

Passengers Request Aggregation and Dynamic Vehicle Routing

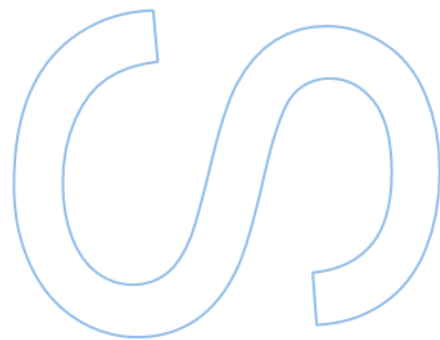
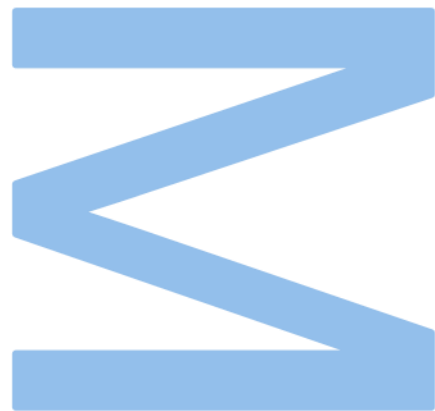
Diogo Miguel Carvalho Castro

Master in Computer Science

Department of Computer Science

Faculty of Sciences of the University of Porto

2024



Passengers Request Aggregation and Dynamic Vehicle Routing

Diogo Miguel Carvalho Castro

Dissertation carried out as part of the Master in Computer
Science

Department of Computer Science
2024

Supervisor

Michel Celestino Ferreira Paiva, Assistant Professor,
Faculty of Sciences of the University of Porto,
Geolink, Lda

GEOLink

Abstract

In today's society, we struggle with significant challenges related to traffic congestion and safe transportation. These issues are mostly perceived during large-scale events such as music festivals or sporting events, where large numbers of people require efficient, affordable, and environmentally friendly transportation options. Traditional solutions such as private vehicle request and ordinary bus routes often fall short, either due to cost barriers, their contribution to carbon emissions and/or vehicle overcrowding. This dissertation addresses these pressing concerns by introducing a novel web application, designed specifically for event transportation, that employs 2 novel algorithms, offering an easy to use alternative to the services available on the market.

The application operates during designated events, allowing attendees to choose their desired destination from all available bus stops within Porto, Portugal, ensuring full city coverage. At a predetermined time, the application aggregates all client requests and determines the optimal number of buses needed to accommodate all users, and efficiently distributed them across the available vehicles. Using a sophisticated routing algorithm, the application generates the most efficient route for each bus, passing through the assigned clients' chosen bus stops and returning to the starting point, available to transport new passengers.

This innovative approach not only provides a well-organized and green solution to event transportation but also ensures affordability and safety. By utilizing professional and experienced bus drivers, the application guarantees a swift and secure journey home for all app users. This project represents a significant stride towards sustainable event transportation, with potential for future development and adaptation to other contexts.

Keywords: Smart Mobility Solutions; Sustainability; Large-Scale Event Transportation; Efficient Transportation; Event-based Transportation Planning; Traffic Congestion Solution; Dynamic Routing; Vehicle Routing Problem; Aggregation Algorithm; Algorithm Development and Analysis; Web Application Development; Database Management; Network Analysis.

Resumo

Na sociedade de hoje em dia, enfrentamos desafios significativos relacionados ao congestionamento do tráfego e ao transporte seguro. Essas questões são principalmente notadas durante eventos de grande escala como festivais de música ou eventos desportivos, onde um grande número de pessoas requer opções de transporte eficientes, acessíveis e amigas do ambiente. As soluções tradicionais como pedidos de veículos privados ou rotas de autocarro regulares frequentemente não são suficientes, seja devido a barreiras de custo, à sua contribuição para as emissões de carbono e/ou à sobrelotação de veículos. Esta dissertação aborda estas preocupações urgentes ao introduzir uma nova aplicação web, projetada especificamente para o transporte durante eventos, que utiliza 2 novos algoritmos, oferecendo uma alternativa aos serviços disponíveis no mercado, fácil de usar.

A aplicação opera durante eventos específicos e oferece aos participantes a possibilidade de selecionar o seu destino desejado entre todas as paragens de autocarro dentro do Porto, Portugal, garantindo uma cobertura total da cidade. Num horário predeterminado, a aplicação agrega todos os pedidos e determina o número ótimo de autocarros necessários para acomodar todos os utilizadores, alocando cada cliente a um autocarro específico de forma ótima. Usando um algoritmo de *routing* sofisticado, a aplicação gera a rota mais eficiente para cada autocarro, passando pelas paragens escolhidas pelos clientes que lhe foram atribuídos e retornando ao ponto de partida, disponível para transportar novos passageiros.

Esta abordagem inovadora não só oferece uma solução bem organizada e verde para o transporte de passageiros durante eventos, mas também garante acessibilidade e segurança. Ao utilizar motoristas de autocarros profissionais e experientes, a aplicação garante uma viagem rápida e segura para casa para todos os utilizadores da aplicação. Este projeto representa um passo significativo em direção ao transporte sustentável de eventos, com potencial para desenvolvimento futuro e adaptação a outros contextos.

Palavras-chave: Soluções de Mobilidade Inteligente; Sustentabilidade; Transporte para Grandes Eventos; Transporte Eficiente; Planeamento de Transporte para Eventos; Solução para Congestionamento de Tráfego; Roteamento Dinâmico; Problema de Roteamento de Veículos; Algoritmo de Agregação; Desenvolvimento e Análise de Algoritmos; Desenvolvimento de Aplicação Web; Gestão de Bases de Dados; Análise de Redes.

Acknowledgments

The journey of developing and writing this dissertation has been a profoundly special and memorable experience, marked by both highs and lows. Each person who shared this time with me played an essential role in shaping the person I am today, offering support when I needed it most.

First and foremost, I extend my heartfelt thanks to Professor Michel Ferreira. Your unwavering enthusiasm and the opportunity you provided to be part of this project have been invaluable. It has been a privilege to work under your guidance on what has become my most significant endeavor to date. For this, you have my deepest gratitude.

I am eternally grateful to my mother, Cassilda Carvalho. Since the day I was born, you have been my constant support, giving me everything I could ever wish for, especially your boundless love and attention. Without you, I would not have had the strength to overcome the toughest challenges. The joy and comfort I found each morning, and the opportunity to pursue a university education, were all possible because of your unwavering dedication, your tears, and your tireless efforts.

To my friends, thank you for always being there to share in the other facets of my life, creating cherished memories and never giving up on me. A special thanks to my girlfriend, Ana Bárbara, who bore the weight of my stress and the trials of life alongside me, always bringing happiness and optimism with her. You never left my side, always guiding me towards the light even in the darkest of times.

Finally, to everyone who played a part in this journey, thank you. I am honored to have been part of yours as well.

Dedicated to my family and my friends who were always there to support me,
especially to Maria Cassilda da Silva Carvalho Castro and Ana Bárbara Carvalho
Barbosa

Contents

Abstract	i
Resumo	iii
Acknowledgments	v
Contents	viii
List of Tables	ix
List of Figures	xiii
Acronyms	xv
1 Introduction	1
1.1 Chapter Organization	3
2 State of The Art	5
3 Design and Development	11
3.1 Project Idea and Motivation	11
3.2 The Database	14
3.2.1 The Application Data	14
3.2.2 SCTP Data	16
3.3 The Network	18
3.3.1 Network Creation	19

3.3.2	Network Visualization and Comparison	24
3.4	The Algorithm	32
3.4.1	Aggregation Of Bookings	33
3.4.2	Route Generation	39
3.4.3	Results	51
3.5	Web Application	56
4	Tests and Analysis	63
5	Conclusions	79
5.1	Future Work	80
	Bibliography	81

List of Tables

3.1	Degree Comparison	24
3.2	Out-Degree Comparison	25
3.3	In-Degree Comparison	25
3.4	Path Comparison	25
4.1	Running Time Comparison	64
4.2	Running Time Comparison with and without memoization	65
4.3	Simplified Routing Time and Distance Comparison	66
4.4	Non-Simplified Routing Time and Distance Comparison	68
4.5	Simplified Route Time and Distance Comparison	71
4.6	Non-Simplified Route Time and Distance Comparison	74

List of Figures

1.1	Alcohol Consumption Per Country	2
1.2	Road Accident Fatalities in 2022	2
3.1	ER Model	16
3.2	Porto City	17
3.3	STCP Bus Heatmap	17
3.4	Original Network	18
3.5	Network Simplification	19
3.6	ER Model	22
3.7	Number of Nodes PieChart	23
3.8	Number of Edges PieChart	23
3.9	Original Network	24
3.10	Simplified Network	24
3.11	Simplified Network Betweenness Centrality	27
3.12	Original Network Betweenness Centrality	27
3.13	Simplified Network Closeness Centrality	28
3.14	Original Network Closeness Centrality	29
3.15	Undirected Original Network Closeness Centrality	30
3.16	Algorithm Architecture	32
3.17	Booking aggregation algorithm Flow Chart	34
3.18	Distribution of Bookings	36

3.19 Grouping of Bookings	36
3.20 Best Case Improbable Scenario Representation	37
3.21 Best Case Probable Scenario Representation	38
3.22 Worst Case Improbable Scenario Representation	38
3.23 Worst Case Probable Scenario Representation	39
3.24 Routing Generation Initial Phase	41
3.25 Route Generation Before Simplification	42
3.26 Route Generation After Simplification	42
3.27 Redefine Stops List Function Flow Chart	43
3.28 Redefining the Stop List Illustration	45
3.29 General Case Loop Flow Chart	46
3.30 Specific Case Loop Flow Chart	46
3.31 Route Generation Result	52
3.32 Route Generation Result	54
3.33 Passenger Icon Unchanged Example	55
3.34 Initial page	56
3.35 Sign Up	57
3.36 Mockup User	57
3.37 Login	57
3.38 Mockup User	57
3.39 Home page	58
3.40 Bookings page	58
3.41 Schedule page	58
3.42 Updated home page	59
3.43 Filled Information	59
3.44 Successful booking creation	59
3.45 Bookings Page	60
3.46 Updated home page	60

3.47 Event track page	60
4.1 Running Time Comparison Line Chart	64
4.2 Porto Simplified Routing Examples	67
4.3 Route Map with unused pathing circled	68
4.4 Porto Without Simplified Routing Examples	69
4.5 Time Comparison Bar Chart	70
4.6 Distance Comparison Bar Chart	70
4.7 Greater Porto With Simplified Routing Examples	73
4.8 Greater Porto Time Comparison Bar Chart	74
4.9 Greater Porto Distance Comparison Bar Chart	75
4.10 Greater Porto Without Simplified Routing Examples	76
4.11 Route Map with very distinct drop off zones	77

Acronyms

STCP	Sociedade Transportes Colectivos do Porto	CSV	Comma Separated Values
GPS	Global Positioning System	TSP	Travelling Salesman Problem
GIS	Geographic Information System	VRP	Vehicle Routing Problem

Chapter 1

Introduction

In today's fast-paced world, the need for efficient transportation is more pressing than ever. Whether commuting from work, returning from shopping, or seeking a swift journey home after an enjoyable event, the demand for quick and convenient travel options is inescapable. The advances of mobile technology has revolutionized transportation, enabling individuals to request private vehicles such as Taxi or Uber rides at the touch of a button, at any time, anywhere. While the value of this advancements can't be denied, this convenience has inadvertently exacerbated traffic congestion and environmental pollution, transforming brief 20-minute commutes into exhaustive, hour-long journeys, while also contributing to a bigger carbon footprint in a time where such matters should not be taken lightly.

Portugal, in particular, struggles with unique transportation challenges. It is among the nations with the highest fatality rates in the European Union due to unsafe driving as seen in the figure 1.2 [11], compounded by one of the world's highest rates of alcohol consumption as shown in the figure 1.1 [10]

These issues are further increased during events such as festivals and concerts, where attendees, often fatigued and inebriated, risk their safety by walking alone at night or entering vehicles driven by strangers.

This thesis aims to address these pressing issues by developing a novel **application** that relies on **two main algorithms**, designed to provide efficient, safe, and environmentally friendly transportation during specified events. In collaboration with **STCP**(Sociedade Transportes Colectivos do Porto), which supplied the transportation infrastructure, including the fleet of buses and designated bus stops, integral to the app's operation, this application enables users to book a drop off location for an event, a **STCP** bus stop, that will then be assigned to a bus, that will have the responsibility of delivering every passenger assigned to it. The passengers will be waiting to depart on a defined waiting zone near the event's location where they will be informed, *à priori*, where to wait, as well as their respective assigned bus. Is important to mention, that the waiting zone should be chosen based on space for both the clients and the buses parking, as to create a safe and comfortable environment for everyone and minimize organizational hazards.

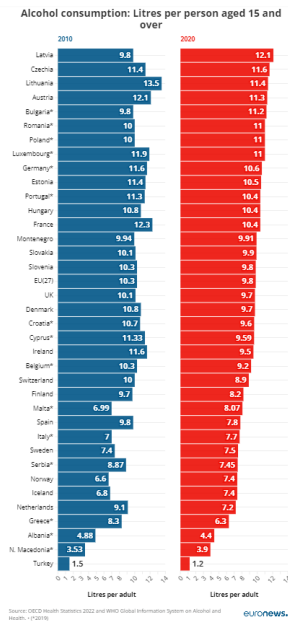


Figure 1.1: Alcohol Consumption Per Country

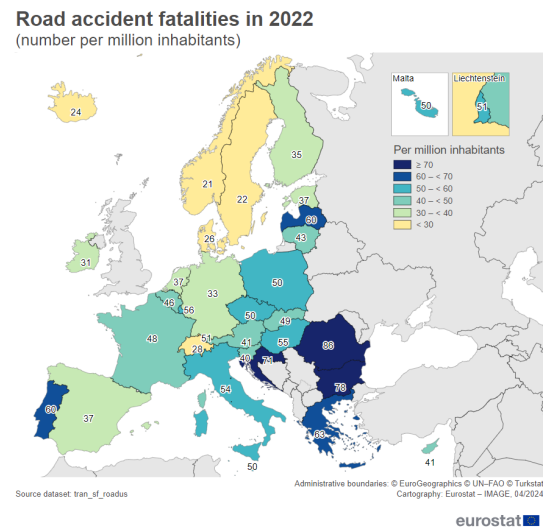


Figure 1.2: Road Accident Fatalities in 2022

Here, when the time is right, the buses will begin to pick up the clients and start their route, delivering every passenger to their chosen bus stop.

When utilizing the application, the users will view the available bus stops within Porto, and select one as their destination for a particular event within a designated time slot. Once a booking is made, the application securely stores the details in a database and the client will receive a confirmation message. As previously stated, the waiting zone location as well as the assigned bus identifier will only be informed around 20 to 30 minutes before the departure time, as too give for every event attendee to book their seat in case they forgot, along with also leaving enough time window for changes in the predefined waiting zone in case some problem arises.

Some minutes before the event scheduled time, the central server will call the **Aggregation** and **Routing** algorithms sequentially. Firstly, the server retrieves the bookings made until that point in time, for that specific event and time slot, and allocates each passenger to one of the available buses. This first task, is handled by the **Aggregation** algorithm, guaranteeing the minimum number of buses needed to transport everyone safely and comfortably, minimizing costs and pollution, while maximizing each buses occupation. The aggregation of passengers, as might be expected, will not be arbitrary, but rather use a deterministic approach to assign the most efficient seats to everyone, guaranteeing each passenger is designated with the best vehicle for it's respective destination or, in other words, find the optimal bus distribution of passengers. Having each bus being assigned with their respective passengers, using the **Routing** algorithm, each bus would then, based on their respective client destinations, be presented with the most efficient travelling route for the driver to follow. Send by the server to each buses' **GPS**, presenting a custom generated path, the algorithm ensures not only a fast delivery of passengers, but a fast

return to the initial point, where the bus would then wait for the next departure time, given the driver time to rest and new passengers to book their own seat (if not already full).

The methodology employed to achieve this objective involves a detailed examination of the technical aspects of the application, including database management, data analysis and engineering, request aggregation and route generation algorithms and the web application itself. During this thesis, we will follow a logical path to the construction of the final product, exploring the key parts, while also debating the issues faced and their respective solutions. The project's performance is evaluated through rigorous testing and comparisons, providing a motive for some decisions that were made, while also identifying potential areas for future development and improvement ideas.

In summary, the contributions of this work are manifold. By offering a green, affordable, and systematic solution to event transportation, the application promises to enhance **user safety**, reduce **traffic congestion**, and contribute to **environmental sustainability**. The following chapters of this thesis will explore these contributions in greater detail and outline the overall structure of the project.

1.1 Chapter Organization

This thesis is divided into 4 chapters, organized in the following order:

Chapter 2 – State of the Art: Presents a comprehensive review of the relevant literature related to this dissertation, offering insights into existing perspectives, projects, and studies within this field of research.

Chapter 3 – Design and Development: Provides a comprehensive overview of the project's design and development, following a logical sequence. It details the work conducted across multiple areas, including Database Management, Network Analysis, Project Structuring, Algorithm Development and Analysis, Web Development, and the final results.

Chapter 4 – Tests and Analysis: Presents the results of the tests and evaluations conducted, providing a comprehensive comparison and analysis, with a primary focus on the developed algorithm.

Chapter 5 – Conclusions: Concluding reflections on the project and recommendations for future work.

Chapter 2

State of The Art

Large-scale events such as music festivals, conferences, and sports gatherings inherently pose significant transportation challenges, often resulting in severe traffic congestion and increased air pollution. The heavy reliance on private transport, particularly taxis and ride-hailing services, exacerbates environmental degradation and diminishes the visitor experience with increased traffic and, consequently, increased travel time. Our study aims to develop a system that leverages aggregation and routing algorithms to optimize transportation for large-scale event attendees. Specifically, this research focuses on dynamically allocating passengers to buses and creating personalized bus routes to reduce the dependence on private vehicles and promote sustainability in event transportation.

Sustainable Event Transportation: To mitigate the environmental impact of large-scale events, numerous studies have explored the implementation of sustainable transportation solutions. Ballarano et al. [1] offers a comprehensive review of sustainability practices at large-scale events, with a focus on promoting public transportation and non-motorized modes like cycling and walking. These strategies are effective in reducing the number of private vehicles on the road, thereby alleviating traffic congestion and lowering emissions—contributing to the European Union’s goal of reducing emissions by 55% by 2030, compared to 1990 levels.

Chirieleison et al. [5] in “Event Sustainability and Transportation Policy: A Model-Based Cluster Analysis for a Cross-Comparison of Hallmark Events” explores how transportation policies impact the sustainability of major public events. Using model-based cluster analysis, the study identifies and compares different clusters of hallmark events based on their sustainability practices and transportation strategies. This cross-comparison sheds light on how various approaches affect sustainability outcomes, offering valuable insights for policymakers and event organizers to improve future event sustainability. The research contributes to the literature by providing empirical evidence on the intersection of transportation policy and event sustainability.

Moreover, Chirieleison et al. [4], in “Event sustainability and sustainable transportation: a positive reciprocal influence”, assessed transportation as part of the event by using the analytical possibilities of the single-case study method and found that more could be done in relation to

transportation in order to influence a more positive perception of event and traffic management.

However, these approaches often lack the flexibility needed to accommodate the diverse travel patterns of event participants. Fixed public transportation routes are inadequate for serving the concentrated and short-term travel demands associated with large-scale events. Non-motorized transportation is also impractical for long distances or for attendees with limited mobility. Our research addresses these limitations by proposing a more adaptive and efficient transportation solution that caters to the varying needs of event attendees.

Mobile Apps and Multimodal Aggregators: Mobile apps have revolutionized urban mobility, enabling real-time information access, travel itinerary planning, and ride-sourcing. Shaheen et al. [39] highlights the transformative impact of mobile apps on multimodal travel by showcasing their ability to integrate various transportation modes. For example, Nazri et al. [29], explores the MyAmble app that was designed to capture transportation behaviors among populations, emphasizing the importance of stakeholder engagement in app development. As a more local example, we also have the TubApp mobile app, which is capable of providing users with itineraries, schedules and real-time bus tracking for the Transportes Urbanos de Braga (TUB) bus fleet, providing a plethora of information to the clients. Additionally, some applications can group passengers heading to the same destination into a single vehicle, saving time for both drivers and passengers. Conversely, other apps allow vehicles to pick up multiple passengers from a common location and drop them off at different destinations, being constricted to their maximum occupation of a personal vehicle, of around 4 to 8 passengers.

Mobile applications are also a great catalyst for better transportation systems as explored by Ragheb et al. [37]. The paper explores how big data, with its high velocity, volume, and variety, can enhance transportation planning through decision support systems (DSS) and transit mobile apps. It examines how big data can manage transportation demands, budgets, and stakeholder needs while addressing environmental and social concerns.

The paper "Mobile Application Development to Solve Vehicle Routing Problems in Marketing or Tour Trip Planning" by Nugroho et al. [30] presents a mobile app designed to address the vehicle routing problem (VRP), a complex optimization challenge in transportation services. The app integrates Mapbox and Google Maps APIs to compute real distances and find the shortest, most cost-effective routes. The study demonstrates that the app effectively handles various conditions, offering a practical solution for optimizing routes in scenarios such as marketing or tour planning.

However, while these applications have significantly improved individual transportation methods, route optimization and adaptability, they often lack the capability to aggregate passengers within the same vehicle and generate a customized route, especially in the context of large-scale events. Our work builds on the success of these apps by incorporating passenger aggregation and personalized bus routing specifically tailored to the demands of large-scale events, for a large number of points, as well as a large covered area.

Vehicle Routing and Aggregation Algorithms: Vehicle routing and aggregation

algorithms are vital for optimizing transportation logistics. The Vehicle Routing Problem (VRP) is a well-established area in logistics, with numerous algorithms developed to address its various variants. Konstantakopoulos et al. [23] provides a detailed classification of these VRP variants, primarily focused on cargo transportation. Applying the VRP framework to a bus route optimization problem for event attendees presents unique challenges, particularly the need for real-time route adjustments and passenger aggregation. These challenges require specialized algorithms that go beyond traditional VRP solutions and are tailored to the specific geographic and infrastructural context of the event location. Our research aims to address this gap by developing real-time passenger clustering and bus route optimization algorithms that enhance the efficiency of event transportation, particularly in the context of Porto and the **STCP** bus network, where the geographical constraints, updated routes and road as well as the real time road works should be considered. Moreover, Perić et al. [32] explores the vehicle routing problem (VRP) with a focus on electric vehicles, emphasizing battery consumption based on road conditions. Their predictive algorithm shows potential cost reductions of up to 80% compared to conventional diesel vehicles, which further underscores our commitment to not only pursuing a more environmentally sustainable solution in this dissertation project but also considering cost-effective alternatives.

Furthermore, passenger aggregation is a key component of our approach. By grouping attendees with similar or nearby destinations, we can maximize vehicle occupancy and minimize travel times. Existing research on ride-sharing and public transportation shows promise in this area, although most studies focus on smaller-scale applications, typically involving groups of two to four passengers. Furthermore, most known application are focused on multiple pickup single destinations while our project intends to do the complete opposite and make use of the full network of roads available. Furthermore, while those applications are capable of making use of the entity of the road network, our project, due to constrictions such as bus volume and width, is constricted to a smaller set of viable streets. We aim to extend these concepts to the more complex and dynamic context of large-scale event transportation, while addressing its inherent challenges.

Geolocation and Mapping Technologies: Geolocation and mapping technologies are foundational to advanced transportation systems, particularly for real-time route optimization and traffic monitoring. Khadem et al. [21] provides valuable insights into multi-objective route planning in hazardous material transportation, with parallels relevant to our research. Integrating real-time data into route optimization is central to our approach.

In the context of event transportation, optimizing bus routes requires consideration of pre-existing routes used by public transport buses and ensuring that selected roads can accommodate large vehicles. By incorporating geolocation technologies like GPS and GIS, our system can obtain real-time vehicle data, allowing us to develop interactive maps that inform attendees of bus locations and estimated arrival times, thereby improving the user experience. For transport operators, these tools facilitate network monitoring, enabling timely adjustments to enhance operational efficiency. Our research enhances both the operational effectiveness and user

experience of event transportation systems through the strategic use of technology.

Environmental Sustainability and Green Transportation: In today’s global context, the need for sustainable solutions to combat climate change is increasingly urgent. Large-scale events present a unique opportunity to advocate for green transportation initiatives. Rehman et al. [38] emphasizes the importance of reducing greenhouse gas emissions by improving energy efficiency in transportation.

Our research aligns with these global sustainability goals by focusing on passenger aggregation and personalized routing, both of which reduce vehicle kilometers traveled and associated emissions. By incorporating green transportation practices into our algorithms, we contribute to the broader effort to make large-scale events more sustainable, reducing the carbon footprint while improving the overall attendee experience.

Challenges in Transportation for Events: Managing transportation for large-scale events presents challenges distinct from everyday urban transportation. These include uncertainty in transport demand, temporary infrastructure requirements, and the complexity of coordinating multiple transportation vehicles.

One of the most significant challenges is the sudden surge in transportation demand, which can overwhelm existing infrastructure and cause delays. Jiang et al. [20] provides valuable insights into managing transportation in high-pressure, unpredictable situations—principles that are applicable to our research. In both cases, quick adaptation to dynamic conditions is essential. For example, buses may need to be rerouted due to unexpected road closures, or schedules might require real-time adjustments in response to unforeseen changes in passenger demand. Additionally, contingency plans must account for potential vehicle breakdowns, ensuring that backup buses are available when needed. Effective coordination among public transit providers, shuttle services, on-demand transport options, and event organizers is also critical to ensuring seamless transportation for attendees. Infrastructure constraints often exacerbate these challenges, particularly in rural or temporary event locations. Temporary infrastructure, such as additional parking facilities or dedicated bus routes, may be necessary to manage the increased transportation demand.

Garrido et al. [16] explores the use of Decision Support Systems (DSS) to improve urban mobility during major city events. It addresses the challenges of increased traffic, unauthorized parking, and environmental impact caused by high private car usage. By implementing a living lab approach, the study involves stakeholders in the co-design and testing of DSS solutions, demonstrated through real-world applications like football matches and concerts in Milan. The findings suggest that DSS can effectively reduce car dependency by promoting shared and micro-mobility options, highlighting its role in creating more efficient and sustainable urban transport systems.

Our research addresses these challenges by developing adaptive and flexible algorithms that respond to changing conditions and maximize the efficient use of available transportation resources. Our goal is to alleviate infrastructure strain and enhance event transportation efficiency solely

through the developed system, regardless of event conditions.

Case Studies: Examining relevant case studies provides valuable context for our research within the broader landscape of transportation solutions. Ferreira et al. [13] presents several case studies on the implementation of smart and sustainable transportation solutions.

One case study discusses the use of Intelligent Transport Systems (ITS) at a major international sporting event, where public transport, ride-sharing, and walking routes were integrated into a unified platform. This system provided real-time information to attendees, enabling them to choose the most optimal and sustainable mode of transport. Dynamic routing was also employed, allowing routes and schedules to be adjusted in real-time based on traffic conditions and passenger demand. The success of this case study highlights the potential of integrating multiple modes of transportation into a cohesive system, supported by real-time data and dynamic routing. Such solutions are particularly relevant for large-scale events, where transportation systems must be scalable and adaptable to manage high volumes of passengers and rapidly changing conditions.

Another pertinent case study involves the implementation of a shuttle bus system at one of the world’s largest music festivals. The organizers faced significant transportation challenges due to the event’s remote location and lack of existing infrastructure. The shuttle bus system grouped passengers from various pickup points and transported them directly to the venue. Real-time tracking and communication tools were integrated into the system, allowing participants to monitor bus locations and receive delay notifications.

This case study demonstrates the feasibility of shuttle bus systems as a solution for transporting large numbers of attendees in areas with underdeveloped infrastructure. However, it also underscores the challenges of synchronizing multiple shuttle routes and ensuring timely transportation for all attendees. Building on these concepts, our research introduces new methods for personalized routing and advanced algorithms for passenger aggregation, aimed at enhancing both the efficiency and sustainability of event transportation systems, as well as providing the needed information to the clients and scheduled services providing a systematic and organized solution to event attendees.

Conclusion: In this section, we have thoroughly reviewed the state-of-the-art literature and explored existing solutions to the transportation challenges identified, along with other relevant parameters related to our project. As our research advances, we will continue to refine and validate our approach, leveraging insights from the literature and best practices from pertinent case studies. Our objective is to develop a transportation solution that is not only efficient and sustainable but also scalable and responsive to the diverse needs of large-scale event attendees.

Chapter 3

Design and Development

During my academic journey as a Master’s Degree student at FCUP, we had the opportunity to develop a full project addressing the current challenges in transportation. This was part of my experience at [Geolink](#), a company born out of research conducted at FCUP, specializing in Vehicle Communication and Geo-spatial databases. [Geolink](#) is particularly focused on optimizing operations related to vehicle fleets, such as taxis. The project was conceived during my tenure at the company and it was a collaborative effort, involving a partnership with [STCP](#), a renowned Portuguese bus company. [STCP](#) manages over 72 bus routes in Porto and its surrounding regions, collectively known as Greater Porto and will provide the buses and bus drivers inherent to the functionality of the project.

3.1 Project Idea and Motivation

As stated in the Introduction chapter of this dissertation, there are numerous current problems inherent to public transportation and safe driving. These issues necessitate the availability of a better solution for anyone wanting to ensure that they return home safe and sound after participating in any activity they enjoy. We decided to focus our project on developing a product that serves as an affordable and efficient solution for occasions such as festivals or public events, like the **Primavera Sound Porto**, an annual musical festival, which is planned to be the first large-scale event to embrace this service and one that we will use for testing purposes. The Primavera Sound Porto festival, held over three days in Porto, typically taking place between May and June, is a counterpart of the official Primavera Sound that has been held in Barcelona, Spain, since 2001. Festivals and events of similar scale like these, bring together thousands of people in the same space. For instance, Primavera Sound 2023 (last year) attracted between 45 and 60 thousand people per day.

In large-scale events such as these, normally, there is a common time when most event attendees decide to head home, generally when the festival day ends or the event finishes, often simultaneously, and predominantly by car. This could be either by requesting a ride on their

mobile device or by using their own vehicle. This not only creates a chaotic environment in the region where the event takes place but also poses a risk to the lives of some inebriated drivers who may lack the motor capacity to drive safely.

Let's consider last year's Primavera Sound attendance as an example. Assuming that **half** of the attendees use a car and share the ride with someone else, we are looking at between 11,000 and 15,000 cars. Even in the best-case scenario where all the cars are fully occupied, assuming the general idea that each vehicle can hold five passengers each, the same area is still overrun with between 4,000 and 6,000 vehicles. This clearly illustrates the scale of the problem and the need for a more efficient and safer transportation solution, which is missing from the current market.

Our project aims to address the risk associated with attending such festivals by developing an app as user-friendly as the ones currently available on the market, while taking into consideration the need for a greener carbon footprint, a cheaper price and a more fluid and organized traffic. With this application, the passenger will be able to choose their destination point within the region of Porto and will be assigned to one of the **STCP** Buses available for transportation. Each bus will be filled to maximum capacity, ensuring all assigned passengers reach their requested destinations using the most efficient routing possible, ensuring we make use of the minimum number of vehicles, while maximizing their occupation and optimizing client distribution. This not only ensures a greener and safer method of transportation for every client, but also a faster one. Based on **STCP**'s current infrastructure, each bus can accommodate eighty-five passengers. With a minimum of 10 buses per event, we can ensure availability for **850** passengers per trip, keeping in mind that the buses will make multiple trips throughout each event. From another perspective, we can also say the we are replacing around 170 cars (with 5 seats each) from the event traffic. It's important to note that, depending on the availability of **STCP**'s buses and bus drivers, the success of the application, and its efficiency in improving the passengers' lives, this number could easily be increased by allocating more buses or even creating unique buses with more space, which would exponentially improve the project's usefulness.

Our initial idea was to deliver each passenger directly to a destination of their choosing rather than to an **STCP** bus stop. However, we encountered a significant issue with this approach. Porto's streets are not fully adapted for buses – many are too narrow, and some turns would challenge even the most skilled drivers. As a result, buses would be unable to take passengers directly to their individually chosen destinations since it's impossible to determine in advance which streets are accessible and which aren't. This approach would risk routing buses into impassable streets or difficult turns, potentially causing traffic congestion and failing to deliver passengers to their destinations. Instead, we developed the app so the client can choose the closest **STCP** bus stop to it's destination, since with this approach, unlike the latter, we have a guarantee that there is at least one path that is navigable. By using this method, we would satisfy all the requirements needed: The passenger will be transported very close to their original location, and the bus would not have to navigate to each individual destination of their assigned passengers, which would oblige it to go to unadapted streets.

Moreover, this approach allows for the simultaneous drop-off of multiple passengers at each bus stop, significantly improving the efficiency of the transportation service. Instead of taking a longer, and often unnecessary, time to transport each passenger to their individual destinations, which all would differ, buses can now drop off several passengers at once on each stop, reducing travel time and enhancing user satisfaction. Additionally, by utilizing the existing **STCP** bus stops to drop off passengers, we can take advantage the already established **STCP** bus route paths defined by **STCP** themselves, which not only, as previously mentioned, are adapted to be navigable by bus, but most likely have a priority lane. This allows us to create a viable map for the buses to follow, eliminating the fear of buses being incapable of navigating certain streets. This not only ensures the feasibility of the bus routes but also contributes to the overall efficiency and reliability of the transportation service provided by the app.

To maximize service efficiency, buses should transport the maximum number of passengers at a given time. Therefore, the app would not operate on-demand but rather on a pre-defined schedule set by the service operator, to give time for clients to book their seats, instead of meeting race conditions where every person would rush to be placed in the earliest buses. As an example how it should work, let's assume a Primavera Sound festival day ends around 00:30-00:40. The operator should schedule the first batch of buses to be ready and waiting outside the venue by 01:00-01:15, prepared to board passengers and depart promptly. Additional services would need to be scheduled afterward to accommodate passengers unable to board the earlier buses, ensuring multiple trips are available. Without race-conditions, people would be comfortable booking their seats, fully knowing when they are due to depart and where they can be placed if have not yet made their reservation.

For when the interval between services should be set there are two approaches we could take: **deterministic time window** or **route time based**.

Deterministic Time Window: Predefine a time window between services, with a large upper bound, such as 2 or 3 hours, to ensure that every bus is available when needed. However, this approach carries the risk of significant time inefficiencies. For example, if the longest bus route takes only half an hour, waiting 2 hours would be excessive. On the other hand, if it takes nearly 2 hours, the schedule would align well.

Route Time-Based: After generating the routes, round up to the nearest hour based on the time required for the longest bus route, and use that as the time interval. This approach requires that the first batch of bus routes be generated and only then, after determining the time for each route, can the next service be scheduled. Since this service has not yet been deployed in the real world, our distance and time metrics may not be entirely realistic, leading to potential unforeseen challenges.

Recognizing that the **Route Time-Based** approach may not be fully reliable before testing the service multiple times, we have opted to proceed with the **Deterministic Time Window** method. This ensures that, in the early stages of the project, we provide a well-coordinated and efficient service to the public.

3.2 The Database

Managing the data generated by client requests is a critical aspect of this project. The efficient operation of the app, namely the aggregation algorithm, relies on the accurate and timely processing of user requests, which include their identifiers and chosen destinations. Not only that, the routing and aggregation algorithms require data about the existing **STCP** bus stops and routes, which will need to be stored and ready to be retrieved and used at any moment. In the next chapter, we will delve into the details of how we handle this task, crucial for the efficiency of the project.

For the application to function as intended, it would require a connection to a database that is available at all times. The database server would be hosted by Geolink, which will store the necessary data for the algorithm to operate effectively. The data will be stored in an encrypted and readily available manner, ensuring both **availability** and **secure storage** to prevent client data to be exposed, guaranteeing that the database remains online ready for both storage and retrieval, while also ensuring that the client's information remains private and secure. These two characteristics of the database are **essential** to implement, as we want to ensure that clients are not left without service when needed, and to protect their customary destinations from exposure, which could pose safety risks. Failing to address these concerns would contradict two of the core motivations behind this project, to ensure the availability of efficient transportation alternatives and guarantee the safety of event attendees.

3.2.1 The Application Data

There will be three essential tables of data needed for the algorithm to function, but all share a common factor: **Spatial Data** [12]. As such, we opted for using **PostgreSQL** [36], an open-source and free Relational Database Management System (RDBMS), which by itself doesn't support Spatial Data, but can be extended with **PostGIS** [35]. **PostGIS** [35] is a free and open-source software that adds support for Geographic Objects to the aforementioned RDBMS, as well as many mathematical functions related to such objects. This option will ensure we have easy access to the data, while also supporting warehousing and guaranteeing fast data retrieval at the request of our app and server.

1. The client's bookings

This table, comprised of four main attributes, will store the data representing the bookings made by clients. The attributes are as follows:

Booking ID: This is used for subsequent warehousing and identification of each individual booking. Primarily useful for relating each booking to a client, it should be a unique value to avoid overlap and collision with other data rows, ensuring no data is accidentally deleted or replaced. It can either be a Serial Integer or a **UUID**, as long as it is unique for each row.

Client ID: This attribute means to relate each booking to the respective client, ensuring each booking is confirmed and associated to a client. As with the previous attribute, can either be a Serial Integer or a **UUID**, as long as this value exists (and its unique) in the clients table where we will store the users data.

Destination: The **STCP** bus stop identifier the client chose.

Time: Timestamp of when the booking was made. While the project is defined to have a limit of seats for each scheduled service, it could be eventually transformed into an unlimited seating service where our own algorithm will decide how many buses and laps will be needed optimally. We still use here because it could be useful to analyse (Data Mining). With this attribute, we could get insights such as: "Do the clients tend to book earlier or closer to the deadline?", "Do clients try to make bookings simultaneously to the same destination in order to go together or most of them book by themselves?", ...

Assigned Bus: Bus identifier for the bus assigned to be responsible for the booking. This identifier is generated only after the booking has been processed or, in other words, been used by the algorithms. It is essential for informing each client which bus they should board.

2. The Bus Stops available to drop off passengers

This table will be responsible for storing the **STCP** Bus Stops data. It has three main attributes:

Stop ID: The string of characters used by **STCP**, to uniquely identify each individual stop, which anyone can search its location by using **STCP**'s website tool: <https://www.stcp.pt/pt/viajar/paragens/>

Stop Name: Also a string, identifying the full name of the stop but, unlike Stop Id, can be the same on multiple stops, but provide a sense of location to the clients. For example, where Stop ID is something like 'JB1', Stop Name will be 'Jardim Botânico'

Stop Point: Coordinates representing the geographical location of the stops, using the 'POINT' type made available by PostGIS

3. The Bus routes between stops

This table will be responsible for holding the data representing the links between 2 bus stops, in a directed manner, while also holding the geographical representation of it:

Origin Stop ID: The **STCP** bus stop ID that is the point of origin of the route. It is a foreign key to the attribute Stop ID within the bus stops table.

Destination Stop ID: The same as the initial attribute but for the destination stop.

Route: An attribute using PostGIS's 'LINESTRING' type, used to store the geographical route between the origin and destination stops

The first table, responsible for storing the bookings data, is populated dynamically by the app's use by clients and can be trimmed down over the time to prevent overuse of memory, while the other 2 need STCP's data to be stored beforehand and permanently. But before explaining how SCTP's data was retrieved, 1 more data table must be defined.

There is a need to relate the routes to the buses. Therefore, we need to create a table that will be able to relate the buses with their next assigned route. Accordingly, the following table was created, in order to store the output of the routing algorithm:

Bus ID: A Bus identifier, which can be either numeral or alphabetic depending on the preferred system, that is unique per available bus.

Assigned Route: A 'MultiLineString' (PostGIS data type) representing the generated path for the next service (route generation algorithm output)

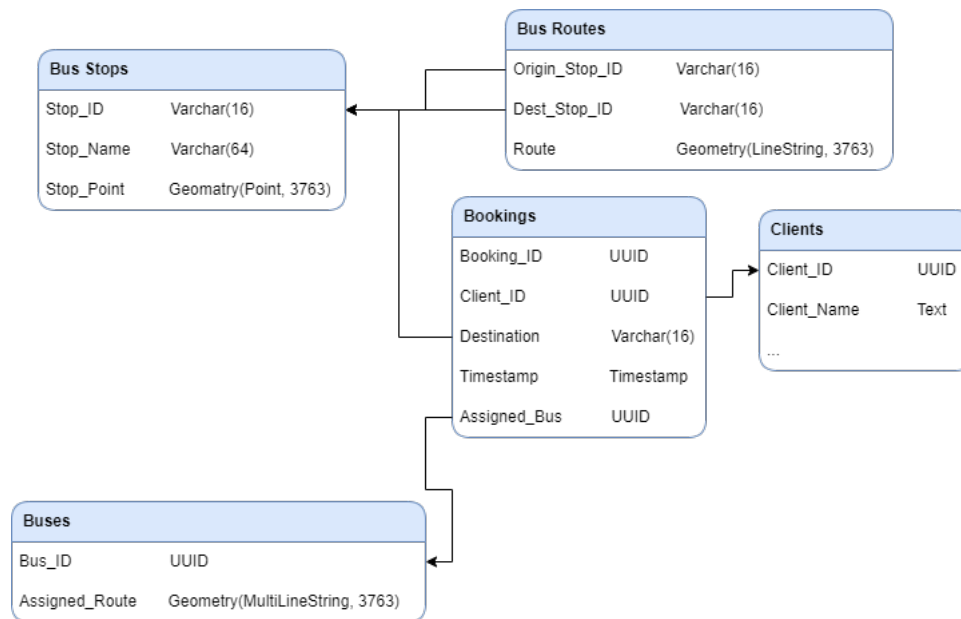


Figure 3.1: ER Model

3.2.2 SCTP Data

To retrieve the data needed to populate both the **STCP**'s stops and routes table in an easy and replicable manner, without any manual insertions needed, we developed a JavaScript program. This program uses network requests to retrieve the necessary information and converts it into a **CSV** file that can be read by PostgreSQL [36] and used to insert the data into the respective tables.

By creating GET requests to:

stp.pt/pt/itinerarium/callservice.php?action=linestops&lcode=*CODE*&ldir=*DIR*

in a loop, where *CODE* is assigned a value retrieved from a list of available *STCP* route codes and *DIR* is assigned values 0 and 1 (original direction of the route and the inverse route), we are able to retrieve (using network data) not only **the stops** each route goes through, but also **connect each pair of consecutive stops** as a **Route**, creating the full network mapping of *STCP* operating zone. This will populate the stops table with all the existing *STCP* stops and, for the Routes table, create a new row for each consecutive stop in an existing *STCP* Route.

For example, in *STCP*'s route 200, the original route starts at stop Bolhão with ID BLRB1, then goes to Mercado do Bolhão with ID MCBL3, and the third stop is Praça D. João I, with ID PRDJ. This previous script will automatically create a stops table data entry for the previous 3 stops but also create 2 new routes on the respective table: starting on Bolhão with destination Mercado do Bolhão and another entry starting on the latter with destination Praça D. João I.

In case there is a creation of more stops, alteration of routes, or even addition of more, it's easy to simply rerun the program, adding the new route codes to the list. With a “click of a button”, we can retrieve the updated data to *CSV* files and automatically import them to the hosted database, being able to keep our data updated easily, being replicable in any situation, even in the case of data deletion.

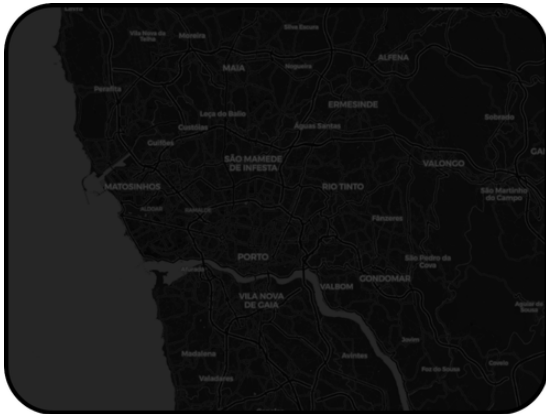


Figure 3.2: Zone of Grande Porto



Figure 3.3: STCP Coverage Heatmap

As illustrated in the image 3.3, comparatively to the entire zone shown in the image 3.2, *STCP* bus routes offer comprehensive coverage, serving the entirety of Porto as well as the surrounding regions. However, this also results in an overpopulation of routes and points, leading to a saturated table with excessive information. This could pose a challenge to data retrieval and, more importantly, significantly increase the computational time of the algorithms. Our goal is to find the fastest possible path between the assigned route points (bus stops) for each bus. Essentially, this is an implementation of the Traveling Salesman Problem (a minimum-cost Hamiltonian circuit returning to the starting vertex), which does not have a Polynomial Solution and belongs to the NP-Complete class of problems. Overcomplicating the routing algorithm with excessive iterations due to the large volume of data, would transform any simple generation

of bus routes into a time-constrained, infeasible problem. Therefore, in order to surpass this obstacle, we would need to do something about the **STCP** network data stored.

3.3 The Network

In the initial stages of development, it was necessary to simplify the network of routes and points provided by **STCP**. Upon retrieving all the necessary data for routing, we found ourselves with a substantial number of stops and routes - precisely, **2941 Stops** and **2891 Routes**, with a route being considered a path between 2 points.

However, a closer examination of the **STCP** routes revealed that a significant portion of these nodes could be deemed redundant for our purposes. This redundancy arises from the fact that many stops are uniquely associated with a specific route, offering no alternative routing options. In other words, when a bus arrives at one of these stops, its next destination is predetermined - it is compelled to proceed to the next stop defined by the route, with no other choices available.

This characteristic of the **STCP** network allows us to make a much needed simplification: we can disregard the stops that are linked to only one destination. By doing so, we focus on the stops that offer multiple potential paths, which are the points where routing decisions actually need to be made. This approach significantly reduces the complexity of the network we need to consider, making the task of route generation and optimization more manageable and much more efficient.

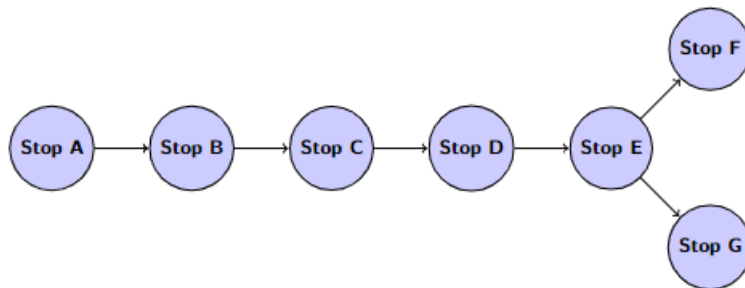


Figure 3.4: Network

The figure 3.4 is an illustrative example of the previously described property of nodes, that is found in several routes. The small graph has represented seven Bus Stops as nodes, where each arrow between 2 nodes indicates that there is at least one existing **STCP** bus route connecting the two. For instance, if there is at least one **STCP** bus line that, after passing through Stop C, goes to Stop D, there is a link from Stop C to Stop D. In this example, unlike the other 6 stops, Stop E has at least one **STCP** route that goes to Stop G and at least one **different** route that goes to Stop F. For example, take into consideration the routes 201 and 303. In both routes, there is a stop at Boavista-Casa da Música (code BCM1). However, on route 201, the bus proceeds to the Agramonte stop (code AGM1), while on route 303, it continues to Boavista-Cemitério

(code BVCM3). In our project, which utilizes all available paths, this stop exemplifies a stop with Stop E characteristics from image 3.4. At this stop, the bus must choose its next direction, unlike most other stops that are served by a single bus line, offering no such choice.

From a programmer's perspective, there is no need to maintain a representation of stops that do not offer more than one routing option, as these only offer a single mandatory destination. In this example, if there is a bus at Stop A, there is no decision on which stop to go to until we reach Stop E. This means that, on Stops A, B, C, and D, we will be obliged to follow to their single available destination stop until we reach Stop E, where we finally can opt to either go to Stop F or Stop G. Even in the case that the last passenger of the bus exits on Stop B, the bus remains on track until the last stop of the route both because there may still be passengers awaiting of the following Stops, but also because the bus can't simply turn around on most streets. The same principle applies to our paradigm: the bus needs to follow the path until there is a node with more than 1 option.

With that in mind, we can simplify this 7 Nodes and 6 Edges network to a **simplified** version of itself but **representative** of the same path data:

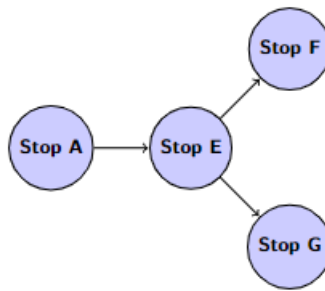


Figure 3.5: Network Simplification

As demonstrated by the figure 3.5 we remained with only 4 Nodes and 3 Edges, a **reduction** of 57% of Nodes and 50% of Edges, while keeping the **same structure** as the figure 3.4 from a high-level view. Those are the percentages we aimed to get when this formula is applied to the real data.

3.3.1 Network Creation

To transform the data into a network, we created a Python script that applies the following algorithm to the database tables:

1. Create a table of Nodes, similar to the Bus Stops table from image 3.1. Unlike the latter, this table will not contain all available stops, but will only include the data that is accepted by the following criteria:
 - Is a Route Starting Stop, that is, a Node that **starts the bus line**. It is identified in

- the data by not appearing as a destination node in the table representing the routes, but appears at least once as an origin node.
- Is a Route Ending Stop, that is, a Node that is the **final destination of a bus line**. In the data representing the routes between two nodes, it is easily filtered by appearing as at least once as a destination node but not as an origin node.
 - Has, **at least**, two outgoing links to different nodes, regardless of the number of incoming routes.
2. Iterate over all the nodes linearly and, for every node, check if the previous criteria verifies. If it doesn't verify, discard the node and proceed to the next iteration;
 3. If the node does contain any of the necessary characteristics, we add them to a list;
 4. Retrieve all the outgoing edges that the current node is the origin of;
 5. Iterate over all the retrieved edges and, for, every edge, we keep following the path while the destination node has only 1 outgoing edge itself, meaning is a **redundant** node;
 6. When the destination node is **not** a **redundant** node, we insert a new tuple element on the list of new edges containing the first node (the first loop element) and the last node (the nested loop element destination);
 7. By the end of the nested loop, we will have a list containing all the **nodes accepted** by the filtration criteria and another list containing tuples of the **new edges**;
 8. We insert all the elements of the two lists in the database in their respective tables;

The previous algorithm can be represented as the following pseudocode:

```

FilteredNodesList = []
NewEdgesList = []
Nodes = getAllNodesFromDatabase()

for Node in Nodes:

    outgoingLinks = GetAllOutgoingLinks(Node, RoutesTable)
    incomingLinks = GetAllIncomingLinks(Node, RoutesTable)

    if len(incomingLinks) >= 1 and len(outgoingLinks) == 1:
        continue

    FilteredNodesList.append(Node)

    for Edge in outgoingLinks:
        destination = Edge.destination
        path = [Node]

        while GetOutgoingLinkCount(destination, RoutesTable) == 1:

```

```

        nextNode = GetOnlyOutgoingLink(destination, RoutesTable).destination
        path.append(destination)
        destination = nextNode

    NewEdgesList.append(path)

InsertIntoDatabase(FilteredNodesTable, FilteredNodesList)
InsertIntoDatabase(NewEdgesTable, NewEdgesList)

```

This algorithm is based on the Douglas-Peucker algorithm [9]. The Douglas-Peucker algorithm is a widely recognized technique used in computer graphics, geographic information systems (GIS), and data compression to simplify a curve or a series of points. The primary objective is to reduce the number of points in a polyline or curve while retaining the essential shape and structure of the original line. The algorithm achieves this by recursively evaluating points and removing those that fall within a specified distance (tolerance) from a straight line drawn between two endpoints. By doing so, it effectively reduces complexity while preserving the most significant features of the shape [19].

The advantage of using the Douglas-Peucker algorithm lies in its balance between simplicity and computational efficiency, making it suitable for both real-time systems and large datasets. It is particularly useful in GIS applications where the simplification of geographic boundaries and transportation networks is crucial for visualization and storage [41]. Additionally, it is employed in trajectory data processing to reduce the number of GPS points without losing meaningful information about movement patterns [27].

While the original algorithm uses a distance threshold to decide which points to remove, our adaptation involves a more context-specific criterion. Instead of using a strict distance-based tolerance, we remove every point that falls under a redundancy property: any point that has only one outgoing edge and is not a bus line starting or ending node is eliminated. This ensures that unnecessary, isolated nodes—those which do not contribute significantly to the overall structure of the route network—are removed, while still maintaining the critical integrity of the transportation paths.

Our approach builds on the Douglas-Peucker algorithm's principles but adapts them to account for the unique requirements of transportation and routing systems of our project, ensuring that we achieve an optimal balance between simplicity and accuracy without introducing gaps in the network's functionality.

After filtering the data in the dataset, we retain all these nodes in a list. We will iterate over this list, attempting to formulate a simplified path between them. Not between each one to every other one, but as explained previously, by suppressing the redundant stops represented in the initial table.

We now have two tables representing the network: the **Nodes** table and the **Edges** table. The Nodes table follows the same structure as the Stops table, while the Edges table mirrors the

structure of the Bus Routes table (both represented in the figure 3.1), with the addition of a new Node ID and Edge ID, respectively. These tables contain the data output by the previous algorithm. In the Nodes table, only non-redundant nodes are included, and in the Edges table, we only have links between pairs of stops appearing in the previous table.

While the data structure defined effectively represents the network of routes and stops in a simplified yet comprehensive manner, there is one important piece missing from the network. Although the network of Stops and Routes has been compressed, the suppressed stops may still be selected by the clients and, naturally, need to be represented somehow and also need to have a path to them.

To address this, we created a new table, **Edge Attributes**, that serves as metadata for the newly created edges by associating Edges with suppressed Nodes:

Edge ID: A foreign Key to the Edge ID in the Edges table, relating each row to a created Edge.

Middle Stop ID and Name: The Stop ID and name of the suppressed Stop, which the Edge with the given Edge ID passes through.

Position: An integer representing the position of the Stop on the new Edge, starting at 1.

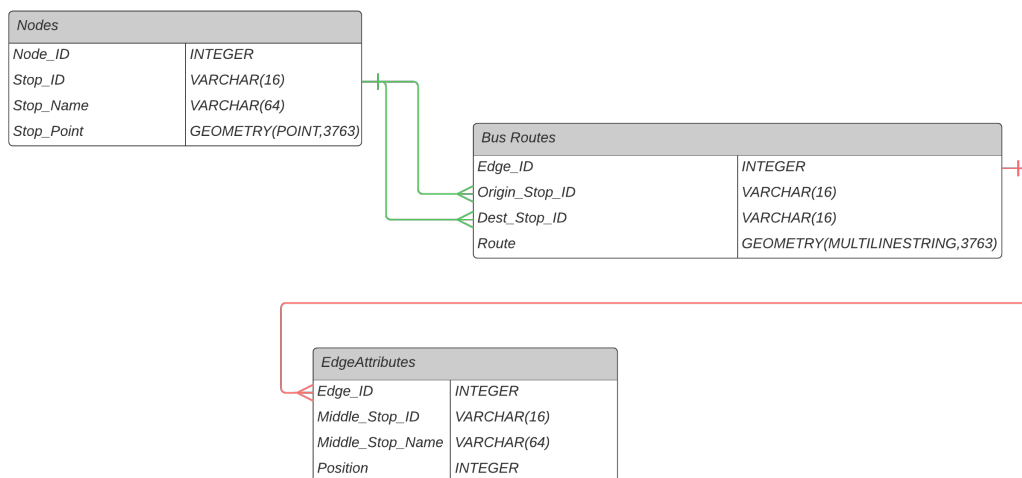


Figure 3.6: ER Model

In order to populate this new table with data, we need to add an extra step to the previous algorithm. While we are iterating through the inner loop, when we follow the edges between the current element of the outer loop and the next node that is not redundant, we keep creating a tuple containing the Node and the position it is in, where position means the sequential position it is on the new edge, and inserting it on a new list. When the outer loop ends (we reach a not redundant node), we insert this new list on another one, containing all the lists of tuples of redundant nodes (eg.: $[[('EXAM', 1), ('PLE', 2)], [('ANOT', 1), ('HER', 2)]]$ – Demonstration

of the data structure) where each list of tuples is respective to a newly created edge, storing it's corresponding omitted stops. Storing the data in this manner enables us to establish a relationship between the elements of the new edges list and their various redundant nodes, given that each position in the list of new edges and this new list will be synchronized.

Using this structure to relate all the nodes to the edges, we have not only the reduced number of Nodes and Edges to help fasten the computation of the algorithms, but also ensure all Stops are accounted for. If a Client were to choose a stop which was suppressed from our simplified data, we only need to get the Edge ID(s) of the Edge Attributes table rows where that stop appears on. As might be expected, there may be more than 1 Edge that goes trough it, as it happens, for example, on stop Asprela, with ID ASP3, that is suppressed from 4 different Edges. This happens because ASP3 is the destination option of 4 different nodes, while it has only 1 destination node itself.

But how did this transformation affect our data dimensionality?

Let's address the statistics:

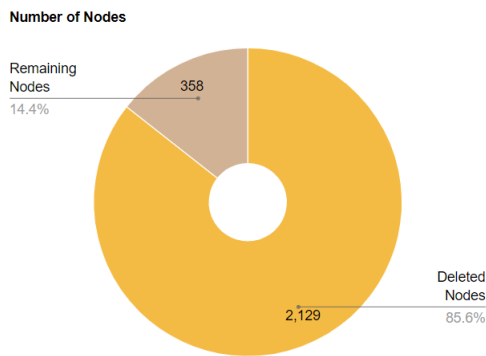


Figure 3.7: Nodes Piechart

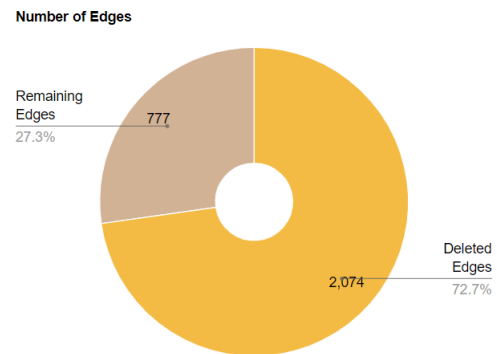


Figure 3.8: Edges Piechart

As seen in the figures 3.7 and 3.8, we were able to **significantly** reduce the size of our initial data set, which comprised a substantial number of nodes and edges. Specifically, we achieved a reduction of 72% in the number of edges and an impressive 85% decrease in the number of nodes. This substantial simplification is instrumental in enhancing the efficiency of the app's algorithm, as it will reduce by a **vital amount** the **computational time**. Importantly, despite this reduction, the simplified data set maintains the integrity of the original data, effectively representing the same information without the inclusion of redundant data. This optimization strikes a balance between data fidelity and computational efficiency, paving the way for the app to deliver reliable and timely results, while also being able to reproduce the same map as before.

Note: Naturally, some **STCP** routes have final stops with no outgoing connections. In reality, when a bus completes its route, it moves to a nearby stop to begin the return journey, instead of placing itself on the same one. To reflect this, we manually connected these stops to ensure full connectivity between all stops and avoid routing issues. As a result, buses can travel continuously

without encountering dead ends.

3.3.2 Network Visualization and Comparison

Using [Gephi](#), an open-source network analysis and visualization software, we can get a better insight on how [STCP](#)'s network is and how it compares to our new simplified version of it:

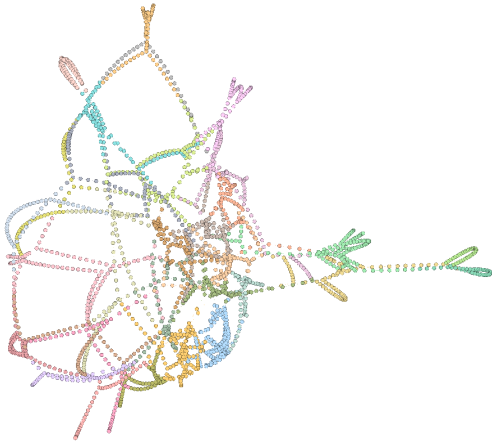


Figure 3.9: Original Network

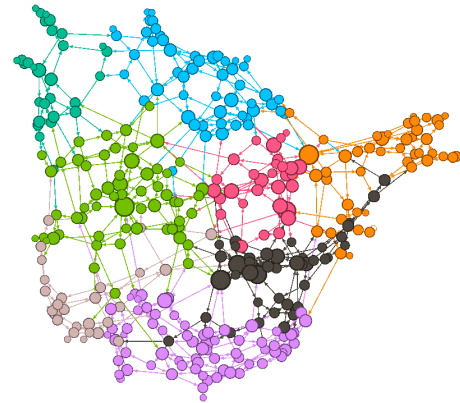


Figure 3.10: Simplified Network

Upon closer examination, it becomes evident that our simplification process effectively eliminates the prevalent one-way paths that were predominant in the original network. Whereas the original network could be characterized as having a central core with numerous tendrils extending outward, the simplified network now presents itself as a unified and singular core. This transformation reflects a consolidation of the network's structure, resulting in a more streamlined and cohesive topology. By removing the excess of straightforward paths found in the original network, we have essentially trimmed the network's complexity and sparsity, paving the way for more efficient routing and navigation. Moreover, this shift towards a singular core not only enhances the network's visual clarity facilitating an easier interpretation and analysis, but also, simplifies its underlying logic.

By using the same tool, we can easily get some comparative metrics of the 2 networks, further demonstrating the effectiveness of the optimization and its advantages:

	Avg. Degree	Max Degree	Min Degree
Original Network	2,325	11	1
Simplified Network	4,247	12	1

Table 3.1: Degree Comparison

	Avg. Out Degree	Max Out Degree	Min Out Degree
Original Network	1.162	6	0
Simplified Network	2,128	6	0

Table 3.2: Out-Degree Comparison

	Avg. In Degree	Max In Degree	Min In Degree
Original Network	1,162	6	0
Simplified Network	2,128	7	0

Table 3.3: In-Degree Comparison

In the analysis of degree distribution, presented on table 3.1, a notable distinction emerge between the two networks. Although there is minimal disparity in the maximum and minimum degrees, particularly evident in the minimum degree due to the retention of nodes initiating or concluding routes, typically possessing only one outgoing or incoming edge by default, an **important** discrepancy arises in the average degree metric value, that is doubled from the original to the simplified network. As for the increase in 1 in the maximum degree metric value, is consequence of redirecting the multiple incoming edges of preceding and redundant nodes to the next non-redundant one.

By looking at the table 3.2, we can notice that, in the initial network, the majority of nodes exhibit an average out degree of one link, indicating that the majority lack multiple routing options, given the impossibility of nodes being completely isolated. This corroborates the impact of our simplification process, which crucially enhances the availability of multiple path options for routing, thereby significantly facilitating the optimization of the algorithm to obtain the best path.

The table 3.3 also confirms that most nodes function primarily as intermediary points within the original network. This is evidenced by the fact that the majority of nodes receive, approximately, only one incoming connection, indicating a high probability that the preceding node in the sequence is merely serving to link two other nodes. Retaining all nodes from the original network would render many calculations redundant; thus, our streamlined approach selectively preserves essential nodes, doubling the connectivity of each node to approximately 2. Consequently, what would have entailed a 30-step iteration loop in the routing algorithm, is now condensed into a mere 2 or 3 iterations, exemplifying the efficiency gained through our approach. This is even more evident on the next metric comparison table:

	Diameter	Avg. Path Length
Original Network	136	44.11
Simplified Network	21	8.17

Table 3.4: Path Comparison

When it comes to emphasize the advantages this simplification yields, the table 3.4 serves as a superior representation compared to other metrics. The diameter, signifying the maximum number of steps required from any node of the network to reach another, portrays these advantages clearly. Within our original network, we had at least 1 node necessitating 136 edges to reach a specific destination. In computational terms, this translates to 136 iterations to traverse a path between these two points, a figure significantly reduced to a mere 21 steps in our simplified network—a remarkable decrease of approximately 75% in path length. This enhanced efficiency is further evidenced by the average path length. Previously, an average of approximately 44 edges were required from a given node to access all others. In the updated network, this number has been drastically reduced to just 8, representing a reduction of around 81%. The computational benefits of this improvement are significant. By more than halving the steps required, on average, to form a path between two nodes, we have made a valuable advancement in optimizing the network. Notably, this optimization was achieved even before making any adjustments to the algorithms themselves.

In the context of network analysis, particularly in networks that faithfully represent geographical data like ours, centrality measures are instrumental in pinpointing the most significant nodes within a graph. Not only we can get valuable insight, we can use that valuable knowledge to optimize and simplify the existing bus connections. Two important measures to evaluate are **betweenness centrality** [14, 15] and **closeness centrality** [2, 14].

Betweenness Centrality [14, 15] quantifies the frequency with which a node serves as a bridge along the shortest path between two other nodes. This measure provides insight into the degree of influence a node wields over the interactions of other nodes within the network. The betweenness centrality $g(v)$ value of node v is calculated with the following formula:

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \quad (3.1)$$

where :

- σ_{st} is the total number of shortest paths from node s to node t .
- $\sigma_{st}(v)$ is the number of those shortest paths that pass through node v (where v is not an endpoint).

By visualizing the network on figure 3.11, where both node size and color correspond to the metric values, it is evident that two stops stand out as the most dominant:

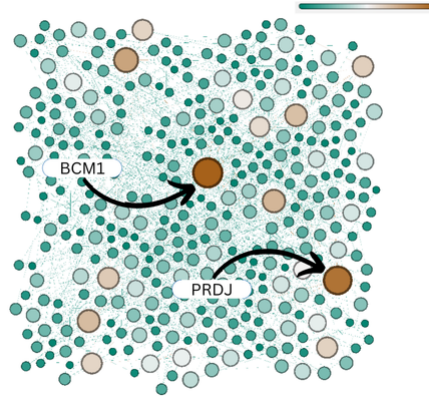


Figure 3.11: Simplified Network Betweenness Centrality

On this metric, for the simplified network, the bus stop at Boavista, Casa da Música (BCM1), emerges as the most central node, followed by Praça D. João I (PRDJ1) which both distinguish themselves from the rest of the nodes on this metric.

Comparatively to the original network demonstrated in the figure 3.12, we notice some differences:

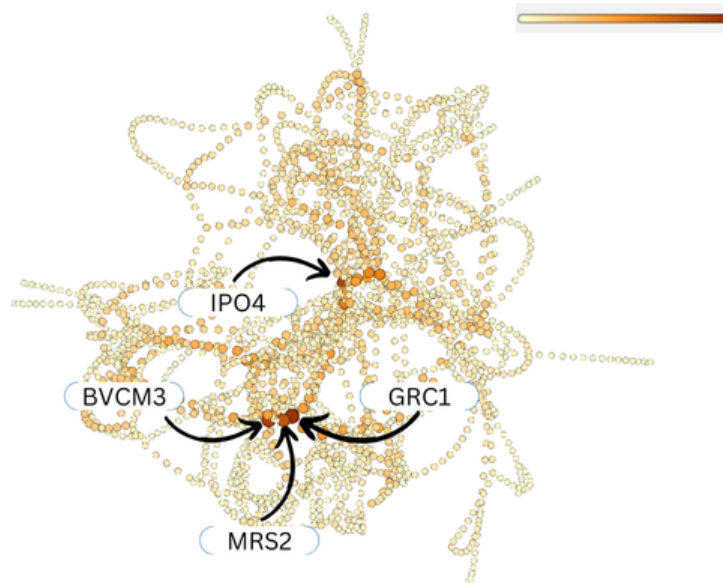


Figure 3.12: Original Network Betweenness Centrality

Unlike the simplified network, the original network displays a more balanced distribution of color, reflecting a similar percentage of nodes with low, medium, and high values. Given the formula for this metric and the uniqueness of most **STCP** bus routes—where many use distinct stops—some nodes are only utilized once, with 1 or 2 incoming or outgoing links, positioning them as potential intermediate nodes. Others, however, are used across multiple routes which, consequently, makes them a viable option for the shortest path of many other nodes. Nevertheless,

certain nodes still stand out, such as IPO (Circunvalção) (IPO4), Boavista Cemitério (BVCM3), Graciosa (GRC1), and Moreira Sá (MRS2), which are not apparent in the simplified network.

Besides the last metric we have **Closeness Centrality** [2, 14]. **Closeness Centrality** [2, 14] is a measure of the proximity of a node to all other nodes within the network. The closeness centrality $C(v)$ of a node v is given by the following formula:

$$C(v) = \frac{n - 1}{\sum_{u \in V} d(v, u)} \quad (3.2)$$

where:

- n is the total number of nodes in the graph.
- V is the set of all nodes in the graph.
- $d(v, u)$ is the shortest path distance between nodes v and u .

In this regard, the three most central nodes in the **STCP simplified network** are the Bus Stop on Avenida dos Aliados (AAL1), the bus stop at Boavista, Casa da Música (BCM1) (appearing once again), and the bus stop at Graciosa (GRC1). However, by analysing the network demonstrated on Figure 3.13, where both node size and color correspond to the metric values, we notice that these values are not substantially different from the majority of the network:

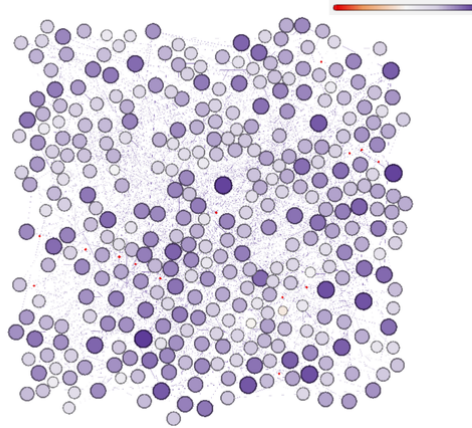


Figure 3.13: Simplified Network Closeness Centrality

Contrary to the figure 3.11, where some nodes would be more "central" and better positioned than others, we observe that the distribution mirrors the pattern in Figure 3.12. The Closeness Centrality values are evenly distributed across the nodes. This outcome is expected, as the removal of intermediary nodes—those that connected non-redundant nodes—has reduced the path length between most nodes. As a result, the Closeness Centrality values for these nodes have, inversely, increased. Avenida dos Aliados (AAL1) leads this metric, followed by Boavista Casa Da Música (BCM1) and Graciosa (GRC1)

As for the original network, we also have an interesting distribution of values, represented in the Figure 3.14:

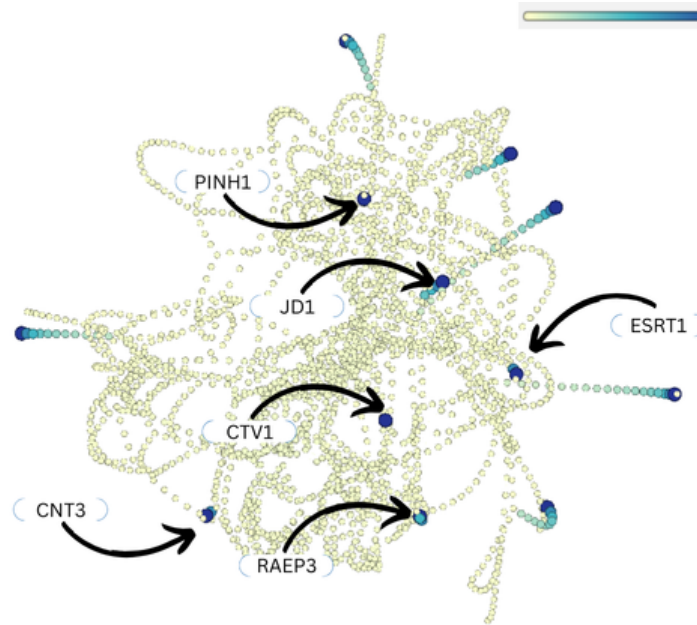


Figure 3.14: Original Network Closeness Centrality

Immediately, we notice a large number of dark blue nodes, indicating the highest Closeness Centrality values. However, as previously mentioned, one characteristic of **STCP** routes makes the original network unsuitable for this metric. Since some stops are the final ones on their routes and this is a directed network, these last stops have no outgoing edges. As a result, the stops before them only have one destination, leading to a favorable path length, as it's the only available route before the path terminates. This explains the color gradient from light green to dark blue along those routes, with the final stop showing a Closeness Centrality value of 0 (since there are no outgoing edges on the node, the path length is 0 and, consequently, we have a Closeness Centrality value of 0). It is important to note that this issue only occurs in the original network, as all nodes in the simplified version have outgoing links. Additionally, this phenomenon is specific to Closeness Centrality and does not affect Betweenness Centrality, which measures how often a node is used as part of the shortest path between other nodes, whereas Closeness Centrality focuses on a node's own paths.

In order to better visualize the **original network** on this metric, we transformed the network from a directed to an undirected one, which instantly presents a very distinct distribution of values, as seen in the Figure 3.15:

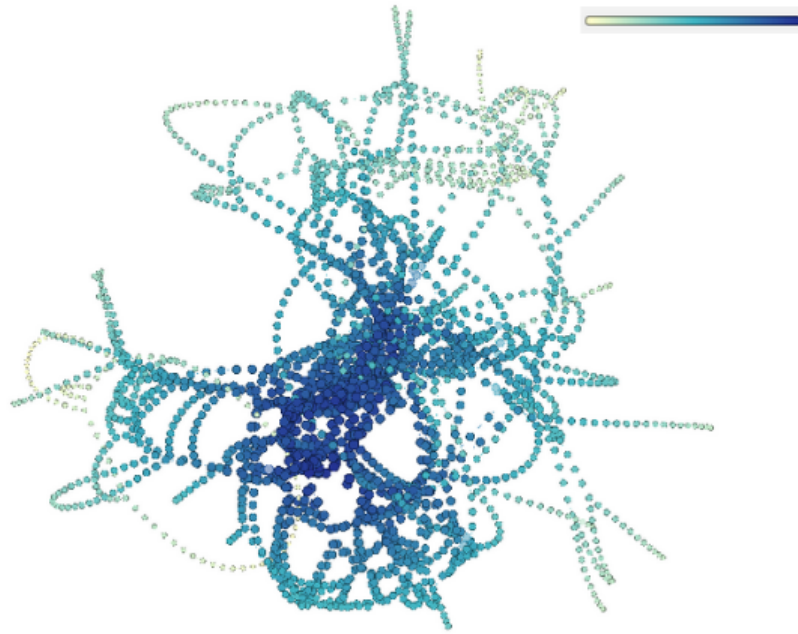


Figure 3.15: Undirected Original Network Closeness Centrality

The new undirected network clearly differentiates itself from the directed network by exhibiting a more balanced and logic distribution. This metric is led by the stop Boavista Cemitério (BVC3), followed by Graciosa (GRC1) and, lastly, by Boavista Casa da Música (BCM3).

Understanding and applying these centrality measures allows us to gain valuable insights into the structure and dynamics of a network. In the context of our work, these measures could be instrumental in optimizing the transportation time by identifying key bus stops (nodes) in the network.

In spite this dissertation being focused on the development of our app, which is mainly motivated by the traffic caused by events of large concentrations of people, this problem spreads all throughout Porto. By analysing the network centrality measures, we can retrieve some important optimization ideas regarding this current problem, that could optimize even further the time of transportation of passengers, whether in the use case of our application or the daily routine **STCP**'s services:

1. Highest Betweenness Centrality

Nodes with the highest betweenness centrality are crucial for maintaining efficient routes through the network since, in our context, it represents the frequency which Bus Stops are used as the shortest path to other nodes. Since our app intends on shortening the travel time as much as possible, bus stops with the higher values on this metric should be the focus of some optimizations, since they will most likely be used by most buses as the shortest path to their next stop.

- Given the frequency with which these nodes are traversed—since they form part of many of the shortest paths between other nodes—connecting these stops to geographically proximate stops will further augment the value of the stop. This is because they will provide new shortest paths to other nodes. While this could be done for every node to balance this value between the network nodes, creating bus routes and dedicated paths incurs costs. Therefore, if we are to invest in this strategy, it would be most beneficial to implement it at these high betweenness centrality nodes as these nodes are likely to be chosen for the majority of paths to other nodes in most cases. By establishing more direct routes between geographically proximate nodes, we could reduce the additional time currently spent navigating indirect routes to other close stops.
- Where feasible, we should consider replacing existing traffic lights around these bus stops with bus dedicated ones. This would enhance the fluidity of travel to and from these nodes, thereby reducing travel time. This strategy would further leverage the central role of these nodes in the network, making public transportation more efficient and user-friendly.

2. Highest Closeness Centrality

Nodes with the highest closeness centrality are also vital for maintaining quick access to all other nodes in the network. In our context, it represents stops that access the other nodes in the network the fastest. Bus stops with the higher values on this metric should also implement optimizations, as they are most likely to provide the fastest routes to all other stops in the network

- Establishing new direct bus routes that connect nodes with the highest centrality values. This strategy creates a faster travel path to nodes that are optimally positioned for rerouting the bus to the requested stops. In essence, nodes with high centrality values are likely to provide the most efficient paths to other nodes in the network. By linking these nodes with a dedicated route, we enhance routing possibilities and avoid unnecessary travel to redundant nodes inherent in existing routes.
- Given the central role of these nodes, it would be beneficial to establish Metro Stations with Bus Interfaces in their vicinity. This approach would enhance the connectivity between the two public transportation methods. By introducing new metro stations at these nodes, we ensure that passengers can conveniently switch to a bus service offering a fast route to their desired destination in cases where the metro network does not cover it.

Having established a comprehensive understanding of the database network and its characteristics, we now turn our attention to the core section of the project: **the app's algorithm**.

3.4 The Algorithm

The algorithm, which is the main focus of our study, leverages the data from the database network to perform its work. The intricate interplay between the database network and the algorithm forms the backbone of our application. Consequently, the algorithm's effectiveness and performance are inherently linked to the quality and structure of the data from the database network. This underscores the importance of the simplification process applied to our initial data.

The algorithm is bifurcated into two primary components: the **aggregation of bookings** and the **route generation**.

Before we delve into the intricacies of each component, let's first elucidate the fundamental premise of the algorithm at a high level, represented by the Figure 3.16:

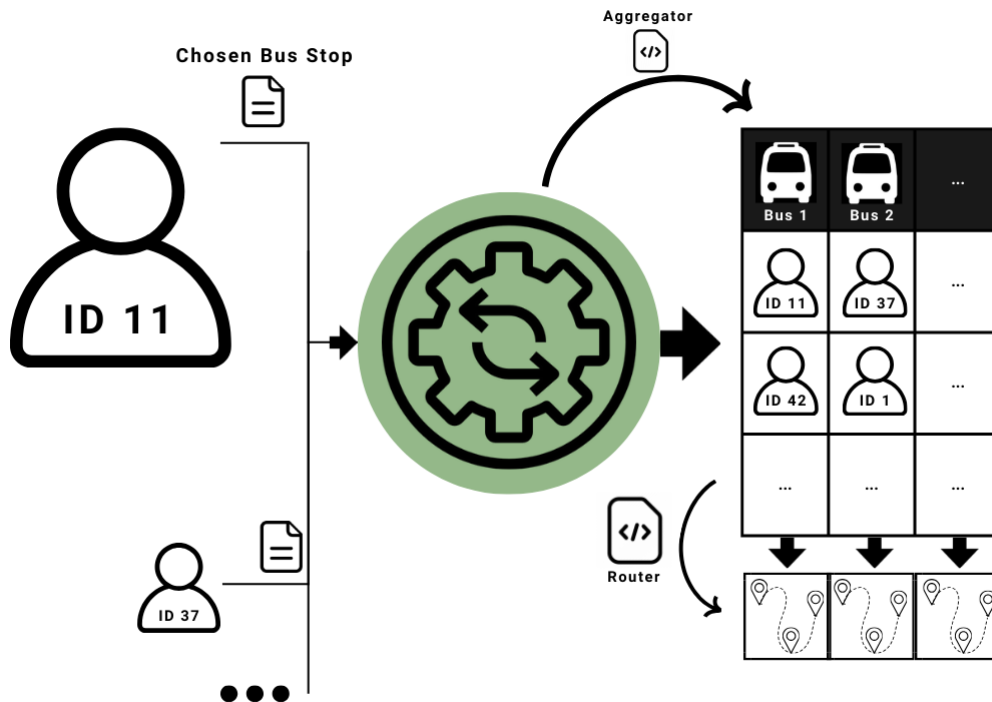


Figure 3.16: Algorithm Architecture

At first, clients reserve their bus rides through the web application, with each booking detail stored in the database. While there is no limit on the number of bookings a client can make, the number of passengers that can be accommodated on the buses for a given event is limited. As a prerequisite for the project's functionality, it is essential to specify the number of buses available for an event, their capacity, and the number of laps they will complete. This information ensures that our application can offer clients access to a bookable service, displaying the maximum capacity, the number of already booked clients, and the date and time of each service start.

At a pre-established time, which is determined based on a sufficient interval prior to the defined time when the bus is scheduled to commence its route, the server requests the database for all the bookings made for that respective service. This interval is designed to accommodate the running time of the algorithm and the physical gathering of passengers on their respective buses and, as such, should be 15 to 30 minutes before the buses take off, depending on the logistics and organization of the event and its location. To ensure that everyone has the opportunity to book their service while seats are still available, we will run the algorithm only during the pre-defined period before the event begins, guaranteeing that all participants are accounted for. There is also the possibility of running the algorithm when there is a service that is fully booked, however, due to the possibility of client cancellation, we would be wasting time and resources computing an output that could be entirely different from when the event actual takes place.

After retrieving the needed data, each booking is allocated to one and only one bus, until every booking is placed. Having each bus filled to maximum capacity (or the maximum number of reservations made), the route will then be generated based on each buses respective bookings.

Each bus driver will be informed of their assigned route to their **GPS**, as the passengers will be informed in which bus to enter by SMS.

3.4.1 Aggregation Of Bookings

The aggregation of bookings serves as the initial component of our application's algorithm. Its primary aim is not only to simplify the subsequent route generation component but also to minimize the total travel area a bus is responsible by, which translates to shorter travel time and, thereby, enhancing passenger satisfaction.

Given our objective to reduce travel time to the greatest extent possible, it would be advantageous to group individuals who intend to travel to the same destination or, more broadly, the same area. Consequently, we have implemented the following Greedy Algorithm:

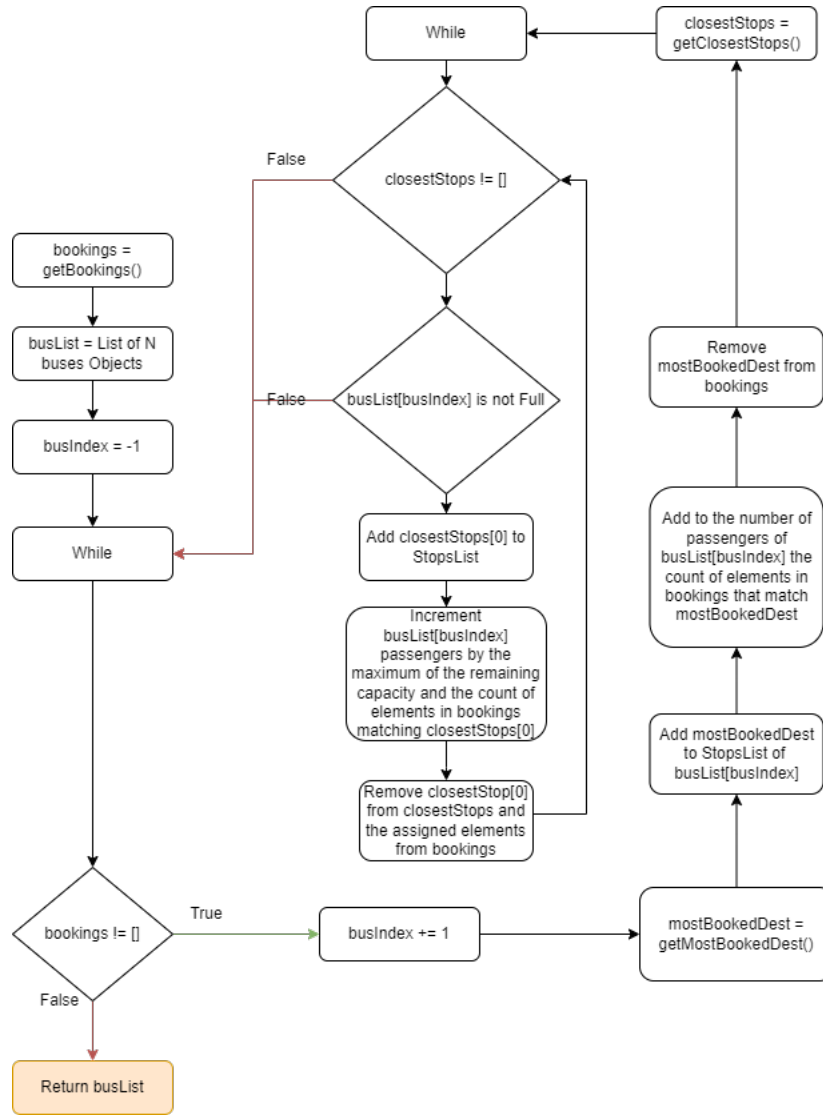


Figure 3.17: Booking aggregation algorithm Flow Chart

The flow chart presented in the figure 3.17, illustrates the logistics of the **Aggregation** algorithm. First, we retrieve all bookings for a specific event date and time, placing them into a list called 'bookings'. Next, we initialize 'busList' with N bus objects, where N **must** be predefined as the total number of buses available for the service. We set 'busIndex' to -1 and enter a while loop to process each booking in the 'bookings' list, aiming to assign every booking to a bus. While there still are elements on the 'bookings' variable, we increment the variable busIndex by 1 and invoke the getMostBookedDest() function. This function returns the most frequently requested booking stop from the bookings list. We then add this stop to the busList[busIndex] stops list and update the number of passengers accordingly, ensuring that the total count does not exceed the bus's capacity, and remove all the assigned elements from the 'bookings' list. At this stage of the loop, we have allocated the most requested bus stop to the bus. We now need to iteratively fill the bus with other bookings, starting from the bookings destined for stops closest to the the most requested booking destination to the most distant ones,

until either there are no more bookings to assign or the bus is full. To accomplish this, we call the `getClosestStops` function, which returns a list of remaining stops from the bookings array, ordered ascendingly by their distance from the `mostBookedDest` stop. This list is assigned to the `closestStops` variable and we then move into the second while loop where we, naturally, verify if either there are still remaining elements in the 'closestStops' list or if the bus is not at maximum capacity. If either of those conditions are `False`, we return to the first while loop. If both those conditions are `True`, we proceed to get the first element of the 'closestStops' list, which is the closest (remaining) request stop to the most requested one, and update the number of passengers of the current bus to be the biggest number between the remaining seats on the bus or the total number of bookings for that stop. By the end of the loop, we will either be out of remaining bookings or have the current bus completely filled. If there are still bookings needing assignment we repeat the process, otherwise, we return the list of buses, each one filled with it's own stops and the own passengers.

Based on the flowchart from the Figure 3.17, the corresponding pseudocode would be the following:

```
bookings = getBookings()
busList = [Bus() for _ in range(N)]
busIndex = -1

while not bookings.isEmpty():
    busIndex += 1

    mostBookedDestination = getMostBookedDestination()

    busList[busIndex].stopsList.append(mostBookedDestination)
    busList[busIndex].passengers += count(mostBookedDestination)

    bookings.remove(mostBookedDestination)

    closestStops = getClosestStops()

    while closestStops and busList[busIndex].passengers <= MaxPassengers:
        nextStop = closestStops.pop(0)

        busList[busIndex].stopsList.append(nextStop)
        busList[busIndex].passengers += min(count(nextStop), MaxPassengers -
            busList[busIndex].passengers)

return busList
```

The previous algorithm is guided by the principle of prioritizing the most frequently booked destination as the key booking and is an important component that simplifies the subsequent route generation as it groups together individuals who intend to travel to the same area, converting geographic randomly distributed points, represented in Figure 3.18 into **drop off zones**, exemplified in Figure 3.19.



Figure 3.18: Distribution of Bookings

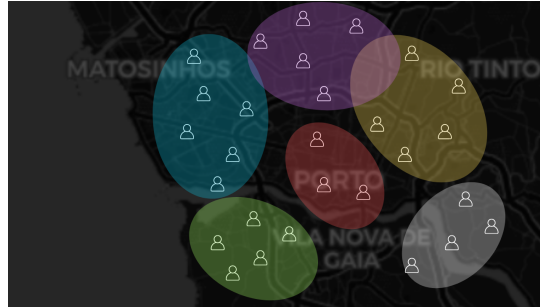


Figure 3.19: Grouping of Bookings

It is noticeable there are no strict maximum travel time or distance restrictions between passengers' destinations. Buses aggregate passengers regardless of their locations, even if they are on opposite sides of Porto. This strategy prioritizes **maximizing bus capacity**. For example, one bus might be fully loaded, while another carries only 2 or 3 passengers, rather than aiming for an equal 50-50 distribution. This approach ensures efficient use of available space. If we were to apply distance or travel time based filters to our algorithm, what would happen would be, for instance, that when there are one or two passengers with distinct destinations far from the rest of the bookings, they may not be placed on any bus, even if there is still available capacity. This situation would also occur in subsequent runs of the algorithm if the bookings remained too far apart, resulting in those passengers being unable to use the service.

The necessity to group bookings by closeness became evident when considering the alternative. For instance, if we were to process bookings in the order they were made, prioritizing earlier clients, we would compel each bus to make numerous stops, often just to drop off one or two passengers, without any proof or guarantee that we wouldn't be grouping together passengers with geographically opposed destinations. This would result in each bus taking a significantly longer and less efficient route, with many buses crossing paths to drop passengers at the same stop.

By grouping the bookings by zone with the central nodes of each one being the most frequently booked destinations, each bus is assigned to a distinct zone and is responsible for fewer stops while each stop will serve a larger number of passengers, enhancing efficiency and, consequently, travel-time and client satisfaction.

Furthermore, we also anticipated the implications on the routing algorithm component. As previously mentioned, this grouping clusters together fewer nodes while being close to each other. Without such grouping, we would need to determine the optimal path from an excessive number of nodes, while each pair of nodes themselves would have an extensive path linking them and, as a result, an increased algorithm running time. Consequently, a larger time interval would be required to ensure each passenger is assigned a seat, necessitating passengers to book even earlier, an unnecessary and inconvenient condition.

Not everything is optimal however, with worst case scenarios still being bound to this grouping

method, but one that is inherent to the project itself and is minimized by our aggregation component logic.

Being each bus equipped with limiting seating not every client that is booked for the same destination or zone, will be placed on the same bus. For instance, let's assume, for an example case, that the buses maximum number of passengers is 10. If we have 11 bookings for the same bus stop, 1 passenger will be assigned to another bus, which will have to maybe go out of their zone to drop that single passenger. Due to how our algorithm is defined, that passenger will be assigned to the best bus (bus with the closest most requested stop) that still has available seating, but that is a problem that could influence the travel time a lot.

Let's analyse the **Best** and **Worst** scenarios assuming the number of Buses is n and each bus has c maximum capacity:

1. Best Case

- Improbable Scenario
 - We have n unique destinations requested by passengers, with each destination being booked by exactly c clients. This will allocate c passengers to each bus, where each bus will have only one stop before returning to its original position. This is represented by the figure 3.20
 - The same passenger allocation could occur if there is only one destination booked that is common for all clients. The minor difference in this case is that all buses will converge at the same bus stop.

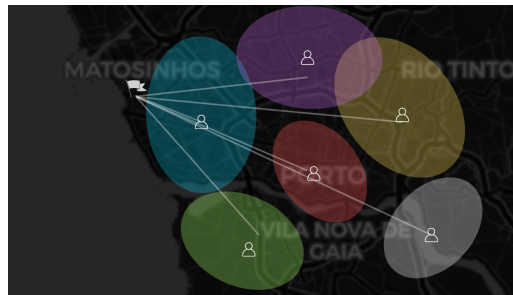


Figure 3.20: Best Case Improbable Scenario Representation

We have n buses, each with a single destination. The lines represent the abstract paths these buses need to follow.

- Probable Scenario
 - We have n fully occupied buses, with all passengers heading to the same geographic zone allocated to the same bus. These buses have geographically distinct drop-off zones without overlap between each other. There are no surplus passengers destined close to an existing **drop off zone** or a destination already assigned to a bus also being allocated to a different one. This is represented by the figure 3.21

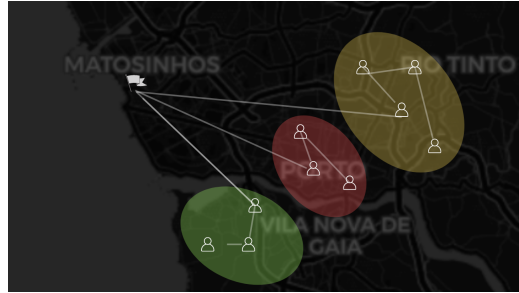


Figure 3.21: Best Case Probable Scenario Representation

In this figure, we have n buses, each responsible for a distinct destination zone, without geographic overlap. Each bus drops off passengers whose destinations are well separated, ensuring no buses travel in the same zone.

2. Worst Case

- Improbable Scenario
 - Each client books a distinct destination (only one booking per destination), with each destination being geographically distant from every other one. Each bus will be filled with c passengers, each of whom will be dropped off at a very different destination. This “unaggregates” zones, resulting in a time-consuming and inefficient route. This is represented by the figure 3.22



Figure 3.22: Worst Case Improbable Scenario Representation

A bus must travel to c geographically distant bus stops, each requested by a single client. This results in an oversized travel zone for the bus, significantly increasing the total travel time to an unmanageable amount.

- Probable Scenario
 - We have $n-1$ buses, each having at least one destination where one or more passengers destined for that stop could not be accommodated on the bus that already passes through it. This leaves a surplus of at least one passenger who will need to be placed on the last bus (last in terms of our algorithm’s bus list iterative order). This will cause the last bus to pass through the booked destination of at least $n-1$ passengers who are geographically located in every other drop-off zone, in addition to the remaining bookings that may or may not be close to each other. This is represented by the figure 3.23

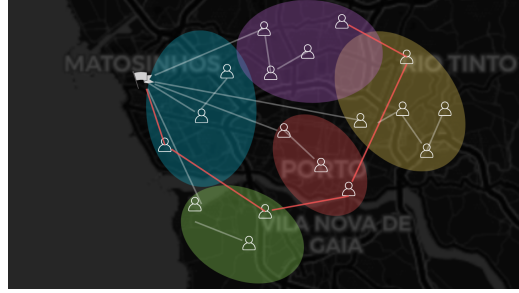


Figure 3.23: Worst Case Probable Scenario Representation

We have $n - 1$ buses, each assigned a well-defined drop off zone. However, seating limitations force some passengers to be left out. Consequently, the last bus carries all these additional passengers, compelling it to pass through all the other $n - 1$ zones, as indicated by the red line.

Having each bus assigned with their respective destinations, the routing algorithm will be called, generating the path needed to pass through them all.

3.4.2 Route Generation

The route generation component of our algorithm aims to construct the most efficient path from a shared origin for all buses, traversing each bus stop as requested by their respective assigned client bookings outputted by the aggregation component. As previously noted, this generated route is constrained to only utilize the streets that are part of the daily **STCP** bus lines. To illustrate, if **STCP**'s bus 200 travels from point A to B via route C, then route C is deemed available for use.

This component is divided into two main steps: getting the best stop permutation and connecting the elements, however, we first need to address the creation of a new table that will store the routing data we will need: the routing table.

3.4.2.1 The Routing table

The Database section of this dissertation features two key tables: Nodes and Edges. The Nodes table represents stops, while the Edges table contains the connections between pairs of stops. Specifically, an edge connects two nodes, and it defines the path between them. However, this data only captures links between directly connected stops in the real world. Unfortunately, it doesn't provide a way to determine the optimal path between nodes that aren't directly linked, which will be needed since we aim to achieve the shortest path to traverse all the requested stops.

To address this limitation, we turned to the pgRouting [33] library that extends the already used one, PostGIS [35]. Specifically, we leveraged the **pgr_dijkstra** function, which applies the Dijkstra [8] algorithm to our Nodes table which results in the creation of a new table—the Routing table—that stores the optimal path between every pair of nodes, as well as

every edge and node passed through and their respective cumulative distances.

Despite using a nearest neighbor heuristic within a Greedy Algorithm we will be seeing forward, utilizing this new data table stored in the database, we know we will always be utilizing the best geographical path available for every path between any two points, as well as guiding the algorithm by the shortest distance possible between any stops. Since the data will be stored permanently in our database as product of a one-time calculation, it will not only increase considerably the quality of our pathing, it also will drastically decrease the algorithm's running time, needing only a single query to efficiently get the best path between each needed pair of nodes. Otherwise, we would need to compute the best path between the iterated nodes every time, following the edges available from the starting node, which is an over complication of an already complex problem

Since we generated a table storing the best path between every pair of nodes, we could also get a table storing the best permutation of nodes for every combination of nodes possible. For example, having an entry for the best path from Node A to Node D, but also having the best path from Node A to Node D that passes through Node C, or another that passes through Node B, or even one other entry that passes on both. This method would be very efficient, as we would only need to query the entries that start on the needed node, and pass through all the other client requested stops and retrieve the one permutation with the lowest total distance. However, this would only be efficient if we had a low number of nodes, which we don't, since we have 2487 nodes (the removed nodes would be needed again since we needed to make sure they are passed on), and as such we would need the following total number of entries:

$$Total = C(2487, 2) \times 2485 = 76819823850 \quad (3.3)$$

This is an absurd number of entries that would not only need **substantial memory capacity** to be stored and an **infeasible time to calculate**, but also **slow the queries** by a significant amount due to the number of entries and data. Moreover, in case of a needed recomputation, such as the addition of new stops or deletion of existing ones, it would require that we lose, once again, an impracticable amount of time to populate it with the updated data again.

As such, we decided to implement our own permutation generation algorithm.

3.4.2.2 Getting the best stop permutation

The first step into generating the best route for a bus is to calculate the best permutation of its assigned stops, that is, the order of which the stops should be travelled in order to get the minimum distance and, consequently, time. Furthermore, due to its extensiveness, we will divide the algorithm into 3 parts: **Initial Phase**, **General Case Loop** and **Specific Case Loop**.

1. Initial Phase

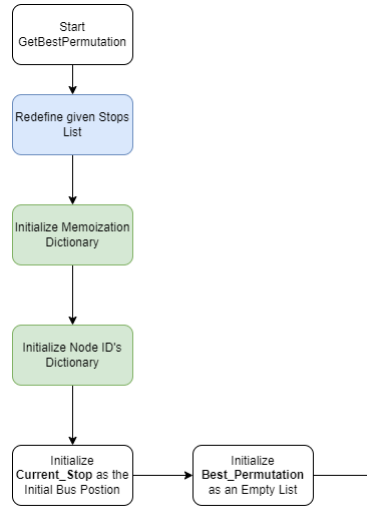


Figure 3.24: Routing Generation Initial Phase

The Routing Generation algorithm initial stage, represented in the flow chart from Figure 3.24, is defined with three main steps. The 2 steps highlighted in green are significant for enhancing the computational speed of the function. **Memoization**, is a technique used to store computed results that will be required more than once and is employed to avoid unnecessary recomputation. This technique is particularly useful on this algorithm for storing distances between pairs of nodes, which are frequently needed during the algorithm's execution which we will be detailing on the next parts. By utilizing a Dictionary, we can bypass the need to repeatedly query the Database for distances, enabling a one-step lookup instead. A second dictionary is initialized following the same principle, storing used Node/Stop IDs to eliminate the need for constant database queries every time this information is required. This techniques could be used in a permanent manner, this is, after some undefined number runs of the algorithm we would have all the data stored in the previous 2 dictionaries, however, it is a excessive memory waste of our server and would render the Database useless, which already stores our data. Accordingly, we decided to utilize the dictionaries per run of the algorithm, for each batch of bookings and buses, resetting them after.

Preceding these initializations, there is a step highlighted in blue. This step, where we Redefine the Stops List, completely changes the way our application will work and is mainly motivated by the structure of the routes STCP provides.

Given the constraint that buses can only utilize routes defined by STCP, there is a constant need to cover additional miles to ensure every passenger is dropped off on a specific stop. In other words, even if three or four stops are just 5 or 10 meters apart, if there are not STCP bus lines that connect them, the bus also cannot directly access them. Instead, it must follow the road indefinitely until there is a stop that allows it to turn around or enter a different road that passes through the other stops. One other prime example of this is when two requested bus stops are on opposite sides of the same road. In such cases, after dropping some passengers on

one side of the road, the bus must travel a very extensive distance until it reaches another node where it can turn around and go back to the same road to drop off passengers on the other side.

To address this problem, we **redefine** the Stops List, utilizing a **simplification technique**. What it aims to achieve is transform a narrow cluster of stops into a single one, converting very close stops into a single drop off stop. We defined the distance threshold to be 500 Meters, eliminating every stop that is, at most, that distance from a given stop. This way, the bus will not require extraordinary amounts of time to redirect themselves to each destination, being able to drop much more passengers per stop, while also being very close to the passengers requested destination and guaranteeing it's satisfaction. The figures 3.25 and 3.26 illustrate the transformation that will be made:



Figure 3.25: Route Generation Before Simplification



Figure 3.26: Route Generation After Simplification

As seen in the figure 3.25, the route generated by not using the simplification, over complicates the routing between the stops, shaping what would be a quick and simple distribution of passengers, into a entangled web of paths, traversing the same roads more than once and even going to the zone of Gaia, without even having any passenger destined there, just so it is able to go in the reverse direction. After using the **simplification** technique, demonstrated in the figure 3.26, we are able to utilize only 1 bus stop to drop off everyone, with every passenger that was not

destined there, to be very close to their respective booked stop (approximately 10 minutes by foot, at maximum). Is is a **very significant change**, but one that is needed to balance the path, the travel time and, most importantly, passenger satisfaction.

The flow chart on figure 3.27 clarifies the algorithm that is being applied to **Redefine** (Simplify) the Stops List:

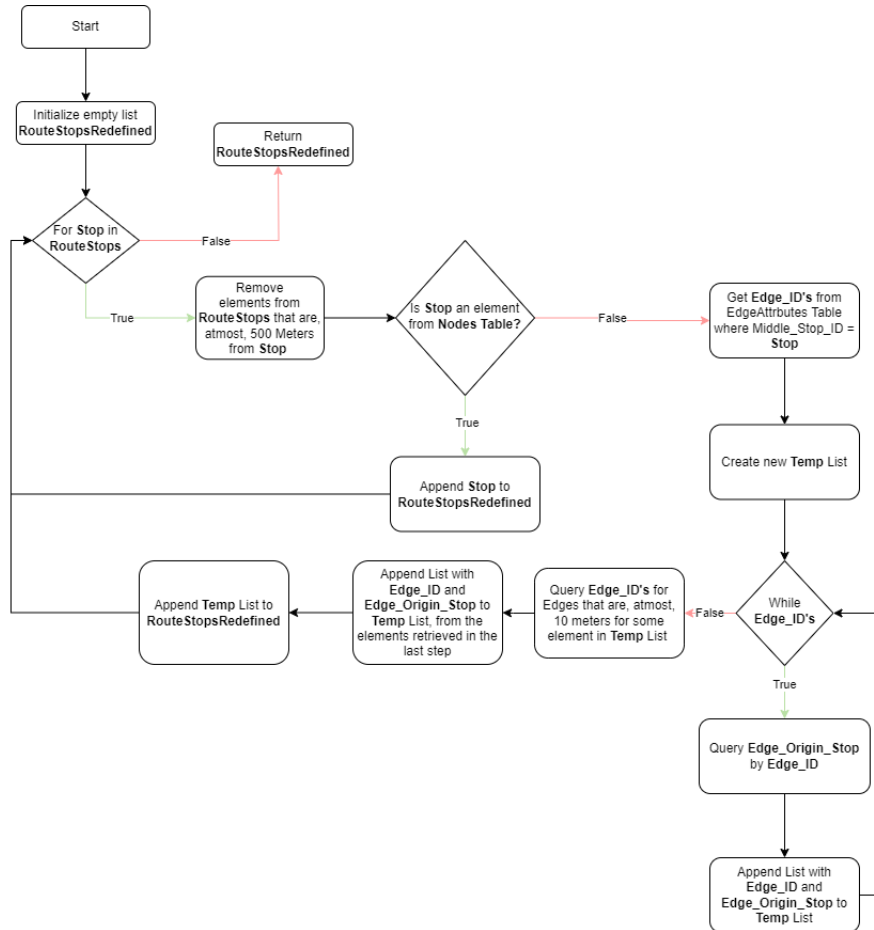


Figure 3.27: Redefine Stops List Function Flow Chart

The function starts by initializing an empty list, **RouteStopsRedefined**, which will hold the updated version of the Bookings List. It then enters a loop to iterate through the original **RouteStops** list. For each element in the list, the function removes any stop that is, at most, 500 meters away from the current stop. It then checks if the current stop exists in the Nodes Table of our database. This table only contains non-redundant stops, so essentially, we are verifying if the current stop is redundant. If the stop is non-redundant (**True**), it is added to the **RouteStopsRedefined** list. If it is redundant (**False**), we need to identify the Edge(s) that passes through it. We query the EdgeAttributes table in the database to retrieve the EdgeIDs that traverse the current stop.

It's important to note that redundant nodes may be traversed by more than one Edge (they are redundant because they only have a single outgoing link to any other stop). Therefore, we

create a temporary list and iterate through the queried Edge IDs. For each Edge ID, we query the database again for its corresponding origin stop and add a list with [Edge_ID, Origin Stop] to the temporary list.

By the end of this loop, the temporary list will be populated with other lists, with each one containing its Edge ID and respective route origin node. In summary, we will have a temporary list containing all the edges that traverse the currently iterated stop, and their respective point of origin.

Upon exiting the loop, we further streamline our stops. We do this by querying each Edge ID and its corresponding origin stop that is within 100 meters of any Edge from the temporary list at we just populated, and adding it to the same list in the same format. This step is **essential** as it allows buses to choose faster routes for dropping off passengers. For example, instead of having to travel from Stop A to Stop B along a specific Edge, which might force the bus to take a detour, the bus can opt for the Edge from Stop C to Stop D, which is the reverse direction of the same road, which has a bus stop 5 or 10 meters away from the requested one.

We conclude the outer loop iteration by adding the temporary list to the RouteStopsRedefined list and continue iterating through the remaining stops that were not suppressed by the threshold. The function ends by returning the RouteStopsRedefined list.

Based on the Figure 3.27, the pseudo-code would be the following:

```
RouteStopsRedefined = []

for Stop in RouteStops:
    RemoveElementsWithinDistance(RouteStops, Stop, 500)

    if Stop in NodesTable:
        RouteStopsRedefined.append(Stop)
    else:
        edges = getTraversingEdges(Stop)
        temp = []

        while edges:
            edge = edges.pop(0)
            edgeId = edge.edge_id
            edgeOriginStop = getEdgeOrigin(edgeId)
            temp.append((edgeId, edgeOriginStop))

        closeEdges = getCloseEdges(temp)
        temp.extend(closeEdges)

        RouteStopsRedefined.append(temp)

return RouteStopsRedefined
```

A significant aspect of the Stops List simplification is that the first element of the list, due to

the ordering done by the Aggregation Algorithm, is always the most booked destination of all the elements of each list. This ensures that this element will always be the first to be processed in the simplification algorithm, ensuring it is never removed and remains on the new list. Consequently, the most booked stop will always be a priority and will deliver the passengers destined for it. Furthermore, the remaining stops that are not removed will be at least 500 meters away from the rest of the list. This spacing creates a more evenly distributed set of drop-off stops.

Before proceeding to the following algorithm section, let's get some insight of what transformation our **Stops List** will go through by looking at figure 3.28:

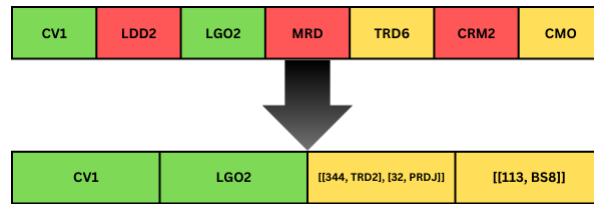


Figure 3.28: Redefining the Stop List Illustration

This figure represents the transformation the Stops List suffers when it is Redefined by the function. The upper list, with 7 elements, mutates, giving origin to the lower list, with only 4 elements. The elements from the list are colored where: Green elements are Non-Redundant stops, Red elements are Stops that were removed by being within than 500 meters from some other element, and Yellow elements are Redundant Stops that were replaced by a list of lists of [Edge_ID, Origin Stop]

After, creating an empty list Best_Permutation and initialize a new variable Current_Stop as the bus stop where the Buses should start (this is dependant where the event will be), we will be iterating through the new redefined bus stops list while it is not empty. During its iterations, we will be faced with 2 cases.

1. General Case

In this initial case, we first assign the **Closest_Stop** variable with the stop from the **Redefined_Stops** list that has the minimum **path** distance to the **Current_Stop** variable, that is, the stop which takes the least amount of time to go to, starting from the **Current_Stop**. The **Current_Stop** variable represents the current stop where the bus is logically located. We make use of the memoization when computing the distance between two stops, as it excludes the possibility of having to recompute the distances between previously calculated nodes, checking instead if the value is already stored in memory and speeding up computational running time.

After identifying the closest stop, we verify if the assigned element is a list or a singular element or, in other words, we check if it is the replacement of a redundant node by a list of lists containing [Edge_ID, Origin Stop] or a non-redundant stop. This case, the general case, represents the scenario where it is a non-redundant node where we simply need to append it to the **Best_Permutation** list and reassign the **Current_Stop** variable with the newly appended stop. The Flow Chart from figure 3.29 pictures the case:

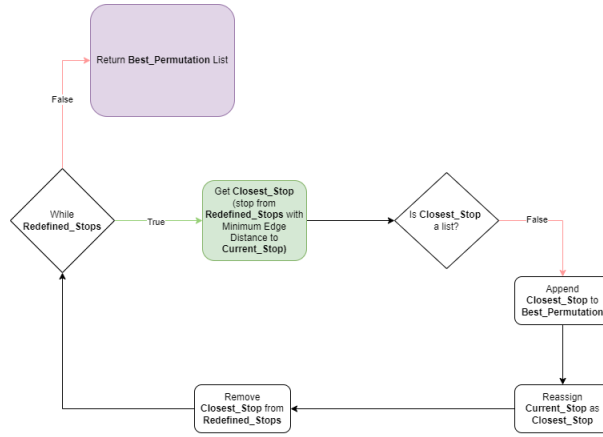


Figure 3.29: General Case Loop Flow Chart

2. Specific Case

This second case happens when **Closest_Stop** is a list. In other words, the element contains the Edge_ID and Origin Stop of the respective edges that pass through some requested stop, whose Origin Node is the closest stop to the logical stop the bus is in, the variable **Current_Stop**. However, since it is a redundant node, it does not offer the possibility for the bus to reroute itself after reaching that node and, consequently needs to fully traverse the Edge the requested node is in. In this cases, we need to account for both distances: the distance to the origin node of the Edge plus the total length of it. The flow chart from figure 3.30 illustrates the behaviour of our loop when we fall into this condition:

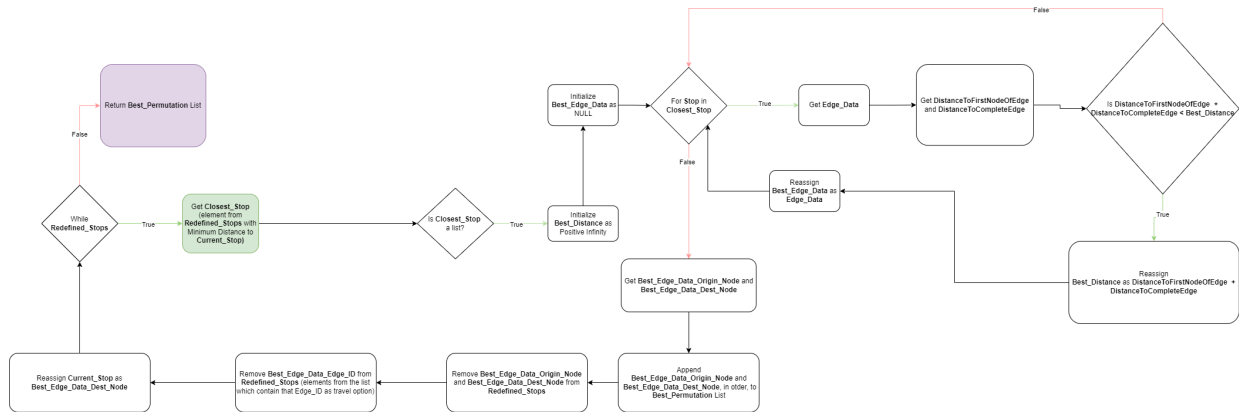


Figure 3.30: Specific Case Loop Flow Chart

After starting in the same way the General Case does, it checks if **Closest_Stop** is a List. In this case, it is, so it follows a completely different path. First, it initializes 3 new variables: **Best_Distance**, which is initialized as Infinity (the highest positive value possible in the respective programming language) and **Best_Edge_Data** is initialized as NULL. **Best_Edge_Data** will contain the Origin Node and Destination Node, which we will be needing know which stops should be appended to the **Best_Permutation** list.

Since in this case, **Closest_Stop** is a List of List, we will be iterating through each element of it. Using a for loop, we get the Edge data for the iterated Edge, using the Edge_ID value of the List and proceed to get the distance metrics needed: **Distance to the origin node of the Edge** and **Distance to fully traverse the Edge**. We then check if the 2 latter variables summed are inferior to the value of the **Best_Distance** variable. If they are inferior to it, we update it to be the sum of the distances previously calculated and also update **Best_Edge_Data** to be the Edge data from the currently iterated element and proceed to the following iteration. In the case the Distances sum is not smaller than the **Best_Distance**, we also proceed to the next iteration of the for loop, but without making any variable alteration.

After iterating through all the list elements from **Closest_Stop**, we have **Best_Edge_Data** as the list with the Origin Stop and Destination Stop of the best Edge to traverse and, accordingly, we append both Stops, in order, to the **Best_Permutation** list and remove the iterated element itself. We proceed by eliminating any occurrence of the Best Edge Origin and Destination Stops from the **Redefined Stops** list on non-redundant elements, as those stops will already be traversed. Alternatively, for other redundant elements, we exclude only the elements that contain the **Best Edge Edge_ID** as an edge option, based on their list of options. In simpler terms, since we must already traverse the selected Origin and Destination Stops, there is no need to retain those stops as non-redundant nodes in the list. Additionally, since we are obligated to complete the chosen Edge, we can also safely discard elements that offer the same Edge as a travel option.

In its entirety (Initial Phase, General Case Land Specific Case), the pseudo-code for the route permutation generation algorithm, represented in the Figures 3.29 and 3.30, would be the following (it's to be noted that memoization and other computational speed optimizations initialized on Figure 3.24 are not represented):

```

redefinedStopsList = redefine(StopsList)
currentPosition = getEventInitialStop()
bestPermutation = []

while not redefinedStopsList.isEmpty():
    closestStop = getClosestStop(currentPosition)

    if isinstance(closestStop, list):
        bestDistance = float('inf')
        bestEdgeData = None

        for stop in closestStop:
            edgeData = getEdgeData(stop)
            totalDistance = edgeData.distanceToInitialNode +
                            edgeData.distanceToCompleteEdge

            if totalDistance < bestDistance:
                bestEdgeData = edgeData
                bestDistance = totalDistance

```

```

originNode = bestEdgeData.originNode
destinationNode = bestEdgeData.destNode

bestPermutation.extend([originNode, destinationNode])

redefinedStopsList.remove(originNode)
redefinedStopsList.remove(destinationNode)
redefinedStopsList.remove(bestEdgeData.edgeId)

currentPosition = destinationNode

else:
    bestPermutation.append(closestStop)
    currentPosition = closestStop
    redefinedStopsList.remove(closestStop)

return bestPermutation

```

Why did we use a Greedy Algorithm

We are currently addressing a challenge involving a set of dynamically selected geographic points chosen by event attendees. This problem can be likened to the Traveling Salesman Problem (**TSP**), and more specifically, to its generalization, the Vehicle Routing Problem (**VRP**). The **VRP** seeks to determine the optimal set of routes for a fleet of vehicles to traverse in order to deliver to a predefined set of customers. Numerous studies on the **VRP** not only delve into the fundamental aspects and definitions of the problem but also explore various solutions, optimizations, and real-world applications. Notable examples include the work of Liu et al.[26], Pillay et al.[34], and Toth et al. [40]. Beyond the classical definition, various extensions of the **VRP** have been proposed, such as the Capacitated VRP (CVRP), the VRP with Time Windows (VRPTW), and the VRP with Pickup and Delivery (VRPPD), each addressing different real-world constraints. These variations complicate the problem even further, as they introduce additional layers of complexity, necessitating more sophisticated algorithms for optimization.

Recent advancements in heuristic and metaheuristic algorithms, including Genetic Algorithms (GA), Ant Colony Optimization (ACO), and Particle Swarm Optimization (PSO), have been applied to the **VRP** with varying degrees of success [3, 25]. These approaches, while not providing guaranteed optimal solutions, often deliver near-optimal results within practical time constraints, making them valuable in real-world applications like logistics, supply chain management, and urban transportation planning [17]. Machine learning techniques, particularly reinforcement learning and deep learning, are also being explored as promising tools to enhance traditional **VRP** solvers by enabling them to adapt to dynamic, real-time conditions [24]. Such innovations are critical as industries increasingly rely on real-time data for route planning, often necessitating dynamic re-optimization to account for factors such as traffic, weather, or vehicle breakdowns.

Both the **TSP** and the **VRP** are classified as NP-hard, meaning they are exceptionally difficult

to solve efficiently. In computer science, NP-hard problems are among the most challenging, as they are at least as complex as the most difficult problems in NP (nondeterministic polynomial time). NP-hardness implies that, unless P equals NP (a question that remains unresolved), no polynomial-time algorithm exists that can solve all instances of these problems. This has profound implications for both theoretical research and practical applications. Given the exponential growth in solution space as problem size increases, even moderately sized instances can become computationally infeasible to solve exactly. Consequently, researchers often turn to approximation algorithms and heuristic methods to find feasible solutions [6]. While many experts in the field consider it improbable that a polynomial-time algorithm will ever be found for NP-hard problems, the quest for better approximation ratios and faster heuristics continues to drive innovation. As computing power and algorithmic techniques advance, there remains a slim but non-negligible hope that breakthroughs, particularly from fields like quantum computing, could one day redefine our understanding of these complex problems [28].

The significance of solving or efficiently approximating solutions for **VRP** and similar NP-hard problems extends far beyond academic curiosity. The ability to optimize routes has direct implications for reducing fuel consumption, lowering operational costs, and improving delivery times. Furthermore, real-world variations of **VRP**, such as those involving stochastic demands or dynamically changing environments, continue to pose major challenges, ensuring that **VRP** remains a fertile area of research with widespread applicability in various domains [7]. Therefore, while NP-hard problems like the **VRP** may not yet have definitive solutions, the continued research and development of advanced algorithms, combined with technological innovations, are poised to yield increasingly efficient and practical outcomes for industries and society alike.

With the previous constraints and variable to be considered into the logistics and implementation of the routing generation algorithm, we decided to approach the problem with a Greedy Algorithm implementation based on the Nearest Neighbour Heuristic (we always follow the local shortest Edge) and there are three main reasons for it.

Firstly, the **time constraints**. Due to the time allocated to fully develop a functional app, we needed to guarantee the algorithm was simple to debug, improve and optimize. We needed to recurrently test the algorithm with many variations as well as making sure the outlier cases are also accounted for and that, in the launch of the application, there would not be errors and passengers not accounted for. Furthermore, due to the interval window the algorithm is supposed to run (around 5 to 10 minutes before departure time), we also require the algorithm to be fast to run and assure there would be no specific cases where the algorithm would need an extraordinary amount of time, so we needed to guarantee minimal overhead and ease of fix in the event something goes wrong.

Secondly, this implementation, conjoined with all the work done that preceded it, offers a **Good Enough Solution**. For starters, since we aggregate the bookings by zone, in most cases, the stops will all be close to each other and, in the case our algorithm does not provide the optimal solution, the amount of travel time saved by a more complex algorithm would not be extraordinary. To add to that, using our pre-calculated **Routing Table** using PostGIS Dijkstra

Algorithm [8], we also know we are always following the best path possible between every 2 nodes, assuring we are always getting the optimal path in every decision the algorithm makes.

Thirdly, the Greedy algorithm offers exceptionally fast computational performance, with the time complexity of a default implementation being $O(n^2)$. With further improvement due to the simplification of nodes, use of the routing table, redefinition of stops and memoization, we could be looking at a time complexity very optimized, ensuring that in the event of a malfunction—such as a blocked road, faulty engine, or other unforeseen incident—we can quickly recompute the solution to guarantee continued service for the remaining passengers or deploy a new bus to complete the route. Additionally, its simplicity in implementation and debugging allows for significant and straightforward opportunities for further improvement.

What other options could we have used?

Brute-force [31]: The most naive approach to **TSP**, one that is easy to implement and debug, but one that also requires a lot a computational time, $O(n!)$, which **does not** get us the much needed **computational speed**, nor does it offer a lot of **room for improvement** and optimization.

Held-Karp Algorithm [18]: Using the Held-Karp algorithm, we would consider all subsets of bus stops and computes the minimum cost to visit each subset, leading to a time complexity dominated by the number of subsets (2^n) and the number of states to compute for each subset (n^2). In total, we would have the time complexity to be $O(n^2 \cdot 2^n)$ which, even though it finds the optimal solution, **does not provide** us with a **fast solution**, which is needed in this system.

Simulated Annealing [22]: Simulated Annealing is a probabilistic algorithm that iteratively improves a tour by exploring neighbors and accepting worse solutions with decreasing probability to avoid local minima, using a temperature that gradually decreases according to a cooling schedule, The complexity depends on a number of factors such as number of iterations, neighbourhood size and cooling schedule, however, we can generally state that is around $O(I \cdot N)$, where I is the number of iterations and N the time to evaluate neighbours at each step. Even though the time complexity it's not overwhelming when considering it is capable of generating an optimal path, the **complexity for debugging and optimizing** is, making an unsuitable candidate for a system that may require in-real time fixing.

While there are further **TSP** algorithms, they fall onto the previous downsides, while our Greedy implementation, provides the perfect approach to the conditions the project is submitted to.

3.4.2.3 Connecting the elements

After generating the best permutation of nodes in the previous section, we need to assign the geographical path to each buses assigned nodes. Since the output from the previous step is a list of non-redundant nodes in a sequential order, we know we have a path between whatever pair

of nodes we need to get the path for in the **Routing Table**. As such, this part is not overly complex, as we simply need to query our **Routing Table** for the best path between every 2 consecutive nodes on our list and concatenate the path geometry data of each Edge, stored on the **Bus Routes** table. For instance, if we have the List [Node A, Node D, Node B, Node C] as the best permutation list obtained, we simply need to query the path from Node A to Node D, the path from Node D to Node B and, lastly, the path from Node B to Node C. Having the edges id's that form the full path between them, we simply need to query the **Bus Routes** table by the respective Edges (by the edge id's) and get their geometrical data.

By developing a straightforward HTML and JavaScript application, we can now **visually represent** the outcomes we obtained. This visualization not only displays the assigned stops for each bus but also animates the bus motion. This animation provides a glimpse of what we would expect the buses to appear like during actual usage of the application and can be used before for verification and optimization purposes.

3.4.3 Results

To better visualize the generated routes, we developed a small application that animates a bus traversing its assigned routes based on provided data, such as stops and the overall route. The animation also depicts the bus dropping off passengers, updating the map icons, which offers a clearer understanding of the algorithm's outputs and also proved invaluable for debugging and refining the algorithm during the development process.

Before showing some of the generated routes and it's pathing, let's first establish what each icon conveys:

-  - This icon represents the bus.
-  - This icon represents a passenger booking destination, indicating where passengers should be dropped off. The icon updates when the bus either passes the location or when the simplification system omits it. If an icon remains unchanged after the bus completes its route, it signifies that the passenger is to be dropped off at an adjacent stop within 100 meters of the indicated location.
-  - This icon marks the start and end points of the edges used to compute the route, primarily useful for debugging purposes.
-  - This icon signifies that the bus has passed a passenger request location and passengers have been dropped off.
-  - This icon represents a passenger request location within 500 meters of a bus stop the bus actually passed, indicating where the simplification technique was applied.

Now, knowing what the icons signify, we can demonstrate, visually, some of the routes created. Let's first look at the route presented in the figure 3.31:



Figure 3.31: Route Generation Result First Example

In this initial example, we present a simple scenario where the majority of bus stops are convoluted, allowing our simplification algorithm to play a significant role. The bus departs from its starting point and follows the river road to reach its first drop-off location, as depicted in the second panel. Notably, many passenger icons (👤) are replaced by check marks (✅), indicating that most passengers can disembark at the first stop, as they are within 500 meters of it. However, due to the structure of the **STCP** bus routes, the bus cannot simply turn back to drop off the remaining passengers. Instead, it must continue its route, as shown in the third panel, and only

is able to turn around in Gaia (south of the river). Coincidentally, the path the bus is obliged to fully traverse, takes the bus past other requested stops, enabling passengers destined for those locations to exit at their intended stops rather than the first one (represented by 🚌).

Given this scenario, it is essential to implement a route information system prior to the start of service that is able to inform the passengers, beforehand, about the bus route and all its stops, ensuring they know where they can disembark. This will be discussed in greater detail in the section 3.5 of this dissertation.

Continuing its route, the bus proceeds to drop off the remaining passengers, who are destined for two remaining stops. In this case, however, after the buses reaches the first stop, it is able to turn around, allowing it to make a single final stop instead of two, as shown in the fourth panel. Once all passengers have been dropped off, the bus returns to its starting position.

On a overall view, this routes takes 46 minutes to complete and travels for approximately 27 Kilometres.

Now, let's examine a more complex example, where there is greater disparity in passenger request locations, as illustrated in Figure 3.32:

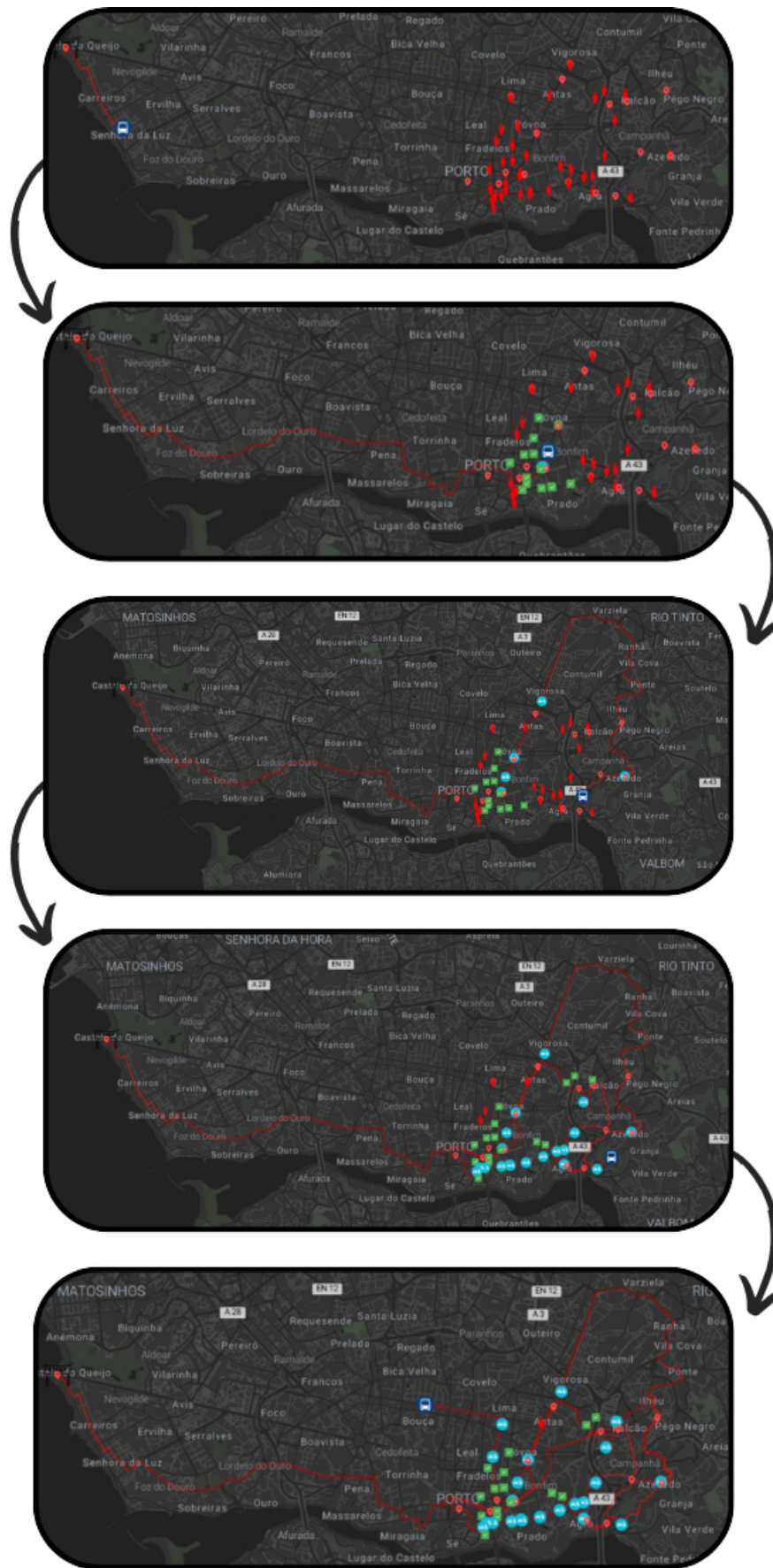


Figure 3.32: Route Generation Result Second Example

Immediately, the zoom out required to capture all the points is noticeable, highlighting that these points are distributed on a larger area than with the previous example, as well as being very distant to the starting position. This makes the route significantly more challenging to complete fast comparatively to the first example. Another notable aspect is that the number of check marks (✓) is roughly equal to the number of drop-off points (🚗), indicating that only half the number of stops were needed to deliver all the passengers, which demonstrates the heavy work our simplification approach is doing on the optimization of the pathing, while we visually verify that each stop that is not stopped at is located very close to one or more that do, guaranteeing every passenger is close to their destination.

On an overall view, this routes needs, naturally, more time to fully complete than the routes from image 3.31, needing around 89 minutes for traversing 54 Kilometers. These metrics will be analysed more in depth in chapter 4

In the route shown in the image 3.32, there is another aspect worth mentioning, represented in the figure 3.33



Figure 3.33: Passenger Icon Unchanged Example

As mentioned in the legend for the icons meaning, the passenger icon (representing passengers booked destinations) can remain unchanged in order to differentiate the points which suffered from edge adjacency simplification from the ones who were omitted from being within the defined radius from other stops. That is the case happening in the image 3.33, where the passenger icon (🚶) does not change. In order to pass through the stop, the bus should traverse the road "Rua de Carlos Amaral Dias" where it is placed, however, we can see the red line, representative of the bus path, pass through an adjacent one. By clicking on the icon, we can see the 100 meters threshold we have defined and visually confirm that the path the bus does pass within the area. In other words, there is a bus stop within that area (which the clients will know about) that the bus will pass through, that was chosen based on that respective edge total travel time (from its point of origin to its end node) being faster than travelling through the edge where the requested stop belongs to, while maintaining the closeness needed to drop the passenger safely.

Having analysed and visualized the results of our algorithms, we need to delve into how the passengers will use this service.

3.5 Web Application

After the development of the algorithm, a web application was designed and implemented to provide users and clients with easy access to this service. It was implemented using Angular for the Frontend and Flask for the Backend, as well as Node.js. This section outlines the design and key features of the web application, demonstrating the design and logistics of the platform as well as its role in facilitating user interaction with the service. It also aims to provide an intuitive interface that simplifies the process for the clients, enhancing accessibility and user experience, providing all the tools needed for a swift and rewarding experience.

The initial page of the web application shown in the figure 3.34 serves as the landing page for users visiting the site. While it is still not defined what should be presented on this first introduction to new clients, it should provide some information about the project's inner workings, objectives, and operation. Here, the clients can either go to the Login Page, if they already have an account, or to the Sign Up page if it is their first contact with the service.

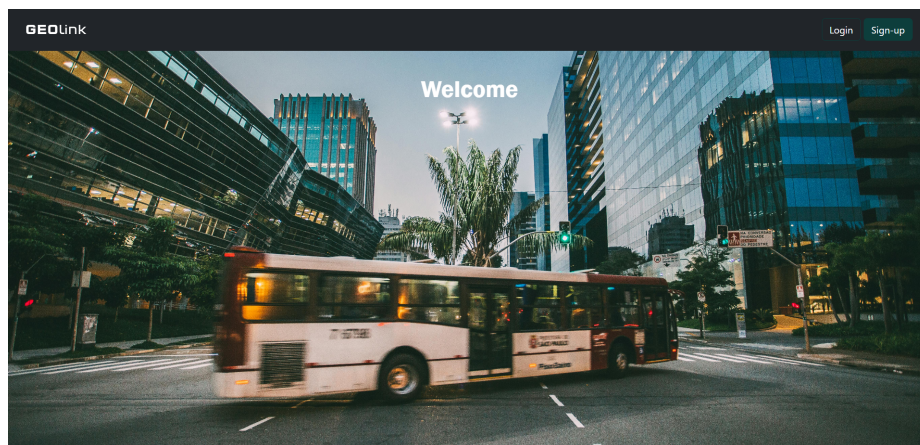


Figure 3.34: Initial page

To create a new account, the users should provide their **email**, **name**, chosen **password** and **mobile phone number**. The **mobile phone number** is needed to assure a correct input, as it will by SMS text, as previously mentioned, users will receive information about the buses assigned to them. To get started using the application, the client needs an account and, as such, shown in the figures 3.35 and 3.36 we created a demo user:

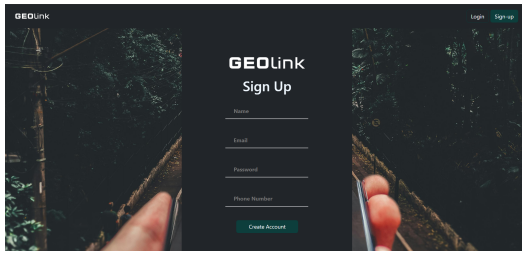


Figure 3.35: Sign Up page

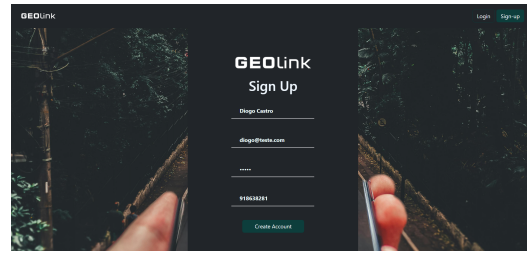


Figure 3.36: Creating an example user

After creating the account, some type of validation will be required. The validation method is still undefined, however, something such as identification document will be needed, otherwise our database would be swamped by inactive or malicious users. After validating it, the user will become activated it and will be able to Login, demonstrated in the figures 3.37 and 3.38:

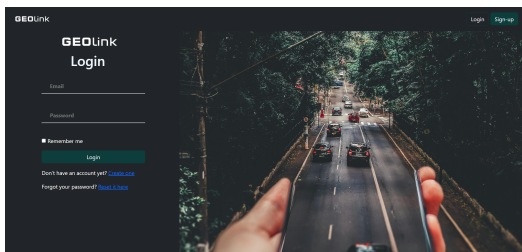


Figure 3.37: Login page

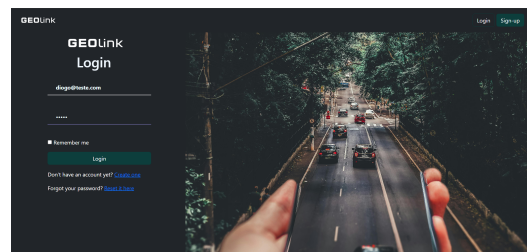


Figure 3.38: Login with example user

If the user forgets the account's password, there is also a functionality to recover it.

Upon logging in, the user is directed to the main page of the application, as shown in Figure 3.39. Here, a booking form is available for clients to request a new booking by selecting a destination, date, and time, the latter two being based on the event the client is attending. To specify the destination, the user must reference the blue pins on the map, which indicate all available STCP stops. Once selected (clicked on), the chosen stop is automatically assigned the destination field:

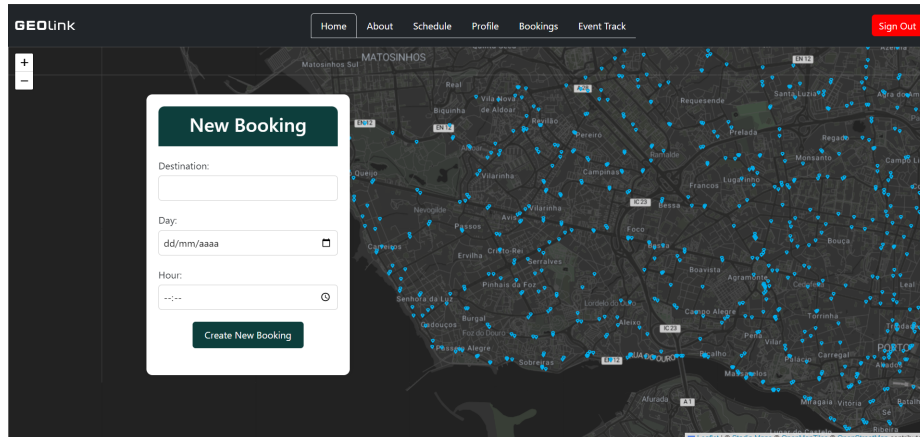


Figure 3.39: Home page

At the top of the page, there is a navigation bar that allows users to access all available functionalities and tabs within the application. The "Bookings" section, shown in Figure 3.40, provides an overview of both upcoming and past bookings:

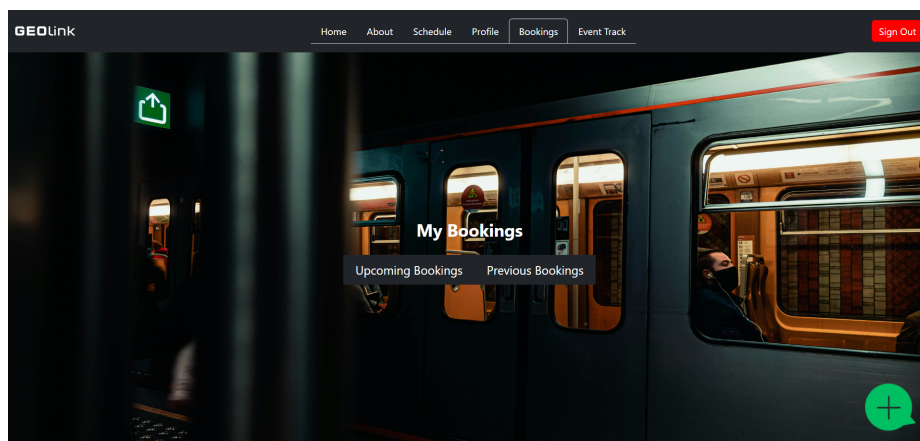


Figure 3.40: Bookings page

Since the created account still hasn't made any reservation, there aren't any booking yet to be shown in this section any bookings. To address this, a new booking can be created.

GEOlink							
Home About Schedule Profile Bookings Event Track Sign Out							
ID	Event Name	Date	Hour	Passengers	Countdown	Ratio	Book
2	Event 1	2024-10-14	09:00:00	63	65 days 13 hours 22 minutes	63/100	Book This Date
3	Event 2	2024-10-15	10:00:00	78	66 days 14 hours 22 minutes	78/150	Book This Date
4	Event 3	2024-10-16	11:00:00	42	67 days 15 hours 22 minutes	42/80	Book This Date
7	Event 6	2024-10-13	13:00:00	60	64 days 17 hours 22 minutes	60/120	Book This Date
8	Event 7	2024-10-17	14:00:00	60	68 days 18 hours 22 minutes	60/90	Book This Date
9	Event 8	2024-10-18	15:00:00	105	69 days 19 hours 22 minutes	105/110	Book This Date
10	Event 9	2024-10-19	13:00:00	100	70 days 17 hours 22 minutes	100/120	Book This Date

Figure 3.41: Schedule page

Navigating to the Schedule page, shown in Figure 3.41, displays a table of all scheduled events using the service, along with the corresponding times when buses will begin collecting passengers. Additionally, the page features a countdown timer for the service, as well as a real-time count of passengers already booked. An important column, labeled **Ratio**, shows the number of booked passengers relative to the maximum capacity for that service, color-coded by occupancy percentage (grey, yellow, orange, red). By clicking the "Book this date" button, users are automatically redirected to the Home page, with the date and time pre-filled in the "New Booking" form, and the map zooming in on a new red marker, shown in Figure 3.42:

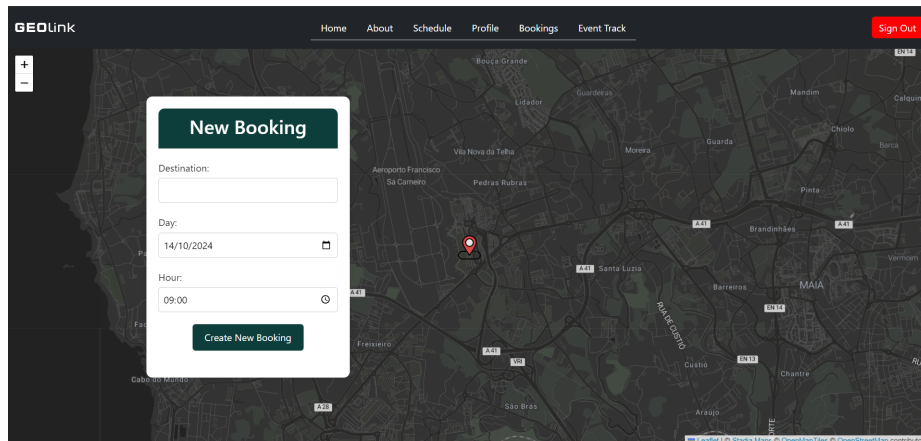


Figure 3.42: Home page with new marker

These new red markers are needed, as they represent the zone where the bus will be waiting for each event, that is, the zone the client should be at, by the time he chose, where the buses will be start checking in people, which clients need to know about.

Having the date and hour filled, the client just needs to click on the pin representing the destination stop he wants and, after clicking on "Create New Booking", check for a success pop up:

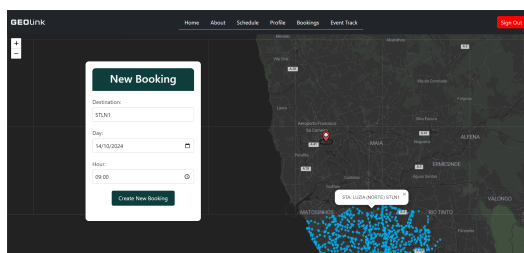


Figure 3.43: Filled form

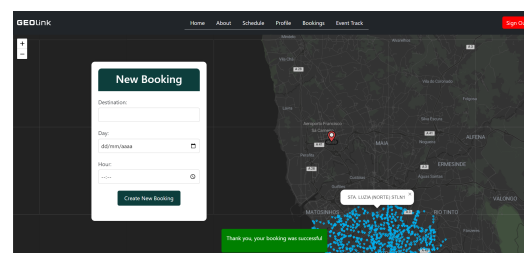


Figure 3.44: Successful booking creation

If we now check our upcoming bookings section in the Bookings tab, we will be able to get all the information needed to the newly booked service:

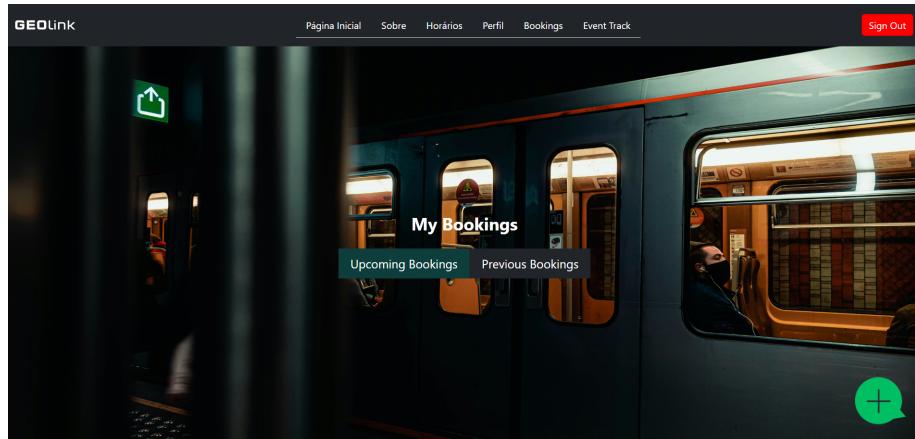


Figure 3.45: Bookings Page

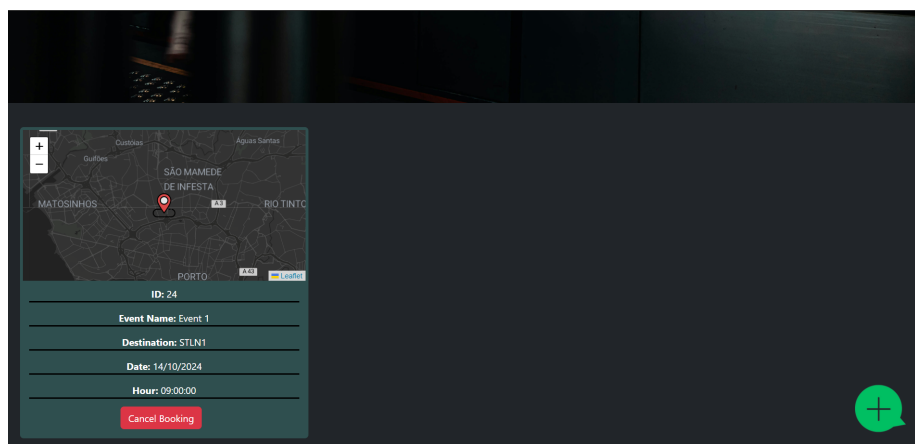


Figure 3.46: Home page with new marker

For the last tab, a **vital feature** for this application: the **Event Track** tab, shown in the image 3.47:

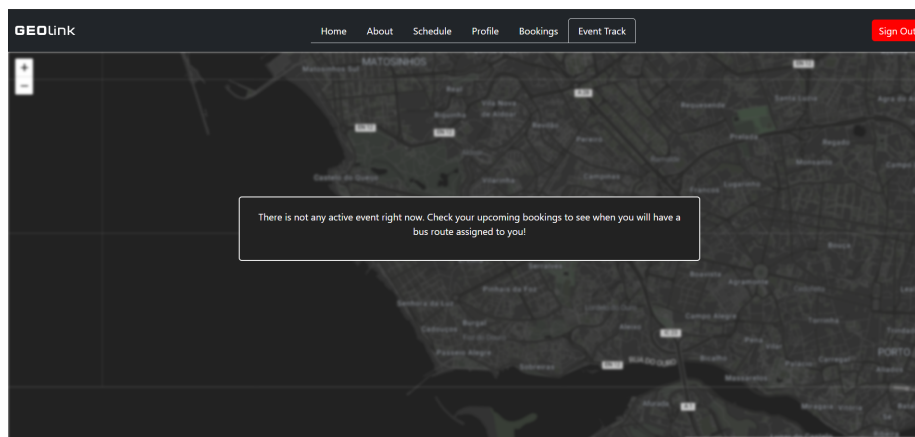


Figure 3.47: Event track page

The unavailability of the tab is due to the fact that the client is **not** currently in a booking since, what this section intends to do is: when the service the client booked reaches its deadline (that can be seen in the schedules tab), plot the full route and all the bus stops the bus that was assigned to him goes through. This way, even though we have route simplifications, the client can decide where to exit the bus by looking at the map that will be presented here.

The development of this platform offers users an enhanced transportation experience, combining accessibility, convenience, and comfort. By addressing various user needs, the application delivers a comprehensive solution to streamline the booking process and improve overall service engagement. It provides all necessary information and detailed guidance, ensuring that even first-time users can easily navigate the system. The platform is designed to be intuitive and user-friendly, allowing individuals to efficiently plan and manage their transportation needs. Ultimately, this application positions itself as a viable alternative to existing transportation options in the market, offering a more seamless and user-centric experience.

Chapter 4

Tests and Analysis

To rigorously test and benchmark our algorithms, as well as visualize the routes generated by our service, we opted to employ real-world data. Rather than fabricating synthetic data and populating our bookings data table with fictitious entries, we leveraged existing taxi data from Geolink. Specifically, we retrieved all the taxi bookings during the Primavera Sound Festival in the years 2021 and 2022. By analyzing the destinations of taxi clients who departed from locations within approximately 200 meters of Parque da Cidade (where the Primavera Sound Festival take place in Portugal), we can anticipate the outcomes we should expect when clients begin using our application. As for the number of buses available, as well as the capacity they have, we decided to define the number of buses available for transportation as 10 and they each have capacity of 85 passengers.

Let's first analyse the running time for both algorithms, presented in the table [4.1](#):

N	Aggregation	Routing		Total Time	
		W/ Simplification	W/o Simplification	W/ Simplification	W/o Simplification
85	7 ms	515 ms	1 746 ms	522 ms	1 753 ms
170	13 ms	608 ms	4 471 ms	621 ms	4 484 ms
255	20 ms	1 455 ms	6 660 ms	1 475 ms	6 680 ms
340	28 ms	2 500 ms	11 010 ms	2 528 ms	11 038 ms
425	34 ms	5 088 ms	18 600 ms	5 122 ms	18 634 ms
510	43 ms	6 479 ms	27 082 ms	6 522 ms	27 125 ms
595	49 ms	8 515 ms	38 785 ms	8 564 ms	38 834 ms
680	54 ms	9 685 ms	45 588 ms	9 739 ms	45 642 ms
765	60 ms	15 262 ms	63 176 ms	15 322 ms	63 236 ms
850	65 ms	19 507 ms	82 799 ms	19 572 ms	82 864 ms

Table 4.1: Running Time Comparison

This table provides a comparison of the running time of computing N client Bookings, measured in milliseconds, divided between the aggregation and routing components. The latter is divided into two rows—one utilizing the route simplification technique and the other without it. The total time rows are simply the sum between the aggregation and the respective routing variation used. Each row's number of bookings processed, in ascending order, is equivalent to 1, 2, ..., or 10 fully occupied buses.

The difference in growth is also very clear when we take the data into a Line Chart, as seen in the figure 4.1

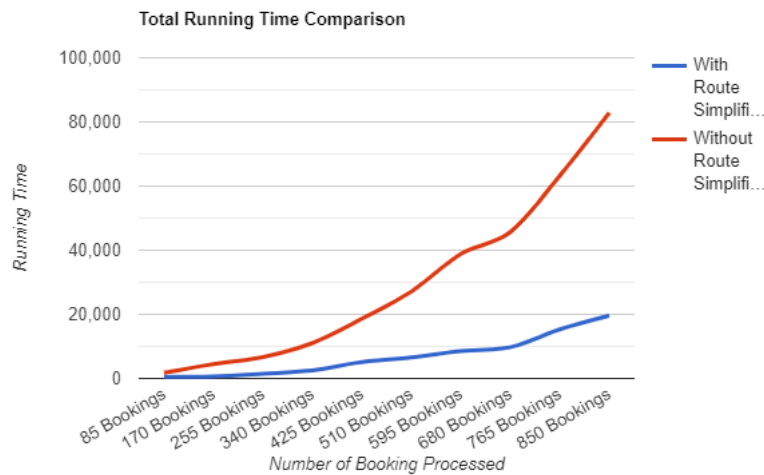


Figure 4.1: Running Time Comparison Line Chart

This figure represents the data from table 4.1 and plots it into a line chart. We can clearly see the exponential growth of the algorithm without using the simplification algorithm, while the version that does use it, is closer to a linear development over the Y-Axis, the number of bookings processed.

The efficiency gains from route simplification are evident in both Table 4.1 and Figure 4.1. This approach significantly outperforms the non-simplified counterpart. On average, the algorithm variant that does not utilize route simplification takes 4.5 times longer to compute.

In some cases, such as when processing two full buses, it can take up to 7 times longer (from 621 ms to 4,484 ms). The impact of the number of bookings processed on the computation time is clearly visible in Figure 4.1, with a stark contrast between the two approaches. The route simplification technique version of the routing algorithm seems to exhibit linear time complexity, whereas the algorithm without it appears to have a time complexity closer to exponential.

We also implemented memoization to optimize performance by storing the results of expensive computations, such as the distance between two points, instead of recalculating them in each iteration. Without memoization, the system would repeatedly access the database to retrieve the geographical coordinates for each point and compute the distance between them. So, how much impact does this have? Let's look at the comparison table 4.2, which compares the computational time of the routing algorithm (without the aggregation component time overhead) with route simplification, both with and without memoization, for each individual bus:

	W/ Memoization	W/o Memoization	Optimization
Bus 1	614 ms	876 ms	29.90%
Bus 2	110 ms	110 ms	0.0%
Bus 3	963 ms	1 779 ms	45.87%
Bus 4	878 ms	1 263 ms	30.48%
Bus 5	2 911 ms	4 196 ms	30.62%
Bus 6	1 556 ms	2 078 ms	25.12%
Bus 7	2 497 ms	3 416 ms	26.90%
Bus 8	1 424 ms	1 680 ms	15.24%
Bus 9	6 480 ms	8 873 ms	26.97%
Bus 10	4 832 ms	6 293 ms	23.22%

Table 4.2: Running Time Comparison with and without memoization

This table provides a comparison of the running time of generating the route for client Bookings, measured in milliseconds, between using memoization and not using it. The columns correspond to one unique bus, having the same Bookings being tested for both cases, providing a direct comparison between the two versions.

In contrast to the table 4.1 where we intend on analyse on the expected running time, in the table 4.2 we compare the difference in running time for each bus route's assigned stops, with and without memoization, rather than the cumulative running time. This approach allows us to directly observe the impact of memoization, as the routes and stops for each bus remain the same in both versions. By avoiding the recalculation of distances and storing them in memory, we can clearly see the full effect on performance. As shown in the 'Optimization' column, besides Bus 2 which does not benefit from memoization since no recalculations are needed (both times are the same), for all other buses, memoization significantly improves running time, with reductions ranging from approximately 15% to 46%, nearly cutting the time in half in some cases. On average, we observe a performance improvement of around 25%, which is substantial.

More important than the algorithm's running time, which doesn't directly affect clients, is our ability to deliver the promised service with minimal travel time. Our **main goal** is provide a

quick transportation service to the clients and, as such, to evaluate the efficiency of our program's output, we calculated the route length and, assuming that the bus can cover 600 meters in 1 minute, the total travel time, as to get insight into how our system would behave in a real world scenario (it's important to remember we used real world data from taxi services destinations). However, this is, undoubtedly, an ideal scenario where we don't account for stops such as traffic lights or stop signs, nor do we consider traffic conditions. However, it allows us to form an opinion on the route generated by our program using real-world data. Let's examine how our service performs under this parameter:

- With Route Simplification

	Time (Minutes)	Distance (Km)
Bus 1	46	27.74
Bus 2	34	20.30
Bus 3	40	24.04
Bus 4	45	26.68
Bus 5	89	53.45
Bus 6	95	57.05
Bus 7	76	45.96
Bus 8	98	58.83
Bus 9	123	73.58
Bus 10	165	98.70
Average	81.1	48.63

Table 4.3: Simplified Routing Time and Distance Comparison

This table presents a comparison of travel time (in minutes) and distance traveled (in kilometers) for each Bus, utilizing the route simplification technique. Note these metrics account for the return of the bus to the initial position, which **does not** waste clients time.

As indicated in Table 4.3, eight out of ten buses complete their routes, delivering all passengers to their destinations, in less than two hours, while having Bus 9 slightly exceeding this time by only three minutes. However, it's important to note that this time also includes the journey from the last stop back to the initial bus position. If we exclude this, Bus 9's trip could also be classified as a "less than two hour" trip. Remarkably, four buses manage to deliver 85 passengers in less than an hour, which can be considered an excellent performance and the values we are aiming at in order to deliver the best service possible. It is also important to notice that the average time of transportation is less around one hour and a half which, considering it accounts for the trip back to the route starting point, is a very promising result. Some of the generated routes for these buses can be seen on the figure 4.2:

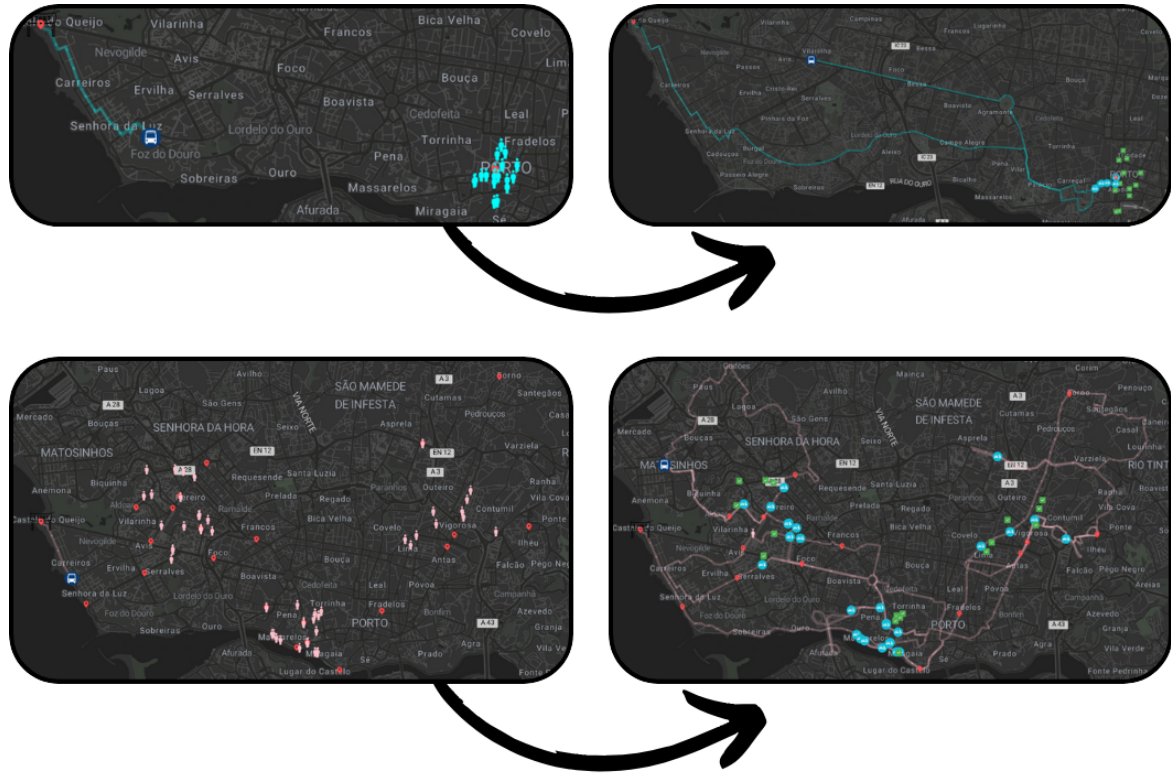


Figure 4.2: Porto Simplified Routing Examples

This figure represents two routes generated by using the simplified routing technique. The first two images are representative of the Bus 2 from the table 4.3 while the two images underneath it are representative of the Bus 10 from the same table

The blue route from image 4.2 represents an excellent example of what simplifying the route can do for the travel time. The bus stops only at only 5 bus stops from the 15 requested. Moreover, by analyzing the data output by the algorithm, we can noticed that the path only was composed by 3 bus stops: The origin, the middle stop denoted by the red pin in the image 4.2 and the final stop (the return to the initial stop); This shows that, to deliver all the passengers, the bus could've stopped only once while also being able to drop every passenger destined to every other stop close enough to their destination. The other fours stops are simply coinciding with it's path and can be stopped upon if requested by a client, however, they were not required in this case in order to deliver everyone. The pink route doesn't get the same benefits as the blue one, but it's also visually noticeable that every stop is passed trough, with each passenger icon being replaced. There is one passenger icon that is not replaced, which is another case of edge adjacency, as explained previously on this dissertation, namely in the Figure 3.33. However, unlike the blue route, which is direct and does not have redundant pathing, the pink route presents, in noticeable way, vast kilometers of road covered that doesn't have on single stop requested on them. For example, the zones circled red in the Figure 4.3:



Figure 4.3: Route Map with unused pathing circled

This figure represents the path generated for a Bus, with 3 zones circled red that mark unnecessarily travelled road

As previously mentioned, this issue stems from the limitation of relying solely on **STCP** bus routes, many of which are extensive and have unique stops. As a result, significant distances must be covered to effectively reach a non-redundant node able to reroute the bus, making it a challenging task to avoid.

- Without Route Simplification

	Time (Minutes)	Distance (km)	Increase in time (%)
Bus 1	89	53.27	93.48%
Bus 2	81	48.78	138.24%
Bus 3	94	56.42	135%
Bus 4	131	78.80	191.11%
Bus 5	147	87.99	65.17%
Bus 6	210	126.21	121.05%
Bus 7	172	103.22	126.31%
Bus 8	174	104.80	77.55%
Bus 9	214	128.63	73.98%
Bus 10	306	183.74	85.46%
Average	161.8	97,19	110,74%

Table 4.4: Non-Simplified Routing Time and Distance Comparison

This table presents a comparison of travel time (in minutes), distance traveled (in Kilometers) and increase (in %) in time comparatively to the table 4.3, for each Bus, **without** utilizing the route simplification technique. Note these metrics account for the return of the bus to the initial position, which **does not** waste clients time.

Downloaded from <http://ajph.org/> on November 10, 2015

This figure represents two routes generated without using the simplified routing technique. The first two

visually confirm that the bus does stop at each individual location.

It's important to remember that the only distinction between Tables 4.3 and 4.4 is the application of the route simplification technique. As a result, the output from the aggregation component remains consistent across both cases. This means that each bus N has exactly the same stops allocated in both scenarios. Therefore, we can make a direct comparison of the times and distances based on the same routing parameters. This allows for a fair and accurate evaluation of the impact of the route simplification technique on our service's efficiency. The bar charts 4.5 and 4.6 provides a good reflection of the impact using the simplification has on the efficiency of the routing:

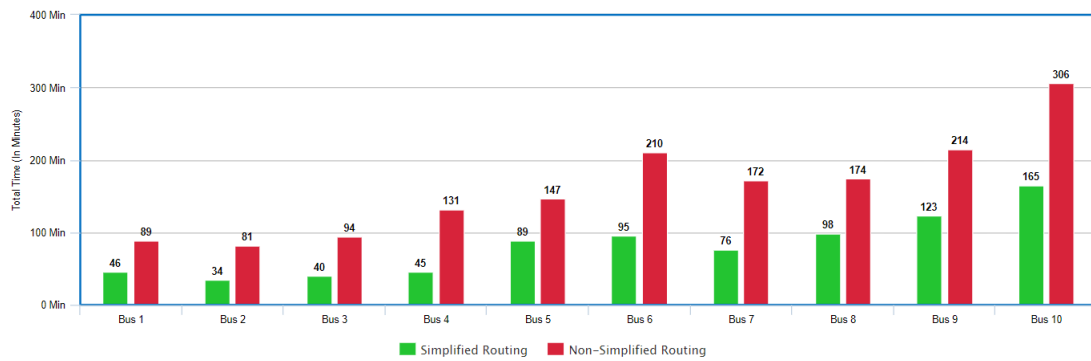


Figure 4.5: Time Comparison Bar Chart

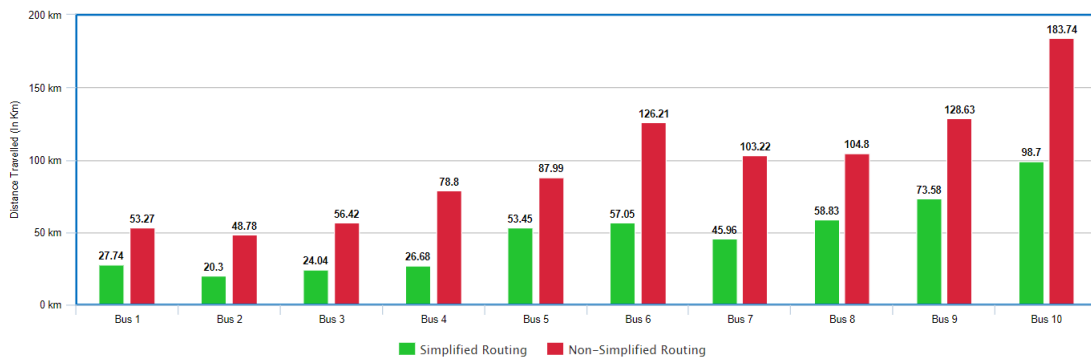


Figure 4.6: Distance Comparison Bar Chart

Referring to Figure 4.4, it is evident that Bus 2 follows an excessively intricate route for a rather confined drop off Zone. The plotted routes are designed with reduced opacity to highlight the frequency of their usage. It is readily apparent that the blue route consists of a recurring zig-zag pattern along the same roads. Furthermore, due to the bus being restricted to specific roads, without using simplification, it becomes a necessity to make a detour through the Gaia zone (below the river area) to reach certain Bus Stops.

Given the narrowness of the drop off zone, as previously mentioned, it serves perfectly as an effective benchmark for comparing the efficiency of the simplification technique in terms of time spent delivering passengers within a small area. Using the simplified Routing, since it only

relies on 1 bus stop to deliver all passengers, the bus takes only around **15 minutes** to reach it. Without using simplification, the bus reaches it's first stop around also **15 minutes**, but, dropping off the last passenger at around **52 minutes**, needs around **37 minutes** to deliver all passenger. In that same time, not only would the passengers from the simplified route would be home already, the bus itself would already be waiting to pick up new passengers (since the full route only takes 34 minutes as seen in table 4.3)

The pink route inherits the same problems. Even though at first glance it appears to be shorter than the pink road with simplification, many roads are being repeated, sometimes even being used **four** times. Besides suffering from a recurring reuse of the same roads, the routes is also characterized by repeatedly travelling back and fourth on confined areas, in addition to the empty areas it goes trough. Once again, this issues are due to the existing **STCP** routes.

In our last parameter test, we decided to see how the application behaves when working with the surrounding areas of Porto, including the Greater Porto Area. By integrating real-world data of taxi services that were requested around Parque da Cidade with destinations outside the limits of Porto, we gained insights into its behavior when compared to the narrower region, in order to know whether the service could be geographically extended:

- With Route Simplification

	Time (Minutes)	Distance (Km)
Bus 1	219	131.99
Bus 2	53	31.70
Bus 3	53	31.74
Bus 4	80	48.11
Bus 5	50	30.15
Bus 6	98	58.55
Bus 7	103	61.86
Bus 8	204	122.41
Bus 9	136	81.42
Bus 10	464	278.66
Average	244.7	87.66

Table 4.5: Simplified Route Time and Distance Comparison

This table presents a comparison of travel time (in minutes) and distance traveled (in Kilometers) for each Bus, utilizing the route simplification technique, accounting for bookings destined outside the Porto region.

At first glance, when looking at the values from table 4.5, some buses seems to take a satisfactory amount of time, with some of them even taking under an hour. However, this is not as good as it seems at first glance. Since the Routing algorithm works with the output from the Aggregation algorithm, buses tend to be responsible for a certain regions and, as such, some

buses still only take passengers to destination inside Porto, as it happens with Buses 2, 5, 7 and 9. For buses transporting passengers to destinations outside of Porto, we face additional challenges. The Grande Porto region encompasses a broad area, offering a wide range of bus stop options. As a result, many stops are requested by only one passenger, making them unique in their usage. Furthermore, stops outside Porto are more geographically dispersed compared to those within the city. This dispersion and low request rate lead to two key issues: the routes between stops become significantly longer, and it is difficult to consolidate these stops within the same bus route. This adds complexity when assigning passengers to buses that still have available seats, as stops outside Porto may be closer to certain destinations than others within the city.

Some examples of routes with these characteristics include those generated for buses 3 and 6. These buses are primarily assigned with booking destined to stops within Porto, except for one or two stops that, while outside the city, are closer to their drop-off zone than any other requested stop. As a result, these buses must take a detour to accommodate those passengers. Although, as shown in Table 4.5, the detours do not add significant time, they do require unnecessary travel to entirely different areas. In other cases, such as with bus 1, the route covers a distinct drop-off zone as intended, without much *zigzagging* or path deviations, but, due to the geographical dispersion of stops and natural long paths connecting them, the journey takes much longer to complete—over three and a half hours in this instance. A worst-case example is Bus 10, which is assigned an unreasonably large coverage area, picking up the remaining stops that could not be assigned elsewhere. This results in a route that takes 500 minutes to complete, making it an unfeasible option for the passengers.

Another notable value is the mean time required when including bus stops outside Porto, averaging 150 minutes. Although this exceeds the average time by nearly 70 minutes compared to routes that only consider stops within Porto with simplification, it is still more efficient than using the same stops without simplification, which takes 165 minutes. Let's focus the comparison these 2 specific buses, Bus 1 and Bus 10, represented in the figure 4.7 respectively:

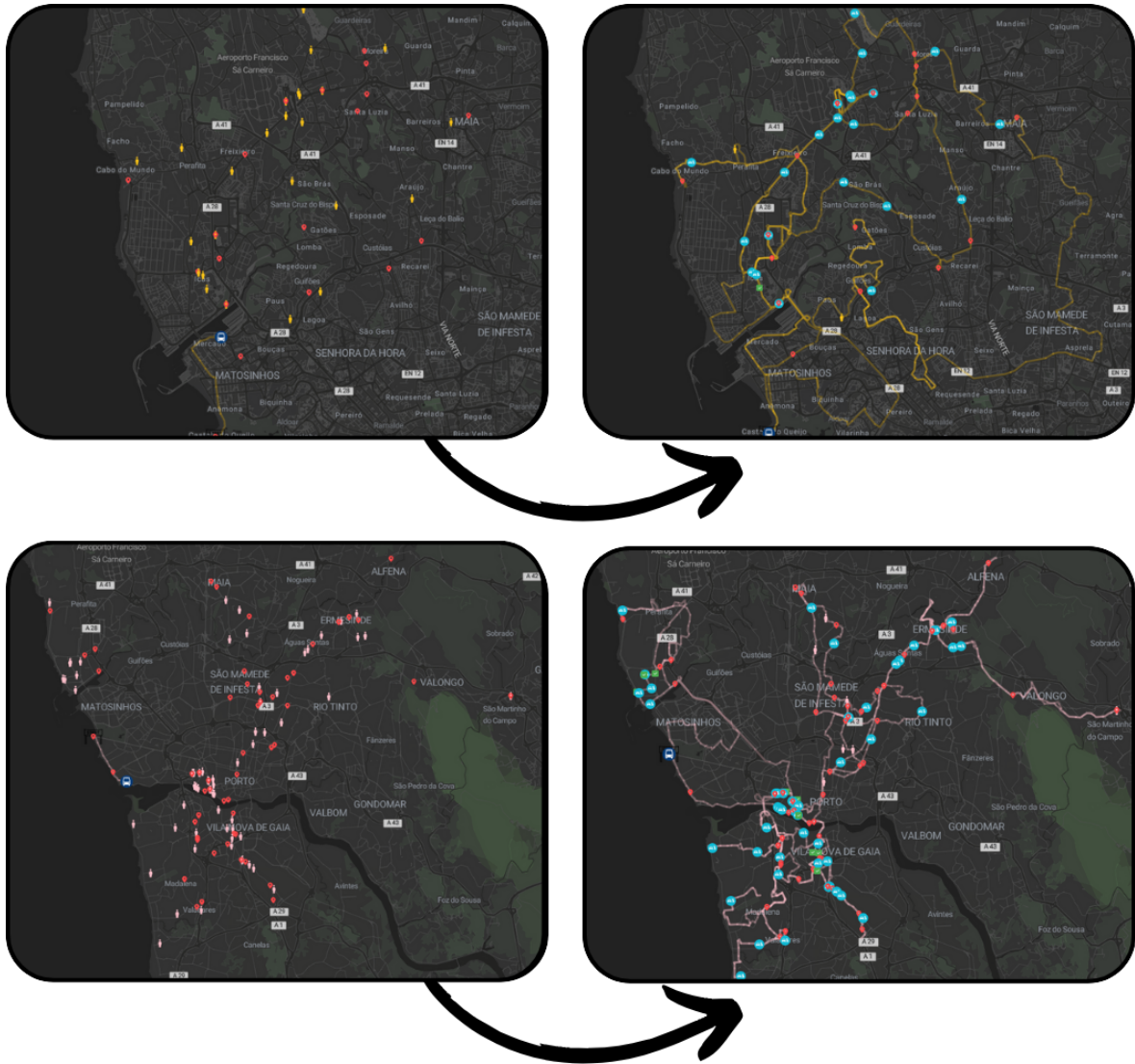


Figure 4.7: Greater Porto With Simplified Routing Examples

This figure represents two routes generated without using the simplified routing technique. The first two images are representative of the Bus 1 from the table 4.3 while the two images underneath it are representative of the Bus 10 from the same table

At first glance, figure 4.7 reveals two key observations: the widespread geographical distribution of stops (which was pointed out before) and the absence of check marks. The absence of check marks suggests that the simplification technique was not applied, as this icon denotes stops omitted due to their proximity within 500 meters. These observations are interconnected; the significant distance between stops results in long, straightforward routes, which limit the application of simplification and, consequently, many stops remain unoptimized, leading to extended travel times. Furthermore, bus 10, represented by the pink route, is exceptionally long and covers a vast area across Greater Porto, making it particularly challenging to optimize, having no feasible way to improve.

- Without Route Simplification

	Time (Minutes)	Distance (Km)	Increase in time (%)
Bus 1	316	189.74	44.30%
Bus 2	110	66.07	107.55%
Bus 3	113	67.51	113.20%
Bus 4	152	91.77	90.00%
Bus 5	175	105.26	250%
Bus 6	179	107.25	82.65%
Bus 7	223	133.72	116.51%
Bus 8	259	155.52	26.96%
Bus 9	223	133.96	63.97%
Bus 10	661	396.64	42.46%
Average	241.1	144.74	93.76%

Table 4.6: Non-Simplified Route Time and Distance Comparison

This table presents a comparison of travel time (in minutes), distance traveled (in Kilometers) and increase (in %) in time comparatively to the table 4.5, for each Bus, **without** utilizing the route simplification technique, accounting for bookings destined outside the Porto region.

This final table further underscores the advantages of the route simplification technique. When compared to Table 4.5, Table 4.6 shows a similar trend to the tables containing only Porto data, with significantly higher time and distance values compared to their counterparts. However, the increases are not as pronounced as in the previous tables (figures 4.3 and 4.4) as seen in the figures 4.8 and 4.9, but also by looking at the average increase in time, which is lower 20% than the increase from not using simplification on stops contained only within Porto. This can be attributed to the fact that bus stops outside Porto, as previously stated, are more widely spaced, and the routes are quite unique, which can easily be seen in the image 4.7. As a result, the benefits of simplification are not as substantial. In other words, the removal of stops and the use of edge adjacency are not applied as extensively, leading to higher values in the simplified route table. Consequently, the value derived from simplification is diminished.

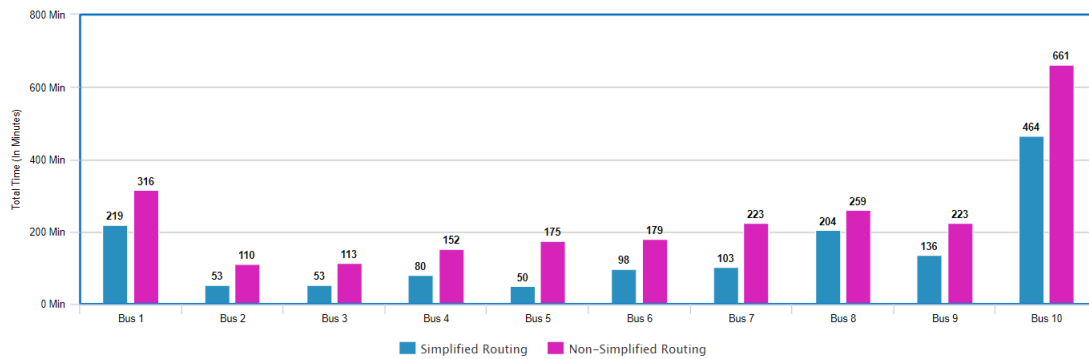


Figure 4.8: Greater Porto Time Comparison Bar Chart

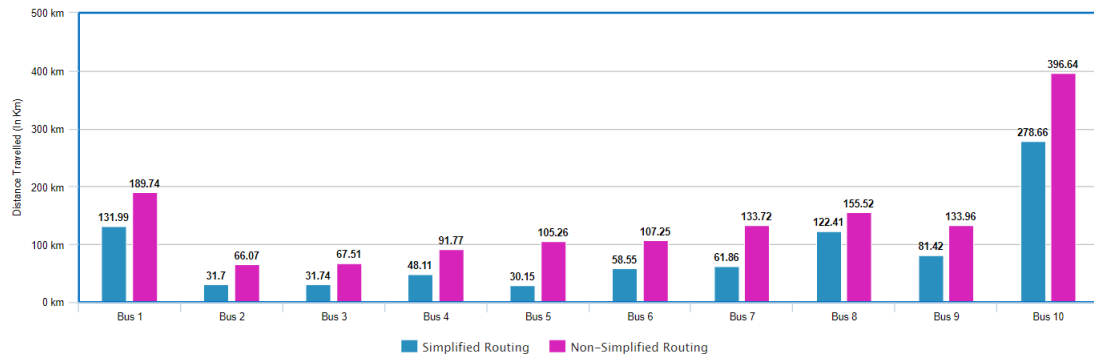


Figure 4.9: Greater Porto Distance Comparison Bar Chart

Furthermore, Bus 10 stands out with a total time of 600 minutes (10 hours), a duration that is absurd. For comparison purposes, we can see what the routes generated for bus 1 and 10 are in Figure 4.7:

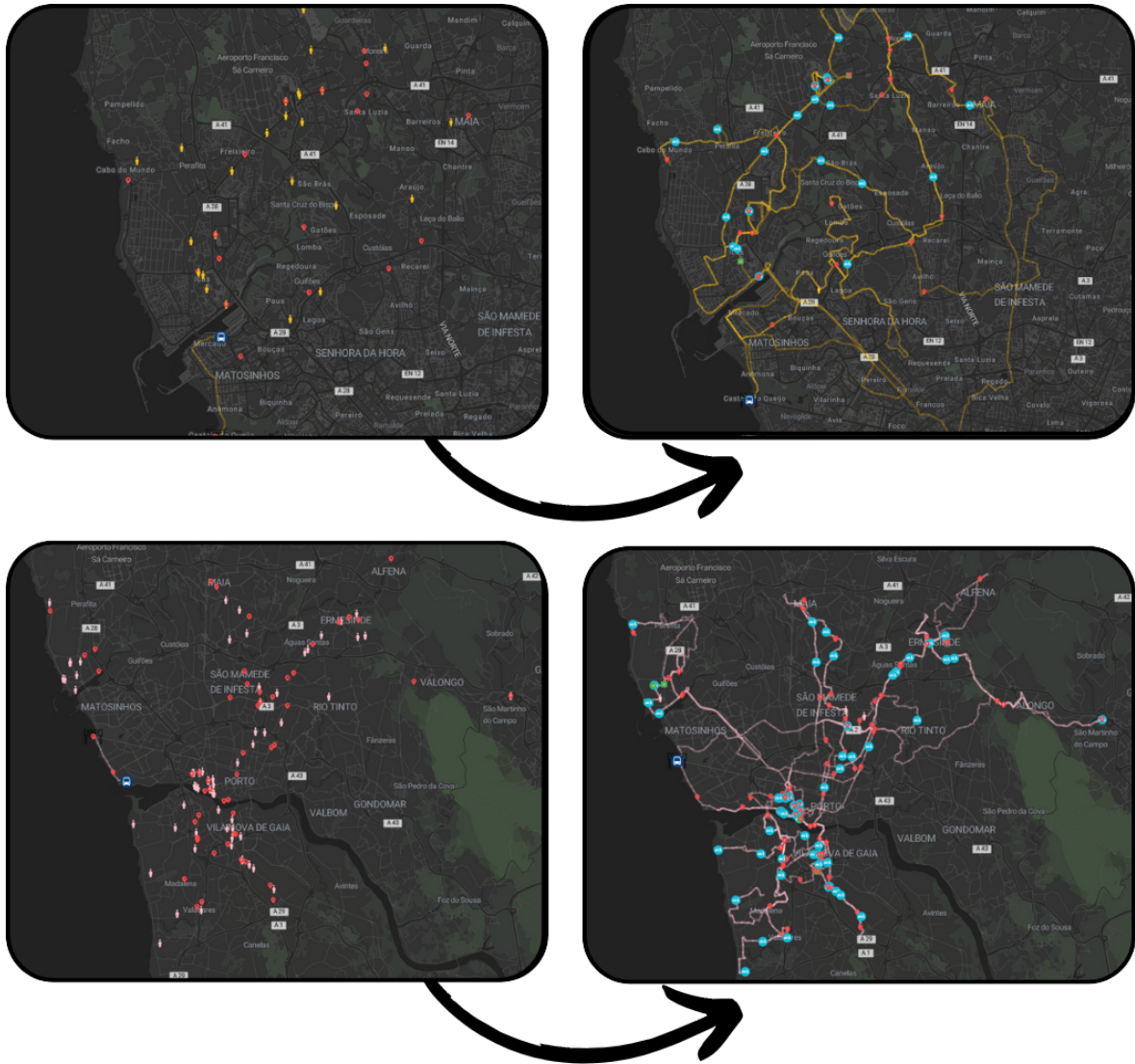


Figure 4.10: Greater Porto Without Simplified Routing Examples

This figure represents two routes generated without using the simplified routing technique. The first two images are representative of the Bus 1 from the table 4.3 while the two images underneath it are representative of the Bus 10 from the same table

Note: As seen in the figure 4.10, as with the figure 4.4, some Check mark icons (✓) appear despite simplification **not** being applied. This occurs due to synchronization issues in the animation. However, we can visually confirm that the bus does stop at each individual location.

Comparatively to 4.7, figure 4.10 does not appear to not introduce significant complications. It maintains the pattern of long paths between points, differing only in additional redundant zig-zag pathing, caused by the lack of edge adjacency, which was still applied in the simplified routing version.

While extending service coverage to regions outside of Porto could be beneficial for many, it also introduces a potential issue of increased travel time, which may not be feasible for clients.

Given the lack of a predictable geographical distribution of bookings, the total time required to deliver all passengers can be unpredictable and potentially excessive.

However, a potential solution to mitigate this routing problem, while still expanding service to a larger population, could be to allocate specific buses to four regions outside of Porto: **Gaia**, **Matosinhos**, **Maia**, and **Gondomar**. This approach would avoid the need for buses to traverse the entire region. Instead, they could follow direct routes to their respective zones, dropping off passengers who intend to travel there. This could alleviate concerns about excessive travel times and improve overall service efficiency.

After analysing the Data from the previous comparison tables, such as 4.3, 4.4, 4.5 and 4.6, there is one aspect that remains consistent across all scenarios: the **last bus** is always the **most time-consuming**. As detailed in the development section, there are both best-case and worst-case probable scenarios. In this instance, the last bus falls into the worst-case scenario, as illustrated in 3.23. Simply put, most buses are being responsible for a well defined geographical drop off zone but always missing some passengers that should be allocated to those vehicles but were left out due to capacity issues. Consequently, since they have to be placed somewhere, most of them also fall short to the other buses, that fill themselves with passengers from other distinct zones. Consequently, they all fall into the last bus, which ends up being responsible for taking the remaining passengers, resulting in a route that meanders across various areas, inherently 'zig-zagging' and detouring frequently, as we can see in the figure 4.11:

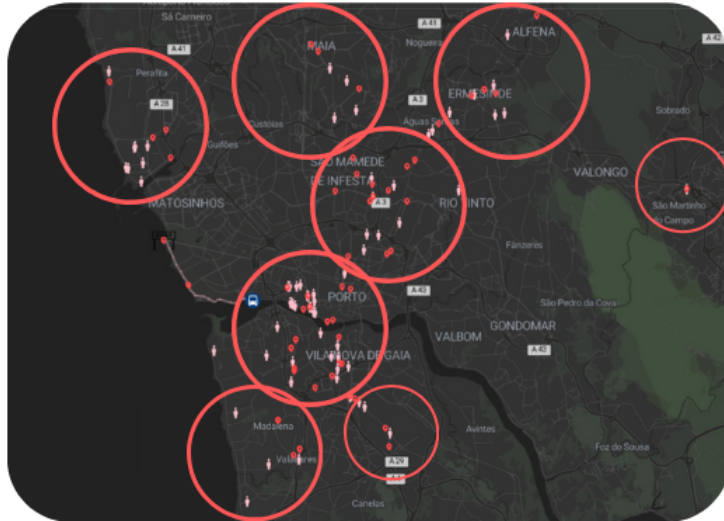


Figure 4.11: Route Map with very distinct drop off zones

This figure presents the passengers distribution for bus 10 (Greater Porto), with eight circles denoting the eight distinct and geographical distant zones assigned to it.

One feasible solution is to delineate a specific zone for each bus. This can be achieved by implementing either a **distance-based** or **stop-based** threshold. For instance, upon assigning the first bus stop to a bus, one could establish a maximum allowable radius that the bus can operate within, or alternatively, set a limit on the number of unique bus stops that can be

assigned to the same bus. This alternative can be implemented; however, it necessitates the allocation of additional vehicles for the project, as there is the possibility of leaving passengers unassigned due to distance or stops threshold limits.

Chapter 5

Conclusions

This dissertation has thoroughly examined the multifaceted challenges associated with large-scale event transportation, providing a new solution and alternative for transportation motivated by the critical issues of traffic congestion, carbon emissions, attendee safety, and transportation inefficiencies. By introducing and analyzing a new algorithm divided in two fundamental components—the aggregation component and the routing component—this research has provided a comprehensive solution to enhance transportation logistics during high-demand events. The development and integration of these algorithms within a web application offer a robust framework for managing event transportation in a more systematic and environmentally conscious manner, as well as providing the users with an easy and intuitive alternative of going home.

The comparative analysis of algorithmic complexity, particularly in relation to other Vehicle Routing Problem (VRP) solutions, has demonstrated the efficacy and advantages of the proposed algorithm. Furthermore, by evaluating both the simplified and non-simplified versions, as well as analyzing the original and the transformed networks of nodes and edges, this dissertation has highlighted the improved performance and adaptability of the proposed methods for the Porto region. These findings underscore the significant advancements in optimizing transportation routes and reducing operational inefficiencies.

In summary, this dissertation contributes to the field of vehicle routing, and sustainable transportation logistics, by presenting an innovative algorithm and a practical application tailored for large-scale events, capable of offering a platform for clients to easily request transportation. The successful integration of these solutions not only addresses the immediate challenges but also sets the stage for future advancements and adaptations in event transportation systems, either in other regions, but also with other methods of transportation. As cities and events continue to grow, the methodologies and technologies explored herein offer valuable insights and potential pathways for further research and development in creating smarter, more sustainable transportation solutions.

5.1 Future Work

While the work done for this dissertation already provides a solid base with a well defined idea, logic and architecture, there are a lot still yet to be defined and done in order to fully lunch the product into the world.

1. **Android Application:** While a Web App already provides the functionality needed and Developing dedicated Android and iOS applications will significantly enhance the user experience and accessibility of the transportation solution. Mobile apps can leverage device-specific features such as location services, push notifications, can improve the responsiveness and reliability of the service, as well as being a direct competitor to private transportation options.
2. **Define Methods of Payment:** One area that requires further development is establishing the payment methods for the service. Conducting a study to identify the most suitable solutions for all parties involved will be essential. Should clients pay on entering the bus or online? Should the payment be a fixed amount or depending on the booking location?
3. **Broader Geographic Implementation:** Explore the application of the solution in other cities and regions beyond Porto. Each new location will require adaptation to local transportation infrastructures, cultural norms, and regulatory environments. This expansion can provide valuable insights into the scalability and adaptability of the solution across different urban contexts. Moreover, we can incorporate real time geolocation technologies into the routing algorithm to make the routes dynamically adaptable to current conditions, such as road work and extreme traffic, and ensure that the passengers complete their journey in the most time and cost-efficient way.
4. **User Engagement and Community Building:** Conduct surveys and interviews in order to get direct insights into user preferences, experiences, and expectations. By understanding user needs and pain points, we can tailor the application to better serve its audience, as well as helping identify areas for improvement.
5. **Data Analysis:** Implement robust data collection and analysis to gather relevant information about user behavior, preferences, and interactions with the web/mobile application in order to improve it's UX/UI. Furthermore, we can analyze user behavior patterns, identifying bottlenecks or areas of problem, and making data-driven decisions to enhance the algorithm's capabilities.

Bibliography

- [1] Dario Ballarano, Sergio Maria Patella, and Francesco Asdrubali. [Sustainable transportation for events: A systematic review](#), 2022. ISSN: 20711050. doi:10.3390/su142315815.
- [2] Philip Bonacich. Power and centrality: A family of measures. *American Journal of Sociology*, 92(5):1170–1182, 1987.
- [3] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation Science*, 39(1):119–139, 2005.
- [4] Alessandro Montrone Cecilia Chirieleison and Luca Scrucca. [Event sustainability and sustainable transportation: a positive reciprocal influence](#). *Journal of Sustainable Tourism*, 28(2):240–262, 2020. doi:10.1080/09669582.2019.1607361.
- [5] Cecilia Chirieleison and Luca Scrucca. [Event sustainability and transportation policy: A model-based cluster analysis for a cross-comparison of hallmark events](#). *Tourism Management Perspectives*, 24:72–85, 2017. ISSN: 2211-9736. doi:https://doi.org/10.1016/j.tmp.2017.07.020.
- [6] Stephen Cook. *P, NP, and the Question of Optimal Algorithms*. Cambridge University Press, 2012.
- [7] Jean-François Cordeau, Gilbert Laporte, Martin W. P. Savelsbergh, and Daniele Vigo. Vehicle routing. In *Handbook of Logistics and Supply-Chain Management*. Elsevier, 2002.
- [8] Edsger W. Dijkstra. [A note on two problems in connexion with graphs](#). *Numerische Mathematik*, 1:269–271, 1959. doi:10.1007/BF01386390.
- [9] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.
- [10] Euronews. [Dry january: Where in europe is drinking alcohol getting worse, and which countries have cut?](#), 2024. Image source: OECD Health Statistics 2022 and WHO Global Information System on Alcohol and Health (2019). Accessed: August 28, 2024.
- [11] Eurostat. [Population in the eu: An overview of demographic trends](#), 2024. Accessed: August 28, 2024.

- [12] Michael R. Evans, Dev Oliver, Xun Zhou, and Shashi Shekhar. [1. spatial big data](#). 2024. doi:10.1201/9781003406969-8.
- [13] Filipe Ferreira, Miguel Ribeiro, Luís Serrano, Marcelo Gaspar, and Jorge Julião. How smart is green? sustainable and intelligent mobility solutions for transporting people and goods. In João Rafael da Costa Sanches Galvão, Paulo Sérgio Duque de Brito, Filipe dos Santos Neves, Flávio Gabriel da Silva Craveiro, Henrique de Amorim Almeida, Joel Oliveira Correia Vasco, Luís Miguel Pires Neves, Ricardo de Jesus Gomes, Sandra de Jesus Martins Mourato, and Vânia Sofia Santos Ribeiro, editors, *Proceedings of the 1st International Conference on Water Energy Food and Sustainability (ICoWEFS 2021)*, pages 636–645, Cham, 2021. Springer International Publishing. ISBN: 978-3-030-75315-3.
- [14] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1): 35–41, 1977.
- [15] L. C. Freeman. Centrality in social networks: I. conceptual clarification. *Social Networks*, 1 (3):215–239, 1978.
- [16] Cristina Isabel Covelli Garrido, Alessandro Giovannini, Annalisa Mangone, and Fulvio Silvestri. [Managing urban mobility during big events through living lab approach](#). *Sustainability*, 2023. doi:10.3390/su151914566.
- [17] Bruce Golden, S. Raghavan, and E. Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, 2008.
- [18] Michael Held and Richard M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.
- [19] John Hershberger and Jack Snoeyink. Speeding up the douglas-peucker line-simplification algorithm. In *Proc. 5th International Symposium on Spatial Data Handling*, pages 134–143, 1992.
- [20] Yiping Jiang and Yufei Yuan. [Emergency logistics in a large-scale disaster context: Achievements and challenges](#), 2019. ISSN: 16604601. doi:10.3390/ijerph16050779.
- [21] Mehdi Khadem, Samaneh Moradi, Majid Ara, Saeed Bahrami, and Adel Pourghader Chobar. A multi-objective location-routing problem model for multi-device relief logistics under uncertainty using meta-heuristic algorithm. *International Journal of Applied Industrial Engineering*, 9:354–373, 09 2021.
- [22] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [23] Grigorios D. Konstantakopoulos, Sotiris P. Gayialis, Evripidis P. Kechagias, Georgios A. Papadopoulos, and Ilias P. Tatsiopoulos. [An algorithmic approach for sustainable and collaborative logistics: A case study in greece](#). *International Journal of Information Management Data Insights*, 1, 2021. ISSN: 26670968. doi:10.1016/j.jjime.2021.100010.

- [24] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations (ICLR)*, 2019.
- [25] Gilbert Laporte. Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416, 2009.
- [26] Baoding Liu. [Vehicle routing problem](#), 2009. doi:10.1007/978-3-540-89484-1_10.
- [27] Nirvana Meratnia and Rolf A. de By. [Spatiotemporal compression techniques for moving point objects](#). *Advances in Database Technology—EDBT 2004*, pages 765–782, 2004. doi:10.1007/978-3-540-24741-8_43.
- [28] Ashley Montanaro. [Quantum algorithms: an overview](#). *arXiv: Quantum Physics*, 2015. doi:10.1038/NPJQI.2015.23.
- [29] Muhammad Irfan Nazmi Mohamad Nazri, Afiza Ismail, Azhar Abdul Aziz, and Suzana Zambri. [2. uitmbustracker mobile application of bus transportation tracker for uitm students](#), 2023. doi:10.1109/icraie59459.2023.10468403.
- [30] Benni Agung Nugroho, Abidatul Izzah, and Kunti Eliyen. [Mobile application development to solve vehicle routing problems in marketing or tour trip planning](#). *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, 7(1):27–33, 2023. doi:10.29207/resti.v7i1.4552.
- [31] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Mineola, New York, 1st edition, 1998. Covers brute force methods for NP-hard problems including TSP.
- [32] Nikica Perić, Slaven Begović, Mohan Muniappan, and Vinko Lesic. [3. vehicle routing problem considering electric vehicle battery consumption](#), 2023. doi:10.1109/iceccme57830.2023.10252897.
- [33] pgRouting Community. [pgrouting: Open source routing library](#), 2024. Accessed February 1, 2024.
- [34] Nelishia Pillay and Rong Qu. [Vehicle routing problems](#), 2018. doi:10.1007/978-3-319-96514-7_7.
- [35] PostGIS Development Team. Postgis. <https://postgis.net/>, 2024. Version 3.2. Accessed January 1, 2024.
- [36] PostgreSQL Global Development Group. Postgresql. <https://www.postgresql.org/>, 2024. Version 11.3. Accessed January 1, 2024.
- [37] Abdel Rahman Ragheb, Mai Abdo, and Khalid S. Al-Hagla. Towards sustainable urban transportation planning in developing countries: “drt” mobile apps as a catalyst for big data-based decision support system, 2022.
- [38] Faheem Ur Rehman, Md Monirul Islam, Mir Zat Ullah, Shabeer Khan, and Mohd Ziaur Rehman. [Information digitalization and renewable electricity generation: Evidence from south asian countries](#). *Energy Reports*, 9, 2023. ISSN: 23524847. doi:10.1016/j.egyr.2023.03.112.

-
- [39] Susan Shaheen, Adam Cohen, and Emily Farrar. *Carsharing's impact and future*, volume 4. 2019. doi:10.1016/bs.atpp.2019.09.002.
- [40] Paolo Toth and Daniele Vigo. *The vehicle routing problem*, 2001. doi:10.1137/1.9780898718515.
- [41] Mahes Visvalingam and J. Duncan Whyatt. *Line generalisation by repeated elimination of points*. *Cartographic Journal*, 30(1):46–51, 1993. doi:10.1179/000870493786962263.