51`

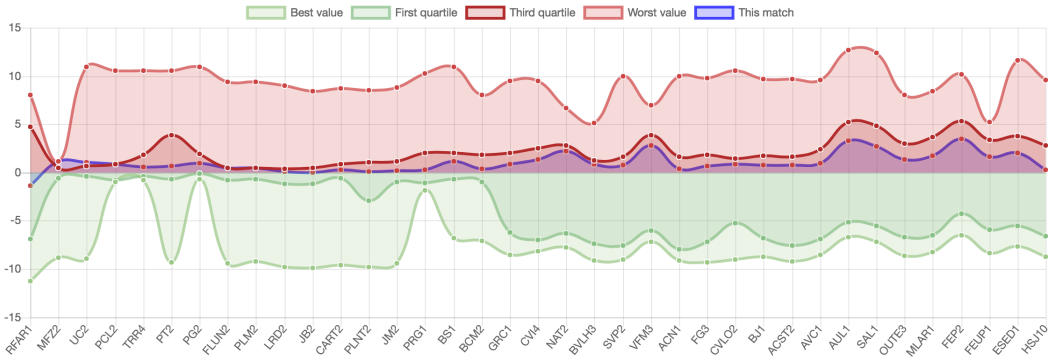


Figure 7.8: Line 204 Hospital de S. João from `15:03:36` until `15:59:51`

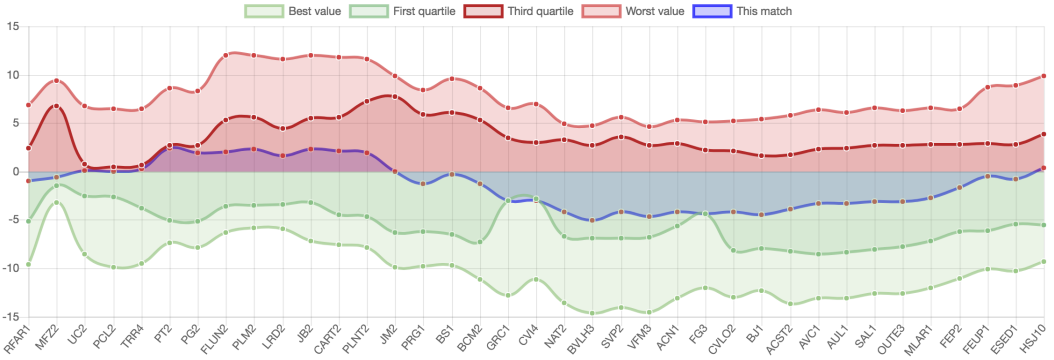


Figure 7.9: Line 204 Hospital de S. João from `17:26:37` until `18:44:52`

Now that the delay plots are presented, we may take some conclusions:

- Under the figure 7.5, the bus behaviour in that day was very similar to the usual behavior, in that time frame. The deviation (distance between the value of the first and third quartiles) is significant in terms of delay, so further inspection is needed to understand if the line tends to present delay, or if it is a consequence of the bus line completion in a very low time interval.
- Under the figure 7.6, the deviation increases, intuitively, due to traffic. The darker inner track limited by the first and third quartile defines a variation that goes almost to 8 minutes, showing that this bus line may present some difficulty on complying with the schedules. Presenting prediction for bus passenger of those bus lines may present errors up to more than 5 minutes.
- The figure 7.7 presents a plot showing that, after the rush-hours, the deviation is much lower. The bus (in blue) performs accordingly with the usual.
- The figure 7.8 shows that, in this period, the bus line works fairly normaly until the third quarter of the line completion. Later, approximating the faculties, it starts worsening.
- The final plot, 7.9 presents a profile that may be correlated with the traffic of *Rotunda da Boavista*, which is one of the main roads of Porto. The deviation increases too, mainly

due to the *street Diogo de Botelho*, which is very dense in traffic regarding *Universidade Católica* and being an access to *Via de Cintura Interna*.

Result presentation 1.2: Detecting problematic lines

Using a longer time to gather estimations results in a great convergence of the percentiles being calculated. Therefore, the estimations are more accurate providing a better long-term estimation.

The figure 7.10, 7.11 and 7.12 present bus lines delay plot referring to the same period of time, on different city flows.

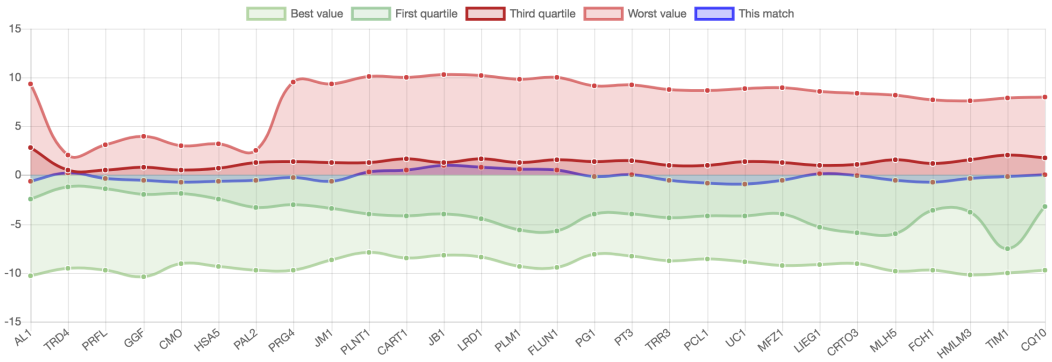


Figure 7.10: 200 Castelo do Queijo, 3 months analysis

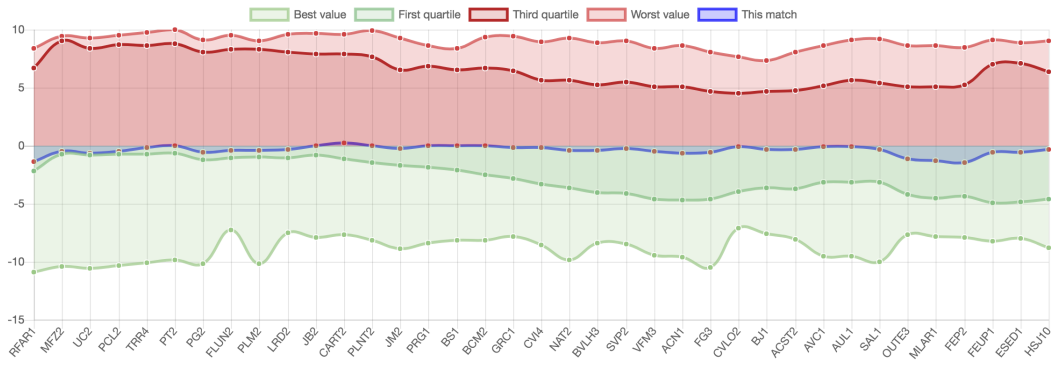


Figure 7.11: 204 Hospital de S. João, 3 months analysis

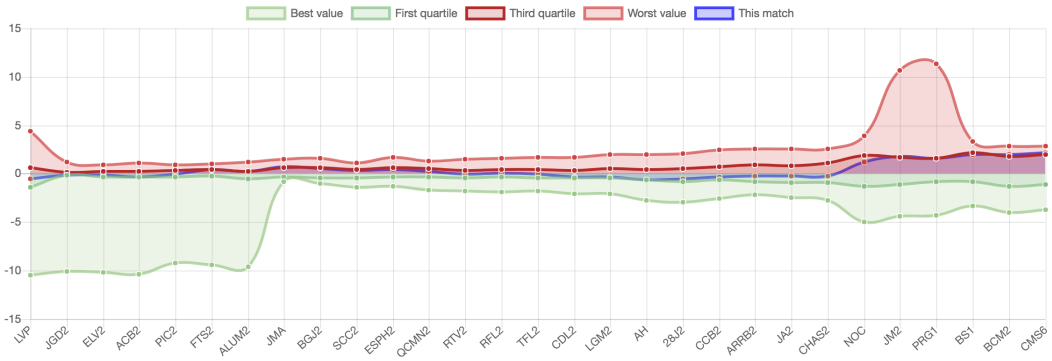


Figure 7.12: 902 Boavista, 3 months analysis

Looking to the figure 7.10 plots, we verify that the third quartile is very close to the median. This means that the variance of the samples in that quartile is low, so it provides a very strong evidence that the line completion usually completes well.

Then, the plot of the figure 7.11 shows that the bus completion of this day was done accordingly with the normal. Despite of that, it also reveals that the line has tendency to present regular delays because the distribution disperses a lot until the third quartile. This bus line is subject to have mobility issues, but it requires further investigation with the transportation company to understand the veracity of this conclusion, despite of being a logical one.

Finally, the plot of the figure 7.12 presents a line that again, does not present mobility problems.

Result presentation 2: Estimated time of arrival given a previously defined static value from the STCP time tables

Imagining that the bus carrier wants solid results, quantified by numbers, rather than a plot, it is possible to deliver the statistical intervals for each of the bus stops, given their schedules, for understanding if the public timetables are or not up to date.

To do that, there is not an automatic method, but using the Estimation Database, it is possible to gather such results.

For exemplification purpose, it is shown the table 7.1, concerning statistics from the line `200 Castelo do Queijo`, having reference time from STCP Website and considering the month March only, for the estimation.

Table 7.1: Statistics for a time query associated to a bus line and bus stop

stop_code	reference_time	best_time	first_quarter	median	third_quarter	worst_time
AL1	07:40:00	07:36:23	07:38:50	07:39:49	07:44:53	07:44:58
PRG4	07:53:00	07:50:12	07:51:56	07:53:04	07:54:12	07:56:52
LRD1	07:59:00	07:56:15	07:57:26	07:58:53	08:01:12	08:02:25
MFZ1	08:06:00	08:02:35	08:05:22	08:06:41	08:08:57	08:09:37
CQ10	08:15:00	08:10:15	08:13:07	08:15:04	08:17:19	08:19:25

It can be seen, over the data set, that it is very usual that the median is very close to the reference times from the STCP Website, indicating that this line worked as expected in

March and that the schedules available on the STCP website are right, for this time frame.

Result presentation 3: Estimated time of arrival given a dynamically chosen value (user given or inferred by the delay plot analysis)

This result is very interesting because it opens possibility for delivering one of the main objectives: building smart schedules.

One example that can be used is the use-case of an user trying to query for the next schedule for a bus that he wants to catch. Using his smart phone, he chooses a bus stop, a line, chooses a time and submits a query like this:

For the Line:*"204 Hospital de São João*

For the Bus Stop:*"For the Bus Stop: Foz"*

For the Input Time:*"For the input time: 08:10 "*

return a recommendation

As a result the system can output something like this:

"You can catch the bus performing the line 204 Hospital de São João at 8:11."

This is possible thanks to the Estimation Database methods which are used for retrieving values for the delay plot. Instead of using the time stamp of a match, it uses an input given by the user. The statistics table has the format presented on the table 7.2.

Table 7.2: Statistics for a time query associated to a bus line and bus stop

best_time	first_quarter	median	third_quarter	worst_time
08:05:46	08:10:05	08:11:04	08:12:55	08:14:50

Because the value of the median corresponds to the most typical value in a sorted sequence, we can consider that the 50% of the values around the median are between 08:10:05 and 08:12:55, providing a solid estimated time of arrival, based on history.

Regarding this result, one may wonder if there is a bus schedule at this time. From the STCP website, there are, in fact, two schedules for this period: one starting at 08:05 a.m and another starting at 08:15 a.m. This means that the person will be able to catch the bus starting at 08:05 (delayed) or the bus which will supposedly arrive at 08:15 a.m.

Validation

Validating these results is again a hard mission. One way of doing it would be contacting the bus carrier and try to make them collaborate by delivering measures of performance (if they have them). Although they were contacted, we are waiting for this information.

Another way of validating these results would be by planting a smartphone to probe the GPS position along time, in a known bus, completing a known line – but it would be a very restricted test, to very small subset of the bus infrastructure.

7.4 Prediction Module Results

The prediction module main goal is to choose the algorithm that best performs to the presented date. Therefore, the main goal of this section is to discuss what model is best suited for being used in the Prediction API. Regarding this, this section presents a comparison of

the regression metrics and a comparison between the values resulting from the best prediction model and the real values.

7.4.1 Regression Metrics Comparison

`GridSearchCV` is used for training and testing the machine learning models with a different range of hyper-parameters. It delivers, also, regression metrics for the best hyper-parameter value combination. Those resulting metrics are presented in the table 7.3.

Table 7.3: Resulting Evaluation Metrics

	Bagging (Decision Tree Estimator)	Bagging (Support Vector Regressor)	Gradient Boosting	Random Forrest
Explained Variance Score	0.982	0.975	0.985	0.984
Mean Absolute Error	179.999	133.562	183.049	183.319
Mean Square Error	38972.588	22614.798	39265.194	39925.414
Mean Square Log Error	1.309e-05	7.682e-6	1.320e-05	1.342e-05
Median Absolute Error	201.664	130.909	206.808	206.723
R^2	0.899	0.941	0.898	0.897

Despite of having a slightly lower explained variance score, using the bagging algorithm with the support vector regressor is an overall better choice because:

- The **coefficient of determination** R^2 is higher, meaning that it is likely that this model is better on predicting accurately.
- The **mean square error, mean square logarithmic error, and the median absolute errors** are lower than the counterpart models.
- The **model is more general**: when plotting the solution, the generated regression is mathematically simpler than the counterparts (see the figures 7.13, 7.14 and 7.15 for a better comparison).

The figure 7.13 represents the regression curve calculated by the Bagging algorithm, using SVR as estimator:

- The *x-axis* represents the bus stops.
- The *y-axis* represents a UNIX epoch in float.
- The dots are the samples used for training the model, each one belonging to a past matching experience (*i.e.* a past bus line completion).
- The green curve represents the regression (built with the regression prediction).

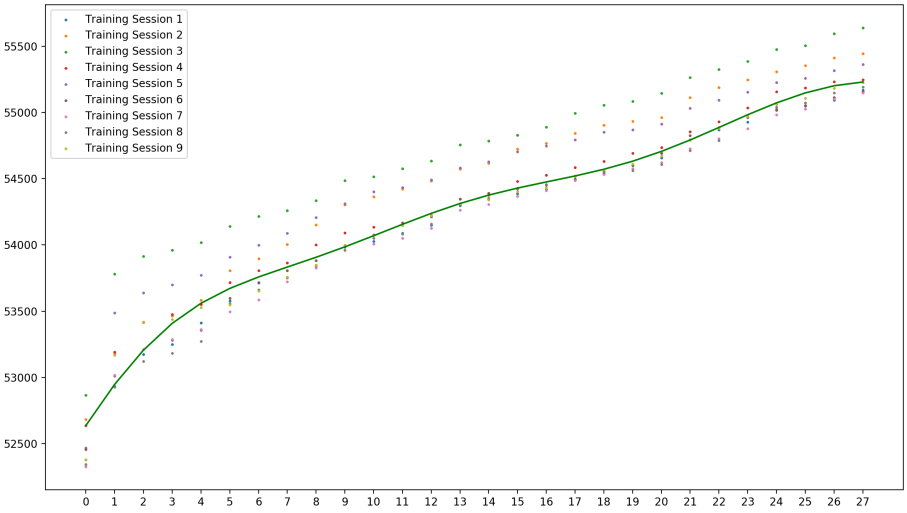


Figure 7.13: Bagging (using support vector regressor)

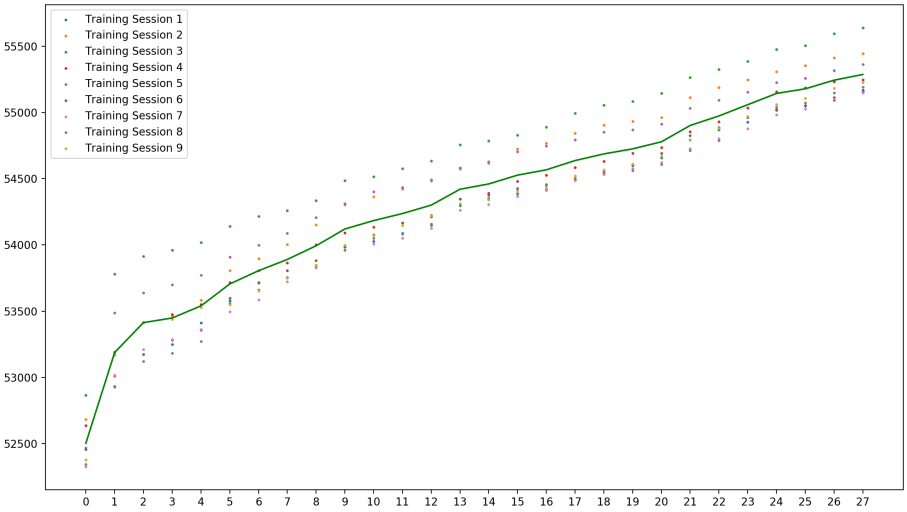


Figure 7.14: Random Forrest

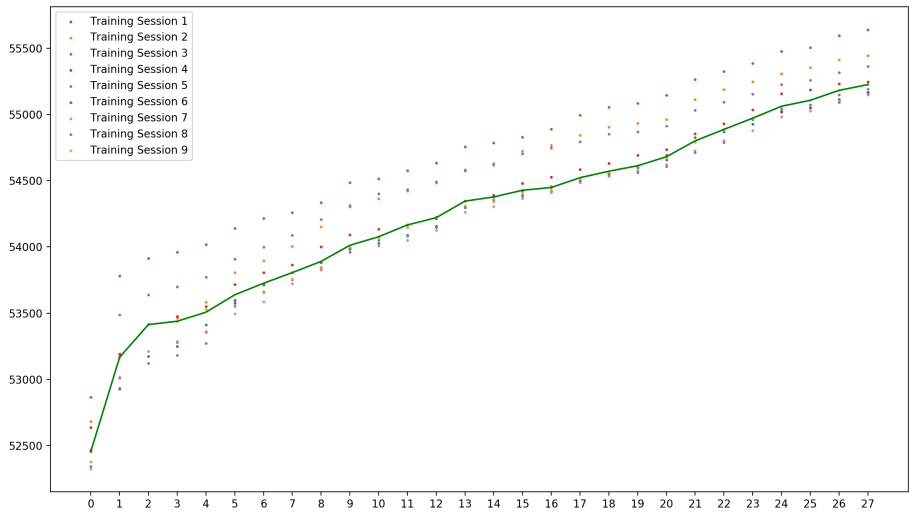


Figure 7.15: Gradient Boosting

Therefore, the model chosen for being used in the Prediction API is Bagging with SVR as base estimator. An important note is that the median error (an mean absolute errors) are approximately 130 seconds, meaning that predicted passing times may have an error in the order of 2 minutes and 10 seconds. This is subject to change, depending on the data variance, something that needs to be studied in more depth, in the future.

7.5 Summary

Despite of being very hard to validate the results, they are interesting and meaningful. About the matching algorithm, we can conclude that:

- It is capable of matching different shape bus lines, which is a good indicator that the algorithm works, due to the decision conditions like *"detecting lines starting and ending in the same place"*, or being tolerant to failures due to gaps existing in the position log records.
- It is not easy to have an accurate metric of accuracy about the matching algorithm without recurring to third techniques like regressions, spatial analysis, or machine learning.
- Certain matches are uncertain and cannot be validated unless there is some communication with the bus carrier to know more about their drivers habits and their operational mode for scheduling lines.

Regarding the delay metrics and estimated times of arrival, we can also conclude that:

- The delay plot is a good tool for visualizing what usually happens in a bus line. Comparing them can give good insights about the line behaviour between a period of time

(a morning, a day, etc). It also enables the understanding of the schedule variation, more precisely, perceive if its variation is higher or lower, a characteristic that raises more accuracy on the recommended estimated time of arrival.

- Obtaining estimated times of arrival is useful for understanding whether the schedules from the STCP website are or not right. Despite the fact that it is only shown one example, it is known that STCP have their schedules up to date.
- Alternatively, estimated times of arrival are also useful and relevant when using a non-static reference time, like those given by bus passengers.

Finally, considering the Prediction Module, the algorithm with best performance is Bagging with Support Vector Regressor as base estimator. Despite of that, looking for those indicators is not enough considering that, if the variance of the data is high, the prediction model is likely to produce inaccurate predictions. It is necessary to study better ways of fetching data, so the error can be further minimized.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This work started by addressing mobility, defining its timeless importance. Vehicular networks are raising new opportunities for building new applications that go beyond safety. This work is integrated in the area of the mobility and smart transportation, proving the applicability of the operational data, passively built, generating meaningful information for bus passengers, municipalities and transportation companies.

The match algorithm is a generic solution, depending on few data to deliver a matching solution. With it, it is possible to guess what bus line is a bus traversing, not only in Porto, but also in other places, if the vehicular network exist and also information regarding bus carrier network is provided. The amount of the data being classified is almost 10 times bigger than the one addressed in the literature, and similarly, it is very hard to make assumptions regarding the validation and the accuracy of the method.

Then, the arrival time estimation follows a completely different approach than the literature, but yet, delivers foundations for using as recommendation for bus passenger, or for delivering reports and decision support systems for the bus fleet operators.

Also prediction is introduced, exploring ensemble learning as a measure for improving the accuracy of the literature studied methods. The estimators perform in a similar way, but the best performing one is bagging, using support vector regressor as base estimator, since it is more general and provides a lower error.

Finally, the applications for bus passengers and bus fleet operators are a proof of the utility provided by this whole architecture. The mobile application has value as proof-of-concept, demonstrating an excellent use-case of the use of the newly generated information. It is very valuable for the bus passenger, because it may help him to decide his schedule depending on the state of bus network. On the other hand, the performance dashboard shows one of the many applications of the newly processed data (in this case, studying bus line completion patterns). Future work will provide real assessment of these applications.

8.2 Lessons Learned

The development history of this work lead to some *lessons* that can be shared as valuable information for students or researchers pursuing the same type of work. Regarding this, the figure 8.1 presents the development history of this master's thesis practical development, and

then, conclusions regarding the work methodologies and technologies choices are drawn.

Figure 8.1: Development phases

Under **Problem Analysis** phase, which took place from Early December until late January, the main objective was to become familiar with the data and separate the problem into smaller tasks. Off the smaller problem set, guessing which bus line was a bus completing was the critical phase because, without it, no time estimations would be possible. Regarding this, two techniques were studied:

- **Graph Matching** consisted in transforming the bus network in a graph, and then, converting also the successive points of the trajectory into a graph to perform complete and incomplete graph matching. The two main technologies used were Neo4j[1] and NetworkX[2]. Both produced results, but they produced a huge overhead regarding network requests and costly data structures, respectively.
- **GIS based Map-Matching** was not possible regarding the non-existence of the full route on the STCP website. Even if the whole data had been there, it would be very difficult to operate and combine such amount of information in a few time.

After spending some time getting familiar with the technologies, making tests and ending up with the conclusion that none of technologies would work, the first learned lesson was

[1] `Neo4j`: The #1 Platform For Connected Data (`https://neo4j.com/`)
[2] `NetworkX`: Software for complex networks (`https://networkx.github.io/`)

simple: **no solution is the best on solving all the problems**. In fact, graph matching was a very good methodology for finding the best route regarding a set of locations, GIS was perfect for making proximity searches, and machine learning was good for finding hidden patterns on data but, none of them was the best for solving the problem of guessing what bus line was a bus completing. **Not falling into the *finding the best technology* loop is very important, and yet another lesson.** None of the technologies is an one size fits it all, so for specific purposes must be implemented specific solutions.

Under the **Algorithm Design** phase, the main issue was the lack of data. In the start of a data science related project, it is usual that someone delivers a set of files with a preview or a small subset of data. This is potentially a bad idea because someone starting a project does not know how much data is sufficient for studying a set of particular events. Also, working with dated data is dangerous. For instance, the initial data granularity was from 1 in 1 second; then, after having access to the data, it was from 15 in 15 seconds. Therefore, some of the efforts related with the amount of data were useless. Regarding this, the lesson to be learned is that **one should assert that the data to be studied has the shape and form of the data currently in production, and that the data comprises several samples because different shapes will produce different effects** which may reveal general failures on the algorithm.

Under the **Architecture Design**, the first version was monolytic: one macro-component with tightly coupled micro-components. The problem is that, as usual, requirements change. Therefore, the lack of modular components started to feel heavier and heavier until each of the components were teared apart. **Decouple your project** is the third lesson to be learnt, as it eases the deployability, maintainability, reusability and scaleability.

Then, under the **Time Estimation, APIs and Apps** phase it was time to study different ways of producing time estimations, building the apps and also the bridge connecting these two. The major problem on each of these development objectives is concerned with the technology choice.

A conservative developer would choose something like PHP for creating the web services, and would program the applications in Java (native Android), because it is easy to program, maintain and document. A non-conservative one would take a chance to experiment the latest trends on technology, like Javascript or Typescript in the different layers of the project, and so, increasing the risk of having to rewrite some of the code again and again because one of the libraries have suddenly changed.

Because I chose the second approach, I would not recommend it unless there is time to learn them or a very strong reason. The Learned lesson is to **chose the technology that best fits the purposes of the architecture. Either regarding database technologies, either regarding web technologies bet on traditional if there is no previous experience and heavy time constraints**. For instance, `AngularJS`, which is an older but well maintained and enterprise level technology would be a better choice than using the latest `Angular`, despite of `Angular` being an interesting solution for future web development projects. Also, in what concerns databases, InfluxDB is a very interesting and promising time-series NoSQL database, but it misses a query analyzer[3], so measuring its performance is a hard mission. Keep in mind **that the latest and trending technology is not always the best as it can be hard to use, hard to maintain and in the vast majority of**

[3]Calculate per query statistics (issue on GitHub) (`https://github.com/influxdata/influxdb/issues/263`)

the times, very incomplete too.

Finally, under the Time Prediction and Enhancement phase, the machine learning methods were studied and the prediction implemented. The databases were optimized for delivering the best performance. The final task, studying the results, is very hard and it can require external input. **If further input is needed from a third-party, like a business partner, it shall be made as soon as possible**.

8.3 Future Work

This dissertation leaves room for future developments and potential expansions. The most critical are:

- *Enhance the algorithm and the time estimations*: the algorithm matches well but with further granularity it would be more accurate and deliver better estimations regarding the possibility of studying the stay times and measuring also the bus acceleration dynamics through the different segments of the bus lines.
- *Porting this work to a Big Data architecture*: the primarily and main objective of this work is to deliver estimated times of arrival. Regarding the fact of having missing link between the data existing in the position log database (vanetV3) and the bus network information, the map-matching problem turned to be one of the main efforts of this work. Having that solved, an obvious optimization of this system would be implementing on top of a *Lambda Architecture* (see 2.5.3).
- *Add further functionality to the dashboard*: the dashboard is being used as proof-of-concept. Some important features can be added, including, showing the exact location of each node on the map in real-time, adding alerts for helping to detect abnormal situations, etc.
- *Integration with the application SenseMyCity*: regarding the project S2MovingCity from the program Carnegie Mellon Portugal, it is expected that the estimated times of arrival produced by this system will be integrated in the application SenseMyCity until December 2017. This application is popular among our colleagues from FEUP, and will give us big human feedback.

Bibliography

[1] *3.3. Model evaluation: quantifying the quality of predictions — scikit-learn 0.19.1 documentation*. URL: http://scikit-learn.org/stable/modules/model_evaluation.html (visited on 26/10/2017).

[2] *Actor Systems • Akka Documentation*. URL: http://doc.akka.io/docs/akka/current/scala/general/actor-systems.html (visited on 01/07/2017).

[3] David Manuel de Oliveira Alves. *Real-time Trip-Planner in Urban Public Transport: Specifications and Preliminary Tests for Lisbon*. 2011.

[4] Sarah Guido Andreas C. Mueller. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, 2016. ISBN: 978-1-44936-941-5.

[5] Albert S. Woodhull Andrew S. Tanenbaum. *Operating Systems: Design and Implementation*. 3rd ed. Pearson, 2006. ISBN: 0131429388, 9780131429383.

[6] Mansur As and Tsunenori Mine. 'Empirical Study of Travel Time Variability Using Bus Probe Data'. In: *2016 IEEE International Conference on Agents (ICA)*. IEEE, Sept. 2016, pp. 146–149. ISBN: 978-1-5090-3931-9. DOI: 10.1109/ICA.2016.050. URL: http://ieeexplore.ieee.org/document/7812995/.

[7] Blaise Barney. *Introduction to Parallel Computing*. URL: https://computing.llnl.gov/tutorials/parallel_comp/#whatis (visited on 01/07/2017).

[8] Clara Benevolo, Renata Paola Dameri and Beatrice D 'auria. 'Smart Mobility in Smart City: action taxonomy, ICT intensity and public benefits'. In: (). URL: http://www.cersi.it/itais2014/pdf/52.pdf.

[9] Paul Black. *greedy algorithm*. URL: https://xlinux.nist.gov/dads//HTML/greedyalgo.html (visited on 29/05/2017).

[10] *Build Amazing Native Apps and Progressive Web Apps with Ionic Framework and Angular*. URL: https://ionicframework.com/ (visited on 08/06/2017).

[11] Annalisa Cocchia. *Smart and Digital City: A Systematic Literature Review*. Springer International Publishing Switzerland, 2014. ISBN: 978-3-319-06159-7. DOI: 10.1007/978-3-319-06160-3. URL: http://link.springer.com/10.1007/978-3-319-06160-3.

[12] *Docker Compose — Docker Documentation*. URL: https://docs.docker.com/compose/ (visited on 08/06/2017).

[13] European Space Agency. *A force that shapes our planet (GOCE)*. URL: http://www.esa.int/Our_Activities/Observing_the_Earth/GOCE/A_force_that_shapes_our_planet (visited on 19/07/2017).

[14] Scott Fortmann-Roe. *Accurately Measuring Model Prediction Error*. 2012. URL: http://scott.fortmann-roe.com/docs/MeasuringError.html (visited on 18/10/2017).

[15] Kenneth Laberteaux Hannes Hartenstein. *VANET Vehicular Applications and Inter-Networking Technologies*. 1st ed. Intelligent Transport Systems. Wiley, 2010. ISBN: 0470740566,9780470740569,0470740620,9780470740620.

[16] Peter Harrington. *Machine Learning in Action*. Manning Publications, 2012. ISBN: 1617290181,9781617290183.

[17] Ronan Harris. *Improving our brand safety controls*. Mar. 2017. URL: https://www.blog.google/topics/google-europe/improving-our-brand-safety-controls/.

[18] *Harvard Kennedy School - Saguaro Seminar :: About Social Capital*. URL: https://www.hks.harvard.edu/programs/saguaro/about-social-capital (visited on 22/05/2017).

[19] *IBM Knowledge Center - Overview of SQL routines*. URL: https://www.ibm.com/support/knowledgecenter/en/SS6NHC/com.ibm.swg.im.dashdb.apdv.sqlpl.doc/doc/c0024289.html (visited on 23/10/2017).

[20] *Infographic: The Four V's of Big Data — IBM Big Data & Analytics Hub*. 2015. URL: http://www.ibmbigdatahub.com/infographic/four-vs-big-data (visited on 18/10/2017).

[21] *Intelligent Transportation Systems - MIT Portugal*. URL: https://www.mitportugal.org/education/transportation/research/intelligent-transportation-systems (visited on 14/10/2017).

[22] Intergovernmental Committee on Surveying and Mapping. *Fundamentals of Mapping*. URL: http://www.icsm.gov.au/mapping/datums1.html%7B%5C#%7Djargon (visited on 19/07/2017).

[23] *Lambda Architecture — MapR*. URL: https://mapr.com/developercentral/lambda-architecture/ (visited on 18/10/2017).

[24] Helena Lindskog. 'Smart communities initiatives'. In: *Proceedings of the 3rd ISOneWorld Conference* April (2004), p. 16.

[25] Diogo Lopes and Susana Sargento. 'Network mobility for vehicular networks'. In: *2014 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, June 2014, pp. 1–7. ISBN: 978-1-4799-4277-0. DOI: 10.1109/ISCC.2014.6912465. URL: http://ieeexplore.ieee.org/document/6912465/.

[26] *Machine Learning: Bridging Between Business and Data Science*. URL: https://www.altexsoft.com/whitepapers/machine-learning-bridging-between-business-and-data-science/ (visited on 18/10/2017).

[27] João Mendes-Moreira et al. 'Comparing state-of-the-art regression methods for long term travel time prediction'. In: *Intelligent Data Analysis* 16.3 (2012), pp. 427–449. ISSN: 1088467X. DOI: 10.3233/IDA-2012-0532.

[28] Usue Mori et al. 'A review of travel time estimation and forecasting for Advanced Traveller Information Systems'. In: *Transportmetrica A: Transport Science* 11.2 (Feb. 2015), pp. 119–157. ISSN: 2324-9935. DOI: 10.1080/23249935.2014.932469. URL: http://www.tandfonline.com/doi/abs/10.1080/23249935.2014.932469.

[29] Taewoo Nam and Theresa A Pardo. 'Conceptualizing Smart City with Dimensions of Technology, People, and Institutions'. In: *The Proceedings of the 12th Annual International Conference on Digital Government Research Conceptualizing*. 2011, p. 10. URL: https://inta-aivn.org/images/cc/Urbanism/background%20documents/dgo_2011_smartcity.pdf.

[30] James Warren Nathan Marz. *Big Data: Principles and best practices of scalable realtime data systems*. 1st ed. Manning Publications, 2015. ISBN: 1617290343,9781617290343.

[31] Regina O. Obe and Leo S. Hsu. *PostGIS in action*. Manning, 2011, p. 492. ISBN: 9781935182269. URL: https://www.manning.com/books/postgis-in-action.

[32] David R. O'Hallaron Randal E. Bryant. *Computer Systems: A Programmer's Perspective*. 2nd. Prentice-Hall, 2011. ISBN: 0136108040,9780136108047.

[33] Sebastian Raschka. *Python Machine Learning*. Packt Publishing, 2015. ISBN: 978-1-78355-513-0.

[34] Rudy Raymond and Takashi Imamichi. 'Bus trajectory identification by map-matching'. In: *Proceedings - International Conference on Pattern Recognition*. IEEE, Dec. 2017, pp. 1618–1623. ISBN: 9781509048472. DOI: 10.1109/ICPR.2016.7899868. URL: http://ieeexplore.ieee.org/document/7899868/.

[35] Robert Osserman. *ellipsoid — geometry — Britannica.com*. URL: https://www.britannica.com/topic/ellipsoid (visited on 20/07/2017).

[36] João G P Rodrigues, Ana Aguiar and João Barros. 'SenseMyCity: Crowdsourcing an Urban Sensor'. In: (2014). URL: https://arxiv.org/pdf/1412.2070.pdf.

[37] Ralph Kimball; Margy Ross. *The data warehouse toolkit : the complete guide to dimensional modeling*. 2ed. Wiley, 2002. ISBN: 0471269433,9780471269434.

[38] Robert E Schapire. 'Explaining adaboost'. In: *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*. 2013, pp. 37–52. ISBN: 9783642411366. DOI: 10.1007/978-3-642-41136-6_5. URL: http://rob.schapire.net/papers/explaining-adaboost.pdf.

[39] Scikit-Learn. *1.11. Ensemble methods — scikit-learn 0.19.0 documentation*. 2017. URL: http://scikit-learn.org/stable/modules/ensemble.html (visited on 18/10/2017).

[40] Jim Scott. 'Lambda Architecture'. In: (). URL: https://spark-summit.org/2014/wp-content/uploads/2014/07/Lambda-Architecture-Jim-Scott..pdf.

[41] *sklearn.model$_s$election.TimeSeriesSplit*. URL: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html%20http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_predict.html (visited on 21/10/2017).

[42] Kehua Su, Jie Li and Hongbo Fu. 'Smart city and the applications'. In: *2011 International Conference on Electronics, Communications and Control, ICECC 2011 - Proceedings* (2011), pp. 1028–1031. ISSN: 2327-4662. DOI: 10.1109/ICECC.2011.6066743.

[43] Richard S. Sutton. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN: 0-262-19398-1.

[44] T. Vincenty. 'Direct and Inverse solutions of geodesics on the ellipsoid with application of nested equations'. In: *Survey Review* XXII.176 (1975). URL: https://www.ngs.noaa.gov/PUBS_LIB/inverse.pdf.

[45] Nobuhiro Uno et al. 'Using Bus Probe Data for Analysis of Travel Time Variability'. In: *Journal of Intelligent Transportation Systems* 13.1 (Jan. 2009), pp. 2–15. ISSN: 1547-2450. DOI: 10.1080/15472450802644439. URL: http://www.tandfonline.com/doi/abs/10.1080/15472450802644439.

[46] Kenneth Cukier Viktor Mayer-Schonberger. *Big Data: A Revolution That Will Transform How We Live, Work, and Think*. 1st ed. Eamon Dolan/Houghton Mifflin Harcourt, 2013. ISBN: 9780544002692,9780544002692.

[47] Roger W.Caves, ed. *Encyclopedia of the City*. Routledge, 2005, pp. 625–626. ISBN: 0415252253 (alk. paper).