PONTIFICAL CATHOLIC UNIVERSITY OF MINAS GERAIS
Graduate Program in Informatics

Pedro Pongelupe Lopes

PONDIÔNSTRACKER: A FRAMEWORK BASED ON GTFS-RT TO IDENTIFY DELAYS AND ESTIMATE ARRIVALS DYNAMICALLY IN PUBLIC TRANSPORTATION NETWORK

Belo Horizonte
2023

Pedro Pongelupe Lopes

PONDIÔNSTRACKER: A FRAMEWORK BASED ON GTFS-RT TO IDENTIFY DELAYS AND ESTIMATE ARRIVALS DYNAMICALLY IN PUBLIC TRANSPORTATION NETWORK

Dissertation presented to the Graduate Program in Informatics at Pontifical Catholic University of Minas Gerais, as a partial requirement to obtain Master's degree in Informatics.

Advisor: Prof. Humberto T. Marques-Neto

Belo Horizonte

2023

FICHA CATALOGRÁFICA

Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

Pedro Pongelupe Lopes

PondiônsTracker: A framework based on GTFS-RT to identify delays and estimate arrivals dynamically in Public Transportation Network

Dissertation presented to the Graduate Program in Informatics as a partial requirement for qualification to the Master degree in Informatics from the Pontifical Catholic University of Minas Gerais

---

Prof. Dr. Humberto Torres Marques-Neto –
PUC Minas (Advisor)

Prof. Dr. Silvio Jamil Ferzoli Guimarães –
PUC Minas (Examining Board)

Prof. Dr. Davide Bacciu –
Università di Pisa (Examining Board)

Belo Horizonte, December 04, 2023.

AGRADECIMENTOS

A conclusão dessa dissertação simboliza um vitória pessoal gigantesca. E não seria possível sem uma série de pessoas envolvidas.

Em especial, à minha namorada Naraiane que, neste ano de 2023, vem compartilhando comigo as mais variadas experiências: alegrias, tristezas, incertezas e boas risadas. Muito obrigado por ter me levado ao hospital naquele dia, sou eternamente grato. Sem o seu companheirismo diário e conversas importantíssimas que tivemos enquanto preparávamos uma boa receita degustando um bom vinho, essa dissertação não existiria.

Ao Professor Humberto que me acompanha em mais uma etapa da minha trajetória acadêmica, certamente a mais complexa para mim, até então. Obrigado por embarcar nas ideias mirabolante que tenho, sempre me inspirando a ser um pesquisador melhor.

À minha família que sempre me apoiou e me proporcionou alento em momentos difíceis, especialmente aos meus pais, Soraya e Eustáquio. Também agradeço à Elaine e Toninho que sempre me tiveram por perto e me mostram mais sobre o mundo. Obrigado por terem me acolhido em Ouro Branco.

Aos meus amigos por vivenciar o processo todo comigo, especialmente durante a pandemia que factualmente moramos juntos. Eram ótimas risadas e comes e bebes de qualidade duvidosa. Ao Daniel, Banana, obrigado em dobro por me acompanhar em fazer o mestrado, me incentivou muito em iniciar, continuar e finalizar, e também obrigado, pela comemoração de aniversário na carreta furacão. Não poderia deixar de salientar como o excelentíssimo Patrick Galdino me diverte em inúmeras ocasiões e faz um ótimo feijão tropeiro. Também ao Paulinho que além de cupido, é um ótimo colega de trabalho e amigo. Toto, João, Vic, Alaor e outros, muito obrigado pela amizade sincera.

Finalmente, agradeço ao Jorge Ben Jor, FBC e Don L que estiveram preenchendo o ambiente com ótimas ideias enquanto essa dissertação foi escrita. Inclusive, estou escutando Jorge Ben enquanto termino este agradecimento. Salve Jorge! Viva São Jorge!

Venceremos!

"Hay que endurecerse, pero sin perder
la ternura jamás."

Ernesto 'Che' Guevara

ABSTRACT

A smart city Public Transportation Network provides mobility services to millions of people daily. Urban computing provides a toolkit to handle acquisition, integration, and analysis, which translates to the improvement of the human mobility of the citizens, mitigating and notifying delays, for instance. Regarding the PTN, many methods rely on two specifications: *General Transit Feed Specification* (GTFS) and *General Transit Feed Specification Real-Time* (GTFS-RT). The first represents the static schedule information, and the second introduces real-time updates from trips, services, and vehicle positions. Despite the qualitative leap with the GTFS-RT specification, GTFS-RT is not as well-adopted as the GTFS because of the non-existence of a matching identifier between the static and real-time data. In this context, we present *PondiônsTracker*[*] which is a *loose coupling* Java framework designed for enriching GTFS data with real-time data to enable delays analysis and to estimate arrivals. So, we present *PondiônsTracker-BH*[†] that is a *PondiônsTracker*'s specialization created to deal with Belo Horizonte's PTN particularities originating from Belo Horizonte's real-time *Application Programming Interface* (API) which we collected every minute for eleven days straight in July and August 2023, summarizing over 246 million entries representing almost 30 Gigabytes. *PondiônsTracker-BH* presented a 76.08% of the matched trips for the schedule during the observation period, then 156,628 out of 205,884 scheduled trips were identified in the real-time data. Analyzing these 156,628 matched trips, we show the delays focus and show that the delays in Belo Horizonte are spatial and temporal related and are *log-normal* distributed. In other words, most of the delays in Belo Horizonte occur at a few bus stops, and these stops are *physically* close to each other and share the same *temporal* patterns.

Keywords: Urban Computing, Human Mobility, Complex Networks, Public Transportation Network.

---

[*] Available at `https://github.com/Pongelupe/PondionsTracker/`

[†] Available at `https://github.com/Pongelupe/PondionsTracker-BH`

LIST OF FIGURES

FIGURE 1 – GTFS data structure 21
FIGURE 2 – The PTN modeled as the graph G 23
FIGURE 3 – *PondiônsTracker*'s architecture diagram 28
FIGURE 4 – Entities Class Diagram 29
FIGURE 5 – *real_time_bus*'s relational model 31
FIGURE 6 – *shapes_summarized*'s relational model 31
FIGURE 7 – Data Providers Class Diagram 33
FIGURE 8 – Integration Module Class Diagram 35
FIGURE 9 – Entries and d_t representations 39
FIGURE 10 – Empty set of entries 39
FIGURE 11 – A more complex scenario of no entries within d_t 41
FIGURE 12 – Example of bus stop couples and their distance 42
FIGURE 13 – Integration Driver Activity Diagram 43
FIGURE 14 – Belo Horizonte's GTFS and RT data 45
FIGURE 15 – Histograms Schedule-Filled Percentage per Routes 49
FIGURE 16 – $DELAY$s Distribution: Bus Stop and Trip 51
FIGURE 17 – 300 Most Delayed Stops 52
FIGURE 18 – Fragment of the 50 Most Delayed Stops 52
FIGURE 19 – Minutes out of schedule of bus stops #14788408 distributed throughout the day 53
FIGURE 20 – Minutes out of schedule of a couple of bus stops distributed throughout the day 54
FIGURE 21 – Minutes out of schedule of a couple of bus stops distributed throughout

the day using generated data 57

LIST OF TABLES

TABLE 1 – Example of merge of e_{n-1} and e_n 40
TABLE 2 – Fields From The Real-Time API 45
TABLE 3 – Workload Overview 47
TABLE 4 – Belo Horizonte's PTN 48
TABLE 5 – Top 5 Routes With The Most Unscheduled Trips 50
TABLE 6 – Delays detailed in whole PTN scale 50
TABLE 7 – Delays detailed in whole PTN scale with generated expected times 56
TABLE 8 – Statuses Distribution for Bus Stops 56

LIST OF ABBREVIATIONS AND ACRONYMS

API – *Application Programming Interface*

CSV – *Comma-Separated Values*

DDL – *Data Definition Language*

GIS – *Geographic Information System*

GTFS – *General Transit Feed Specification*

GTFS-RT – *General Transit Feed Specification Real-Time*

OKF – *Open Knowledge Foundation*

PTN – *Public Transportation Network*

SQL – *Structured Query Language*

CONTENTS

1 INTRODUCTION **15**
1.1 Objectives 16
1.2 Master Thesis Structure 17

2 THEORETICAL FOUNDATION **18**
2.1 Smart Cities 18
2.1.1 Urban Computing 18
2.1.2 Human mobility 19
2.2 GTFS and GTFS-RT specifications 20
2.2.1 GTFS 20
2.2.2 GTFS-RT 21
2.3 Complex Networks and Graphs 22
2.3.1 Public Transportation Networks as a complex network 23

3 RELATED WORK **25**

4 PONDIÔNSTRACKER **28**
4.1 Entities 29
4.2 Data Module 31
4.2.1 GTFS Data Importer 31
4.2.2 Real-Time Data Collector 32
4.2.3 Data Providers 32
4.3 Integration Module 34
4.3.1 Trip Extractor 35
4.3.2 Trip Matcher 36
4.3.3 Trip Selector 37
4.3.4 Trip-Bus Stop Linker 38
4.3.5 Trip Expected Time Generator 41
4.3.6 Integration Driver 42

4.4 PondiônsTracker-BH 44
4.4.1 Real-Time Data Collector 44
4.4.2 Belo Horizonte's RealTimeService 45

5 RESULTS **47**
5.1 Schedule Analysis 47
5.2 Delay Analysis 50
5.3 Comparison Between Generated and Real Data 55
5.4 Limitations 58

6 CONCLUDING REMARKS **59**

REFERENCES **61**

1 INTRODUCTION

A smart city *Public Transportation Network* (PTN) provides mobility services to millions of people daily. This is far from being a trivial task because many major cities have hundreds of bus lines operating every day. The complexity and importance of this network have been a study object for a long time, and to create and manage the schedules, many approaches could use manual or computational methods. The many computational methods work with the *General Transit Feed Specification* (GTFS) (GOOGLE, 2005) and its real-time extension, *General Transit Feed Specification Real-Time* (GTFS-RT) (GOOGLE, 2009) which are industry standard specifications for sharing schedules and associated geographic information.

On the one hand, GTFS represents the static data of the PTN and its main entities are the trips, routes, stops, stop times, and fares. Thus, this specification has been promoting research in multiple fields, such as creating multimodal applications, ridesharing, and data visualization (ANTRIM; BARBEAU et al., 2013). On the other hand, GTFS-RT introduced a whole new dimension with real-time updates from trips, services, and vehicle positions. With GTFS-RT new research topics were introduced, such as measures of disparities in service provision, temporal variability, the role of relative travel times and costs in mode choice (WESSEL; WIDENER, 2017; AEMMER; RANJBARI; MACKENZIE, 2022; WRONA; GRZENDA; LUCKNER, 2022).

Both GTFS and GTFS-RT specifications are based on open data, which the *Open Knowledge Foundation* (OKF) (OKF, 2004) defines as "open data and content can be freely used, modified, and shared by anyone for any purpose."Some cities provide their open data through *open data portals*, which aggregate a wide range of datasets, such as urban planning, health, and tourism (AUER et al., 2007). These cities, which use their data to enable political efficiency and social and cultural development for their citizens, are considered smart cities (ALBINO; BERARDI; DANGELICO, 2015). Many smart cities follow OKF's mission "to create a free, fair, and open future, advancing open knowledge as a design principle beyond just data."These portals have been contributing to the massive popularity of these specifications.

In order to work with the huge volume of data daily produced in smart cities, urban computing provides a toolkit to handle acquisition, integration, and analysis of the data from multiple sources (ZHENG et al., 2014). Regarding the PTN, these tools aim to improve the human mobility of the citizens by creating models based on individuals' movement patterns. So, one common approach is to model the PTN as a complex network, in which the bus stops represent the nodes set, and the edges set is represented by the arcs connecting two nodes, in other words, the streets (FERBER et al., 2012). This modeling also embraces PTN's additional information about the sets previously mentioned, information as bus coordinates and delays. The fact that the PTN is updated, and basically, every new data input from hundreds of buses around a city expresses the complexity of this network.

Although the qualitative leap with the GTFS-RT specification, GTFS-RT is not as well-adopted as the GTFS. One may associate this condition with the lack of real-time data, but the issue occurs mainly due to the ownership of the data. Then, transit agencies provide real-time data, and the government provides the static data, commonly, leading to the situation where there is no matching identifier between the two datasets.In other words, many cities have *both* GTFS and a real-time service but does not have GTFS-RT. This issue was named as *no matching identifier issue* and for instance, Raghothama, Shreenath e Meijer (2016) describe this scenario for Rome back in 2016 and Wessel, Allen e Farber (2017) identified and proposed a method to shorten this gap for Toronto in 2017.

Yet in 2023, GTFS-RT is unavailable for many major cities due to the no matching identifier issue. Then, in this dissertation, we propose *PondiônsTracker*, a framework to enrich GTFS data with real-time data to mitigate this issue. *PondiônsTracker* is designed to work with as many cities as possible, so its components are replaceable and have their behaviors defined in interfaces. Thus, we validate our hypothesis that *PondiônsTracker* mitigates the no matching identifier issue using Belo Horizonte's data which also has the issue. Finally, our main contribution is to provide the PTN as graph enriched with delay data when GTFS-RT is unavailable.

1.1 Objectives

The main objective of this master's dissertation is proposing and validating PondiônsTracker, which is a framework to identify delays and improve the estimated arrival task in Public Transportation Networks in cities with buses, real-time data, and GTFS. So, reach the main objective, we use Belo Horizonte's data with the following specific objectives: 1. Collecting data from the real-time API and combining with the GTFS; 2. Understanding if Belo Horizonte's delays are spatial and temporal dependent by analyzing delays among bus stops; 3. Comparing the arrival times defined at the GTFS with the arrival times generated by *PondiônsTracker-BH*;

1.2 Master Thesis Structure

This master thesis is organized as follows. In Chapter 2, we present the theoretical reference, then we discuss the related work. Next, we describe the methodology used to build PondiônsTracker in Chapter 4. In Chapter 5, we present the results using PondiônsTracker to Belo Horizonte. Finally, we discuss our conclusions and future work.

2 THEORETICAL FOUNDATION

In this Chapter, we present concepts and techniques used throughout this work. We start presenting smart cities, urban computing, and urban mobility concepts, and then, information about how cities export their data on open data portals. Then, we show the GTFS and its real-time extension, GTFS-RT, which provides all the data required to execute our methodology. Finally, we define the PTN as a complex network after a gentle introduction to complex networks and graphs.

2.1 Smart Cities

There are many definitions of smart cities, Nam e Pardo (2011) describe a smart city is a city whose "data infuses information into its physical infrastructure to improve conveniences, facilitate mobility, add efficiencies, conserve energy, improve the quality of air and water, identify problems, and fix them quickly, recover rapidly from disasters, and collect data to make better decisions, deploy resources effectively, and share data to enable collaboration across entities and domains".

In addition, a smart city must be connected to its citizens and various systems, for instance, transportation, health care, and more. The dimensions of a smart city are its networked infrastructure enabling political efficiency and social and cultural development. It is social inclusion and social capital in urban development. Another key point is the measurement of performance indexes, which help keep the focus on resources and time where they are needed (ALBINO; BERARDI; DANGELICO, 2015). Finally, it is worth highlighting that the assessment of a smart city must be adapted to each specific scenario to develop the quality of life of its citizens.

2.1.1 Urban Computing

Urban computing is an interdisciplinary toolkit that uses the data generated by the sources in urban spaces to tackle the major issues that cities face (ZHENG et al., 2014). This toolkit comprises processes of acquisition, integration, and analysis of the data generated, which sources could be sensors, devices, vehicles, buildings, and humans, for instance. The problems faced are also interdisciplinary, they could be related to many fields, such as transportation (SILVEIRA et al., 2015; HUANG et al., 2020; WESSEL; WIDENER, 2017; AEMMER; RANJBARI; MACKENZIE, 2022; WESSEL;

ALLEN; FARBER, 2017; WRONA; GRZENDA; LUCKNER, 2022), and health (RO et al., 2020; SARAN et al., 2020; CHANG et al., 2020; SONKIN; ALPERT; JAFFE, 2020).

2.1.2 Human mobility

Human mobility aims to study the movement of humans through time and space. And its impacts on the environment, understanding human mobility benefits many study fields, such as traffic forecasting (WRONA; GRZENDA; LUCKNER, 2022), urban planning, and tourism (CARVALHO; MORAIS; CUNHA, 2018). Before introducing techniques and methods using urban computing, it is fundamental to briefly introduce its history, which dates back to the 19th century with the *Laws of Migration* (RAVENSTEIN, 1885). These laws try to explain predictions of migration patterns using socio-economic factors, despite the observational character, they are non-quantitative. Later on in the 1940s, Stouffer (1940) introduced the *Law of Intervening Opportunities* that Barbosa et al. (2018) defines as "the number of people going a given distance is directly proportional to the number of opportunities at that distance and inversely to the number of intervening opportunities."In other words, when migrating, the further you are willing to displace, the more opportunities you should have. However, unexpected suitable opportunities may appear before the person arrives at the original destination, called intervening opportunities.

Zipf (1941) applied his law from observing the rank-frequency dependence in linguistics, the eponymous Zipf's law. This infers that the frequency of a word ranked z has the statistical dependence of $f_z \sim 1/z$, in terms of usage (BARBOSA et al., 2018). When Zipf applied this law to cities, the rank z was not a constant yet varied due to two competing forces: Diversification and unification. The first expresses the likelihood of the population living near the source of raw materials to shorten the distances to the production center, leading to multiple centers of a small population. The second force describes the inclination of populations to concentrate in urban centers causing the minimization of work required to transport finished products to consumers, leading to a large population in few centers.

In the 1990s, human mobility methods allowed to model more complex human dynamics by incorporating a space-time prism using *Geographic Information System* (GIS) techniques (MILLER, 1991). So, the models used spatial and temporal constraints upon an individual's movement patterns to enrich the analysis. Yet, in the 1990s, despite upgrading the models, the data available failed to calibrate the models, which issue was solved in the 21st century (BARBOSA et al., 2018). GPS data usage became well-adopted with mobile phones, and this scenario led to the development of more complex data mining methods.

Also, in the 21st century, we live Zipf (1941)'s scenario of few centers with large populations, but along with urban computing methods from this century led to the improvement of mobility methods in the urban space to a level of real-time services to mass populations. These models are based on the premise that every day, many citizens are going to displace over the whole city, this could be done in many ways, such as walking, cycling, driving, and using public transportation. Consequently, these models help cities make decisions considering their own data. Thus, understanding the patterns underneath and their history is the key point to create effective models, such as identifying delays and other public transportation questions.

2.2 GTFS and GTFS-RT specifications

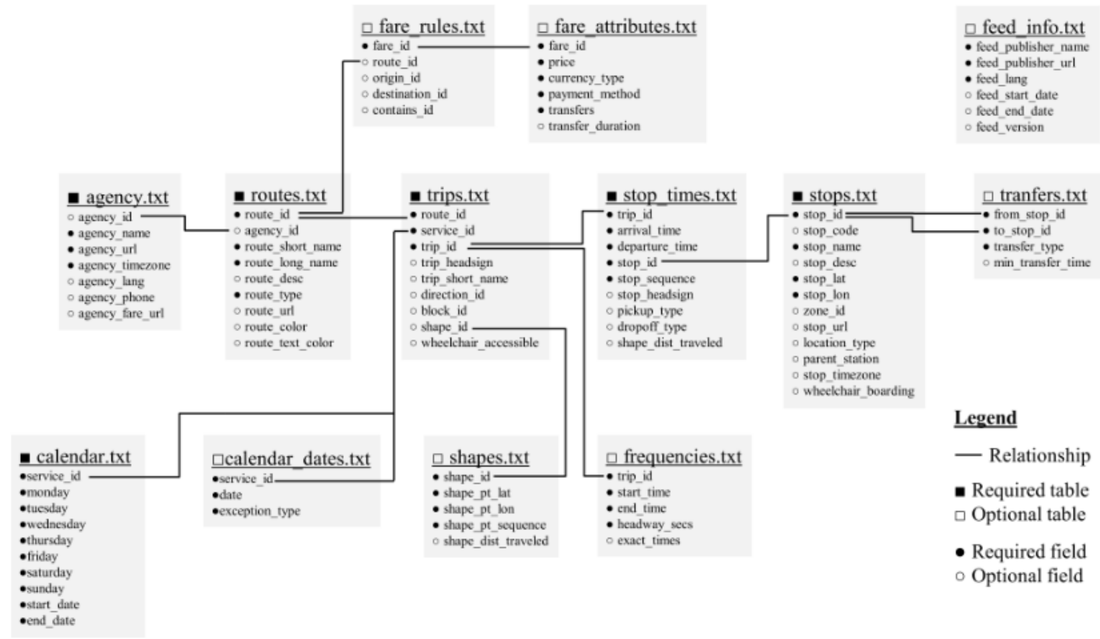
2.2.1 GTFS

The General Transit Feed Specification defines a common format for public transportation schedules and associated geographic information (GOOGLE, 2005). Google has defined publish patterns for public transit agencies to improve collaborative work between public transit agencies and developers. This standardization helps developers design applications that consume that data interoperable (MCHUGH, 2013).

This specification comprises *feeds*, a series of text files collected in a ZIP file. This set of files of a feed is called *dataset*, and each file in a dataset corresponds to an *entity* from the Figure 1 (WONG, 2013). A *record* represents a single entity, a data structure comprised of several different field values, represented in a table as a row. A *field* of one record represents a property of this entity, represented in a table, as a column. Neither all files from the dataset nor all fields of one record are required, GTFS has three kinds of *field values*: 1. *Required*, the field must be included in the dataset, and a value must be provided in that field for each record. 2. *Optional*, the field may be omitted from the dataset. 3. *Conditionally required*, the field or file is required under certain conditions outlined in the field or file description. Outside of these conditions, this field or file is optional (GOOGLE, 2005).

The GTFS has been studied in many research fields. The specification provides the Collective Transportation Network that is used to model a multimodel urban transportation network, which is a network that has at least two modes. The user has to transfer between modes (SMARZARO; DAVIS; QUINTANILHA, 2021). The multimodel urban transportation network can be understood as the composition of the street network and the infrastructure for each modal, bus and train, for instance. Furthermore, GTFS is also used for ridesharing or carpooling applications where users share a private vehicle to make the same or similar trip (ANTRIM; BARBEAU et al., 2013).

Figure 1 – GTFS data structure



Source: Wong (2013)

2.2.2 GTFS-RT

On the one hand, the GTFS provides static data of schedules, maps, and fares, on the other hand, it neither supplies vehicle positions nor trip updates nor service alerts, natively. The GTFS-RT is an extension of GTFS, allowing public transportation agencies to provide real-time updates about their fleet. Currently, it supports three types of information served via HTTP and is updated frequently. Its transmission is based on regular binary files, and there are no constraints on how frequently the feed should be updated or retrieved (GOOGLE, 2009). The types are listed as:

- **Trip updates** - delays, cancellations, changed routes.
- **Service alerts** - stop moved, unforeseen events affecting a station, route, or the entire network
- **Vehicle positions** - information about the vehicles, including location and congestion level

The delays worsen the user experience with the PTN, GTFS-RT has enabled many researchers to estimate, analyze, and mitigate delays, due to the temporal features available. One approach is to stream significant delay changes in real-time (WRONA; GRZENDA; LUCKNER, 2022), in other words, the user can be notified in real-time with schedule deviation. Another approach to mitigate delays is to collect real-time data and use the temporal series to predict delays and improve the timetable (WESSEL; WIDENER, 2017; AEMMER; RANJBARI; MACKENZIE, 2022).

GTFS and GTFS-RT specifications are important in many research fields, such as human mobility. Then, it is necessary to comprehend these two specifications to develop a framework to fulfill the GTFS-RT, which could be unavailable, with real-time data from an API combined with GTFS. Finally, our approach uses data structures alike GTFS-RT, as further discussed in Chapter 4.

2.3 Complex Networks and Graphs

In the late 1990s and early 2000s, computational models started to deal with huge datasets, which have been only increasing since then. In this scenario, complex networks appear to address this issue based on graph theory designed to model real-world networks, and these models are suitable to represent relationships and interactions between entities through link orientation and labels (ALBERT; BARABÁSI, 2002). Complex networks present compact structures, also called *small worlds*, with short distances between nodes, high levels of correlations, and self-organization (FERBER et al., 2012). The popularization World-Wide Web and its applications, chemistry, drug design, natural language processing, and recommender systems are examples of applications in which complex networks could be used. To understand a complex network and its metrics, we need to comprehend some graphs concepts.

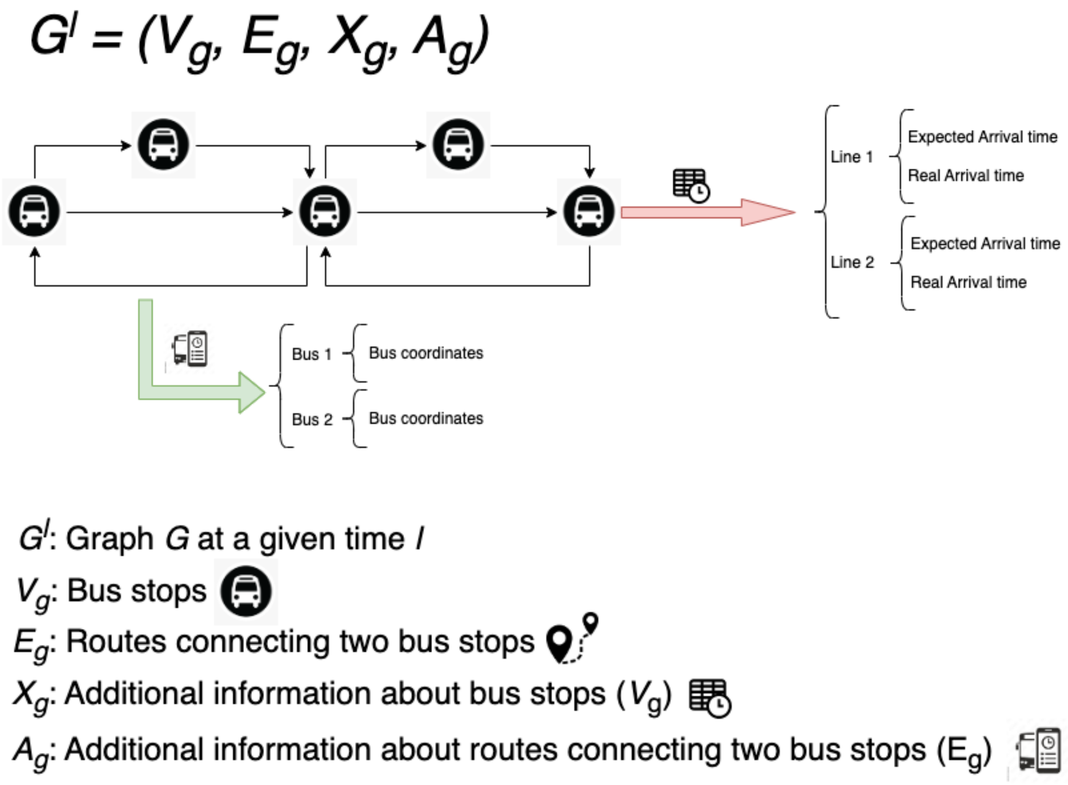
Bacciu et al. (2020) state that "a graph has a compositional nature, being a compound of atomic information pieces and a relational nature, as the links defining its structure denote relationships between the linked entities". To formally define a graph, the mathematical notation is $g = (V_g, E_g, X_g, A_g)$ where V_g is a set of *vertexes* and E_g is a set of *edges*, or *arcs* connecting a couple of nodes (BONDY; MURTY, 1976). Two more sets, X_g and A_g, represent additional information on the node set and the edges set, respectively. In other words, the nodes and edges in some applications have extra features. So, each node $u \in |V_g|$ has a unique feature vector associated therefore, each edge is associated with and feature vector $a_{uv} \in A_g$.

When couples of nodes are ordered, i.e., $E_g \subseteq \{\{u, v\}|u, v \in V_g\}$, in a graph, this graph is *directed* and its edges are *oriented*, otherwise the graph is *undirected* and its edges are *non-oriented*. In both directed and undirected graphs, a *walk* is a sequence of edges that join a sequence of vertexes, a *trail* is a walk in which all edges are distinct, and a *path* is a trail in which all nodes are distinct (WILLIAMSON, 2010).

2.3.1 Public Transportation Networks as a complex network

The public transportation network is one of the networks that is part of the lives of millions of people around the globe, but it hides a high complexity degree (FERBER et al., 2012). The PTN can be represented as a *directed* graph $G = (V_g, E_g, X_g, A_g)$, where the set of nodes V_g represents the bus stops and the set E_g represents the directed arcs that connect different bus stops. X_g and A_g are the sets of additional information about the nodes (V_g) and the arcs (E_g), respectively. Figure 2 shows G.

Figure 2 – The PTN modeled as the graph *G*



Source: The authors

X_g contains the bus arrival time on each bus stop in V_g. Which is comprehended as all bus trips that pass by any bus stop v in V_g during a day, for instance. And, A_g contains the bus entries collected between two sequenced bus stops, B_x and B_{x+1}. In other words, that is all the information provided by a bus while traveling through any arc e of E_g. The graph G is a *dynamic* graph because the routes change over the days, so the structure of this graph is not fixed. But, when we approach this graph in a defined time interval, it has a *static* topology because the bus stop locations and routes are not on the same day. The field called *service_id* from the GTFS defines the routes for a given day, so this field defines the topology of the graph and the graph can be approached in a static way for the same *service_id*, our analysis in Chapter 5 are based on this premise.

Thus, this graph can be modeled using two data groups, static data, and RT data. The first group is represented by the GTFS data which is used to generate V_g and E_g sets, since both sets do not change over the delimited time interval. The second group of data is represented by the real-time bus entries producing X_g and A_g sets, which change over the delimited time interval due to the *service_id*, meaning that the traffic will be different on Monday than Tuesday, for example.

3 RELATED WORK

In this Chapter, we present related work identifying delays and strategies to mitigate them in public transportation networks. Also, we approach the issue that GTFS-RT could be unavailable despite having GTFS and real-time data and how the literature has dealt to this issue.

Raghothama, Shreenath e Meijer (2016) use the combination of GTFS data and real-time data to analyze delays in the public transportation network of Rome and Stockholm, These cities had divergent data available, resulting in differences in the analysis used GTFS for the static data. For real-time data, Rome's data were provided by the Mobility Control Center of Roma Servizi della Mobilità, but there was no identifier to connect the real-time data and the static data. So, the link between sets was inferred by comparing the expected arrival time and the real arrival time within a threshold. The results show the average delay for a stop, the average delay for a trip, and the average delay for a trip at a stop.

Stockholm's real-time data provided by the Swedish Transportation Administration and it was easy to link with the static data, unlikely Rome. Also, the real-time data was composed of real-time updates on routes, disruptions, delays, and arrival-departure time of buses, trains, trains, and boats at each stop, so characterizing the GTFS-RT specification. Raghothama, Shreenath e Meijer (2016) used the dataset to examine three questions: 1. Are the delays in the public transport network spatially dependent? 2. What are the factors contributing to delays? 3. What methods best suit the analysis of large real-time data streams? The first question approach was to calculate the mode and standard deviation of the delays and apply a simple Moran's I test, which determines a significant spatial auto-correlation. The second was tacked using an OLS model in which the average delay is the dependent variable, and maximum speed, road types, number of lines, and number of lanes are the independent variables. Finally, regarding the third question, the authors point out that spatially aware, computationally intelligent, and machine-learning techniques to work with this kind of dataset.

We replicated the delay analysis that Raghothama, Shreenath e Meijer (2016) had done in Rome, and we discussed the three questions analyzed with Stockholm's data using Belo Horizonte data. Belo Horizonte's dataset resembles Rome's because no easy-matching identifier exists. Although, we address the issues with a different approach but also incorporating the key ideas of the algorithm used in Rome, as further discussed in

Many papers identify delays in public transportation networks using the GTFS-RT data, despite the differences in measurement and mitigation of delays, it is a common approach to model the public transportation network using graph theory. Wessel e Widener (2017) describe that a random delay along legal, physical, or social constraints are the factors that produce delays. To mitigate these factors, the planners added to the schedule a safety margin delay, called schedule padding. Furthermore, the schedule padding is given by the time required to operate on any given segment of the route for each segment, the padding is the time difference between the fastest and the average time of the remaining entries. A fixed value for schedule padding may heavily influence the transportation experience by delaying routes without needing to. Then, the takeaway is that the padding must be proportional to the random delay, which is caused by some unpredictable factors such as: traffic conditions, wrecked vehicles, or the number of red lights.

We molded the public transportation networks using graph theory as well, and we adapted Wessel e Widener (2017)'s concept of schedule padding to measure the delays and to estimate arrival-departure windows. Because the arrival-departure window of a bus to a bus stop depends on whether the bus is delayed, or on time, or ahead of schedule, which expresses a negative, neutral, positive padding to the expected arrival time, respectively.

Wessel, Allen e Farber (2017) point out that GTFS' analysis that researchers have been dealing with could be better addressed using real-time data to enrich GTFS data. Because the GTFS is based on schedules, which are projections about the services rather than observations. In other words, the research questions should take into account the events that interfere with the real composition of the network, the randomness of a random complex network. Despite a brief mention of the GTFS-RT specification, Wessel, Allen e Farber (2017) reveal one common issue with it, that is, the non-compatibility of the real-time data and the GTFS. So, they propose an algorithm based on the monitoring of vehicles in real-time as their locations are updated to unify the two datasets. After the data collection, the algorithm's following steps are: 1. Delimiting trips and blocks; 2. Spatial matching and positional error handling; 3. Determining stop times; 4. Constructing the retrospective GTFS package. Finally, using the enriched GTFS, they demonstrate the usage in Toronto.

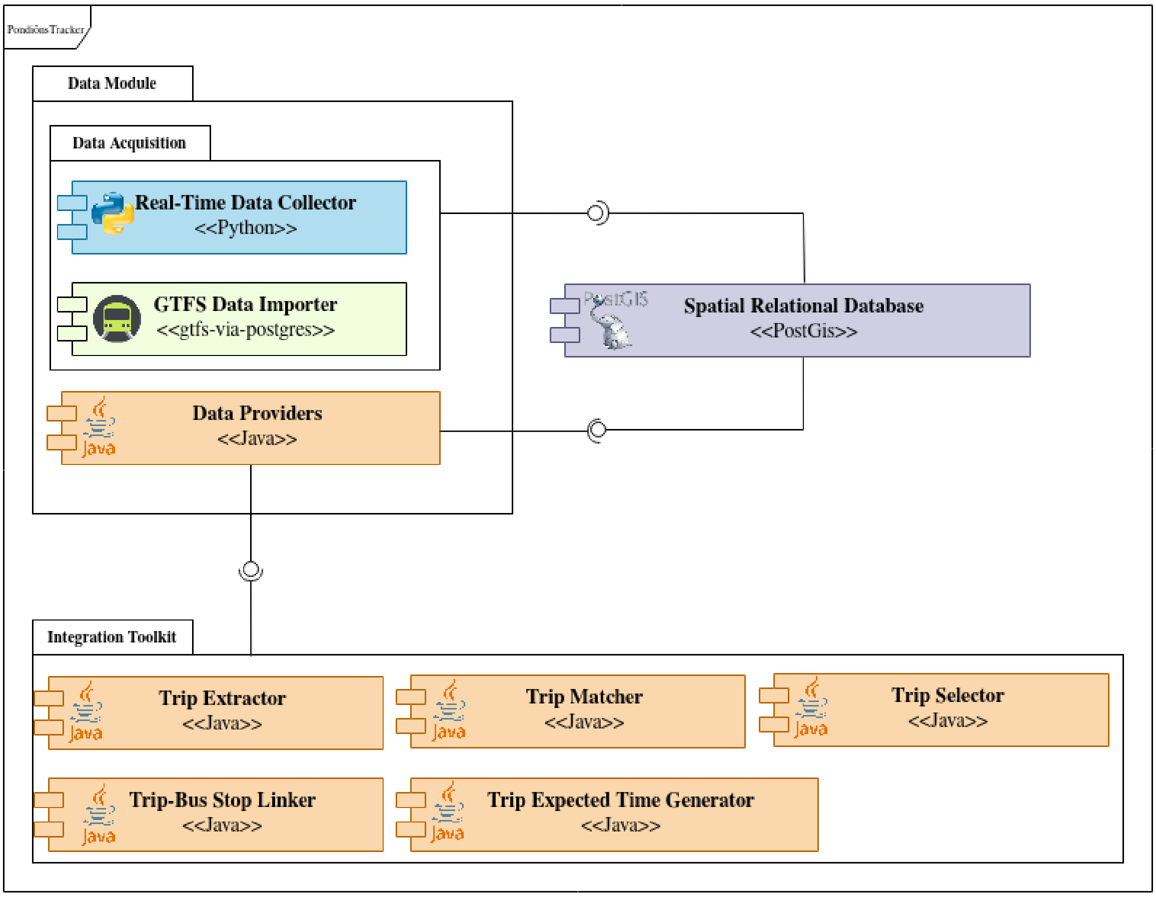
The Wessel, Allen e Farber (2017)'s algorithm is really interesting; we approach the real-time data in a very similar outline. To delimit trips, we used the vehicle's current distance en route instead of the headsign. To do spatial matching and positional error handling, they used Open Source Routing Machine's map-matching algorithms to match entries to bus stops, we opted for keeping a low-coupling architecture using the Postgis and GTFS. For determining stop times, we used the same approach of estimating time by linear interpolation from the surrounding vehicle reports, but we propose a different heuristic to choose the reports to interpolate. The architectural decisions we made and our justifications are further explored in the following chapter, Chapter 4.

Given the state-of-the-art literature discussed in this chapter, this dissertation contributes to GTFS and GTFS-RT because we present a tool to link GTFS with real-time data in cities where GTFS-RT is unavailable, allowing analysis such as delay analysis. Then, we contribute to delay analysis due to the reproduction of questions using the proposed framework.

4 PONDIÔNSTRACKER

In this Chapter, we present *PondiônsTracker**, which is a framework to enrich GTFS data with real-time data enabling *collecting* and *integrating* . The name *PondiônsTracker* is a small gag from the sonority of the expression *bus stop* when pronounced in Portuguese with the accent from Minas Gerais, and its architecture is divided into two components: the data module, and the integration module as shown in the architecture diagram from Figure 3. In the following Sections of this Chapter, we present the entities used by the components, then we deep dive into each component and finish presenting *PondiônsTracker-BH*.

Figure 3 – *PondiônsTracker*'s architecture diagram



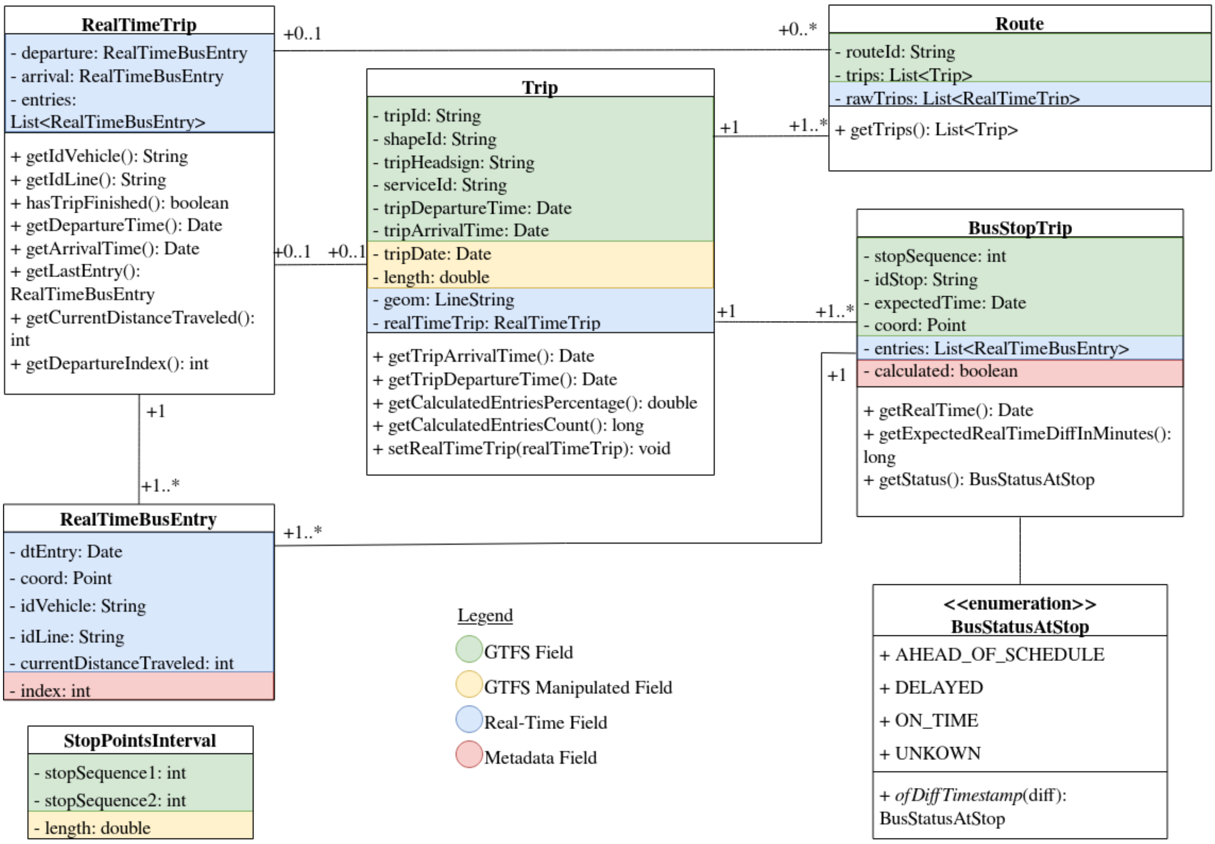
Source: The authors

*Available at `https://github.com/Pongelupe/PondionsTracker/`

Loose coupling is the key design principle of *PondiônsTracker* architecture to work with as many cities as possible, and this can be achieved by smaller software components that can be replaced or extended and still work with the other components with few adjustments, because the behaviors of the main components are defined in interfaces, so an object-oriented language as Java suits this scenario. The major external dependency is the *PostGIS*† database, which provides spatial functions and stores the data.

4.1 Entities

Figure 4 – Entities Class Diagram



Source: The authors

In this Section, we present PondiônsTracker's entities used in all components, the entities mix three types of fields: fields from the GTFS specification, fields to represent the real-time data and some auxiliary fields, Figure 4 presents the entities class diagram with the getters and setters suppressed to improve the readability. The fields highlighted in green are from the GTFS, and those highlighted in yellow are not originally from the given entity, but they are easily obtained by manipulating other fields, such as the

---
†Available at `https://postgis.net/`

Trip's *geom* and *BusStopTrip*'s *coord* which are spatial data from *shape_id*, *stop_lat* and *stop_lon*, respectively. Finally, the fields highlighted in blue are from the real-time data, and those highlighted in red were designed to store metadata produced during the components' execution.

The *Route* and *Trip* entities are defined at the GTFS specification, and we enriched both with more fields, the $rawTrips$ was added to the *Route* to link with all the related raw real-time trips collected. Four new fields were introduced to the *Trip*, two in yellow and two in blue, the yellow couple represents the spatial, in which $length$ represents the *geom* length, that is a *LineString* with the bus stops are linked by the path programmed, in meters. Furthermore, in the blue couple, $realTimeTrip$ corresponds to the real trip of a scheduled trip, which is a zero or one relationship, because one $Trip$ can have at most one related *RealTimeTrip*, but it also is unavailable, leading to a null reference. And *busStopSequece* is a *List* of all stops of this trip ordered by its $stop_sequence$, in other words, a list of *BusStopTrip*.

The *BusStopTrip* works as an entity that merges two GTFS's entities: *stop_times* and *stops*. The first brings *stop_sequence*, *stop_id*, and *departure_time* that could be null depending on the local GTFS. And, *stops* provides the spatial information achieved using the *stop_id*. The field *entries*, which is highlighted in blue, represents all the *RealTimeBusEntry* related to this stop, which may be even zero, in other words, it is a list containing every bus that was around this stop. Finally, the *calculated* field in red is a flag representing that this *BusStopTrip* has no actual entry related to it, so an artificial entry was created and associated with this stop, as further explained in Section 4.3. Finally, the method *getStatus* gets the *BusStatusAtStop* of the current *BusStopTrip*, and this operation enables the delay analysis.

Then, for every bus positioning, around or not around a bus stop, is entry from *RealTimeBusEntry*, which is the entity that summarizes all *PondiônsTracker*'s real-time required blue fields in a group of five as follows: 1. *dt_entry*: Timestamp of the occurrence; 2. *coord*: The location of the occurrence. Mainly given by lat/lon coordinates; 3. *id_vehicle*: A identifier for the vehicle executing the trip; 4. *id_line*: A identifier for the route/trip being executed; 5. *current_distance_traveled*: The current distance traveled in a trip. And a *RealTimeTrip* is achieved by ordering the entries by its *dt_entry* then group them by *id_vehicle*, which is the concept that a single bus can only travel a single trip at a time. Finally, the field in red *index* represents the position of an entry in a trip.

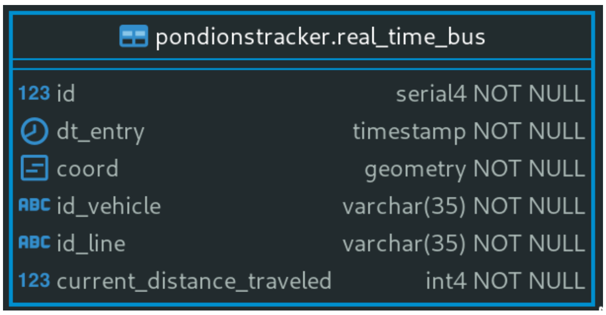
4.2 Data Module

The Data Module's main goal is to encapsulate operations of the *PostGIS* from the other components. There are three artifacts inside this module, *DataImporter*, *RealTimeDataCollector*, and *DataProviders*, the first two components encapsulate writing, and the last encapsulate reading. Also, there are two *DataProviders*, one that works with GTFS data and the other works with real-time data, *GTFSProvider* and *RealTimeProvider*, respectively.

4.2.1 GTFS Data Importer

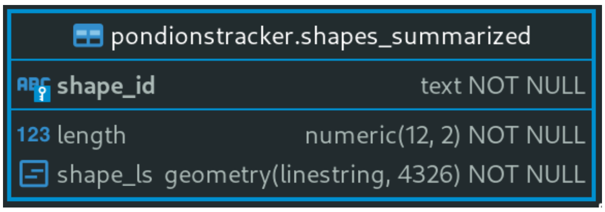
The *GTFS Data Importer* is a component that imports the GTFS data to the *Postgis* and creates the basic structure needed. So, given an instance of *Postgis* running in a cloud service or local or dockerized environment, for instance, the first step is to execute the *Data Definition Language* (DDL) required, which is defined at *schema.sql* that is a *Structured Query Language* (SQL) file located at *scripts/sql/*. *schema.sql* defines two tables, and the first is called *real_time_bus* is a table to store the real-time data. In other words, each row is an instance of a *RealTimeBusEntry*. And, for this table, there are two indexes, one is over *dt_entry* and the other over *dt_entry* combined to *id_vehicle*, they are designed to improve the performance of the queries executed by the *Data Providers*, further explained in Subsection 4.2.2. Figure 5 shows its relational model.

Figure 5 – *real_time_bus*'s relational model



Source: The authors

Figure 6 – *shapes_summarized*'s relational model



Source: The authors

Figure 6 shows the relational model of the second table created, called *shapes_summarized* and is an auxiliary table that stores the geometric type *LineString* and its length for all trips. This table is needed because the tool used to import the GTFS data does not provide this structure or a similar one. The tool used is the *gtfs-via-postgres* [‡], an open-source

[‡] Available at `https://github.com/public-transport/gtfs-via-postgres`

solution to import static data that Google recommends. It takes as input the GTFS files, which could be in the *.txt* or *.csv* format, and create a new table in the database for each entity.

Finally, the last step to fully initialize the database is to execute an SQL script that populates the table *shapes_summarized*, which is *shapes_summarized_populate.sql* and it is located in the same folder as *schema.sql*. This SQL script takes the *shapes* table grouped by the *shape_id* as input to the *PostGIS*'s function *ST_MakeLine* that produces a *LineString* from the given points.

We provide a *shell* script that executes all the steps described previously, this script is called *init.sh*, and it is in the folder *scripts*. This script takes two inputs: the first argument is the path to the folder where the GTFS's files are located, and the other argument is the schema name that *gtfs-via-postgres* is going to import to. *init.sh* uses *PostGIS*'s environment variables [§] to connect to the database, then execute three commands. The first loads *schema.sql*, the second uses *gtfs-via-postgres* to import the GTFS, and the third executes the *shapes_summarized_populate.sql*.

4.2.2 Real-Time Data Collector

The *Real-Time Data Collector* is the component responsible to collect the real-time data and insert it into *bus_real_time*, encapsulating the writing into the database. In other words, this component will incorporate the data from real-time traffic API provided by some external source into the table, such as traffic agencies. Regarding extensibility, *bus_real_time* can be enriched with more data, instant speed, and direction, for instance.

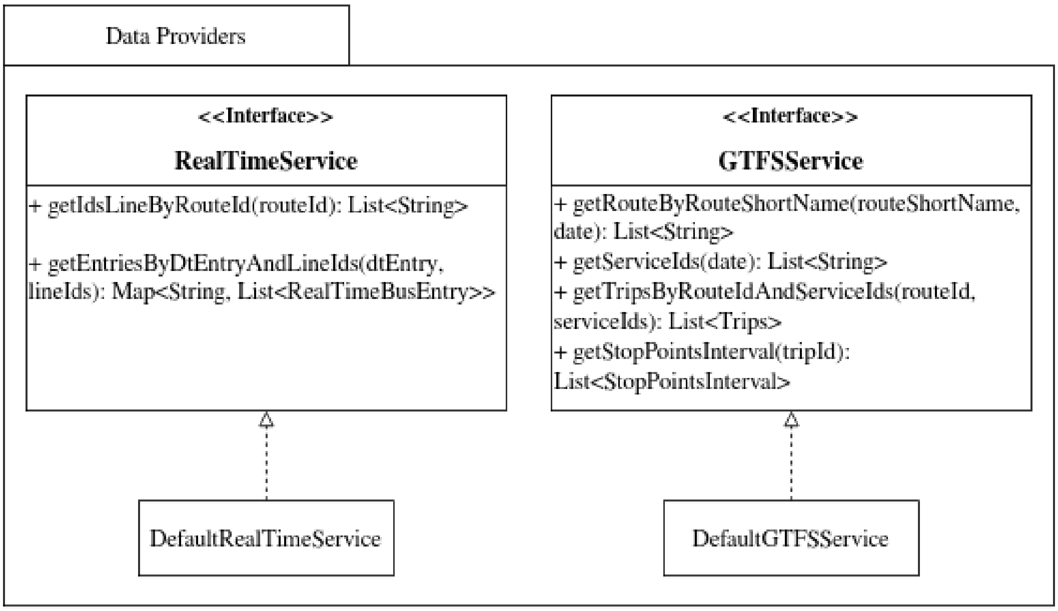
The Real-Time Data Collector may be implemented in any language since it must adapt to each real-case scenario to insert valid entries into the table because it depends on the local city provider. To illustrate a record, an API provides the following data: A bus b of line 123 is at a bus stop near a drugstore and has already traveled 3 km on its route at 4 p.m. The corresponding entry would be: 1. *dt_entry* as 16:00; 2. *coord* as the point composed by the latitude and longitude of the bus stop near a drugstore where the b is; 3. *id_vehicle* as the b's vehicle id; 4. *id_line* as 123; 5. *current_distance_traveled* as 3000, which is 3km transformed to meters. And the *id* is generated and managed by the database.

4.2.3 Data Providers

To isolate the access to the *PostGIS*, the Data Providers are designed to supply all the data required in other modules. Figure 7 represents the class diagram for

[§] Available at `https://www.postgresql.org/docs/current/libpq-envars.html`

Figure 7 – Data Providers Class Diagram



Source: The authors

these components, then it is composed of two interfaces: *RealTimeService* and *GTFSService*, which represent the real-time data and the GTFS data, respectively. Also, each of these interfaces has a default implementation, the *RealTimeService* implementation is *DefaultRealTimeService* and the *GTFSService*'s default implementation is given by *DefaultGTFSService*. The Data Providers are available at the Maven Repository and can be accessed by adding the following dependency to a *pom.xml* in a Java 17+ project.

```xml
<dependency>
  <groupId>br.pondionstracker</groupId>
  <artifactId>data-module</artifactId>
  <version>1.0.0</version>
</dependency>
```

The *DefaultRealTimeService* implements the two methods defined by the interface. The first is called *getIdsLineByRouteId*, It takes a route id as input and produces a list containing all the line identifiers related to the given route, the default implementation is to return a list of a single item, which item is the inputted route id. In other words, this method establishes the relationship between the two datasets heavily dependent on each city context. Although this method is likely to be overridden, the default implementation assumes that the RT provider will use the *route_id*.

The second method is called *getEntriesByDtEntryAndLineIds*, it takes a target date and the *id_lines* as input and produces a *java.util.Map* whose key is *id_vehicle* and value is a list of entries related to the *id_vehicle*. So, the main idea, which is also implemented by the *DefaultRealTimeService*, is to retrieve all the entries from a day of some lines ordered by *dt_entry* grouped by *id_vehicle*, the *TripExtractor* is going to take advantage of this grouping, as further approached in Subsection 4.3.1.

The *GTFSService*'s default implementation is given by *DefaultGTFSService* which implements the four methods defined. The first is called *getRouteByRouteShortName*, it takes a *route_short_name* and a target date as input and produces a *java.util.Optional* of a *Route*, the *Optional* is going to be filled if the *route_short_name* exists at the given date, empty otherwise. The default implementation fills the *trips* object using two other methods available, *getServiceIds* and *getTripsByRouteIdAndServiceIds*. The method called *getServiceIds* takes a target date as input and retrieves all the valid *service_ids* at the given date, that is, all the available services for that day of the week.

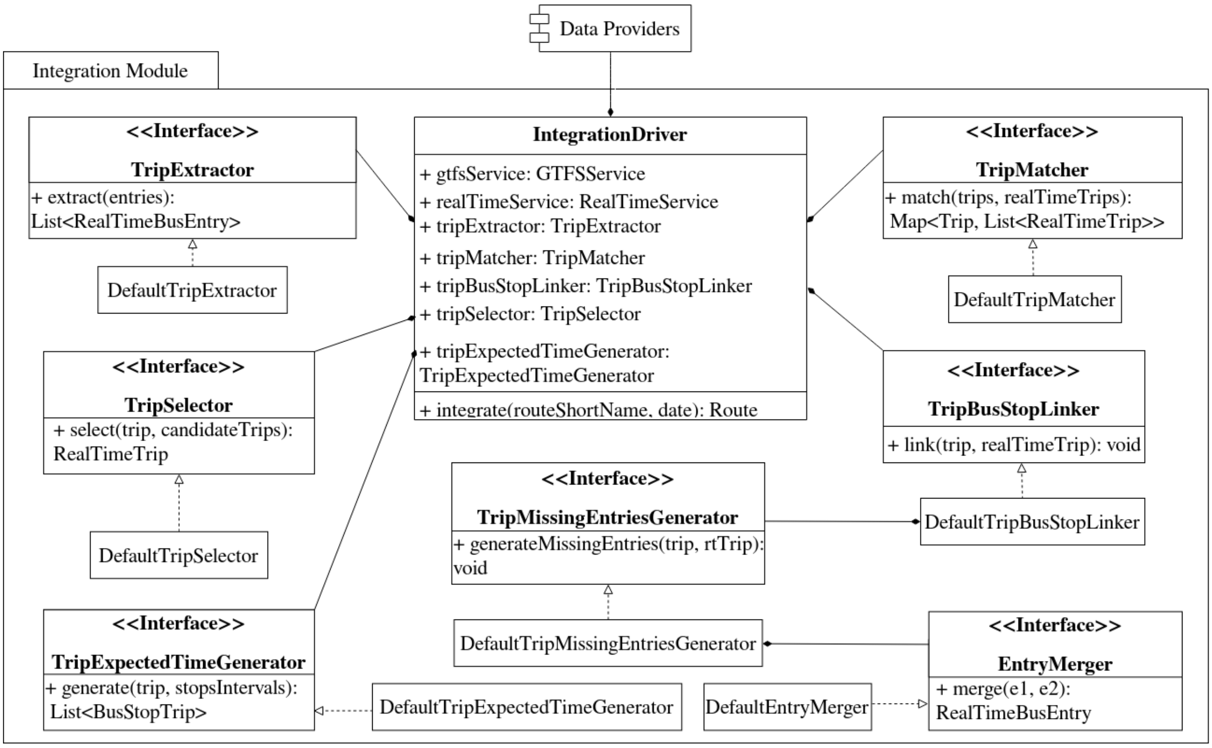
The method called *getTripsByRouteIdAndServiceIds* takes a *route_id* and a *List* of *service_id* as input and produces a *List* of trips that is the schedule of each trip of that route. The default implementation retrieves the following fields: *trip_id*, *shape_id*, *service_id*, *trip_headsign*, *shape_ls*, and *length*. Then, for each trip, the *busStopSequence* is filled with its stop sequence, stop id, expected time, and the bus stop coordinates. Finally, the last method is called *getStopPointsInterval*, it takes a *trip_id* as input and produces a *List* of *StopPointsInterval*. That is, the trip's bus stops in sequence and the distance between a couple of stops.

4.3 Integration Module

The Integration Module's main goal is to provide functions and methods to work with the transit data previously collected using the *GTFSService and RealTimeService*. So, this module exposes seven interfaces, which are *TripExtractor*, *TripMatcher*, *TripBusStopLinker*, *TripSelector*, and *TripExpectedTimeGenerator* as shown in Figure 8. In the following Subsections, we get into details of the interface contracts and its *default* implementation. In the last Subsection, we approach the *Drivers*. Finally, all the interfaces and their default implementation are available at the Maven Repository and can be accessed by adding the following dependency to a *pom.xml* in a Java 17+ project.

```xml
<dependency>
  <groupId>br.pondionstracker</groupId>
  <artifactId>integration-module</artifactId>
  <version>1.0.0</version>
</dependency>
```

Figure 8 – Integration Module Class Diagram



Source: The authors

4.3.1 Trip Extractor

The *TripExtractor* is an interface that declares one method which is called *extract* that takes a *List* of *RealTimeBusEntry* as input and produces a *List* of *RealTimeTrip*. This component's main goal is to extract the trips from entries because the data from the real-time API may not have a direct link to the GTFS, which leads to scenarios where APIs supply more data than the scheduled trips. In other words, a trip represents a *trail* in the graph. In addition, the APIs might return some unwanted trips, such as the displacement from the bus garage to a bus stop, and the other way around, the *default* implementation does not apply any strategy to filter out these trips. The implementation is defined in Algorithm 1 and depends on three premises: 1. The entries must be from the same *idVehicle*; 2. The entries must be sorted by *dtEntry* in order to preserve their chronology; 3. The *currentDistanceTraveled* can only increase for the same trip.

After postulating these premises, it is iterated over entries, then for each entry, it *currentDistanceTraveled* is compared with a temporary variable d defined on line 2. If the entry's *currentDistanceTraveled* is *lower* than d, represented by the *if* command on line 8, it implies that a trip has just finished due to the third premise at this point, it is assigned the *previousEntry* to the *arrivalEntry*, on line 19. Otherwise, when d is

Algorithm 1 *DefaultTripExtractor*'s implementation

```
trips ← []
d ← 0                                            ▷ currentDistanceTraveled
departureEntry ← null
arrivalEntry ← null
previousEntry ← null
for i ≤ entries.length; i + + do
    entry ← entries[i]
    if entry.currentDistanceTraveled ≥ d then
        if arrivalEntry ≠ null then
            trips.add(newTrip(departureEntry, arrivalEntry))
            reset temp vars to initial state
        else
            d ← entry.currentDistanceTraveled
        end if
        if departureEntry is null then
            departureEntry ← entry
        end if
    else
        arrivalEntry ← previousEntry
        d ← entry.currentDistanceTraveled
    end if
    previousEntry ← entry
end for
trips.add(newTrip(departureEntry, previousEntry))
return trips
```

greater than or equal to the entry's *currentDistanceTraveled* and the *arrivalEntry* is still null, and the entry might be the *departureEntry* or an intermediate entry. Finally, after iterating over all entries, it is added to *trips* the last trip that is composed of the variables *departureEntry* and *previousEntry*, the *previousEntry* contains the *last* entry of the List inputted, then *trips* are returned.

Algorithm 1's output is all the trips delimited by its vehicles and timestamps, but they are not linked to a route yet. The complexity of this algorithm is $O(n)$, in which that n corresponds to entries generated by a single bus in an interval, and it is necessary to iterate once over n to delimit the trips. Despite the $O(n)$ complexity, it is worth noticing that n corresponds to entries produced by a single bus in an interval, then Algorithm 1 *might* be executed over a hundred times for a single day of observation, for instance.

4.3.2 Trip Matcher

The *TripMatcher* is an interface that declares one method, which is called *match* that takes two *Lists* as input, one of *Trip* and the other of *RealTimeTrip*. Then, it produces a *Map* of all *RealTimeTrip*s related to a *Trip*. So, this component is responsible for linking the static to the real-time data, the *default* implementation tries to unify the scheduled block with the delimited trips from real-time data by comparing *Trip*'s and *RealTimeTrip*'s *departureTime*. To do so, the key idea is to iterate over all the trips from *trips*, and for each scheduled trip, all real-time trips that *departureTime* is within an interval i is reserved to the block otherwise, it is discarded.

This interval represents the trip's maximum initial shifting and is a required argument to instantiate *DefaultTripMatcher*, which is used to initialize the final field, *maxTripInitialShifting*. That is a minute interval that works to mitigate minor deviations from the schedule at the start of a trip that is likely to happen due to real-time data unpredictability. Then, *maxTripInitialShifting* works both for delayed and ahead-of-schedule lefts because it is taken into account in an absolute value, for instance, when $maxTripInitialShifting = 5$ it matches a trip that is 5 minutes delayed or 5 minutes ahead of schedule. Finally, to exactly match on-time trips, $maxTripInitialShifting = 0$.

DefaultTripMatcher's complexity is $O(rt)$, in which r is the size of *trips* and t is the size of *realTimeTrips*. r is constant because its source is the GTFS data, and t's size is unstable due to the real-time data, so $\Omega(n^2)$, or even lower when $r > t$. Despite having the lowest complexity, lower than $\Omega(n^2)$, this scenario is undesired because it represents that there are more scheduled trips than trips collected, which indicates some degree of information loss. When $r = t$, it represents the best-case scenario regarding information recovery because there is the same number of scheduled trips and trips collected, corresponding to a complexity of strictly at $\Omega(n^2)$. In most cases, $r < t$ is due to the unwanted trips, then $\Theta(rt)$.

4.3.3 Trip Selector

The *TripSelector* is an interface that declares one method, which is called *select* that takes a *Trip* and *List* of *RealTimeTrip* as input and produces a *List* of *RealTimeTrip*. This component's main goal is to *select* the *RealTimeTrip* trip that better fits the *Trip*. The default implementation selects the *RealTimeTrip*, which has the latest *departureTime* that fills the two following conditions:

1. The last entries *dtEntry* must be after the *departureTime*;

2. The trip must have traveled at least a *tripMinPercentageTraveled* percentage of the *Trip*'s *length*.

tripMinPercentageTraveled is a required argument to instantiate *DefaultTripSelector* and it represents the *RealTimeTrip*'s minimum percentage traveled of the *Trip*. In other words, it is a parameter that defines the concept of a completed trip. *DefaultTrip-Selector* has a complexity of $O(n)$ because it iterates over the candidate real-time trips and selects the trip with the latest *departureTime* that fills the preconditions defined above.

It is necessary to select the candidate trip because there are some unwanted trips, such as the displacement from the bus garage to a bus stop and the other way around. Among all trips, some unfinished trips get distinguished by not going through all bus stops on a route due to many factors, such as 1. Bus breaking down. 2. The bus' communication system is malfunctioning. 3. A bus' last entry recorded is still en route.

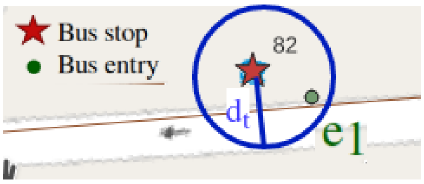
4.3.4 Trip-Bus Stop Linker

The *TripBusStopLinker* is an interface that declares one void method which is called *link* that takes a $Trip$ and a $RealTimeTrip$ as input and fills each *entries* set from $Trip$'s *busStopSequence*. In other words, this component's main objective is to link a real-time trip to the bus stop from a scheduled trip, which binds static data alongside real-time data. Although this component can be used alone, and it was designed to work with the output of the previous component to enrich the trips with some initial route information. The default implementation assumes that the $RealTimeTrip$ have been around every bus stop from their route.

Furthermore, in the context of the default implementation, the entry is considered to be around the bus stop if it is *at* or *near*. Matching the entry and the bus stops is not a simple task due to GPS position errors caused by the lack of synchronization between the datasets. These errors are pretty common and well-expected, it is unlikely that the entry's *coord* is exactly *at* the bus stop for many reasons, such as the bus broke or was simply too fast, and the dataset missed capturing the instant. These issues are reported in Wessel, Allen e Farber (2017), and we adopted their concept of a distance threshold between an entry and the bus stop for positioning, and from here, we are going to reference this threshold as d_t, so an entry within d_t is considered *at* a bus stop. Figure 9 illustrates an entry within d_t, and Figure 10 shows a couple of entries, e_1 and e_2, which are not within d_t, for instance.

So, the *DefaultTripBusStopLinker* associates each bus stop from the scheduled trip with a set of entries there is within d_t. However, an entry that is within d_t is not necessarily part of the set because a single entry might be within d_t of more than one bus stop. For instance, one avenue has a couple of bus stops on the same route which are on opposite sides, and consequently, they are in different directions of the route. One entry between these two bus stops is related to only one of them, one bus cannot be at two stops simultaneously. Thus, the key idea is to match entries and bus stops within d_t and

Figure 9 – Entries and d_t representations



Source: The authors

Figure 10 – Empty set of entries

Source: The authors

in the same direction as the route.

The default implementation key idea is to iterate over *Trip*'s *busStopSequence* and for each bus stop s_x it is searched for all the valid entries within d_t, then scanning the *RealTimeTrip*'s *entries* set. The three following premises are required to relate an entry e to a bus stop s_x:

1. an entry e can be associated with only one s_x;
2. a s_x can be related to more than one entry e;
3. a s_x can be related to zero entries.

The complexity of this operation is $O(n \log_k)$, in which n is the number of stops in a trip and k is the size of entries of the real-time trip. The first premise postulated assures the complexity of $\log_k$ when iterating through the entries set because the search for a stop s_x starts at the index of the last entry related to a previous bus stop.

After executing the previous operation, each set of entries related to each bus stop should have *at least* one entry, but sometimes, the set is empty. And this does not imply that the bus has not been around a stop on the trip, it might be only a GPS positioning error. For example, if a bus is at a certain speed, then it passes by a bus stop without stopping because there are not any boarding nor landing at that given stop. Consequently, the set of entries will be empty, and the third premise represents it. In other words, we know that the bus had been around to the stop, but there was not a single entry close enough, as e_2 from Figure 10. In this scenario, the *DefaultTripBusStopLinker* executes an extra step to generate an artificial entry at the stop with an empty set using the closest couple of entries, described by interface *TripMissingEntriesGenerator*.

The *DefaultTripMissingEntriesGenerator* design is to iterate over each bus stop s_k from *Trip*'s *busStopSequence* with no entry related to it. Then, each s_k is going to be linked to a new artificial entry e_g that is going to be generated by the merge of a couple of entries. The *EntryMerger* is the component in charge of merging two entries, the strategy behind it is to execute a linear interpolation between the two entries *coords* and other attributes, which is a simple operation whose complexity is *constant*. To illustrate, if the merged entry $e_{(n-1,n)}$ is the product of e_{n-1} and e_n then it is going be as shown in Table 1.

Table 1 – Example of merge of e_{n-1} and e_n

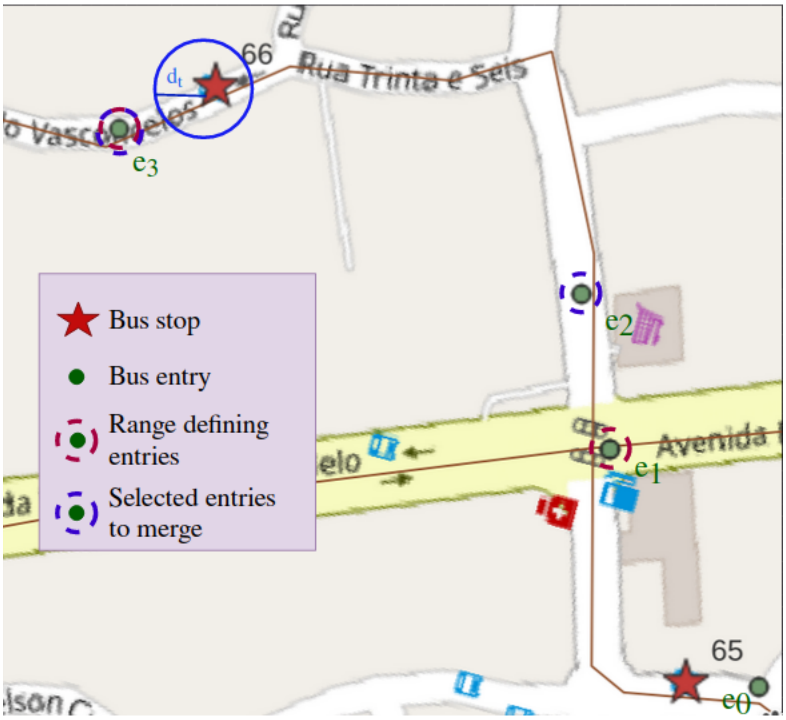
	dt_entry	*coord*	*id_vehicle*	*current_distance_traveled*
e_{n-1}	10:00	coord e_{n-1}	1	5000
e_n	10:02	coord e_n	1	5100
$e_{(n-1,n)}$	10:01	coord $e_{(n-1,n)}$	1	5050

Source: The authors

The task of selecting which couple is going to be merged is complex as well and it is performed by the interface *EntryMergerSelector*. For instance, given a bus stop s_n, in which n represents the sequence of that stop on the trip and s_n has no entry related to it, then an artificial entry e_g is going to be generated by the merge of two other entries. In the default implementation, the first step in choosing these entries to merge is to comprehend that one is before and another is after s_n to guarantee that the bus had passed by the stop, and, the fact that multiple entries may exist in between. So, the key idea is to search for the two closest entries to s_n from an interval of entries ranging from a *Lower Bound Entry* and an *Upper Bound Entry*.

On the one hand, the *Lower Bound Entry* must be the entry with the latest *dt_entry* related to any other s_x where $x < n$, that is the representation of the latest entry before s_n. On the other hand, the *Upper Bound Entry* must be the entry with the earliest *dt_entry* related to any other s_x where $x > n$, which is the representation of the earliest entry after s_n. Figure 10 represents a scenario where the interval contains only the boundary entries so that they will be merged. Figure 11 pictures a more complex scenario in which to generate an entry related to the bus stop S_{66} is needed to evaluate the following set: $E = \{e_1, e_2, e_3\}$, then choose the two nodes with the lowest distance to S_{66}, that are e_2 and e_3. In this case, the *Lower Bound Entry* is e_1, and the *Upper Bound Entry* is e_3.

Figure 11 – A more complex scenario of no entries within d_t



Source: The authors

The *DefaultTripMissingEntriesGenerator* complexity is $O(2n+c)$ because the complexity of finding out each boundary entry is $O(n)$ and the merge algorithm adds the constant c. Then, the final complexity to generate an entry associated with a bus stop is $O(2n)$, consequently, the *DefaultTripBusStopLinker* complexity is $O(n \log n + 2n)$ which is the sum of the complexity of the two steps.

4.3.5 Trip Expected Time Generator

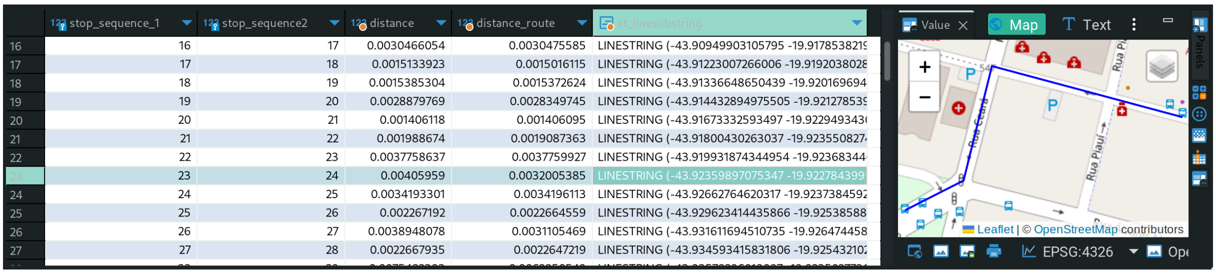
The *TripExpectedTimeGenerator* is an interface that declares one method, which is called *generate* that takes a *Trip* and *List* of *StopPointsInterval* as input and produces a *List* of *BusStopTrip*, The default implementation generates the expected stop time using the average route speed with the complexity of $O(n)$ where n is the number of stops.

The *DefaultTripExpectedTimeGenerator*'s key concept is that the stop times are incremental in the trip and are incremented at each stop. In other words, for each bus stop, we increase to the initial *arrival_time* some time interval until the point it gets to the *departure_time* at the last bus stop. This time interval referenced is calculated for a bus stop S_n using two values: the distance between S_{n-1} and S_n and the average trip speed. The latter is constant due to the length of the trip and the required *arrival_time* and *departure_time* fields, which are used to infer the total route duration, then the average trip speed is the division of the route length by the trip duration.

Calculating the distance between each S_n and S_{n+1} is not as straightforward as it looks because of the route, the distance is not as simple as a line connecting these stops, and the paths may contain turns and direction changes. An approach to this issue is the usage of Open Source Routing Machine's routing algorithms are similar to those used by Wessel, Allen e Farber (2017) for map-matching. We relaid on the GTFS data, the *shapes_summarized* SQL table, and the following *PostGIS* functions: *ST_LineLocatePoint*(), *ST_Length*() and *ST_LineSubstring*(). Our approach consists of grouping and ordering the stops by couples using the *stop_sequence*, consequently generating a route segment connecting each couple stop whose distance is used to calculate the time needed to travel this segment.

Figure 12 represents a bus stop couple for each record in which the first two columns indicate origin and destination stops, and the third and fourth columns are the linear distance and trip distance, respectively. For instance, the highlighted record shows the route segment from bus stop 23 to 24, whose linear distance is around 400 meters and the trip distance is 320 meters, so there is an 80-meter difference.

Figure 12 – Example of bus stop couples and their distance



	stop_sequence_1	stop_sequence2	distance	distance_route	st_linesubstring
16	16	17	0.0030466054	0.0030475585	LINESTRING (-43.90949903105795 -19.9178538219
17	17	18	0.0015133923	0.0015016115	LINESTRING (-43.91223007266006 -19.9192038028
18	18	19	0.0015385304	0.0015372624	LINESTRING (-43.91336648650439 -19.920169694
19	19	20	0.0028879769	0.0028349745	LINESTRING (-43.914432894975505 -19.921278539
20	20	21	0.001406118	0.001406095	LINESTRING (-43.91673332593497 -19.922949343
21	21	22	0.001988674	0.0019087363	LINESTRING (-43.91800430263037 -19.923550827
22	22	23	0.0037758637	0.0037759927	LINESTRING (-43.91993187434954 -19.92368344
23	23	24	0.00405959	0.0032005385	LINESTRING (-43.92359897075347 -19.922784399
24	24	25	0.0034193301	0.0034196113	LINESTRING (-43.92662764620317 -19.9237384592
25	25	26	0.002267192	0.0022664559	LINESTRING (-43.929623414435866 -19.92538588
26	26	27	0.0038948078	0.0031105469	LINESTRING (-43.931611694510735 -19.926474458
27	27	28	0.0022667935	0.0022647219	LINESTRING (-43.934593415831806 -19.92543210

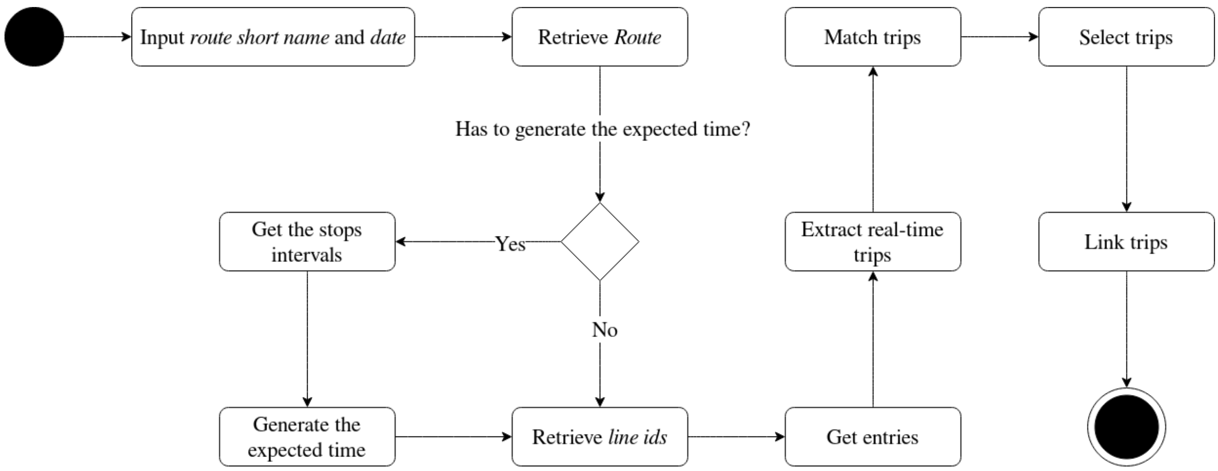
Source: The authors

Finally, given the building blocks, *Route Expected Time Generator* takes as input routes information to calculate the average speed and route duration, and a list of records such as Figure 12, which we iterate through leading to the complexity of $O(n)$. For each record, the route initial *arrival_time* is increased with the time needed to travel from a couple of bus stops at the average speed of the route. In which *could* cause minor deviations to the original *departure_time* due to the precision of the calculus executed.

4.3.6 Integration Driver

The *Integration Driver* is the component that is composed, directly or indirectly, by the other components discussed in this Section to drive the execution from simple inputs as the *route_short_name* and a target date to produce a *Route* instance completely filled. This component provides a default interaction not only between the *Integration Module* components but also to the *Data Providers* from the *Data Module*. Despite defining how the components interact, the *Integration Driver* depends on the abstraction rather than the implementation, enabling the user to adapt the components to the particular scenario. Also, it uses the default implementation with default values for each component that is not overwritten by the user building a new instance.

Figure 13 – Integration Driver Activity Diagram



Source: The authors

Figure 13 shows the activity diagram of the *integrate* method, which starts manipulating the static data, then the real-time data, and finally, linking the two datasets. Also, Figure 13 also provides a high level point-of-view of the process to connect the datasets .The first step is to retrieve the *Route* which is provided by the *GTFSService getRouteByRouteShortName* method. Next, if the expected time at each bus stop for every *Trips* of that *Route* is missing, then, before proceeding, they are generated using the *TripExpectedTimeGenerator*. It takes the *stopsIntervals* as input provided by the *GTFSService getStopPointsInterval* method. This scenario is due to the fact that the *arrival_time* and *departure_time* fields, from GTFS's *stop_time* ¶ entity is required just for the first and last stop of the trip.

After manipulating the static data, the next step starts by getting the *idsLines* using the method *getIdsLineByRouteId* from the *RealTimeService*. Which is used as input to get all entries from a day grouped by its *idVehicle* using the method *getEntriesByDtEntryAndLineIds*. In sequence, for each vehicle, the trips are extracted in *parallel* using the *TripExtractor*, then collected into a single List, called *realtimeTrips*.

¶Available at `https://developers.google.com/transit/gtfs/reference#stop_timestxt`

Finally, *Integration Driver* links the two datasets by matching the *Route*'s *Trips* with the real-time trips previously extracted using *TripMatcher*'s *match* method, that uses a 5-minute interval, so $maxTripInitialShifting = 5$. After matching, it is required to select the most suitable real-time trip corresponding to a scheduled trip, which is executed by the *TripSelector*, and the default implementation defines that a valid trip must have at least 85% of its trajectory traveled. In other words, $tripMinPercentageTraveled = 0.85$. Then, the *TripBusStopLinker* links the bus stops with their respective entries for each scheduled trip with a real-time trip associated with, the Driver defines a 50-meter range as the default value for $distanceThreshold$. Then, the *Route* is returned.

4.4 PondiônsTracker-BH

In this Section, we present *PondiônsTracker-BH*[‖] that is a *PondiônsTracker*'s specialization created to deal with Belo Horizonte's PTN particularities. So, we have implemented our own *Real-Time Data collector*, and we have overwritten the method *getIdsLineByRouteId* from the *RealTimeService*.

4.4.1 Real-Time Data Collector

Belo Horizonte has a scenario similar to Rome's described in Raghothama, Shreenath e Meijer (2016), in which different agencies provide the GTFS and real-time data. The GTFS is published by the *BHTrans*[**], which is the local agency responsible for urban mobility planning. And the real-time data is provided by *Transfacil*[††] which administrates the local bus services. So, we developed our *Real-Time Data Collector* in Python 3.11 to collect real-time data from Belo Horizonte's API.

All entries collected were stored at *real_time_bus* SQL table, but the API has its own fields, which were translated to insert into the table. The API provides nine fields. Table 2 describes each field and its translation to *real_time_bus*'s columns. Despite representing the *id_line* in the database, the NL field is not a straightforward identifier to any GTFS entity, in other words, all bus entries retrieved from the API cannot be *directly* related to any trip or route. This scenario leads us to override *RealTimeService*'s *getIdsLineByRouteId* method, as further discussed in the following Subsection.

‖ Available at `https://github.com/Pongelupe/PondionsTracker-BH`

** Available at `https://dados.pbh.gov.br/dataset/gtfs-estatico-do-sistema-convencional`

†† Available at `https://dados.pbh.gov.br/dataset/tempo_real_onibus_-_coordenada`

Table 2 – Fields From The Real-Time API

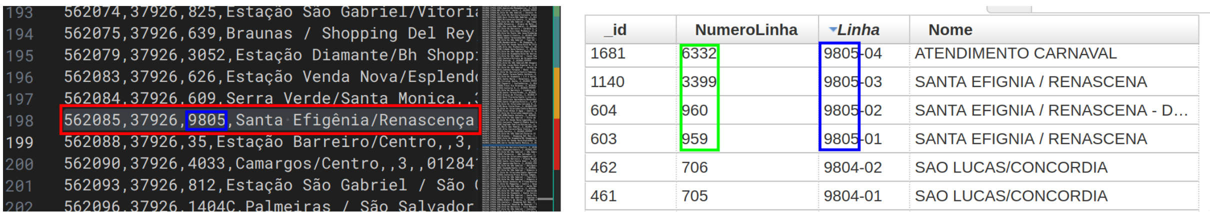
Field name	Description	*real_time_bus* related column
EV	Event code	-
HR	Timestamp	*dt_entry*
LT	WGS84 Latitude	*coord*
LG	WGS84 Longitude	*coord*
NV	Id vehicle	*id_vehicle*
VL	Instant speed	-
NL	Id line	*id_line*
DG	Vehicle's direction	-
SV	Trip way	-
DT	Distance displaced	*current_distance_traveled*

Source: The authors

4.4.2 Belo Horizonte's RealTimeService

We created *BHRealTimeService* to override *getIdsLineByRouteId* method and use the default implementation for the other methods, then *BHRealTimeService extends DefaultRealTimeService*. So, our implementation has to convert the *route_id* using a *Comma-Separated Values* (CSV) conversion file [‡‡] with 4 fields: *_id*, *NumeroLinha*, *Linha* e *Nome*.

Figure 14 – Belo Horizonte's GTFS and RT data



Source: The authors

Figure 14 shows Belo Horizontes's GTFS Routes on the left and the conversion table from the real-time data on the right. The first observation is that the column $NumeroLinha$'s domain is the NL, then $NumeroLinha = NL = id\_line$. The column $Linha$ resembles GTFS's *trip* entity and *routes* relationship because it is also a *one-to-many* relationship, in which a single bus line has multiple $Linha$ records. For instance, the real-time API supplies an entry e that its NL is 959, which value corresponds to an

[‡‡] Available at `https://dados.pbh.gov.br/dataset/tempo_real_onibus_-_coordenada/resource/150bddd0-9a2c-4731-ade9-54aa56717fb6`

image from the domain of the column $NumeroLinha$ on the conversion table. Then, 959 is going to be converted to $9805 - 01$, consequently $id\_route = 562085$.

To illustrate the *one-to-many* relationship, in Figure 14 the route highlighted in red whose $route\_id$ is 562085 has four related $NumeroLinha$ values, the group highlighted in green from the column $NumeroLinha$ on the conversion table. Because the group highlighted in blue has the $route\_short\_name$ as the prefix. In this scenario, *BHRealTimeService*'s *getIdsLineByRouteId* is going to return the group in green for $route\_id = 562085$, from the route highlighted in red.

5 RESULTS

In this Chapter, we use Belo Horizonte as a study case to validate *PondiônsTracker* using *PondiônsTracker-BH*. We used the GTFS file published on July 23^{th} 2023 and collected data from the real-time API for eleven days straight, from 29-07-2023 to 08-08-2023. During this period, our script was executed every minute, summarizing over 246 million entries representing almost 30 Gigabytes, Table 3 presents the entries for each day collected.

Table 3 – Workload Overview

Date	Day-of-Week	Entries
29-07-23	Saturday	22,319,765
30-07-23	Sunday	22,635,117
31-07-23	Monday	22,583,380
01-08-23	Tuesday	22,432,739
02-08-23	Wednesday	21,970,073
03-08-23	Thursday	22,050,579
04-08-23	Friday	22,402,865
05-08-23	Saturday	22,642,955
06-08-23	Sunday	22,786,254
07-08-23	Monday	22,109,606
08-08-23	Tuesday	22,405,222
Total	-	**246,338,555**

Source: The authors

We also present an analysis of the data collected, which is based on the output of *PondiônsTracker-BH*'s *IntegrationDriver* for *all* routes scheduled in the GTFS during the observation period. Firstly, we compare the expected with the real scenarios of Belo Horizonte's PTN over the observed period, and then we deepen into out-of-schedule incidents, especially delays. The following analysis are divided into three groups: Weekdays, Saturday and Sunday. This happens due to the GTFS *service_id* field, as previously discussed in Chapter 2.

5.1 Schedule Analysis

Table 4 shows that every observed day have 294 *Routes* planned. For weekdays are planned $22,774$ *Trips*, $14,100$ for Saturdays and $9,133$ for Sundays. Table 4 shows the schedule-filled percentage of trips filled with real-time trips during the observation period. This translates to the percentage of trips planned by the trips collected and matched from the real-time API.

Table 4 – Belo Horizonte's PTN

Date	Routes	Scheduled Trips	Matched Trips	%
29-07-23	294	14,100	11,424	81.02%
30-07-23	294	9,133	7,594	83.14%
31-07-23	294	22,774	17,184	75.45%
01-08-23	294	22,774	16,948	74.42%
02-08-23	294	22,774	16,914	74.27%
03-08-23	294	22,774	16,973	74.53%
04-08-23	294	22,774	16,854	74.01%
05-08-23	294	14,100	11,372	80.65%
06-08-23	294	9,133	7,679	84.08%
07-08-23	294	22,774	16,849	73.98%
08-08-23	294	22,774	16,837	73.93%
Total	**3,234**	**205,884**	**156,628**	**76.08%**

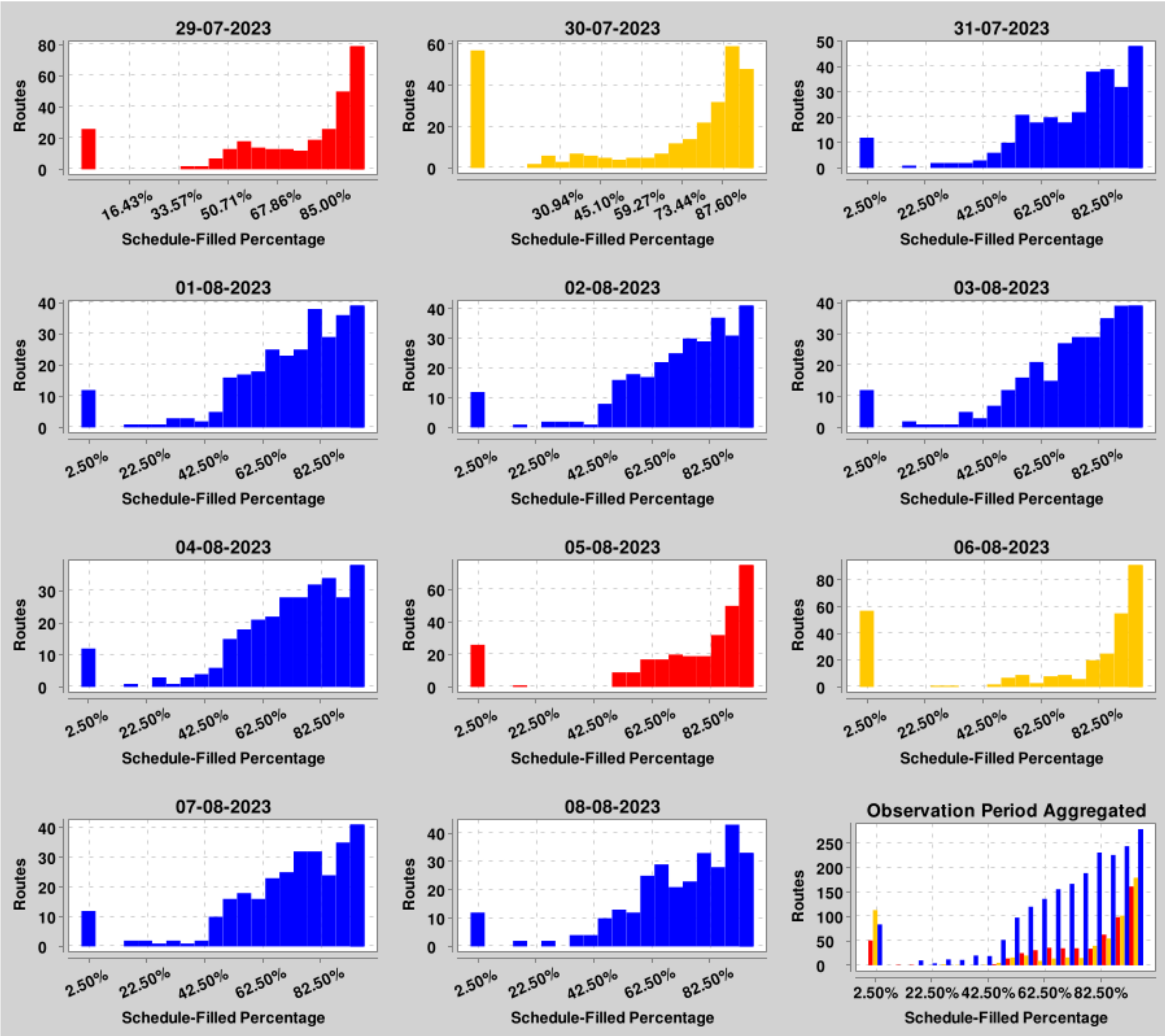
Source: The authors

The schedule-filled percentage values correspond to *all* the $3,234$ routes scheduled, and it has an average for all the period of 76.08%, but this ratio is *indirectly proportional* to the number of scheduled trips. Weekdays have the biggest number of $159,418$ trips defined and the lowest average for matched trips of $74,37\%$, comprised by 31-07, 01-08, 02-08, 03-08, 04-08, 07-08 and 08-08. Saturday's average is 80.84% with $28,200$ scheduled trips, comprised of days 29-07 and 05-08. Finally, Sunday's average is 83.61% with $18,266$ scheduled trips comprised of days 30-07 and 06-08.

Figure 15 shows the histograms of the schedule-filled percentage distributed throughout each day, and the last histogram presents the aggregated distribution for the observation period. The red, yellow, and blue bars represent Saturday, Sunday, and Weekdays. All histograms present that the schedule is not uniformly distributed, and despite the averages shown in Table 4, for each day, the most considerable individual frequencies are associated with the schedule-filled percentage of at least 82.50%, as shown in the aggregated histogram.

All histograms show the existence of considerable sets of routes whose schedule-filled percentage is *at most* 42.50%, and, especially, routes with near 0 routes reported. These sets are likely caused by the unpredictable and randomness of the network, leading to unwanted and unfinished trips, as discussed in the previous Chapter. Which decreases the averages and points to a high standard deviation, representing that the schedule is

Figure 15 – Histograms Schedule-Filled Percentage per Routes



Source: The authors

not uniformly distributed. Then, plenty of routes are completed (100%) or at least near 82.50%, despite some associated with lower percentages.

Regarding the real-time API, there are 4,095 collected trips from 63 different routes which were not defined at Belo Horizonte's GTFS. In other words, the API provided 4,095 trips with a valid *NV* unscheduled trips. Table 5 shows the top 5 routes with the most unscheduled trips. For instance, the route 82 has a high schedule-filled rate with real-time trips despite having the most unscheduled trips. On Sundays, the GTFS does not schedule any trip for route 82, but the API provided entries regarding this route twice during the period observed. The first time with 147,897 entries leading to 505 trips collected and the second time with 149,422 entries leading to 551 trips collected , then once added gets to the 1,056 shown in Table 5.

Table 5 – Top 5 Routes With The Most Unscheduled Trips

Route	Unscheduled Trips
82 - Estação São Gabriel / Savassi Via Hospitais	1,056
61 - Estação Venda Nova/Centro-Direta	791
52 - Estação Pampulha / Avenida Antonio Carlos	612
50 - Estação Pampulha / Centro - Direta	309
85 - Estação São Gabriel / Centro Via Floresta	257

Source: The authors

5.2 Delay Analysis

To analyze the delays in the PTN, firstly, we take a look at the delays on a global scale of the network, and for the context of this section, to be *on-time* denotes that the expected time is equal to the real-time within a maximum fifty-nine-seconds span, so a *delay* is a time after the expected and a *ahead-of-schedule* is a time before the expected, with equal or greater than one minute. In this context, Table 6 presents data over the matched trips aggregated by the day of the week, in which the trips entirely out of schedule represent the most extensive set of trips. The definition of this group is that every trip does not have a single entry on time. In other words, for these trips, the buses were either delayed or ahead of schedule for all expected times at every bus stop.

Table 6 – Delays detailed in whole PTN scale

	Weekday	Saturday	Sunday
Total trips matched	118,559	22,796	15,273
Trips entirely out of schedule	60,244	10,899	7,148
Trips with departure or arrival on time	39,403	8,731	5,988
Trips with departure and arrival on time	324	95	56
Trips entirely on time	1	2	1

Source: The authors

Furthermore, Table's 6 second and third most extensive sets are related to trips that are on-time at the beginning and the end of the trip. The third group represents if the trip is on time on both ends, and it is contained by the second group, which requires at least one end on time. The second group also contains the last group, a particular case of the bus being on time for *all* bus stops in the trip, which occurred only four times during the observation period for the same route, coincidentally. The route is *331 - Estação Barreiro/Conjunto Antonio Teixeira Dias Via Upa*, which has 32 bus stops, representing a length of 8,948.92 meters, almost 9 kilometers, and the four expected and real date-times are listed as follows:

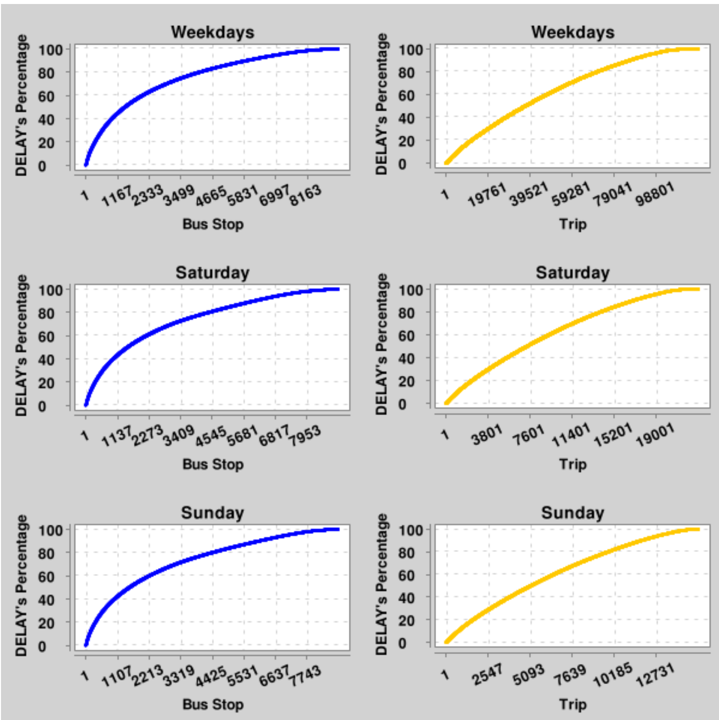
1. Expected departure and arrival: Jul. 29 15:30:00 - 15:56:27
   Real departure and arrival: Jul. 29 15:30:03 - 15:57:03;

2. Expected departure and arrival: Jul. 30 08:20:00 - 08:46:27
   Real departure and arrival: Jul. 30 08:20:31 - 08:46:15;

3. Expected departure and arrival: Aug. 04 05:40:00 - 06:06:27
   Real departure and arrival: Aug. 04 05:40:30 - 06:06:00;

4. Expected departure and arrival: Aug. 05 17:10:00 - 17:36:27
   Real departure and arrival: Aug. 05 17:10:45 - 17:36:49.

Figure 16 – $\boldsymbol{DELAY}$s Distribution: Bus Stop and Trip

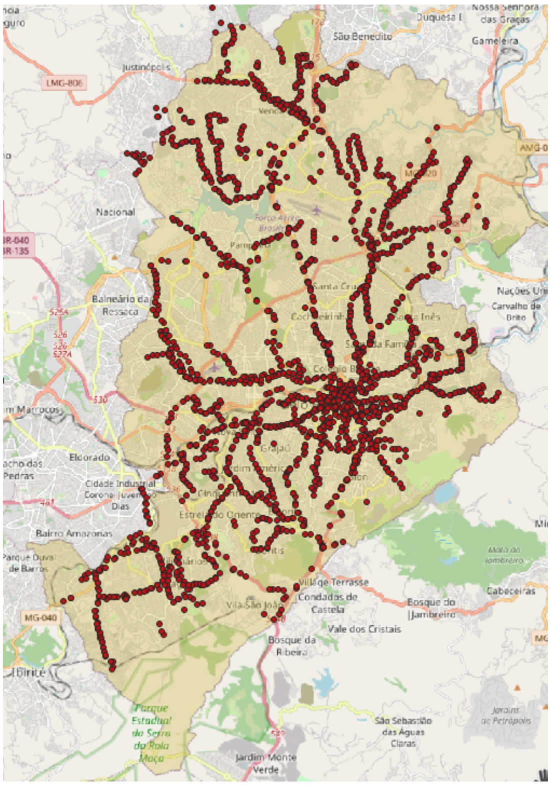


Source: The authors

Until this point, our analysis has not taken into account the impact of each distinct status in the PTN, and for weekdays, the statuses are divided as: 1. DELAY representing 89.8%; 2. AHEAD_OF_SCHEDULE representing 6.9%; 3. ON_TIME representing 3.3%. The predominance of $DELAYED$ in the PTN does not imply that the network is not working nor completely stopped because the $DELAYED$s are *log-normal* distributed between the bus stops and trips, as shown in Figure 16. So, this leads to the scenario where few bus stops and trips cause more delays. For instance, for all delays reported for weekdays, 200 trips represent 14.05%, and 200 bus stops 73.75%. This distribution also occurs on Saturdays and Sundays, as shown in Figure 16.

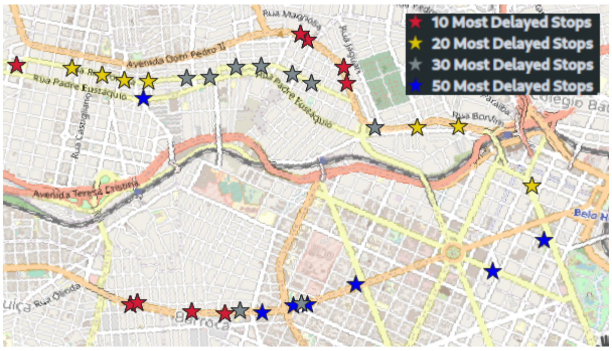
The distribution indicates a small set of bus stops is responsible for most delays, but it does not imply any spatial relationships between this set. Figure 17 shows 2, 115 out of 9, 309 most delayed bus stops, representing around 60% of the total delay and only 22.72% of the bus stops. It is observable that some main avenues and pathways are contemplated, such as the following: 1. *Avenida Dom Pedro II*; 2. *Rua Padre Eustáquio*; 3. *Avenida Amazonas*.

Figure 17 – 300 Most Delayed Stops



Source: The authors

Figure 18 – Fragment of the 50 Most Delayed Stops



Source: The authors

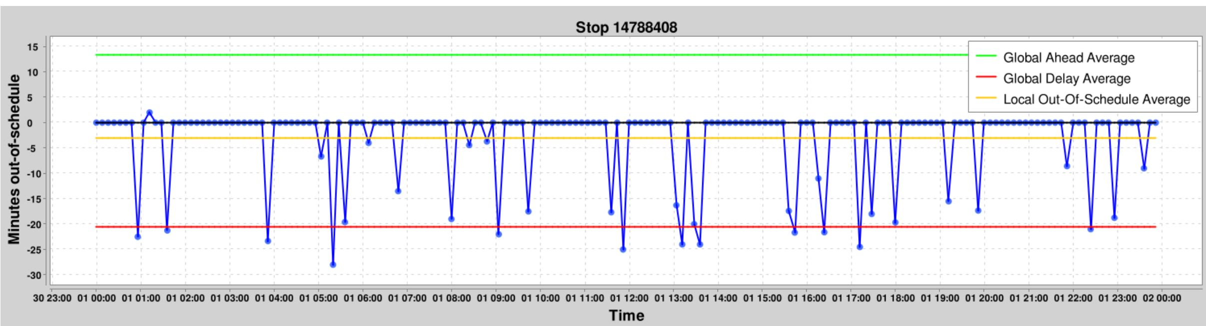
These three pathways are represented in Figure 18, which shows a fragment of Belo Horizonte's PTN and indicates the 50 most delayed stops divided into groups regarding the number of delays. So, the stops in red accumulate the most delays, and the group in blue gathers the fewest for this figure, yet these stops represent 4.58% of the network. Also, Figure 18 reveals the spatial relationships between these stops because they are on the same pathway. For instance, the *Avenida Amazonas*, the avenue located in the lower part of the figure, is one of the most critical avenues in the PTN, containing dozens of stops throughout its 8.97 kilometers, *Avenida Amazonas* has 10 out of the 50 most delayed stops. As follows, we list the 5 most delayed stops for weekdays during the observation period:

1. #14793268 - *Avenida Dom Pedro II 1520* with $7,309$ delays reported at;
2. #14791617 - *Avenida Amazonas 7309* with $7,009$ delays reported at;
3. #14790997 - *Avenida Dom Pedro II 1980* with $6,692$ delays reported at;
4. #14784438 - *Avenida Sinfronio Brochado 773* with $6,528$ delays reported at;
5. #14788981 - *Avenida Amazonas 3410* with $6,272$ delays reported at.

Despite pointing to the spatial relationship between the stops, Figures 17 and 18 fail to provide a temporal association. This is relevant because the PTN is dynamic, not a framed snapshot, so bus stops may be physically close and unaffected by each other, and for instance, a couple of bus stops on opposite sides of the same avenue. Also, the delays occur sparsely during the day, and not all delays are caused by the same causes. For illustration, a delay caused by a flat tire is different from one caused by a traffic jam, and a delay reported in the morning is unlikely to affect another from hours later, and so on.

Figure 19 represents the stop whose *stop_id* is #14788408, it is located at *Rua Indiana 135*, and is a bus stop that is not on a busy pathway nor has significant delays to the PTN. But this stop exemplifies the behavior of the delays over the hours of the day. The $y - axis$ represents the average minutes of all out-of-schedule statuses at that stop grouped by 8-minute intervals, in which negative values represent delays and positive ahead-of-schedule, and the $x - axis$ is the time of the day. Also, the following three constants are displayed:

Figure 19 – Minutes out of schedule of bus stops #14788408 distributed throughout the day



Source: The authors

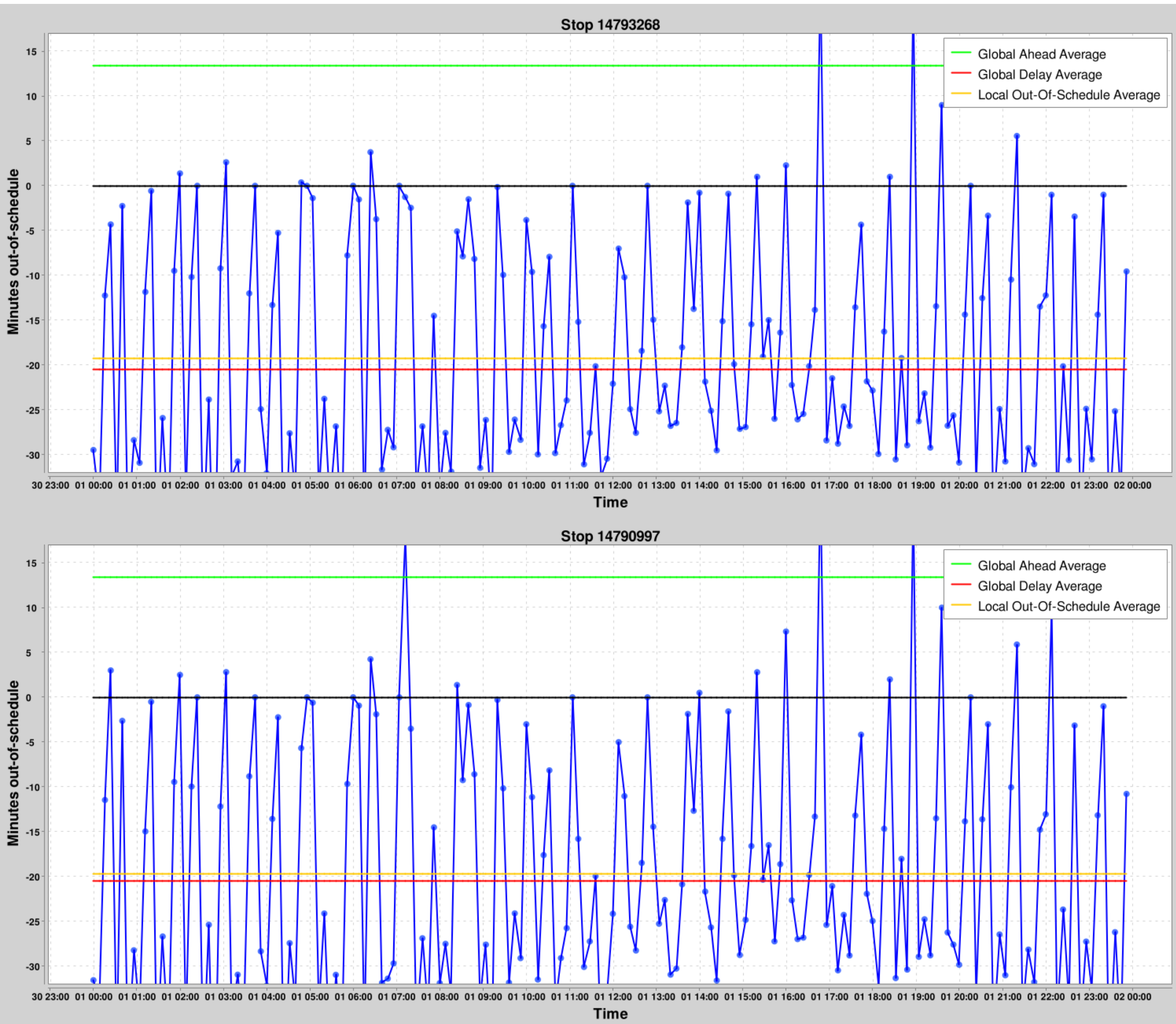
1. *Global Ahead Average*: 13.42 minutes that represents the average of ahead-of-schedule in the PTN in green;

2. *Global Delay Average*: 20.49 minutes that represents the average of delay in the PTN in red;

3. *Local Out-Of-Schedule Average*: The average number of minutes out-of-schedule for that given bus stop in yellow.

For instance, using stop #14788408, each blue dot in Figure 19 denotes the average minutes out-of-schedule for this stop in an instant in time. So, around 1:15 A.M., all the buses heading to this stop had an average of 2 minutes ahead of schedule, and around

Figure 20 – Minutes out of schedule of a couple of bus stops distributed throughout the day



Source: The authors

21:50 A.M., all the buses heading to this stop had an average of 9 minutes delayed. Since the green and the red constants are global averages, they are unreliable for analyzing the delays due to the *log-normal* distribution, but the yellow gives a more reliable insight to create a time-window interval, so for this stop, on average, the buses are delayed 3 minutes.

The stops #14793268 and #14790997 are the first and third most delayed in the PTN, respectively. These stops are 462 meters from each other on the same avenue, *Avenida Pedro II*, and share 2,590 common trips, so they are spatially related. Furthermore, they also have a temporal relationship, and Figure 20 works similarly to Figure 19 and shows these stops on weekdays. Observing Figure 20, the similarities of delay distribution in these stops demonstrate the temporal relationship because both stops have similar frequencies of all statuses simultaneously: on time, ahead of schedule, and delayed. For instance, the early morning period from 4:00 A.M. to 7 A.M. is practically equal with periods of intense delays, while the afternoon period starting at 5:00 P.M. reveals for both stops trips ahead of schedule. Also, these stops have a close *Local Out-Of-Schedule Average*, which is 19.29 minutes for #14793268 and 19.68 for minutes #14790997, both are delays.

5.3 Comparison Between Generated and Real Data

The analysis shown in the two previous sections was only possible because Belo Horizonte's GTFS defines the expected time for all bus stops on every trip. Belo Horizonte's GTFS provides this data despite not being required fields. As discussed in the previous Chapter, the *Trip Expected Time Generator* generates the expected times when missing. In this section, we executed this component with Belo Horizonte's data and compared the expected times generated with those defined at the GTFS. Table 7 displays the same data presented in Table 6 but increasing it with the generated expected times.

Table 7 shows that our generated data has some similarities and differences to the provided data and using the generated data, Weekdays, Saturdays, and Sundays have fewer trips entirely out of schedule, 3.99%, 3.69%, and 5.81%, respectively. On the one hand, this decrease points to a redistribution of the status ON_TIME throughout the network. On the other hand, this redistribution affected the four trips entirely on time, which are missing. Another point is that for all cases, except one, the generated schedule has increased the number of trips with departure and/or arrival on time, and this occurs due to the minor alterations to the *departure_time* field caused by the *DefaultTripExpectedTimeGenerator*, which were used. Saturday's trips with departure or arrival on time is the scenario that the generated data was outperformed by 0.09%, virtually, they performed equally.

Table 7 – Delays detailed in whole PTN scale with generated expected times

		GTFS	Generated
Weekday	**Total trips matched**	118,559	118,559
	Trips entirely out of schedule	60,244	57,843
	Trips with departure or arrival on time	39,403	39,526
	Trips with departure and arrival on time	324	393
	Trips entirely on time	1	0
Saturday	**Total trips matched**	22,796	22,796
	Trips entirely out of schedule	10,899	10,497
	Trips with departure or arrival on time	8,731	8,723
	Trips with departure and arrival on time	95	130
	Trips entirely on time	2	0
Sunday	**Total trips matched**	15,273	15,273
	Trips entirely out of schedule	7,148	6,733
	Trips with departure or arrival on time	5,988	6,022
	Trips with departure and arrival on time	56	73
	Trips entirely on time	1	0

Source: The authors

Furthermore, Table 8 presents statuses throughout the bus stops for the GTFS and the expected times generated. For both scenarios, the $DELAYED$ status is predominant in the network, followed by the $AHEAD\_OF\_SCHEDULE$ and ON_TIME, respectively. Also, Table 8 shows an expressively increase in $AHEAD\_OF\_SCHEDULE$, and an expressively decrease for $DELAYED$, and a minor decrease for ON_TIME when using the generated data. Despite the $DELAYED$ status having the most occurrences, it does not denote that the PTN is completely stopped or delayed due to the *log-normal* distribution of this status, as discussed in the previous Section.

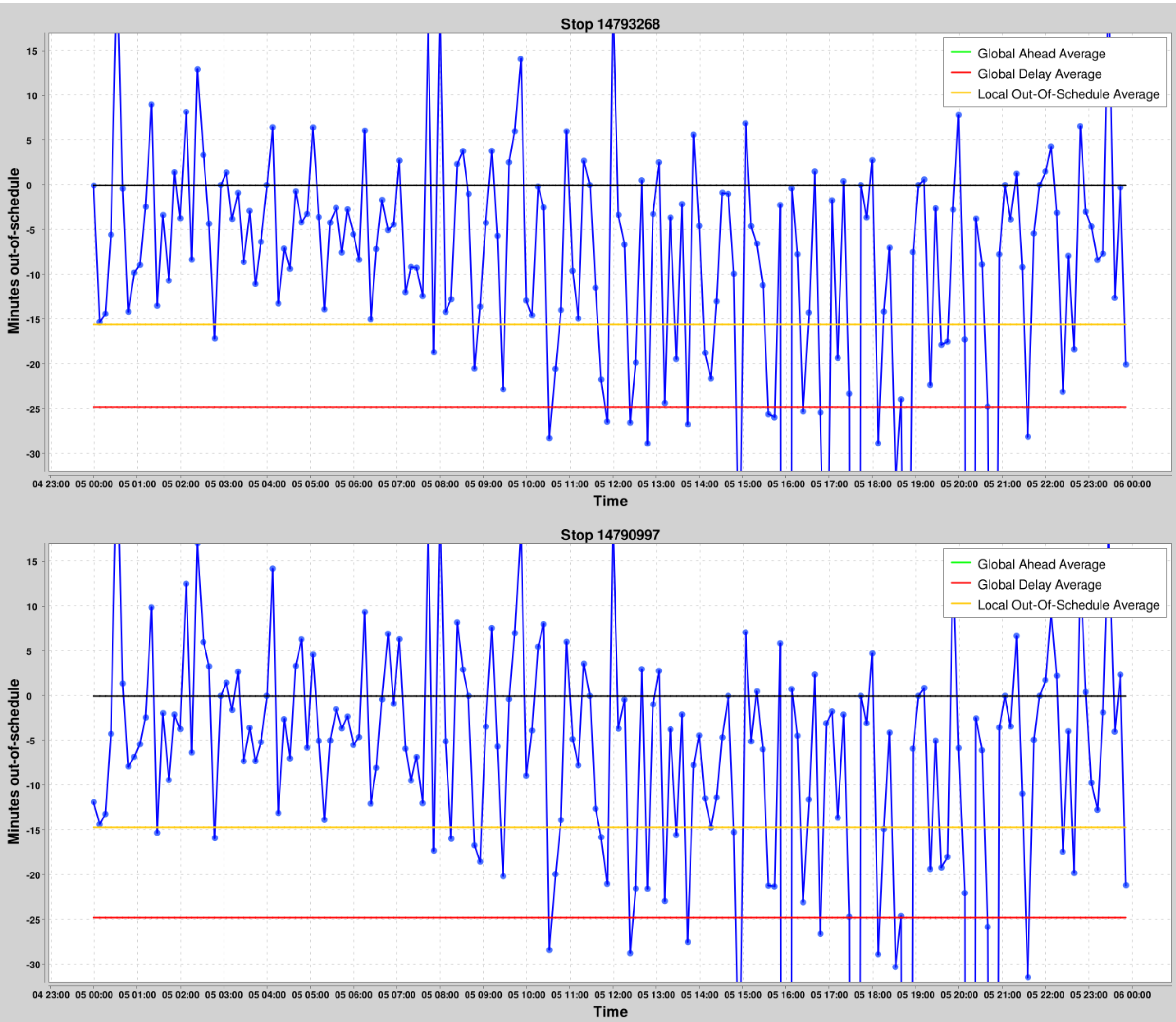
Table 8 – Statuses Distribution for Bus Stops

		GTFS	Generated
Weekday	ON_TIME	3.3%	3.2%
	$AHEAD\_OF\_SCHEDULE$	6.9%	17.8%
	$DELAYED$	89.8%	79.0%
Saturday	ON_TIME	3.9%	3.5%
	$AHEAD\_OF\_SCHEDULE$	6.5%	18.4%
	$DELAYED$	89.6%	78.1%
Sunday	ON_TIME	4.4%	3.9%
	$AHEAD\_OF\_SCHEDULE$	5.4%	18.5%
	$DELAYED$	90.2%	77.6%

Source: The authors

Finally, Figure 21 presents the same couple of stops from Figure 20 but using the generated expected times. With the generated data, the *Global Ahead Average* and the *Global Delay Average* are 38.57 and 24.75 minutes, respectively. This denotes that the interval of out-of-schedule is wider using the generated data. Still, on the other side, the *Local Out-Of-Schedule Average* show smaller delays for the stops #14793268 and #14790997, 15.54 and 14.68 minutes respectively. This data points out that despite increasing the global averages, our approach decreased the average delay in two of the most delayed stops from the PTN by 3.75 and 5 minutes, respectively.

Figure 21 – Minutes out of schedule of a couple of bus stops distributed throughout the day using generated data



Source: The authors

5.4 Limitations

We acknowledge some limitations, mainly due to the desynchronization between the datasets. For instance, the *Real-Time Data Collector* is the most fragile component due to the third-party real-time traffic API interface, which creates minor issues, such as providing some unwanted trips. And some significant problems, such as the impossibility of linking an entry to a trip. Also, the crucial point is that the size and quality of the real-time data heavily depend on the API refresh timeout, in which smaller timeouts translate to more bus positioning, which improves the accuracy of generating artificial entries.

We acknowledge the lack of methods to make a comparison, we discussed about all the others similar scenarios found in literature in Chapter 3. Also, *PondiônsTracker* fails to capture unexpected events that affect the traffic such as concerts because these events *may* not be represented by the GTFS. Thus, two routes are scheduled in the GTFS, but they did not have any entry reported by the real-time API, which are the two following routes: 1. *720 - Circular Saúde MG20* missed 175 trips; 2. *912 - Conjunto Taquaril/Praça Che Guevara* missed 210 trips.

6 CONCLUDING REMARKS

In this master dissertation, we present *PondiônsTracker*, a framework used to collect and integrate GTFS and real-time data. Then, we validate our framework by specializing it to work with Belo Horizonte, *PondiônsTracker-BH*. We collected data from the real-time API for eleven days straight, summarizing over 253 million entries in 2023 and combining it with Belo Horizonte's GTFS data to analyze its PTN.

Our analysis compares the expected schedule with the actual schedule, which uncovers some gaps between the schedule defined at the GTFS and the data collected at Belo Horizonte. First, there are a couple of routes, representing 385 trips the API has not reported at least one entry, and others 200 that have no matched trip, despite having reported entries. This scenario shows a lack of information from the real-time data provider. Also, this analysis points out to the fact that the GTFS data provider under-scheduled trips, which were returned by the API.

Also, we shed light on some delay analysis using the expected times defined at the GTFS and the expected times generated by the *Trip Expected Time Generator*. The performance of the generated dataset shows that the component is a viable option when the stop times are not defined. Belo Horizonte's data demonstrates that only four out of over 150 million trips were entirely on time compared to the schedule, and there are many trips entirely out of schedule. This denotes how complex the PTN of a major city can get and points to the open questions about delays currently being researched in the literature.

Regarding delays in Belo Horizonte's PTN, we show that delays are the most predominant status concerning the schedule. They follow a log-normal distribution throughout the bus stops and might be both spatial and temporal related. This scenario supplies substrate to guide the local programs and initiatives to address mobility issues because it is the right of the citizens to transit around their city, and it directly affects thousands of people in a daily routine. Also, a concise PTN plays a significant role in approaching climate change questions because it could replace several individual vehicles with a few collective ones, for instance.

The analysis performed over Belo Horizonte was only possible because *PondiônsTracker* enabled the link between the two datasets because of the nonexistence of GTFS-RT. Developing, validating, and, mainly, sharing *PondiônsTracker* to as many cities as possible to analyze their data when the GTFS-RT is unavailable is our main contribution in this master thesis. This is translated to *PondiônsTracker* architecture, which takes loose coupling as principal, and the *DataProviders* and *IntegrationModule*'s all components are available as Maven dependencies. Furthermore, the *IntegrationModule* provides many components that are planned to be used as building blocks, so for each particular city, the different blocks can be adapted and combined. The *IntegrationDriver* executes a default sequence of steps to unify the two datasets into a single SQL schema, which schema could have been created and populated by the shell script provided, *init.sh*.

As future work, we intend to explore Belo Horizonte's PTN further, then collect wider time intervals and apply deep learning for graph techniques. This toolkit can explore the impact of different bus stops on each other, even if they are apparently unrelated, such as two stops from different neighborhoods. Thus, graphs algorithms that explore shortest path may show delay patterns and methods to mitigate them. Also, *PondiônsTracker* can be incorporated to geovisualization tools such Layerbase (LOPES; MARQUES-NETO, 2022). In addition, *PondiônsTracker-BH* output, the base of all analysis, can be used as input for a deep graph network. For instance, state-of-the-art techniques such as Flock of Starlings can be exploited.

Also, in future work, we intend to reproduce the results obtained in Belo Horizonte in other cities, not restrained to Brazil. It is an opportunity to improve human mobility in smart cities without the GTFS-RT but with multiple providers, as described in Raghothama, Shreenath e Meijer (2016). Finally, in future work, further exploring the PTN delays combining temporal and spatial dimensions because vehicles in the same pathway will get stuck *together* in a traffic jam, and a slow pace of traffic during rush hour follows the same principle, for instance.

REFERENCES

AEMMER, Z.; RANJBARI, A.; MACKENZIE, D. Measurement and classification of transit delays using gtfs-rt data. PUBLIC TRANSPORT, Springer, v. 14, n. 2, p. 263–285, 2022.

ALBERT, R.; BARABÁSI, A.-L. Statistical mechanics of complex networks. REVIEWS OF MODERN PHYSICS, American Physical Society (APS), v. 74, n. 1, p. 47–97, jan 2002. Disponível em: <https://doi.org/10.1103%2Frevmodphys.74.47>.

ALBINO, V.; BERARDI, U.; DANGELICO, R. Smart cities: Definitions, dimensions, performance, and initiatives. JOURNAL OF URBAN TECHNOLOGY, v. 22, p. 2015, 02 2015.

ANTRIM, A.; BARBEAU, S. J. et al. The many uses of gtfs data–opening the door to transit and multimodal applications. LOCATION-AWARE INFORMATION SYSTEMS LABORATORY AT THE UNIVERSITY OF SOUTH FLORIDA, v. 4, 2013.

AUER, S. et al. Dbpedia: A nucleus for a web of open data. In: ABERER, K. et al. (Ed.). THE SEMANTIC WEB. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 722–735. ISBN 978-3-540-76298-0.

BACCIU, D. et al. A gentle introduction to deep learning for graphs. NEURAL NETWORKS, Elsevier BV, v. 129, p. 203–221, sep 2020. Disponível em: <https://doi.org/10.1016%2Fj.neunet.2020.06.006>.

BARBOSA, H. et al. Human mobility: Models and applications. PHYSICS REPORTS, v. 734, p. 1–74, 2018. ISSN 0370-1573. Human mobility: Models and applications. Disponível em: <https://www.sciencedirect.com/science/article/pii/S037015731830022X>.

BONDY, J. A.; MURTY, U. S. R. GRAPH THEORY WITH APPLICATIONS. [S.l.]: Macmillan Education UK, 1976. 1-226 p. ISBN 978-1-349-03521-2.

CARVALHO, A.; MORAIS, E. P.; CUNHA, C. R. Location based mobile services & Context-aware: An approach to the tourism sector. PROCEEDINGS OF THE 32ND INTERNATIONAL BUSINESS INFORMATION MANAGEMENT ASSOCIATION CONFERENCE, IBIMA 2018 - VISION 2020: SUSTAINABLE ECONOMIC DEVELOPMENT AND APPLICATION OF INNOVATION MANAGEMENT FROM REGIONAL EXPANSION TO GLOBAL GROWTH, p. 6828–6836, 2018.

CHANG, S. L. et al. Modelling transmission and control of the covid-19 pandemic in australia. NATURE COMMUNICATIONS, v. 11, n. 1, p. 5710, Nov 2020. ISSN 2041-1723. Disponível em: <https://doi.org/10.1038/s41467-020-19393-6>.

FERBER, C. O. O. von et al. A tale of two cities. JOURNAL OF TRANSPORTATION SECURITY, v. 5, 06 2012.

GOOGLE. GTFS Static Overview. 2005. `https://developers.google.com/transit/gtfs/`. Accessed: 2023-19-06.

GOOGLE. GTFS Realtime Overview. 2009. `https://developers.google.com/transit/gtfs-realtime`. Accessed: 2023-19-06.

HUANG, J. et al. Understanding the impact of the covid-19 pandemic on transportation-related behaviors with human mobility data. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining. New York, NY, USA: Association for Computing Machinery, 2020. (KDD '20), p. 3443–3450. ISBN 9781450379984. Disponível em: <https://doi-org.ez93.periodicos.capes.gov.br/10.1145/3394486.3412856>.

LOPES, P.; MARQUES-NETO, H. Layerbase: Uma solução para visualização e análise temporal de dados georreferenciados. In: Anais do VI Workshop de Computação Urbana. Porto Alegre, RS, Brasil: SBC, 2022. p. 70–83. ISSN 2595-2706. Disponível em: <https://sol.sbc.org.br/index.php/courb/article/view/21445>.

MCHUGH, B. Pioneering open data standards: The gtfs story. Beyond transparency: open data and the future of civic innovation, Code for America Press San Francisco, p. 125–135, 2013.

MILLER, H. J. Modelling accessibility using space-time prism concepts within geographical information systems. International Journal of Geographical Information Systems, Taylor Francis, v. 5, n. 3, p. 287–301, 1991. Disponível em: <https://doi.org/10.1080/02693799108927856>.

NAM, T.; PARDO, T. A. Conceptualizing smart city with dimensions of technology, people, and institutions. In: Proceedings of the 12th Annual International Digital Government Research Conference: Digital Government Innovation in Challenging Times. New York, NY, USA: Association for Computing Machinery, 2011. (dg.o '11), p. 282–291. ISBN 9781450307628. Disponível em: <https://doi.org/10.1145/2037556.2037602>.

OKF. Open Knowledge Foundation. 2004. `https://okfn.org/`. Accessed: 2023-05-20.

RAGHOTHAMA, J.; SHREENATH, V. M.; MEIJER, S. Analytics on public transport delays with spatial big data. In: Proceedings of the 5th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data. New York, NY, USA: Association for Computing Machinery, 2016. (BigSpatial '16), p. 28–33. ISBN 9781450345811. Disponível em: <https://doi.org/10.1145/3006386.3006387>.

RAVENSTEIN, E. G. The laws of migration. Journal of the Statistical Society of London, [Royal Statistical Society, Wiley], v. 48, n. 2, p. 167–235, 1885. ISSN 09595341, 23972343. Disponível em: <http://www.jstor.org/stable/2979181>.

RO, J. W. et al. Compositional cyber-physical epidemiology of covid-19. Scientific Reports, v. 10, n. 1, p. 19537, Nov 2020. ISSN 2045-2322. Disponível em: <https://doi.org/10.1038/s41598-020-76507-2>.

SARAN, S. et al. Review of geospatial technology for infectious disease surveillance: Use case on covid-19. Journal of the Indian Society of Remote Sensing, v. 48, n. 8, p. 1121–1138, Aug 2020. ISSN 0974-3006. Disponível em: <https://doi.org/10.1007/s12524-020-01140-5>.

SILVEIRA, L. M. et al. MobDatU: A New Model for Human Mobility Prediction Based on Heterogeneous Data. Proceedings - 33rd Brazilian Symposium on Computer Networks and Distributed Systems, SBRC 2015, IEEE, p. 217–227, 2015.

SMARZARO, R.; DAVIS, C. A.; QUINTANILHA, J. A. Creation of a multimodal urban transportation network through spatial data integration from authoritative and crowdsourced data. ISPRS International Journal of Geo-Information, v. 10, n. 7, 2021. ISSN 2220-9964. Disponível em: <https://www.mdpi.com/2220-9964/10/7/470>.

SONKIN, R.; ALPERT, E. A.; JAFFE, E. Epidemic investigations within an arm's reach – role of google maps during an epidemic outbreak. Health and Technology, v. 10, n. 6, p. 1397–1402, Nov 2020. ISSN 2190-7196. Disponível em: <https://doi.org/10.1007/s12553-020-00463-0>.

STOUFFER, S. A. Intervening opportunities: a theory relating mobility and distance. American Sociological Review, v. 5, p. 845–867, 1940.

WESSEL, N.; ALLEN, J.; FARBER, S. Constructing a routable retrospective transit timetable from a real-time vehicle location feed and gtfs. Journal of Transport Geography, v. 62, p. 92–97, 2017. ISSN 0966-6923. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0966692317300388>.

WESSEL, N.; WIDENER, M. J. Discovering the space–time dimensions of schedule padding and delay from gtfs and real-time transit data. Journal of Geographical Systems, v. 19, p. 93–107, 2017.

WILLIAMSON, E. Lists, Decisions and Graphs. S. Gill Williamson, 2010. Disponível em: <https://books.google.com.br/books?id=vaXv_yhefG8C>.

WONG, J. C. Use of the general transit feed specification (GTFS) in transit performance measurement. 2013. Tese (Doutorado) — Georgia Institute of Technology.

WRONA, P.; GRZENDA, M.; LUCKNER, M. Streaming detection of significant delay changes in public transport systems. In: GROEN, D. et al. (Ed.). Computational Science – ICCS 2022. Cham: Springer International Publishing, 2022. p. 486–499. ISBN 978-3-031-08760-8.

ZHENG, Y. et al. Urban computing: Concepts, methodologies, and applications. ACM Trans. Intell. Syst. Technol., Association for Computing Machinery, New York, NY, USA, v. 5, n. 3, set. 2014. ISSN 2157-6904. Disponível em: <https://doi-org.ez93.periodicos.capes.gov.br/10.1145/2629592>.

ZIPF, G. National Unity and Disunity: The Nation as a Bio-social Organism. Principia Press, Incorporated, 1941. ISBN 9780598847560. Disponível em: <https://books.google.com.br/books?id=wFezAAAAMAAJ>.