# Optimizing bus networks from multimodal travel flows

**Vasco Dias Silva**

Thesis to obtain the Master of Science Degree in

## Information Systems and Computer Engineering

Supervisors: Prof. Rui Miguel Carrasqueiro Henriques
Drª. Anna Carolina Nametala Finamore do Couto

## Examination Committee

Chairperson: Prof. José Luís Brinquete Borbinha
Supervisor: Prof. Rui Miguel Carrasqueiro Henriques
Member of the Committee: Prof. Vasco Miguel Gomes Nunes Manquinho

**October 2021**

# Acknowledgments

First and foremost, a huge thanks goes out to my supervisors, Prof. Rui Henriques and Prof. Ana Finamore. Their council and feedback was a huge driving force for this work. Our meetings were always uplifting experiences that allowed me to move on from various impasses, either because of our productive discussions, their rich new ideas or their encouraging comments.

I am profoundly grateful to my family, in particular my parents and my brother. Being at home during the development of this thesis, their effort in guaranteeing ideal working conditions for me was outstanding. They also encouraged me in my work and provided unconditional support in the hard times.

A big thanks goes out to all my close friends. In particular I would like to thank: Rodrigo Moreira for our interesting discussions that allowed me to spring my work forward; Gabriel Quintas for our conversations and our Stellaris and D&D time that provided much needed distraction and laughs during the quarantine time; Hugo Fernandes for our conversations about the past and LoR and Minecraft time that got me through the low times; Filipe Almeida for his enthusiasm and positive mindset and for getting me into some of his projects which gave me some much needed time away from my computer.

# Abstract

Mobility is a key part of modern civilization. The COVID-19 pandemic made apparent the necessity to adapt our public transport systems to the ever-changing users' needs. In this work we propose a solution to improve the bus network in the city of Lisbon, Portugal. The system takes individual trip data given by smartcard validations at CARRIS buses and METRO stations, uses them to estimate the origin-destination demand in the city, and then designs a network that better fits that demand. Route scheduling is optimized separately from the topology. To these ends, Genetic Algorithms are used in both single and multi objective approaches. The single objective formulation is based on a human rating of Pareto Optimal networks drawn from multi objective solutions. A regression model is used to infer the weighted sum of the different objectives. The single objective optimization processes proved to improve on the multi objective optimization results with reductions in objective functions up to $28.3\%$. The system manages to reduce the number of routes in the network from 309 to 200 which then corresponds to a daily reduction in $59.8\%$ of distance covered by buses when frequencies are optimized. All the passenger related objectives, including travel time and transfers per trip are improved with only the unsatisfied demand slightly increasing from $0.7\%$ to $1.3\%$.

# Keywords

Public Transport Network; Optimization; Route Planning; Multimodality; Sustainable Mobility

# Resumo

Mobilidade é um dos fatores determinantes na sociedade moderna. A pandemia COVID-19 realçou a necessidade de adaptar a oferta nos transportes públicos à constante oscilação na procura. Neste trabalho, propomos uma solução para melhorar a rede de autocarros na cidade de Lisboa, Portugal. O sistema utiliza dados de viagens recolhidas através da validação de cartões à entrada dos autocarros da CARRIS e nas estações de METRO para inferir a procura origem-destino na cidade e posteriormente, desenhar uma rede de autocarros que melhor se ajuste ao cenário de procura. A configuração da rede e as frequências são otimizadas separadamente. Para o efeito, algoritmos genéticos são usados para otimização multiobjetivo e otimização uniobjetivo. A formulação do problema uniobjetivo é baseada num processo de avaliação manual das redes pareto-ótimas que resultam do processo de otimização multiobjetivo. Regressão linear é usada para inferir pesos para uma soma ponderada dos diferentes objetivos. Os resultados dos processos uniobjetivo melhoraram em relação aos resultados multiobjetivo, com reduções na função objetivo de até $28.3\%$. O sistema reduz o número de rotas na rede de 309 para 200, o que corresponde a uma redução de $59.8\%$ na distância diária percorrida pelos autocarros depois de otimizadas as frequências. Todos os objetivos relativos à qualidade dos serviço, incluindo tempos de viagem e transferências por viagem, são melhorados com a exceção da procura não satisfeita que aumenta ligeiramente de $0.7\%$ a $1.3\%$

# Palavras Chave

Rede de Transportes Públicos; Otimização; Planemento de Rotas; Multimodalidade; Mobilidade Sustentável

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Acronyms

| | |
|---|---|
| **AON** | All-or-nothing |
| **ACO** | Ant Colony Optimization |
| **BFS** | Breadth First Search |
| **CRGA** | Candidate Route-set Generation Algorithm |
| **CSV** | Comma Separated Values |
| **CNDP** | Continuos Network Design Problem |
| **DNDP** | Discrete Network Design Problem |
| **FSLC** | Fixed String Length Configuration |
| **GTFS** | General Transit Feed Specification |
| **GA** | Genetic Algorithm |
| **MNDP** | Mixed Network Design Problem |
| **MMNDP** | Multimodal Network Design Problem |
| **NAP** | Network Analysis Procedure |
| **NSGA-II** | Non-dominated Sorting Genetic Algorithm II |
| **OSRM** | Open Source Routing Machine |
| **OSM** | Open Street Map |
| **OD** | Origin-Destination |
| **PSO** | Particle Swarm Optimization |
| **RNDP** | Road Network Design Problem |

| | |
|---|---|
| **TAZ** | Traffic Analysis Zone |
| **TNFSP** | Transit Network Frequencies Setting Problem |
| **TNDFSP** | Transit Network Design and Frequencies Setting Problem |
| **TNDSP** | Transit Network Design and Scheduling Problem |
| **TNDP** | Transit Network Design Problem |
| **TNTP** | Transit Network Timetabling Problem |
| **TNSP** | Transit Network Scheduling Problem |
| **TSP** | Traveling Salesman Problem |
| **UTNDP** | Urban Transportation Network Design Problem |
| **VSLC** | Variable String Length Configuration |

# Part I

# Foundations

# 1 Introduction

**Contents**

Mobility is a central aspect in modern societies, allowing people to take part in a multitude of activities that are the identity of our days. With the worldwide population increasing, and more people using the roads, problems such as congestion and air pollution arise. One way to address these challenges, is to provide the community with an efficient public transport network. Public transports carry more individuals and generally pollute less than personal vehicles. Yet, they have to be convenient for the users, otherwise they will continue to use their personal vehicles.

The COVID-19 pandemic radically changed people's habits and even during periods of normal activity, over time, demand will change and efforts should be made to make sure the Public Transport Networks meet individuals' needs. Additionally, new environmentally friendly modes of transportation, such as car and bike sharing, are available and therefore it is important to explore synergies between all the existing modes of transportation. By working as an extension of the existing network, passenger load can be properly distributed and, in turn, clutter can be reduced, improving user experience. Aligning the offer of the different transportation modes also provides users with a wider array of choices and has the potential to satisfy more demand, improving flexibility and coverage.

## 1.1 Research Problem

The goal of this work is to propose an improved design to CARRIS bus network, one that uses the available buses in a more efficient way and serves the clients as best as possible with the available resources, while trying to reduce the environmental impact of the network. Here, quality of service means providing a service with good city *coverage*, low *travel times*, low *transfer rates*, low *waiting times* and, satisfying *safety guidelines* such as maximum occupancy and recommended individual distances.

Even though the CARRIS network is the focus of this work, the system is extendable and applicable to the optimization of any transportation network. All the optimization and design methods in this work are valid for any transportation network, given the necessary data.

This work uses previous contributions by Cerqueira et al. [2] for alighting stop estimation and origin-destination (OD) matrix inference as starting points. Our work is based on actual data collected from the integrative VIVA-card fare collection system and official routing publications which are used to estimate the travel demand that then serves as input to an optimization algorithm that will find a (near) optimal design.

In this work, we work with traffic data from October 2019. The CARRIS bus network deployed at that time can be seen in Figure 1.1. The network has 309 routes and 2,193 stops. Each route has, on average, 26.2 stops and each station, on average serves 3.7 routes. For the month of October 2019, we have 6.2 million smartcard validations at the bus entrances, spanning 4 days. These validations, as well as validations on the Metro network, are used as the input for the work of Cerqueira et al. [2] for inferring

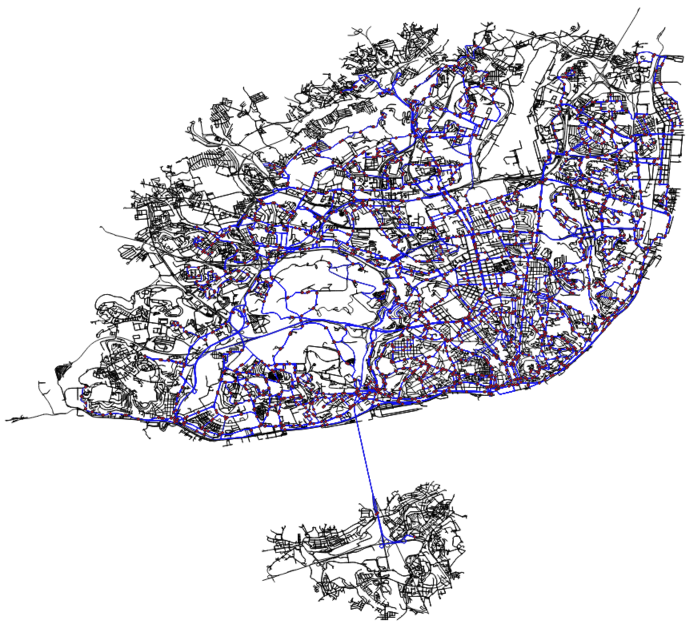


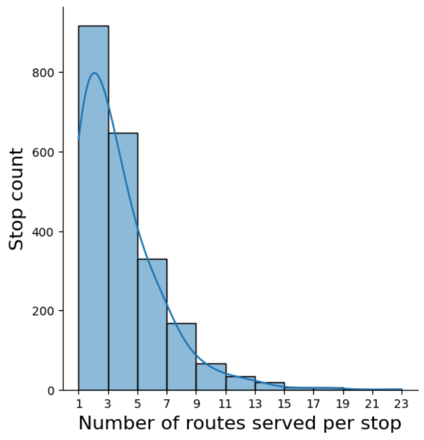
**Figure 1.1:** CARRIS network in October 2019

the OD demand. In Figure 1.2, we present the distribution of some variables regarding the data and the network.

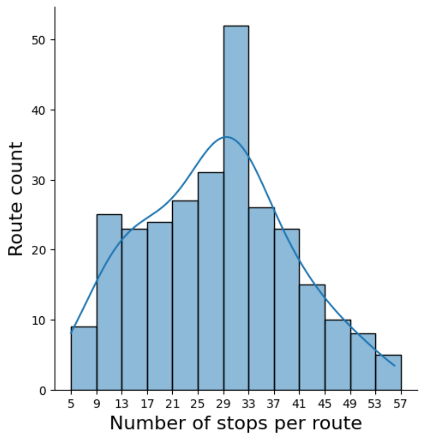
In Figure 1.2a, we can see that most stops serve only a small number of routes but there are some central stops that serve more routes. Corpo Santo is the stop that serves the most routes, a stop in downtown Lisbon, near Cais do Sodré, which is a transportation hub where passengers can exchange between, buses, ferry-boats, trains, trams and metro. It is also near to Terreiro do Paço and Baixa do Chiado, all destinations in the heart of the historical part of the city which justifies its centrality as it is likely used a lot by people visiting the city. In Figure 1.2b we can see that routes vary greatly in size. The biggest route works only at night and it connects Cais do Sodré to Gare do Oriente, traversing the whole city. This route has small bus frequency. The smallest route works during the day and it has a high bus frequency. This small route operates in the historic part of the city. In Figure 1.2c and Figure 1.2d, we can see how the validations are distributed. Most validations happen in a small number of routes and stops. The distribution of validations per station even follows a power law distribution.

## 1.2 Research Contributions

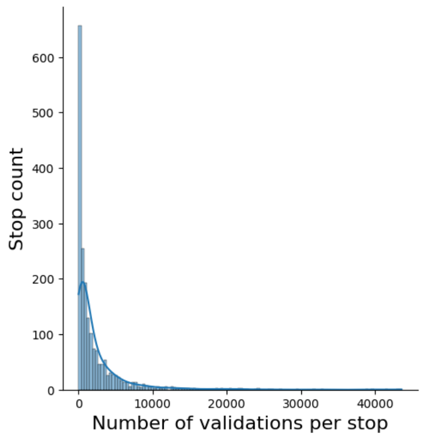
The present work provides a comprehensive comparison of multi objective and single objective genetic optimization methods when applied to Network Design. To the best of our knowledge, previous works in this line of research, when confronted with a multi objective problem, convert the problem to a single objective formulation and then solve it using single objective algorithms. Those algorithms heavily reduce



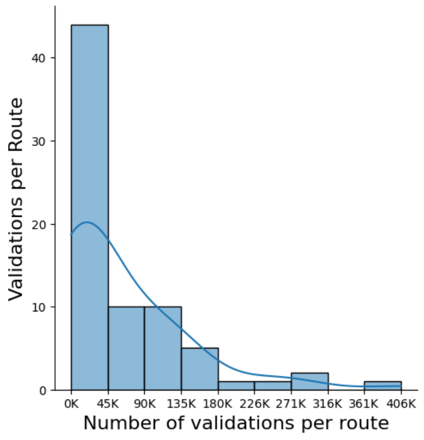
**(a)** Number of routes passing through each stop



**(b)** Number of stops covered by each route



**(c)** Number of validations in each stop during October 2019



**(d)** Number of validations in each route during October 2019

**Figure 1.2:** CARRIS network and traffic statistics in October 2019

the scope of the crafted solutions as it needs the user to know what kind of compromise they want between the different objectives a priori, which is no easy task. In this work, we aim to inform the user of the possibilities with the results from multi objective optimization and then provide the best possible network that goes towards the user's preference with a single objective optimization process. We do this by rating the Pareto Optimal networks and then using a linear regression to infer the weights for a single optimization process.

This work also aims to assess the importance of context in the Network Design process. Route configuration and route frequencies are optimized in different optimization stages to allow frequencies to better serve the different use patterns we find in the network throughout a work day.

## 1.3 Document Outline

This document is divided into three major parts. Part I introduces the necessary concepts, algorithms and techniques used in this area of research in Chapter 2 and goes over previous work done in transit network optimization in Chapter 3. Part II is all about describing the implemented solution and grounding our choices in the related work. In Chapter 4 we go over the preparations needed to be able to optimize a bus network, in Chapter 5 we discuss metrics to evaluate a bus network and how they can be computed and in Chapter 6 we talk about how we model this problem as a genetic problem, i.e. a problem solvable with Genetic Algorithms. In Part III, we present and discuss our results and reflect on what could be done differently in Chapter 7 and talk about possibilities for future work in Chapter 8.

# 2
# Background

**Contents**

This Chapter reviews essential concepts and establishes a notation for the remaining parts of this work. We first look at how we model a bus network in Section 2.1. Section 2.2 covers essentials on optimization problems.

## 2.1 Bus Networks

Generally, networks (or graphs), $G = (N, E)$, are represented as a finite non-empty set of nodes $N$ and a set of edges (or links) between those nodes $E \subseteq \{(i, j) \mid i, j \in N\}$. These links can be directed or undirected. If links are directed, then $(i, j)$ and $(j, i)$ are different edges. An example of a simple undirected network can be seen in Figure 2.1. Links can have quantities associated with them which can represent a variety of things, depending on the application context. In our problem, they will be costs and we will represent the cost associated with edge $(i, j)$ as $c_{ij}$.

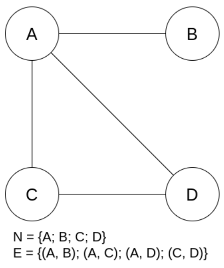


**Figure 2.1:** Example of a simple network

Our work will focus on designing a bus network on top of the existing road network. In an urban traffic network, road sections will be links and road interceptions, as well as bus stops, will be nodes. The cost associated with each link will be a reflection of how long it takes to travel through that road section and the operation costs of vehicles using that section. Links can be directed or undirected depending on the road network being dealt with, if there are one way streets, then directed links should be used. One small example of a bus network can be seen in Figure 2.2.

Usually, the units of a bus network are routes, which are (usually) acyclic paths on the road network to which buses are assigned. A route $r$ has associated a given bus frequency $f_r$ or a bus headway $h_r$ (time interval between buses). Given a road graph $\langle N, E\rangle$, a bus network is a non-empty set of routes $R \subseteq \{(i_1, \ldots, i_k) \mid i_1, \ldots, i_k \in N;\ 2 \leq k \leq |N|; (i_1, i_2), \ldots, (i_{k-1}, i_k) \in E\}$.

An Origin-Destination (OD) pair is representative of movement from an Origin to a Destination. $W \subseteq \{(s, t) \mid s \in W^O \subseteq N,\ t \in W^D \subseteq N\}$ is the set of all OD pairs with $W^O$ and $W^D$ being, respectively, the origin set and the destination set. The demand of the network is $Q = \{q_w \mid w \in W\}$. In other words, for each OD pair, we define the number of times that passengers need to make the trip within a time frame. We can conveniently represent this information in an OD matrix, $D$, with dimension $|W^O| \times |W^D|$ in which $D_{st} = q_{(s,t)}$.

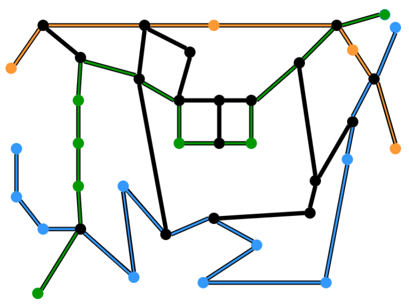


**Figure 2.2:** Example of a simple bus network on top of a road network (black). There are three routes in this bus network, green, orange and blue. Colored nodes are stops on a route while black nodes are road intersections. Note that, a station or road segment can serve more than one bus route.

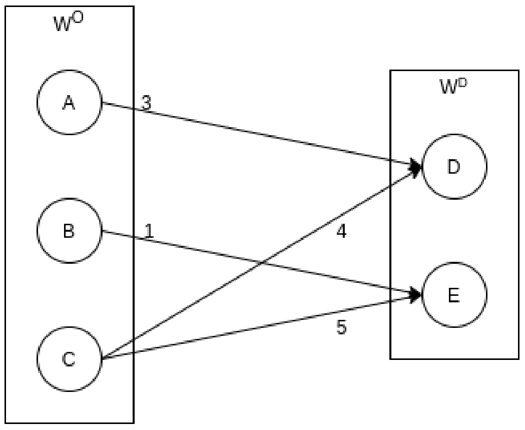


**Figure 2.3:** Example of demand between points in the Origin Set and the Destiny Set. The numbers along the lines represent the amount of passengers doing the trip withing a specific time frame. The OD matrix for the example is shown in Equation 2.1.

As an example, if we have the demand depicted in Figure 2.3, we would have the following OD matrix:

$$D = \begin{array}{cc} D & E \\ \end{array} \\ \begin{bmatrix} 3 & 0 \\ 0 & 1 \\ 4 & 5 \end{bmatrix} \begin{array}{c} A \\ B \\ C \end{array} \tag{2.1}$$

## 2.2 Optimization

Finding the best elements, $\mathbf{x}^*$, from a set of alternatives, $\mathbb{X}$, according to a set of criteria, $F = \{f_1, f_2, ..., f_m\}$ is the essence of optimization. $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ is a set of what is called design or decision variables and they model the real world entities that make up a solution in the problem domain. In the context of this work, we are trying to find the optimal bus network for the city of Lisbon given the OD demand.

$f_i : \mathbb{X} \mapsto Y, \quad (i = 1, 2, ...m)$, with $Y \subseteq \mathbb{R}$, are the criteria or objective functions. Usually these types of problems are constrained, for example, in our network design problem we can only go so high in the frequency of a given route because CARRIS only has so many buses. Illustrating, one can formalize optimization problems as follows:

$$\begin{aligned} \underset{\mathbf{x} \in \mathbb{X}}{\text{minimize}} \quad & f_i(\mathbf{x}), \quad (i = 1, 2, ..., m) \\ \text{subject to} \quad & h_j(\mathbf{x}) = 0, \quad (j = 1, 2, ..., o) \\ & g_k(\mathbf{x}) \leq 0, \quad (k = 1, 2, ..., p) \quad . \end{aligned} \tag{2.2}$$

We say we have a single objective problem if $m = 1$ and a multi objective problem if $m > 1$. Problems may have no objective function, in which case we call them, feasibility problems. This is the case of the famous 8-queen problem [1], in which one has to place 8 queens in a board of chess without any of them being able to attack another.

If $\mathbb{X} \subseteq \mathbb{R}^n$, then we can see potential solutions as vectors, design vectors, i.e. $\mathbf{x} = (x_1, x_2, ..., x_n)^T$ and we can define some concepts that will helps us understand the main advantages and drawbacks of optimization solutions.

The space spanned by the design vector is called design space and the space formed by the objective function values is called the solution space.

We usually refer to the way the objective functions' values change with the different assignments as the landscape. For example, if we have a non constrained, single objective optimization problem with a single variable, in which we try to minimize $y = f(x)$, then the landscape is just the plot of the function. Landscapes can have some features that certain algorithms deal well with and others, not so much. The goal is to reach the lowest point in the landscape, i.e. the assignment to the design variables that minimizes the objective functions, this is a global minimum. If a given point is the lowest in the vicinity, this is a local minimum. We can similarly define global and local maximum, but those are relevant if we are trying to maximize the objective functions only. If the objective function does not change in a neighborhood of points, that is a plateau. A landscape example can be seen in Figure 2.4.

One of the easiest ways to deal with multiple objective functions is to define a new single one, $f'$ that is a weighted sum of all $f_i \in F$, thus converting a multi objective problem into a single objective one:

$$f'(\mathbf{x}) = \sum_{i=1}^{m} w_i f_i(\mathbf{x}) \quad . \tag{2.3}$$

Now we can define the optimal set of elements as $\mathbf{x}^* \in \mathbb{X} : f'(\mathbf{x}^*) \geq f'(\mathbf{x}), \ \forall \mathbf{x} \in \mathbb{X}$. This formulation should only be used if the objective functions share the same big-O notation, otherwise some functions

[1] https://en.wikipedia.org/wiki/Eight_queens_puzzle

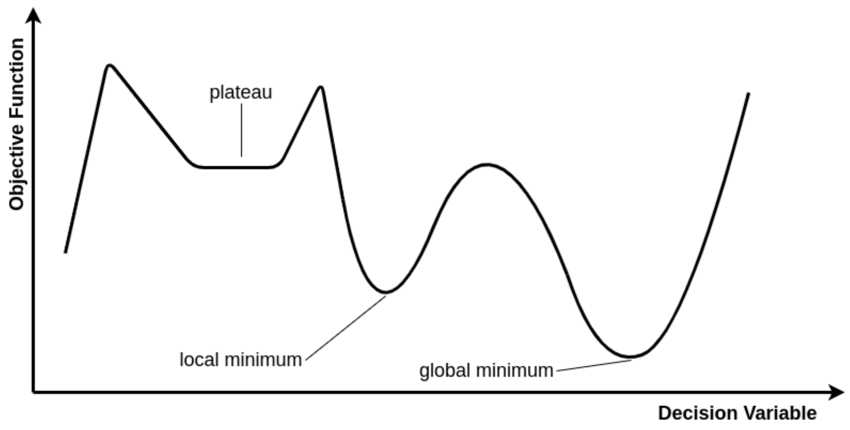


**Figure 2.4:** Landscape example with global and local minimums as well as a plateau.

can become irrelevant.

We can also incorporate the constraints into the objective function with the so called penalty functions. The idea is to steer the algorithm naturally to feasible solutions by penalizing the objective function when constraints are not met. First, it is note worthy that $h(\mathbf{x}) = 0 \Leftrightarrow h(x) \leq 0 \land -h(x) \leq 0$ and so we can disregard all equality constraints. One possible penalty function is the quadratic loss function. We can incorporate the inequalities in our objective with the quadratic loss function as follows:

$$f''(\mathbf{x}) = f'(\mathbf{x}) + \sum_{k=1}^{p} v_k \ \max\big(0, g_k(\mathbf{x})\big)^2 \quad . \tag{2.4}$$

One must be careful while using these formulations for semi-optimal solutions, if the solution is not optimal then the constraints might not be satisfied.

### 2.2.1 Pareto Optimality and Pareto Frontier

Pareto optimality is a relevant definition when dealing with multi objective problems. A solution is Pareto optimal if we cannot improve one objective without damaging the quality of the remaining objectives. A Pareto optimal solution compromises the different objectives in an optimal way. Different Pareto optimal solutions represent different balances between the objectives. The set of all Pareto optimal solutions is called the Pareto Frontier. Pareto optimality is tightly coupled to the definition of domination. A solution $\mathbf{x^1}$ dominates another solution $\mathbf{x^2}$, in which case, we write $\mathbf{x^1} \prec \mathbf{x^2}$ if, and only if:

$$\begin{aligned} &\forall i \in \{0, ..., m\} : f_i(\mathbf{x^1}) \leq f_i(\mathbf{x^2}) \quad \land \\ &\exists j \in \{0, ..., m\} : f_j(\mathbf{x^1}) < f_j(\mathbf{x^2}). \end{aligned} \tag{2.5}$$

Now we can define the Pareto Frontier, $\mathbf{X}^*$ as the set of points that are not dominated by any other solution. In other words:

$$\mathbf{x}^* \in \mathbf{X}^* \Leftrightarrow \nexists \mathbf{x} \in \mathbb{X} : \mathbf{x} \prec \mathbf{x}^* \tag{2.6}$$

An example of a Pareto Front for a constrained bi-objective problem in objective space can be seen in Figure 2.5. Multi objective algorithms strive to find an approximation of this front for the given problem.

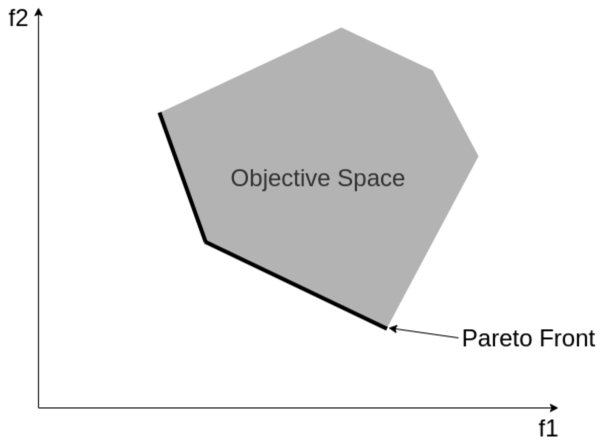


**Figure 2.5:** Example of a Pareto front in objective space.

### 2.2.2 Single Objective versus Multi Objective Optimization

Single objective problems have a single optimal solution, while in multi objective problems all the solutions in the Pareto Front are optimal in the Pareto sense. Turning multi objective problems into single objective problems, as we did in Equation 2.2, means that we will be searching for a single optimal solution in the Pareto Front. Changing the constants $w_i$ will lead us to a different equilibrium between the objectives, but, nonetheless, we have to have an idea of the kind of balance we want between the different objectives upfront, which is not always easy to assess and is very dependent on the problem domain. One could run the single objective algorithm multiple times with different weights but an evenly distributed set of weights fails to produce an even distribution of points from all parts of the Pareto set [3], leaving options unexplored. We can search for the Pareto Front and study the solutions found there. That will give us an idea of the kind of balance we want and then we can run a single objective algorithm to try and optimize that solution further, now with a clear idea of what we are looking for, specifically.

### 2.2.3 Multi Objective Optimization Quality Metric - Hypervolume

Hypervolume is a measure to compare solution sets. In Multi Objective Optimization, the products of the algorithms are approximations of the Pareto Front, which is a solution set. In order to compare the performance of different algorithms or different parameterizations, we can use the hypervolume metric.

The hypervolume of a solution set is essentially the union of the hypervolumes defined by the solutions in the solution set and a reference point or negative utopia point which is just a bad solution we use as a baseline. In Figure 2.6 we have the example of an hypervolume (in this case, area) for a bi-objective problem. Because we are trying to minimize objectives, the higher the hypervolume, the better the solution set.

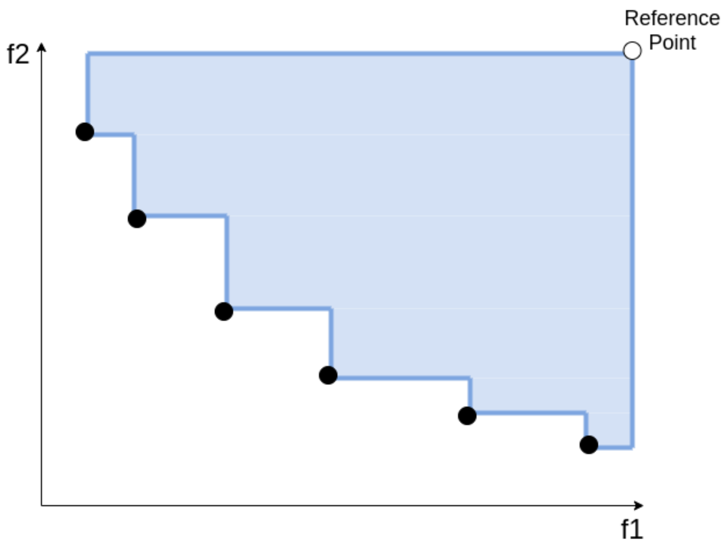


**Figure 2.6:** Example of hypervolume for a bi-objective solution set. Black points are individual solutions.

### 2.2.4 Searching for Solutions

Classic approaches to solve optimization problems are deterministic. For example, if all objective functions and constraints are linear and we are working with non-discrete variable domains, we can use the simplex algorithm, or if the function to optimize is smooth and unimodal, then gradient based methods like Newton-Raphson method or Gradient Descent work very well. However, if the function is not smooth, we might have to revert to Branch and Bound and if $\mathbb{X} \not\subseteq \mathbb{R}^n$ we can use design space search. These algorithms are guaranteed to find us the best assignment to the design variables (except for gradient based methods). However, for problems of realistic size, we cannot pay the time price to have the optimal solution.

Monte-Carlo algorithms allow us to trade solution quality for a reasonable time requirement. These algorithms model the exploration of the state space as a stochastic process allowing us to cover a wider area, but the exploration will not be systematic, which means we can miss the optimal solution. As such, the results are not reproducible but the solutions provided will almost always be good, though not necessarily optimal.

A heuristic is a rule of thumb. They reflect the quality of a solution candidate by relying on domain specific knowledge and they are usually used to guide search, for example, Hill-Climbing might use an heuristic as a proxy for the objective function as they are usually faster to compute. For feasibility problems, i.e. no objective function, heuristics provide a way to evaluate how promising a certain search direction is and thus, are a good way to speed up searches. A metaheuristic is a general method, independent of the domain, to solve optimization problems that usually uses information about the objective functions or heuristics. Sometimes the words heuristic and metaheuristic are used interchangeably but the latter is commonly used as a word for algorithms that blend randomization and local search. Apart from Hill Climbing, all the algorithms detailed bellow are metaheuristics.

Yang [4] uses a good metaphor for optimization algorithms that can also help us create a taxonomy for the algorithms. Imagine that we are trying to find a treasure in a hilly landscape and that the treasure is placed in the highest point in that landscape. Understandably, it is not a good idea to sweep the whole area looking for the treasure if the landscape is wide, this is what systematic approaches do. Now, we can just run around randomly looking for the treasure but this will likely lead us nowhere. Because we know that the treasure is in the highest point in the landscape we can try to go up the steepest cliff but we might end up on top of a smaller peak, this is what the Hill-Climbing algorithm does. One good idea might be to mix the random walk with information about the local aspect of the landscape so we can know where it is worth to look, this is the basis of Simulated Annealing. We can also mark already explored spots with a flag so we know not to look there again. This is what Tabu Search does. We might also try and gather a group of people to search with us and share the information as we go along, this is what Particle Swarm Optimization does. Alternatively, the group might agree upon some marks that can be left on the landscape to guide others to a better search spot, this is what Ant Colony Optimization does. Finally, we can fire hunters that are not doing so good in their search and replace them with new ones while keeping the best hunters searching, this is the idea of Genetic Algorithms.

We will focus on the aforementioned Monte-Carlo strategies as the optimal network design problem is hard to solve [5] and systematic methods become intractable for problems of realistic size.

The metaheuristics we are going to discuss work, in an iterative way and at each time step $t$, a new potential solution $\mathbf{x}_t$ or a family of solutions $X_t = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_L\}$ is generated. Going back to our metaphor, every time we are looking for the treasure alone, we are using an algorithm that keeps only one candidate at each time step, these are trajectory-based algorithms. When we are looking for the treasure with a group, we keep multiple candidates at each time step, these are population-based algorithms. The way new candidates are generated is highly dependent on the problem at hands. In purely mathematical settings $\mathbf{x_{t+1}}$ is obtained by offsetting $\mathbf{x_t}$ with a vector drawn from a predetermined distribution. The sequence $\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_t}$ is a random walk, or random flight, hence the name trajectory-based. If the setting is not purely mathematical then, traditionally we define a set of operators that generate a new state or set of states from an existing one. Population-based methods use these operators in conjunction with other special operators to create new populations from older generations.

#### 2.2.4.A Hill Climbing

The idea of Hill Climbing [6] is very similar to the idea of Gradient Descent. The algorithm was designed to maximize objective functions, not minimize, hence the climbing. This is a trajectory-based algorithm so only one solution candidate is kept at a time. The idea is to keep the generated candidate with lower $f$ value at each time step. If we were maximizing, this would be going up hill through the steepest path. If at any given point the successors of a candidate do not provide an improvement, the algorithm stops. This is very susceptible to plateaus and local minimums as they will make the algorithm stop. One idea to improve the original algorithm is to do random restarts at different points in the landscape every time the algorithm stops in hopes that one of the trials will lead to the global minimum.

#### 2.2.4.B Simulated Annealing

In metallurgy, annealing is used to make metals softer. The materials start by being heated above their recrystallization temperature and then cooled at a slow rate so their crystals can organize in the lowest energy state possible. If the cooling is too fast, then the crystals will organize in a semi-optimal state that represents a local minimum of energy. This process inspired the creation of Simulated Annealing [7].

Simulated Annealing uses randomness to try and escape local minimums. This randomness is controlled by the temperature parameter, $T$, that, just like in annealing, decreases over time. The algorithm starts with a solution candidate $\mathbf{x}_1 \in \mathbb{X}$. The objective function $f$ (or the heuristic) is representative of the system's energy and so, every time we generate a new solution, there will be a change in energy in the system, $\Delta E = f(\mathbf{x}_{t+1}) - f(\mathbf{x}_t)$. This change will always be accepted if $\Delta E \leq 0$. If the new generated solution is not better, i.e. $\Delta E > 0$, then we accept it with a given probability $e^{-\frac{\Delta E}{T}}$. The probability $P(\Delta E)$ of accepting a new solution in the system becomes:

$$P(\Delta E) = \begin{cases} e^{-\frac{\Delta E}{T}} & \text{if } \Delta E > 0 \\ 1 & \text{otherwise} \end{cases} . \tag{2.7}$$

The worse the new solution generated, the less likely it is to be maintained. The lower the temperature, the less probable it is that increases in energy are accepted. By the end, it is very improbable that the solutions get worse between two iterations as the temperature should be almost zero by then. The way temperature decreases with time is called Temperature Schedule and it has high influence in the quality of the solutions and in the run time of the algorithm. If temperature decreases very fast, then the algorithm will be fast but the solution might be pour. On the other hand, if temperature decrease is sufficiently slow then optimality is guaranteed but the algorithm can take longer than a systematic strategy.

There are many Temperature Scheduling policies. The important thing is that temperature should start high and go towards zero. One possible strategy might be to reduce T every m iterations in the following way:

$$T_{t+m} = (1-\epsilon)T_t, \quad 0 < \epsilon < 1 \quad . \tag{2.8}$$

#### 2.2.4.C Tabu Search

Tabu Search, proposed by Glover [8], is an improvement on Hill Climbing and it aims to explore a more diverse range of candidates by marking explored states as tabu to try and escape local minimums. If at any point, the steepest descent (as we are minimizing) leads to a tabu state, we take the second most steepest. This marking of states can be achieved with a simple list, that has to be circular because design spaces can me enormous. One other alternative, if it is possible to measure distances between elements of $\mathbb{X}$, is to mark as tabu every state in a given neighborhood of explored states.

#### 2.2.4.D Genetic Algorithms

Genetic Algorithms (GA's) [9] draw inspiration from nature's evolutionary process in which the most fit survive and reproduce. It is a population-based algorithm, so, at each time step, we consider a plethora of potential solutions. In genetic algorithms, each individual is a chromosome which is defined by a particular sequence of genes. Practically speaking, solutions are a sequence of elements, (usually) of the same datatype and depending on the problem, they may have either a fixed or variable length. For example, for the problem of Transit Network Design, a possible representation might be a string of routes that make up the network in which each route is pre-generated with a heuristic procedure before the actual genetic algorithm takes place.

The algorithm starts by creating a random population. Then, each individual is evaluated according to the fitness function which is basically a reflection of the potential of that individual and thus, it is usually based on the objective function or on heuristics. Higher values of fitness should be returned for better states. Based on the fitness values, individuals are selected for reproduction, this process is random, but the higher the fitness function of an individual, the higher the probability that it is selected for reproduction. Reproduction, in these algorithms, is represented by a crossover operation. Usually, single-point crossover is used but generally, one can use multi-point crossover. In single-point crossover a crossover point is chosen randomly and offspring are created by crossing over the parents at crossover point. The first individual will be his first father till crossover point and will be his second father after crossover point. The second will be the opposite, second father till crossover point and first father thereafter. The idea is easily extendable for multi-point cross-over. Offspring are then mutated with a small independent probability. This is just changing randomly a value in the string. In the Transit Network Design Problem we might just replace a random route with one other route not present in the network at the moment. In problems where the representation has variable size, we might also add probabilities to

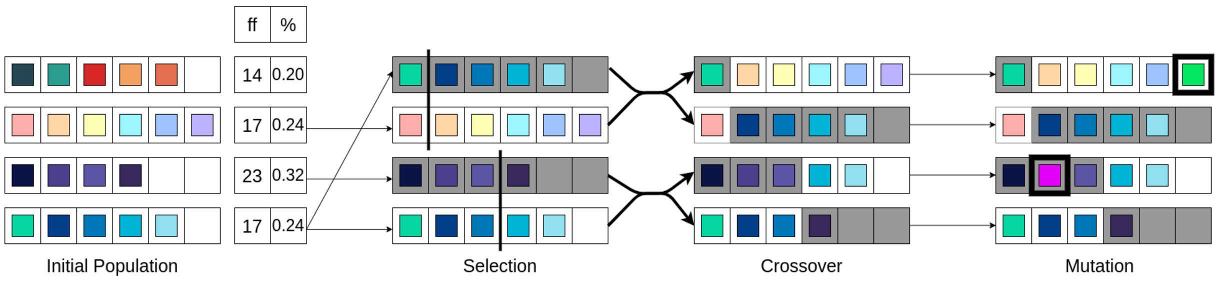


**Figure 2.7:** Iteration of a Genetic Algorithm on the Network Design Problem with a population of 4. Each route is represented by a color. The second crossover can be seen board wise in Figure 2.8.

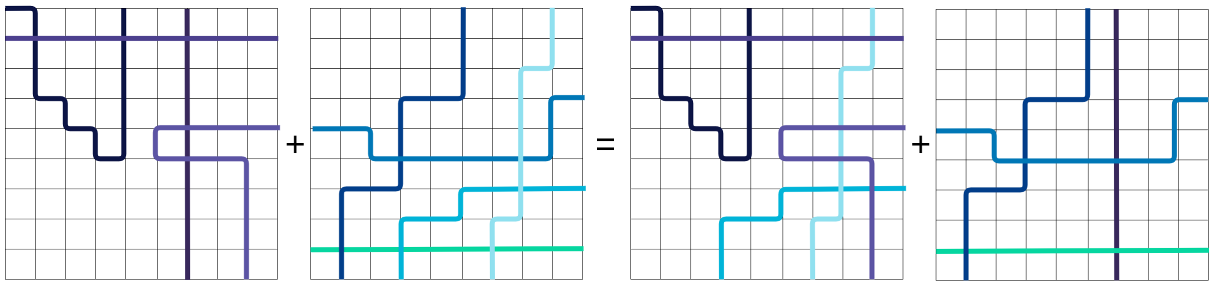


**Figure 2.8:** Single point crossover with bus networks. This is the second crossover operation in Figure 2.7

add and remove genes at random and in random positions. In fact if we want to allow our network to have a number of routes between a given interval, we have to use a variable size string to represent individual networks. After mutation, we now have a new generation to be evaluated and reproduced. A diagram of an iteration for Network Design Problem can be seen in Figure 2.7, the second crossover operation of the iteration is showed in more detail with actual routes laying on top of a lattice road network in Figure 2.8. The algorithm stops, for example, if the best solution in the population has not increased significantly for a given number of iterations.

#### 2.2.4.E Ant Colony Optimization

Ant Colony Optimization (ACO) [10] is inspired by the intelligent behavior that emerges from a group of ants scavenging for food. When ants leave their colony to look for food, they start by exploring their environment in a random manner. While they wander about, they leave a pheromone on the path so they can know how to return once they find food. If they find food, they will come back through the same path, leaving pheromones behind once again. The path ants take on their way to food is highly influenced by pheromones left by other ants. A high concentration of pheromone on a path will very likely lead ants into it. This behavior is very simple, but because pheromones evaporate over time, the colony as a whole will behave in quite an intelligent way. The longer a path to a food source is, the longer an ant will take to traverse it, which means, that there is more time for the pheromone to evaporate while the ant comes and goes. On the other hand, shorter paths can be traversed more times within the same time, which means they will have a high pheromone concentration and will thus lead more ants into them. This will establish a positive feedback loop and lead colonies to take very efficient paths to food sources after they scavenge for a while.

This process can be mimicked by computers in order to find minimal cost paths through graphs. If you can model an optimization problem as finding minimum cost paths in graphs, you can solve it with Ant Colony Optimization. The famous Traveling Salesman Problem (TSP) is the prime example of one such problem, and the cost of paths in this problem is merely their length.

The algorithm works in an iterative manner. In each iteration, each ant must traverse the graph, building its own solution, i.e. finding a path from the colony to the food source. Then the pheromone concentration in each edge on the graph is updated and the process repeats. The algorithm can halt once the best path (highest concentration of pheromones from the colony to food source) hasn't changed in quality significantly for a given number of iterations, similarly to the stopping of Genetic Algorithms.

Previously we referred to network nodes with the letters $i$, $j$, and $k$. For the remainder of this section we will refer to them as $x$, $y$ and $z$ to avoid confusion with ant identifiers.

Each ant builds its solution iteratively, selecting edges one at a time. From node $x$, ant $k$ will move to one of the nodes in $A_k(x)$, the set of nodes the ant can reach from $x$. Making the transition from $x$ to $y$ will be dependent on two factors, the attractiveness of the movement, $\eta_{xy}$, and the concentration of pheromone on the edge $(x, y)$ at time t, $\tau_{xy}(t)$. The attractiveness of the movement is domain dependent and is usually computed with the aid of heuristics. The probability of ant $k$ taking edge $(x, y)$ at time t, $p^k_{xy}(t)$ becomes:

$$p^k_{xy}(t) = \frac{[\tau_{xy}(t)]^\alpha [\eta_{xy}]^\beta}{\sum_{z \in A_k(x)} [\tau_{xz}(t)]^\alpha [\eta_{xz}]^\beta} \quad , \tag{2.9}$$

where $\alpha$ and $\beta$ are two adjustable parameters that control the impact of the attractiveness and the pheromone intensity.

Once every ant builds its solution, pheromone concentration can be updated as follows:

$$\tau_{xy} \leftarrow (1 - \rho)\tau_{xy} + \sum_k \Delta\tau^k_{xy} \quad , \tag{2.10}$$

where $\rho$ is the pheromone evaporation coefficient and $\Delta\tau^k_{xy}$ is the amount of pheromone left behind by ant $k$ and is usually the following:

$$\Delta\tau^k_{xy} = \begin{cases} Q/L_k & \text{if ant } k \text{ takes edge } (x,y) \text{ in its path} \\ 0 & \text{otherwise} \end{cases} \quad , \tag{2.11}$$

where $L_k$ is the cost of the solution found by ant $k$ and $Q$ is a constant.

#### 2.2.4.F Swarm Particle Optimization.

Much like Ant Colony Optimization, Particle Swarm Optimization (PSO) [11] is a nature inspired meta-heuristic. Only this time we are drawing inspiration from the way flocks of birds and schools of fish look for food. Individuals in these swarms move with the group with a certain degree of randomness and when one finds food, it lets the others know, so they can move in the direction of the food and feed themselves.

This algorithm is population-based. The elements of the population are called particles, and each particle $p$ has a position $p.\mathbf{x} \in \mathbb{R}^n$ and a velocity $p.\mathbf{v} \in \mathbb{R}^n$. The position of a particle can encode a solution in solution space much like in the genetic algorithms and the velocity can encode the search direction i.e. which states to explore next. The fact that Swarm Particle Optimization works in $\mathbb{R}^n$ does not mean that $\mathbb{X} \subseteq \mathbb{R}^n$. The quality of a position refers to the quality of the solution it encodes (objective function or heuristics).

The algorithm starts by randomly initializing particles' position and velocity. Then it works iteratively and in each iteration, first the velocities are updated and second the positions are updated. Once updates are not meaningful, the algorithm may halt. For the velocity update, each particle may consider only particles on its neighborhood or all the particles in the population. If the whole population is considered, the algorithm will converge in a single solution, while in the neighborhood case, particles can group around different minimums. Let us define *best* as the best position in the system and $p$.best as the best position the particle $p$ has been on. The updates then are:

$$\begin{gathered} p.\mathbf{v} \leftarrow p.\mathbf{v} + (\text{random}_u(\mathbf{0}, \mathbf{c}) \otimes (p.best - p.\mathbf{x})) + \\ (\text{random}_u(\mathbf{0}, \mathbf{d}) \otimes (\textit{best} - p.\mathbf{x})) \end{gathered} \quad , \tag{2.12}$$

$$p.\mathbf{x} \leftarrow p.\mathbf{x} + p.\mathbf{v} \quad , \tag{2.13}$$

where $\otimes$ is the element wise multiplication and $\mathbf{c}$ and $\mathbf{d}$ are learning rate vectors.

#### 2.2.4.G Non-dominated Sorting Genetic Algorithm II

Non-dominated Sorting Genetic Algorithm II (NSGA-II) is the only multi objective algorithm we will be analyzing in this work and it was proposed by Deb et al. [12]. It is, as the name shows, a genetic algorithm. We still work with populations of candidate solutions, but this time, we do not have the concept of fitness, we have ranks which tell us how many solutions a solution dominates and by how many solutions it is dominated. The solutions with rank $0$ are always the current approximation of the Pareto Front. When selecting individuals for the next generation we are also looking too keep our sample as diverse as possible as we would like to have an approximation of the Pareto Front that is scattered in the objective space (or solution space if we so desire) and not focused around a small amount of points. For this reason, the tie breaker for solutions in the same rank is the crowding distance which is a measure of how far a given solution is from its closest neighbors.

In terms of the main loop, this algorithm is very similar, we start the iteration by reproducing multiple solutions. In this case, as we do not have fitness, two individuals are chosen at random, then the individual with lower rank is kept. If both individuals are ranked equally, the one with smaller crowding distance is discarded. This is done again to form a reproducing pair which then reproduces using the crossover operator we discuss in Section 2.2.4.D. This tournament is done $N$ times, where $N$ is the size of the original population, leaving us with a population of size $2N$. The members of this population are then mutated at random as discussed in Section 2.2.4.D.

The element that changes significantly from the Classic GA to this one, is the selection of individuals for the next population. This requires two steps, nondominated sorting and crowding distance sorting.

In the nondominated sorting, we start by computing, for each solution $\mathbf{x}$, the domination count, $n_{\mathbf{x}}$ which is the number of solutions that dominate $\mathbf{x}$ and the set of solutions that $\mathbf{x}$ dominates, $S_{\mathbf{x}}$. After these quantities are computed for every candidate, the solutions with $n_{\mathbf{x}} = 0$ belong to $F_0$, the first front, meaning that these solutions have rank $0$ and are, thus, our current approximation of the Pareto Front. The next step is to decrement the domination count of every solution $\mathbf{x}' \in S_{\mathbf{x}}$ for every solution $\mathbf{x} \in F_0$. Now every solution with $n_{\mathbf{x}} = 0$ belongs to $F_1$, these are the solutions with rank 1. This process is repeated until every solution is ranked.

The next step is the crowding distance computation. The crowding distance is a density estimation metric which is measured within each front. This means that the crowding distance computation is done for every front separately. For each objective, the front is ordered by the values of those objectives. The solutions with maximal and minimal objective values are assigned a crowding distance of $\infty$. Then a sweep is made, and for each solution $i \in F_j$:

$$distance(i) = distance(i) + \frac{f_l(i+1) - f_l(i-1)}{\max_{k \in F_j}\{f_l(k)\} - \min_{k \in F_j}\{f_l(k)\}} \quad . \tag{2.14}$$

To make this step clearer, Figure 2.9 shows the placement of consecutive solutions in objective space.

Now we can define a partial order $\prec_n$ as:

$$i \prec_n j : i, j \in \mathbb{X} \Rightarrow \big(rank(i) < rank(j)\big) \vee \big(rank(i) = rank(j) \wedge distance(i) > distance(j)\big) \quad . \tag{2.15}$$

Now we order the population according to the partial order $\prec_n$ and the first $N$ elements will make

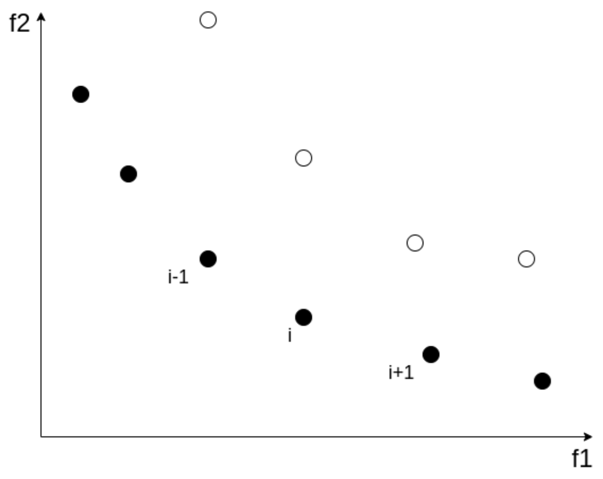


**Figure 2.9:** Crowding distance calculation. Points marked with filled circles are part of the same front, in this case, the Pareto Front.

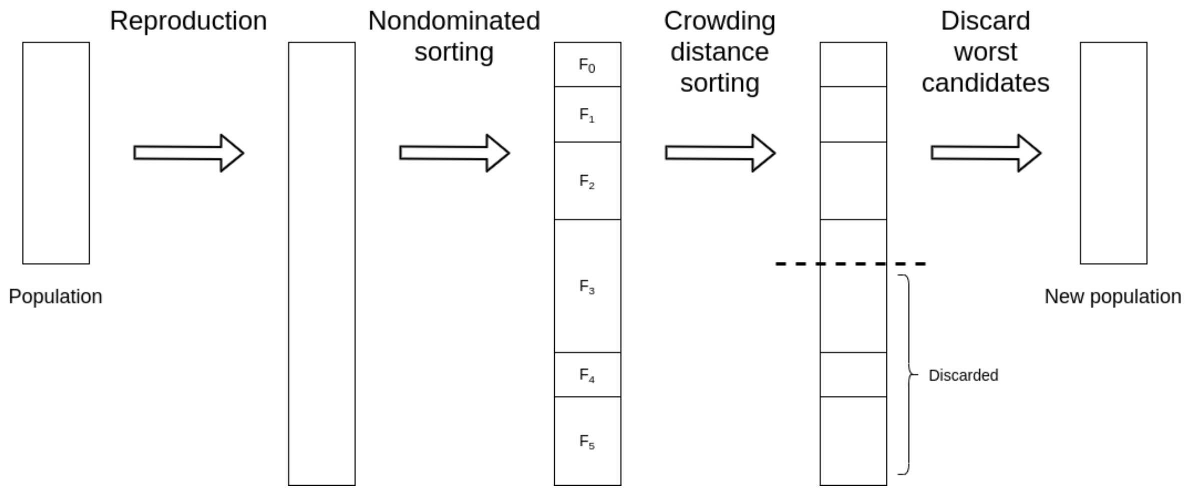


**Figure 2.10:** An iteration of NSGA-II.

up the subsequent generation. This cycle is repeated for a given number of iterations and with each iteration, the first front $F_0$ becomes closer and closer to the real Pareto Frontier. A diagram summing up the functioning of the algorithm can be seen in Figure 2.10.

The way NSGA-II handles constraints is by modifying the definition of domination that was introduced in Section 2.2.1. A solution $\mathbf{x}^1$ is said to constrained-dominate a solution $\mathbf{x}^2$ if any of the following holds: i) solution $\mathbf{x}^1$ is feasible and $\mathbf{x}^2$ is not; ii) both solutions are infeasible but solution $\mathbf{x}^1$ has smaller overall constraint transgression; iii) both solutions are feasible and solution $\mathbf{x}^1$ dominates solution $\mathbf{x}^2$.

# 3 Related Work

Contents

This chapter explores how similar problems to the targeted in our work are addressed in the literature so we can chose the appropriate approach to tackle our situation.

Every strategic, tactical and operational decision related to public transportation networks is part of the Urban Transportation Network Design Problem (UTNDP) [13]. Strategic decisions are usually long-term and related to the infrastructure supporting the urban transport network. Building new streets or establishing a bus network are examples of such decisions. Tactical decisions are related to the effective use of the infrastructure and include, for example, determining the orientation of one-way streets or setting the frequency of buses in a given route. Operational decisions are commonly short-term and are mostly related to scheduling and flow management. The prime example for operational decisions is the scheduling of traffics lights.

UTNDP is usually divided into two categories, the Road Network Design Problem (RNDP) and the (Public) Transit Network Design and Scheduling Problem (TNDSP) [13].

RNDP usually looks at the flow on the road network in a homogeneous way, not distinguishing between public and private vehicles. RNDP can be divided into subproblems depending on the nature of the decision variables at hand. Discrete Network Design Problem (DNDP) deals with discrete decisions, for example, deciding the direction of one way streets or determining which directions it is possible to turn in interceptions. Continuos Network Design Problem (CNDP) takes care of the continuous decisions such as the scheduling of traffic lights or the expansion of capacity of road segments. Mixed Network Design Problem (MNDP) deals with a combination of continuous and discrete design decisions [13].

TNDSP is usually focused on a single mode of transportation and include all strategic, tactical and operational decisions it entails. TNDSP can be further divided depending on the decisions we are looking to do. Transit Network Design Problem (TNDP) is concerned with topology, that is, the routes that make up a transport network. Transit Network Design and Frequencies Setting Problem (TNDFSP) determines the frequency on each route in addition to the topology. Transit Network Frequencies Setting Problem (TNFSP) establishes frequencies for each route and has the network topology as input. Transit Network Timetabling Problem (TNTP) plans time-tables for staff and buses given the topology and the route frequencies. Lastly, Transit Network Scheduling Problem (TNSP) decides route frequencies and time-tables given a network topology [13].

TNDSP studies commonly focus on a single mode of transportation at a time but in reality, all modes of transportation influence each other. One can consider another class of problem, the Multimodal Network Design Problem (MMNDP) tackles at least two modes at a time, even if the design decisions regard only one mode.

Figure 3.1 provides a simple acronym-meaning correspondence and a scheme of the taxonomy described above.

UTNDP is usually formulated as a bi-level or follower-leader problem [13]. These problems are

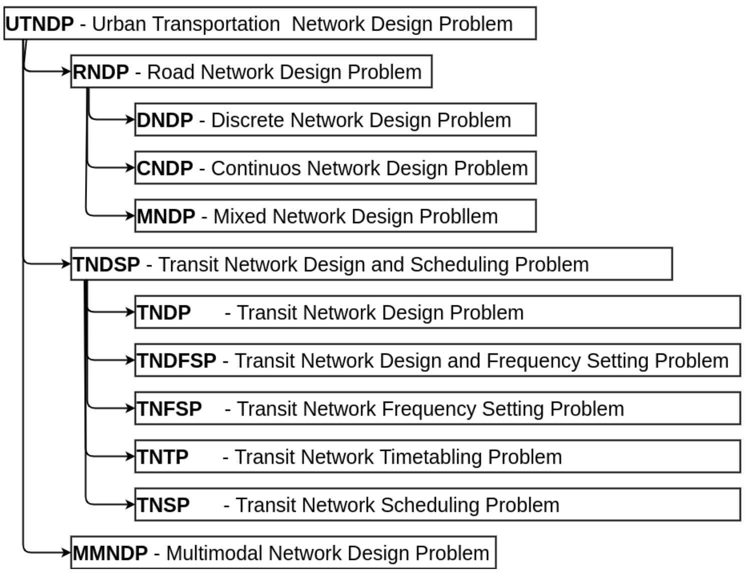

UTNDP - Urban Transportation Network Design Problem
RNDP - Road Network Design Problem
DNDP - Discrete Network Design Problem
CNDP - Continuos Network Design Problem
MNDP - Mixed Network Design Probllem
TNDSP - Transit Network Design and Scheduling Problem
TNDP - Transit Network Design Problem
TNDFSP - Transit Network Design and Frequency Setting Problem
TNFSP - Transit Network Frequency Setting Problem
TNTP - Transit Network Timetabling Problem
TNSP - Transit Network Scheduling Problem
MMNDP - Multimodal Network Design Problem

**Figure 3.1:** Urban Transit Network Design Problem taxonomy.

optimization problems, but there are two levels, ultimately, we want to solve the upper level problem which is, in itself, an optimization problem that is restricted by the solution to the lower level problem, which is also an optimization problem. The upper level problem aims to provide the best design decisions possible, but it so happens that judging this quality implies knowing how the users respond to the design decision made. Users use the network in a way that is optimal to them, so a second level must be solved. Essentially we want to judge our design by the way the users will be using it.

In most works, a single-level, i.e. normal optimization problem, as described in Section 2.2, is used [13]. This is possible by simplifying user behavior, making it possible to assess user response without having to solve an optimization problem. All-or-nothing (AON) is an approach in which all demand is assigned to the shortest route. Now, this is more complicated for passenger assignment in bus or train networks, because users tend to prioritize transfer minimization, i.e. not having to switch between buses or trains, even if that leads to a longer trip. Furthermore, if two routes can satisfy users' needs without the need for transfers, people tend to catch the first bus or train that comes around even if that one takes longer. This is not as simple as predicting traffic of private vehicles in an AON policy, but is still simpler than solving an optimization problem.

In Section 2.2 we saw that optimization problems are solved using exact or mathematical methods, heuristics and metaheuristics. These are exactly the tools used in the literature to solve UTNDP [13].

Our goal in this work is to improve CARRIS network, however, we want to explore synergies with metro. This means the problem falls in the MMNDP, however, most works addressing MMNDP try to build more than one network in parallel. For example, Beltran et al. [14] propose a method to build a bus network that is composed of two sub-networks, a green sub-network, composed by green vehicles and a non-green sub-network composed of traditional vehicles. They use Genetic Algorithms and minimize the user and operating costs on both networks while reducing the overall environmental impact of the whole system. In our case we want to find the optimal design for only one network while considering the presence and passenger flow in the others. By modeling passenger behavior in a simplified way, we can assess how effective a given bus network design might be in the context of the whole Lisbon Transportation Network and thus, we can predict the flows in a given design. For that reason, most of the works reviewed here fall in the TNDSP.

## 3.1 Complexity of TNDP

Newell [5] points out the non-convexity of TNDP which is illustrated by the fact that increasing the frequency of buses in some routes, i.e. increasing operation costs may lead to an increase in user costs. Newell points out that the sources of this non-convexity are the waiting times, either when entering the network, or transferring between routes, not the links that compose the network. Baaj and Mahamassi [15] also mention the combinatorial explosion of the problem caused by its discrete nature. Additionally, the problem is usually formulated with multiple objectives and, while bus frequencies are usually depicted as decision variables in problem formulations, the network itself, in terms of routes and the stops in each route, is not depicted in problem formulations at all. This makes Mixed Integer formulations like the one proposed by Wan and Hong [16] rather cumbersome and ineffective to solve TNDP. In Wan and Hong's formulation, the presence of each node in each route is modeled as a binary variable. This makes it so that a small network with 10 nodes and 19 links, ends up having 363 binary decision variables, 30 integer decision variables and 303 continuous decision variables, when we try to solve TNDP for it. Additionally, the formulation requires the user to specify the number of routes and only takes into consideration the operation costs. The problem was then solved by commercial Mixed-Integer Programming packages. Since we will be solving TNDFSP for a network much larger than the one in Wan and Hong, we will ignore exact methods from here on out and it is also the reason why we did not discuss any exact method in detail in Section 2.2.4.

## 3.2 TNDP Solutions

Jia et al. [17] tackles TNDP for the bus network in Xi'an, a city in Northwest China. Unlike the majority of solutions, this study focus on optimizing topology from the point view of Complex Network Theory. Their solution tries to minimize the product between the betweenness and resistance of arcs between specific OD pairs. The idea is that arc betweenness in the bus network will cause many routes to pass through the same road interceptions, causing congestion, and resistance will further slow down the road network usage. Furthermore, if a road segment with high betweenness is compromised (accidents, or congestion) the functioning of the network as a whole will suffer heavy penalties as many routes will be stalled. They place constraints for route length, station capacity and even transfers, but these are not the focus of optimization. This study tries to improve network sustainability and robustness, no efforts are put forth to make optimizations in the perspective of the users nor in the perspective of the service provider. They do not use any conventional optimization algorithm, instead they develop their own method which takes as input the current network topology. The method did improve Xi'an's network sustainability and cut down its average path length which is also a good indicator of network quality as it reflects how efficient routes and transfers between routes are. They also make the point that high degree stops with low clustering coefficients should be avoided (potentially decomposed into interconnected nodes) as they constitute single points of failure. The main takeaway is that we can use some concepts of Network Theory to evaluate how robust and sustainable our network is without looking at OD demand.

Yu et al. [18] used ACO to solve TNDP. While most works start from the road network and build the bus network from scratch, this work focus on revising an existing network. Their goal was to maximize passenger flow per unit length and minimize transfers. Their rational behind this objective was that, by minimizing transfers and maximizing passenger flow, you are making great use of your resources while providing a good service, thus user costs and operator costs are being minimized as a consequence. In their model, OD pairs work as the nest and the food source for several sub-colonies and, in that way, by modeling the pheromone as passenger density, solving TNDP becomes the ant food scavenging problem which is solvable by ACO. Here while $\tau_{xy}$, the pheromone concentration, is updated in each cycle as normal, $\eta_{xy}$, the attractiveness of the edge is the passenger flow density on link $(x, y)$. At the end of each cycle, the line with the highest passenger flow density is added as an alternative line and no more than one alternative is generated for each OD pair. Note that ACO can be parallelized since ants act independently, and this paper uses a variant of ACO that exploits that. They tested their work on an existing network in the city of Dalian (a chinese city) with 3,200 links and 2,300 nodes with 89 lines. The algorithm was able to reduce the number of lines to 61 and did so without compromising satisfied demand. Additionally, the optimized bus lines are shorter than the existing ones and the amount of demand satisfied with no transfers went from 41% to 51%.

## 3.3 TNDFSP Solutions

Pattnaik et al. [19] used GA to solve TNDFSP. The work focus on minimizing traveling times for users' benefit and minimizing operating time of buses for the benefit of the operator, given some feasibility constraints regarding bus frequencies and capacity. Their approach worked in two phases, first a Candidate Route-set Generation Algorithm (CRGA) generates a set of routes that will compete for the solution route set. Then a GA computes the optimal set of routes and their frequencies. CRGA essentially computes the shortest paths between every OD pair and then generates alternative routes by temporarily removing links from the road network and computing the shortest paths in that incomplete network, improving route diversity. If a generated route does not meet the constraints for route length, it is discarded. In this work, path cost is measured in travel time, and transfers between routes apply a heavy time penalty. The GA model adopted in this work represents a bus network as a string of routes in which each route is represented by a binary string of fixed length. The binary string for each route is just an identifier. The GA is used in two different ways in this work. In Fixed String Length Configuration (FSLC), every network in the population has the same number of routes. In Variable String Length Configuration (VSLC), networks have a variable number of routes. VSLC can benefit from insertion and deletion operators, which respectively add and remove routes to a network at random. As with normal mutations, insertion and removal occur with a small probability. FSLC can be used if we know roughly how many routes the network should have, because this way we can run the algorithm a few times for different numbers of routes and choose the best one. If we do not know this value, VSLC has to be used. Computing the fitness function for each network involves frequency setting for every route and the computation of the objective function which implies assigning demand to the routes in the network. In this work the method developed by Baaj and Mahamassi [20] is used. The methods are tested in a subset of a network in Madras, South India, but there is no comparison to the network operating there. The objective was to compare FSLC and VSLC and assess the influence of the insertion and deletion operators. The work estimates the best parameters for the GA (population size, crossover probability, mutation probability, insertion probability and deletion probability) using sensitivity analysis and then runs both FSLC and VSLC to compare results. FSLC produced significantly best results leaving no demand unmet, but it needed more computing time as the GA has to run for different network sizes. The insertion and deletion operators proved to improve the results of FSLC.

Chien et al. [21] also used GA in the context of TNDFSP but in a different way. Their goal was to determine optimal feeder bus routes, not an optimal network. Feeder routes bring people from (to) several points to (from) a central hub such as a major intermodal transfer station or a central business district. In their work they consider only dummy networks modeled as grids and buses only travel from left to right (or vice versa) in the horizontal direction and can go up and down in the vertical direction. Their objective function also minimizes user and operator costs. The supplier cost is a function of the round trip time of the buses and the headway. The user cost involves user access cost (how far do users have to walk to be able to catch a bus), user wait cost and user in-vehicle cost. They show that their cost function is convex and that makes it possible to compute the optimal bus headway given the topology of the route. In the GA, each individual is a route and the fittest routes are the ones with lower cost functions. Crossover must be done between routes that share a link, that way we can combine two routes and produce two new ones that have the links after crossover point swapped. They tested their approach in small networks and found that, when the parameters are properly tuned, the GA can find the optimal route. The optimal route was confirmed to be the global optimum by an exhaustive search algorithm.

Bielli et al. [22] used GA to solve TNDFSP. They use a variant of the classic GA called Cumulative Genetic Algorithm (CGA) in which members with high $ff$ values are saved and used with new populations. The idea is to maintain good solutions actively reproducing instead of potentially ruining them in a crossover operation. Their GA modeling is quite different from that of [19]. Bielli et al. use a fixed length representation in which each chromosome is a network and each gene represents a pre-generated line with a frequency and an on/off switch that indicates whether the route is active in the network or not. This means that every pre-generated route will exist in every network but with different frequencies and it can be active or not. This also makes reproduction very straight forward. There is no special considerations here, a simple crossover operation, as defined in Section 2.2.4.D, is used. Mutation is also simple, an individual subject to a mutation will have a route with a different frequency that can go from being inactive to being active or vise-versa. The $ff$ values are computed in each iteration after the OD demand is spread through the network. Their $ff$ is a weighted sum of efficacy, efficiency and quality metrics. Efficacy metrics measure the extent to which the network is able to meet demand, for example number of pedestrians is the number of users that do not use the bus network and the lower this value, the better. Efficiency metrics measure the amount of resources needed for the network to work, for example, number of buses, the fewer buses needed, the better. Finally, quality measures offer the possibility to assess the quality of service and environmental consequences, for example, average number of transfers or pollution indexes. The work was tested on a city in middle-north Italy called Parma and the best result provides an improvement of $90\%$. This result was obtained after 66 iterations with a mutation probability of $0.1$ and a crossover probability of $0.8$ (no population size specified). The routes used are taken from previous projects on the matter and make a total of $80$ candidate routes. One is the actual Parma bus network, then a network was greedily generated to satisfy OD demand with short routes and the last two networks are based on geometrical routes, the first one consisting of circular routes and the second one of semicircular routes that connect two suburban nodes through the city.

Fan and Machemehl [23] developed the last GA approach to TNDFSP reviewed here. Their objective function considers, user costs, operator costs and unsatisfied total demand costs. The relative importance of the costs can be tweaked. The constraints considered focus on controlling the lower and upper limit of bus headway, route length, trip length and number of routes and making sure the service can be provided by the available fleet and controlling the load factor on any given route. Similarly to Pattnaik et al., their approach starts with a CRGA which uses Dijkstra's Algorithm and Yen's k-shortest paths algorithm to generate routes between OD pairs. Dijkstra's algorithm finds the shortest path between two nodes in a graph and Yen's algorithm uses an algorithm to find shortest paths between nodes as a base to compute the k-shortest paths. Then, a GA is used, modeling the problem in similar ways to Pattnaik et al. and Chien et al.. The Network Analysis Procedure (NAP) assigns demand to routes and sets the frequency for each route, allowing for the computation of the $ff$. The algorithm was tested on an example network and sensitivity analysis was used to find the best parameters for the GA. No comparison with existing networks was done. They did compare, however, different methods on the same network and found that the GA outperforms other population based methods and can produce solutions of the same quality as Simulated Annealing and Tabu Search. They found that after 80 iterations the algorithm produces no further improvements so that can be used as a stopping criteria. They found the optimal crossover probability to be $0.8$ and the optimal mutation probability to be $0.1$, just like Bielli et al. The optimal population size was 60.

## 3.4 Other Optimization Techniques

Botee and Bonabeau [24] propose solving the TSP (traveling salesman problem) using ACO in which the parameters are determined by a genetic algorithm. This is an interesting proposition and one that can be taken into account even if their work is not focused on the TNDSP. The parameters in the metaheuristic algorithms are very important for the algorithm to find good solutions, they often balance exploration (which is what allows the algorithms to escape local minima) and exploitation (which is what allows the algorithms to reach the minima) and finding the optimal parameters is, in itself, an optimization algorithm. They were able to improve the best known solution for a benchmark TSP problem with 30 cities.

Along this section, trajectory-based algorithms were not included on purpose. While trajectory-based algorithms might be a good option to make local adjustments to an already existing network, if our goal is to design a network from scratch, their computational complexity to cover a given search sub-space is considerably higher than that of a population-based strategy. By maintaining multiple solutions and exchanging information between them, population based methods provide a much wider coverage of the search space in small amount of time. Zhao and Gan [25] did use Hill Climbing and a Hybrid Algorithm combining Tabu Search, Simulated Annealing and Hill Climbing to make local adjustments to a bus network in Florida and were able to reduce transfers and improve network coverage.

To the best of our knowledge, PSO was not applied to TNDP or TNDFSP studies yet, perhaps because of the difficulty to model the velocity vectors in discrete problems. The TNFSP can be easily solved with PSO due to its continuous nature. Nevertheless, it is worth pointing out that Miandoabchi et al. [26] used a Hybrid Algorithm combining PSO and Simulated Annealing among other Hybrid Algorithms to solve MMNDP. Their goal was to optimize the coexistence of buses and private vehicles by constructing new streets, adding lanes to existing streets, determining the direction of one-way streets and determining bus exclusive lanes. The Hybrid Algorithm with PSO outperformed the others.

# Part II

# Solution

In the second part of the present work, we introduce and discuss each component and phase of the proposed methodology for optimizing route planning. The overview of the developed solution can be seen in Figure 3.2.

The central part of the work is optimization of the CARRIS bus network. Optimization is conducted in two separate steps. In the first one, we are looking to optimize the route configuration of the bus network without concerning ourselves with frequency related metrics like waiting times and fleet constraints. In the second one, we aim at optimizing the frequencies on each route, first, deciding a single frequency for the whole day and then introducing context and optimizing the frequency of each route for different parts of the day. The inclusion of context information is still unexplored in this line of research, to the best of our knowledge.

We could not find any study using multi objective optimization. When faced with more than one objective, other works in the literature resort to the weighted sum approach (introduced in Section 2.2). This heavily limits the variety of solutions that can be found. In contrast, we want to use multi objective optimization to assess the eligible networks and then use single objective optimization to find the best network possible that represents the balance we are looking for.

The OD matrix estimation was made using the work of Cerqueira et al. in [2]. This is a fundamental input to the optimization process as it is exactly around the clients' needs that we want to design around.

The majority of studies in this line of research consider the bus network isolated. In this work, we want to account for the fact that the CARRIS network can be used in conjunction with the METRO network. Interfacing both networks through a walking network is also a concern.

The bus network is constructed on top of the road network. For us to be able to compute bus paths on the road given a stop sequence, we have to have bus stops as part of the road network. This is a challenge because bus stop locations and the road network configuration are obtained with GPS devices that are prone to errors so we have to project stops onto the network and assess the best projection for each stop.

There are groups of stops that exist in the close vicinity of each other to divide the passengers waiting for the buses between different spaces. From the optimization point of view, these stops are redundant and can be clustered, this will also be a discussion point in this part of the work.

Route generation is also a central part to the proper functioning of the first optimization phase as routes are the genes in this problem that get swapped around during the application of genetic operators. The objective and constraints and how they are computed is also a very important part of this work.

In the next chapters we discuss in detail the previously identified problems. In Chapter 4 we explore the steps that need to be executed prior to optimization. In Chapter 5 we go over the methods and algorithms we used to assess the quality of networks. In Chapter 6 we discuss how we modeled the TNDP and TNFSP as Genetic Problems.

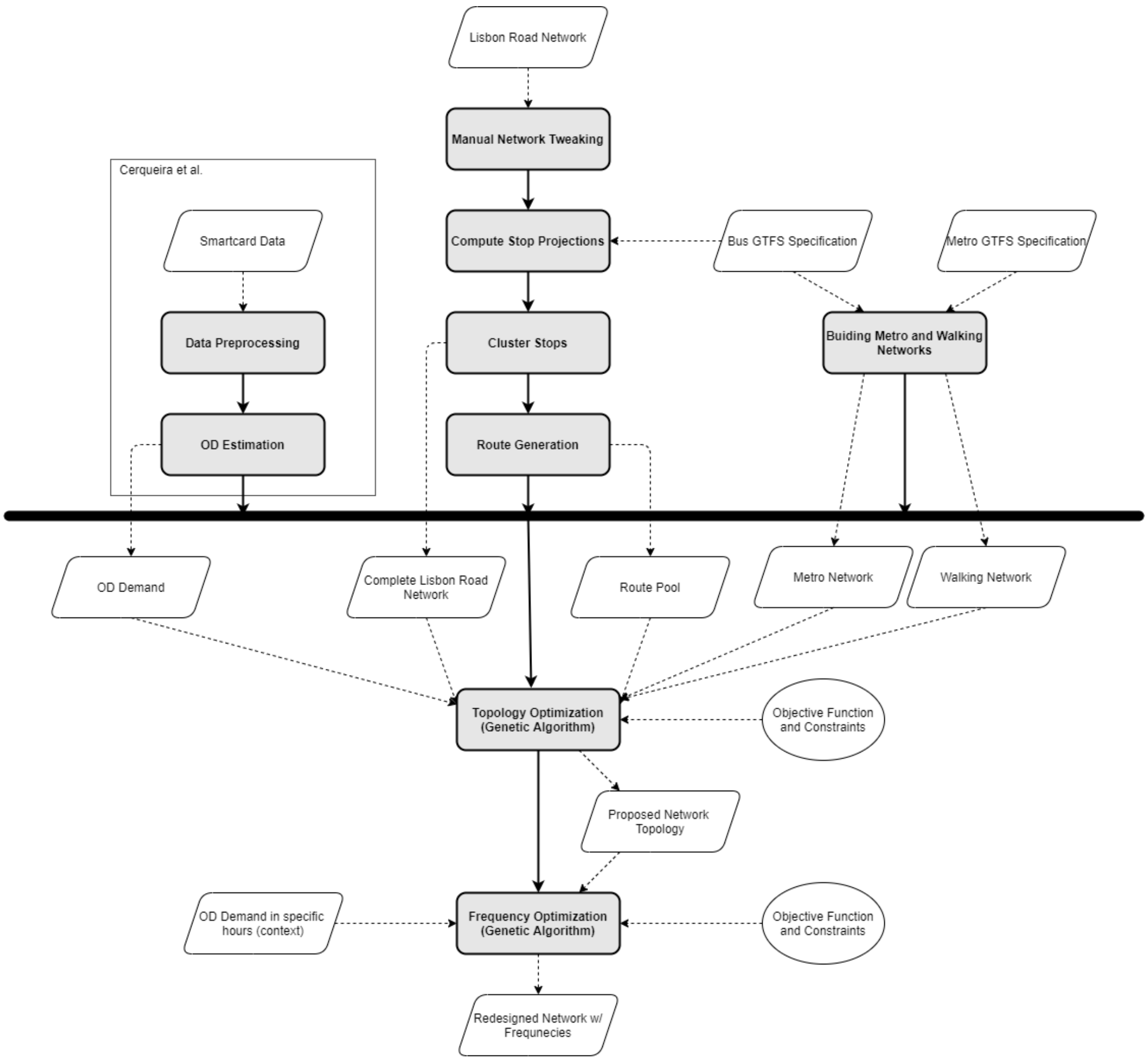

Lisbon Road Network
Manual Network Tweaking
Cerqueira et al.
Smartcard Data
Compute Stop Projections
Bus GTFS Specification
Metro GTFS Specification
Data Preprocessing
Cluster Stops
Buiding Metro and Walking Networks
OD Estimation
Route Generation
OD Demand
Complete Lisbon Road Network
Route Pool
Metro Network
Walking Network
Topology Optimization (Genetic Algorithm)
Objective Function and Constraints
Proposed Network Topology
OD Demand in specific hours (context)
Frequency Optimization (Genetic Algorithm)
Objective Function and Constraints
Redesigned Network w/ Frequnecies

**Figure 3.2:** Proposed route optimization methodology.

# 4

# Preprocessing

**Contents**

In this chapter we address the problem of preparing the targeted networks, including the road network and bus network, so that they can be used in an optimization process in the most efficient way possible. In Section 4.1, we discuss how to make bus stops a part of the road network which is necessary so we can compute paths between the stops on a route. In Section 4.2, we go over some manual adjustments made to the road network due to its incompleteness. In Section 4.3, we tackle stop clustering which allows us to reduce the number of stops in the network without compromising the accuracy of our results. Lastly, in Section 4.4, we present all the networks involved in the optimization processes and how they interact so we can evaluate the bus networks.

## 4.1 Fixing Stops on the Road

General Transit Feed Specification (GTFS) is a format for service providers to share their public transportation schedules and associated geographic information. The GTFS format includes only information that is supposed to be available for the users as it is usually used within systems that facilitate travel planning. A GTFS feed is a collection of Comma Separated Values (CSV) files with the *.txt* extension contained within a unique *.zip* file. There are 6 mandatory files and 7 optional files. The mandatory files are *agency.txt* which provides information about the service provider and the timezone under which it operates, *routes.txt* which provides a listing of the operating routes, *trips.txt* which associates individual trips to the routes, *stop_times.txt* provide information about which trips go through which stops, *stops.txt* which is a list of the existing stops and their location and *calendar.txt* which indicates in which days of the week a given trip will be served. The optional files may provide information about travel costs, sporadic trips and the specific routing of each trip, which may be different between trips serving the same route.

Open Street Map (OSM) is a project to create an editable map of the world that is solely based on crowdsourced data.

While the maps in the OSM database are complete, their accuracy cannot be guaranteed. Furthermore, the location of the GTFS stops are assumed to have been obtained with a GPS device, which is usually accurate within 5 meters. However, the presence of buildings, trees or even the weather can severely hurt the accuracy of the measurements.

In our work, bus stops are part of the road network. This way, we can define a route as a path on the network that starts and ends in a bus stop and may have an arbitrary number of intermediate stops. Because we cannot guarantee the accuracy in neither the road segments nor the stop locations, it may not be clear to which road segment a GTFS stop should be assigned to. For example, in Figure 4.8b, there are some stops located between two road segments which makes it difficult to decide in which of the segments a stop should projected.

To that end we use the work of Vuurstaek et al. [1] to choose the optimal projection of each GTFS stop on the road network. Their work is summed up in Algorithm 4.1 The road network itself was retrieved from the OSM database using a python package called OSMnx which already provides the network as a NetworkX directional multi-graph. NetworkX is another python package used in this work that is focused on representation of networks and implementation of graph algorithms.

**Algorithm 4.1:** Optimal Projection Assignment

```
 1: $S_P$, $S_G$ ← project_stops(stops.txt)
 2: $G_G$ ← build_stop_connection_graph($S_G$, stop_times.txt)
 3: $G_P$ ← build_projections_graph($G_G$, $S_P$)
 4: repeat
 5:     removed_one ← false
 6:     handle_triples($G_G$, $G_P$)
 7:     handle_maximal_sequences($G_G$, $G_P$)
 8:     handle_stars($G_G$, $G_P$)
 9:     reduce_cycle_breakers($G_G$, $G_P$)
10: until not removed_one
11: components ← decompose($G_G$, $G_P$)
12: for component ∈ components
13:     assign_by_enumeration(component)
```

According to Vuurstaek et al. [1], the first step is to find a set of candidate projections for each GTFS stop present in the GTFS feed. Let $S_G$ denote the set of stops in the GTFS feed and $S_P$ the set of projected stops. Each GTFS stop $g \in S_G$ is projected to all the links for which the distance between the GTFS coordinates and its projection does not exceed a given threshold. A projection $p \in S_P(g)$ is considered to be the point in a given link, that is closer to the stop point in the GTFS specification. The identifier and location of each stop can be found in the *stops.txt* mandatory file of the GTFS feed. To obtain the shape of a given link, one can resort to the NetworkX graph obtained with the OSMnx package which has an optional *geometry* parameter for each link that has a complex shape.

The second step is to build the stop connection graph $G_G = (S_G, E_G)$ and the projections graph $G_P = (S_P, E_P)$, both directed graphs. $G_G$ is built based on the GTFS specification of the existing routes. An example can be found in Figure 4.1. If a station $v$ follows a station $u$ in a route, then the edge $(u, v)$ should be in $E_G$. $G_G$ is built in an iterative way such that, if at any point, the addition of an edge induces cycles in the graph, the set of nodes that, if removed, would break those cycles is determined and the one with the least amount of candidate projections is marked as a cycle breaker. An example of such a case is shown in Figure 4.3. $G_P$ is then built based on the stop connection graph and the the set of projected stops $S_P$. For each projection $u_i \in S_P(u)$ and projection $v_j \in S_P(v)$ for which $(u, v) \in E_G$, the edge $(u_i, v_j)$ should be added to $E_P$ with a cost equal to the cost of the shortest path between projection $u_i$ and projection $v_j$ on the road network. An example of a projections graph can be seen in Figure 4.2. From the GTFS specification, one can build the stop connection graph resorting to the *stop_times.txt* file that, for every trip, has the sequence of stops covered.

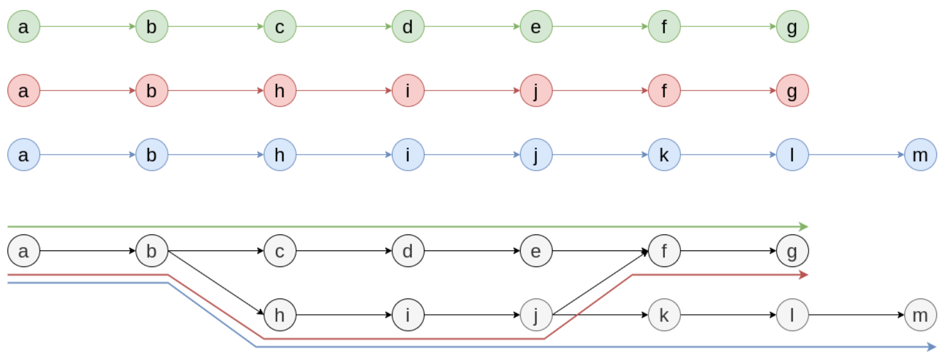


**Figure 4.1:** Example of a stop connection graph. Built from 3 routes, green, red and blue. (Adapted from Vuurstaek et al. [1])

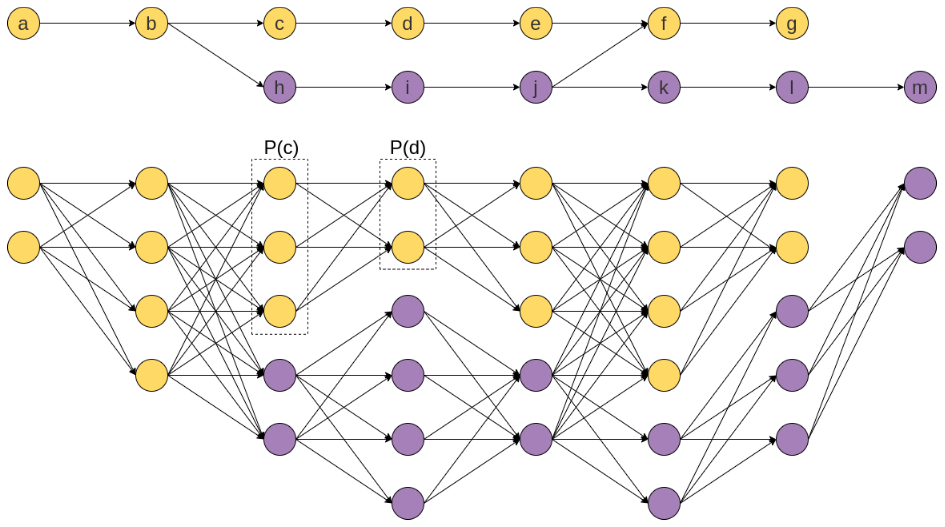


**Figure 4.2:** Example of a projections graph. Built from the stop connection graph presented in Figure 4.1. (Adapted from Vuurstaek et al. [1])

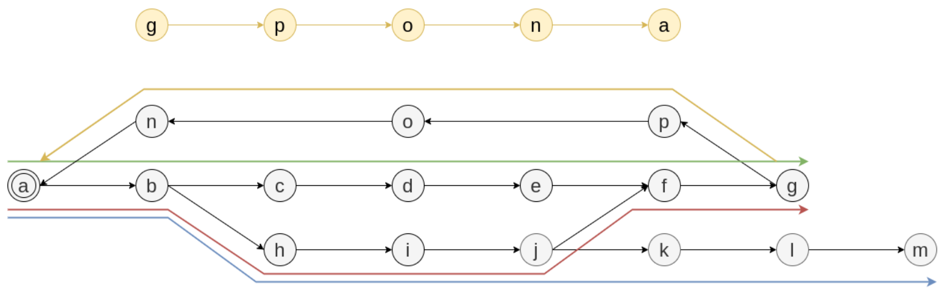


**Figure 4.3:** Example of a cycle breaker. If we were to introduce the yellow route in the stop connection graph presented in Figure 4.1 we would introduce 2 cycles. The nodes that could break those cycles are $a$, $b$, $f$, $g$, $n$, $o$ and $p$. In this case $a$ would be chosen as a cycle breaker because it is the stop with the smallest amount of projections. (Adapted from Vuurstaek et al. [1])

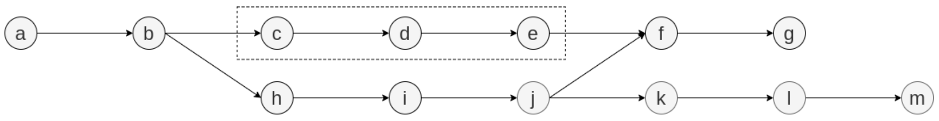


**Figure 4.4:** Example of a triple. The triples present in this graph are $\langle b, c, d\rangle$, $\langle c, d, e\rangle$, $\langle d, e, f\rangle$, $\langle b, h, i\rangle$, $\langle h, i, j\rangle$, $\langle j, k, l\rangle$ and $\langle k, l, m\rangle$. (Adapted from Vuurstaek et al. [1])

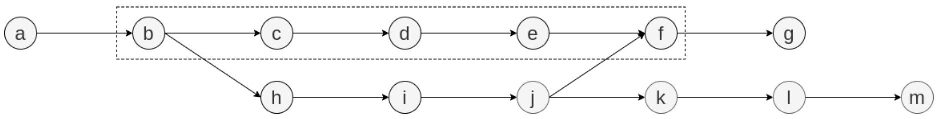


**Figure 4.5:** Example of a maximal non bifurcating sequence. The sequences in this graph are $\langle a, b\rangle$, $\langle b, c, d, e, f\rangle$, $\langle f, g\rangle$, $\langle b, h, i, j\rangle$ and $\langle j, k, l, m\rangle$. (Adapted from Vuurstaek et al. [1])

After having $G_G$ and $G_P$ complete, the algorithm enters a cycle that fixes some candidate projections and discards others until a cycle is no longer able to rule out candidate projections. The cycle starts by looking at triples. A triple is a sequence of 3 nodes $\langle g_0, g_1, g_2\rangle$ in which $deg_{in}(g_1) = 1$ and $deg_{out}(g_1) = 1$. An example can be seen in Figure 4.4. The idea is to compute the shortest path between every projection of $g_0$, $p_{0,i} \in S_P(g_0)$ and every projection of $g_2$, $p_{2,j} \in S_P(g_2)$. Then if a projection of $g_1$, $p_{1,k} \in S_P(g_1)$ appears in every path, we can fix that projection. On the other hand, if $p_{1,k}$ appears in none of the paths, we can discard that projection as a candidate. After processing the triples, the algorithm processes maximal non bifurcating sequences, meaning chains of nodes in which every middle node $g_i$ has $deg_{in}(g_i) = 1$ and $deg_{out}(g_i) = 1$. These sequences are delimited by sources, sinks, cycle breakers, assigned vertices or vertices that have more than one outgoing or incoming edge. The way to fix or remove candidates is exactly the same, in fact, triples are just a special case of maximal non bifurcating sequences that make the whole process more efficient. The next phase involves processing stars. Stars are subgraphs that have a center node with one incoming edge and more than one outgoing edge or one outgoing edge and several incoming edges and all the nodes connected directly to this center. The way to fix or abandon candidates is similar once more, we compute the shortest paths between all the projections of the nodes with edges directed to the star center and all the projections of the nodes with edges coming from the star center and consider the presence of the projections of the central node on those paths. Examples of stars and maximal no bifurcating sequences can be found in Figures 4.6 and 4.5 respectively. Because fixed projections work as boundary nodes for maximal non bifurcating sequences, at the end of each cycle we remove cycle breakers that become redundant with the fixation of the projections in the current cycle.

Once the main cycle of the algorithm is no longer able to rule out candidates, we have to assign projections by solution enumeration. For that process to be efficient, $G_G$ is divided into components delimited by nodes that are sources, sinks, have assigned projections or are cycle breakers. An example of such components can be found in Figure 4.7.

In Vuurstaek et al. [1] there are three steps of the algorithm in which the way they were accomplished

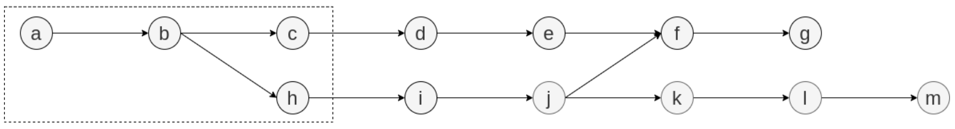


**Figure 4.6:** Example of a star. The stars in this graph are $\langle a, b, c, h\rangle$, $\langle e, j, f, g\rangle$ and $\langle i, j, f, k\rangle$. (Adapted from Vuurstaek et al. [1])

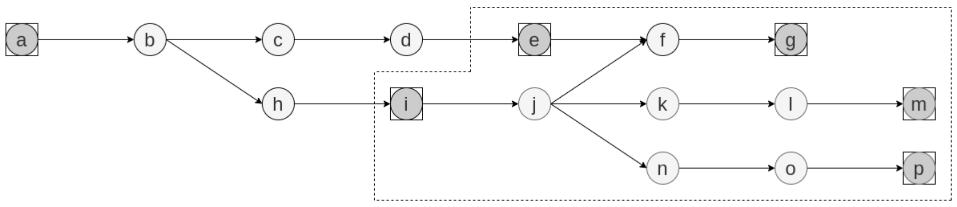
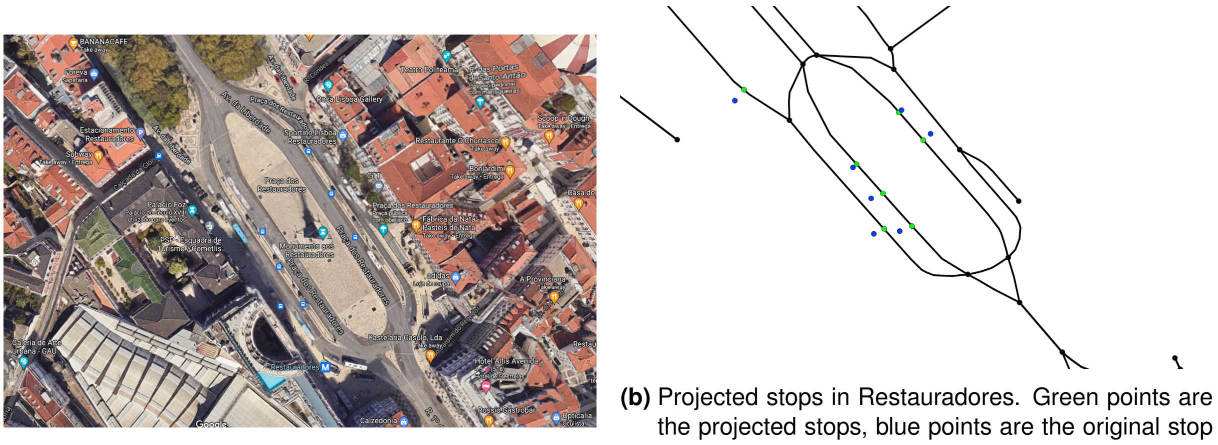


**Figure 4.7:** Example of a component. Boundary nodes are in gray with a square around the vertex. The components in this graph are $\langle a, b, c, d, e, h, i\rangle$ and $\langle e, f, g, i, j, k, l, m, n, o, p\rangle$. (Adapted from Vuurstaek et al. [1])

is not clear. The first two are very closely related and are the marking of nodes as cycle breakers and the unmarking after some projections have been fixed. The way we accomplished it in this work was by maintaining an incomplete version of $G_G$, let us call it $G'_G$ in which cycle breakers and fixed nodes do not exist. If, at any point, we introduce the edge $(u, v)$ in $G_G$, we introduce it in $G'_G$ as well, and then verify the existence of paths from $v$ to $u$. If paths exist, the set of common nodes between the paths is determined and the node $g$ for which $|S_P(g)|$ is the lowest is marked as a cycle breaker and removed from $G'_G$. After each cycle, every fixed node is removed from $G'_G$ and, for each cycle breaker, we assess if its inclusion in the graph would lead to the existence of cycles, if it does not, it is introduced in $G'_G$ and unmarked. The other thing that is not clear, is the decomposition of $G_G$. The way we did it in this work was by performing a Breadth First Search (BFS) with two queues. If at any point a node was to be added to the normal BFS queue and it is a boundary node, meaning a source, a sink, a cycle breaker or an assigned node, it is instead introduced in the second queue. Every node the algorithm looks at is added to a component. Once the first queue is empty, we begin another BFS with the second queue, in which we start adding nodes to a different component and boundary nodes are added to the first queue. This continues until every node is explored. In the end, every component might have several Weakly Connected Components, (wCC) so, to reduce the computational effort of assigning projections by solution enumeration, the components are divided according to their wCC.

As an example, in Restauradores, downtown Lisbon, there are a few stops whose correct projection on the road may not be obvious as their location is in between two road segments. Our assignment and a comparison to a ground truth can be seen in Figure 4.8.


**(a)** Ground truth for stop locations in Restauradores.

**(b)** Projected stops in Restauradores. Green points are the projected stops, blue points are the original stop points and black points are road junctions.

**Figure 4.8:** A comparison of the real location of stops in Restauradores and the obtained projections. Portugal is a right-hand traffic country, proving the correctness of the assignments in this case.

## 4.2 Network Preparation

The road network we obtained, using the python package OSMnx, was incomplete which did not allow us to compute all the paths buses take in the current CARRIS network. The bigger problem was that the South-North way was missing in the $25^{th}$ of April bridge that connects Almada to Lisbon, as well as some connections that were essential for buses to be able to make that trip. This can be seen on a map in Figure 4.9. This would make it impossible for any generated route to go from Almada to Lisbon (even though the mirror was possible) and would thus influence our results. It is also worth mentioning that the A2 exit to Eng. Duarte Pacheco Avenue was missing and was also introduced manually. This can be seen on a map in Figure 4.10.

These segments were the most problematic as they are central to the navigation between Lisbon and Almada. Other segments may not be included in the network which would affect our final results.

One alternative to having a graph representation of the road in memory would be to use the Open Source Routing Machine (OSRM) which also works with Open Street Map and the segments that were missing are in the Open Street Map database. OSRM also accounts for the kinds of vehicle for which it is routing, traffic lights and the geometry of the paths. We did not use OSRM because, in early development phases, we were thinking that path finding in the road network would be needed during optimization and, therefore, it wouldn't be computationally efficient to send HTTP requests to the Routing Machine. Additionally, we need a graph representation of the road network so we can project stops onto the road network, as discussed in Section 4.1.

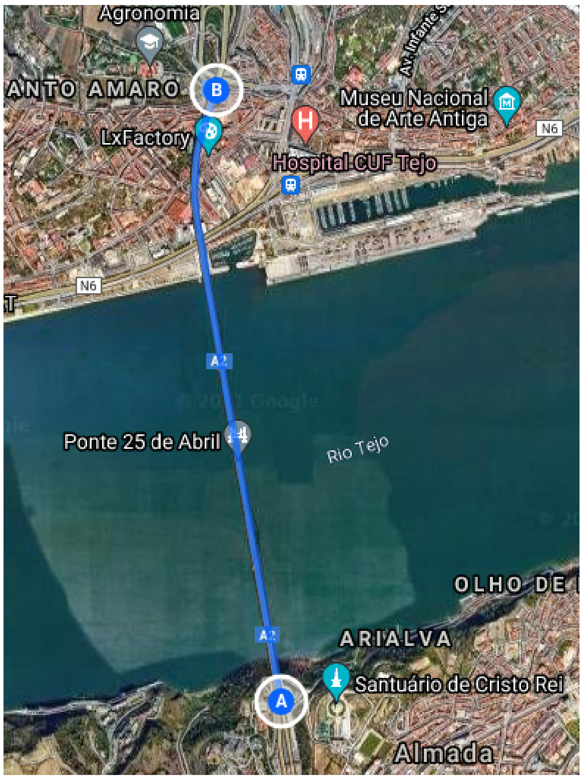


**Figure 4.9:** North-South segment missing from the network obtained with OSMnx.

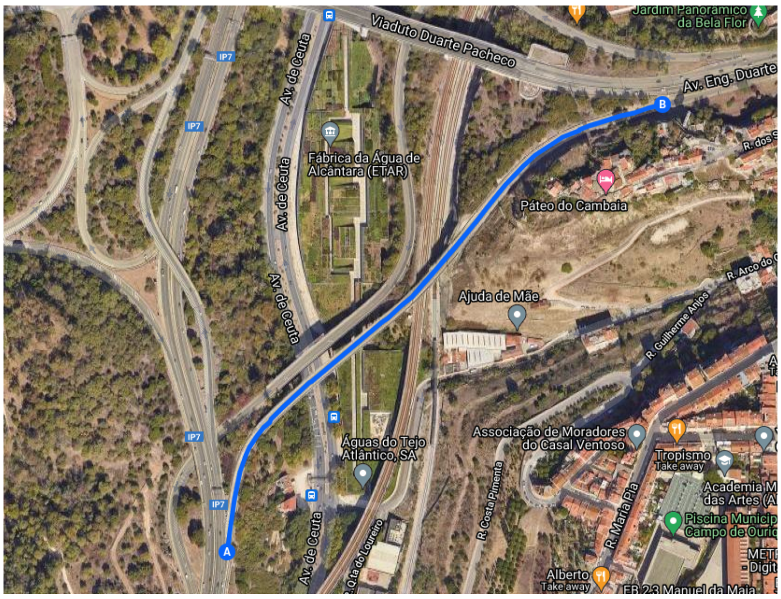

Viaduto Duarte Pacheco
Jardim Panorâmico da Bela Flor
Av. Eng. Duarte
B
IP7
Av. de Ceuta
Fábrica da Água de Alcântara (ETAR)
Páteo do Cambaia
Ajuda de Mãe
Águas do Tejo Atlântico, SA
Associação de Moradores do Casal Ventoso
Tropismo Take away
Academia M das Artes
Piscina Municip Campo de Ouriq
Alberto Take away
A

**Figure 4.10:** A2 exit missing from the network obtained with OSMnx.

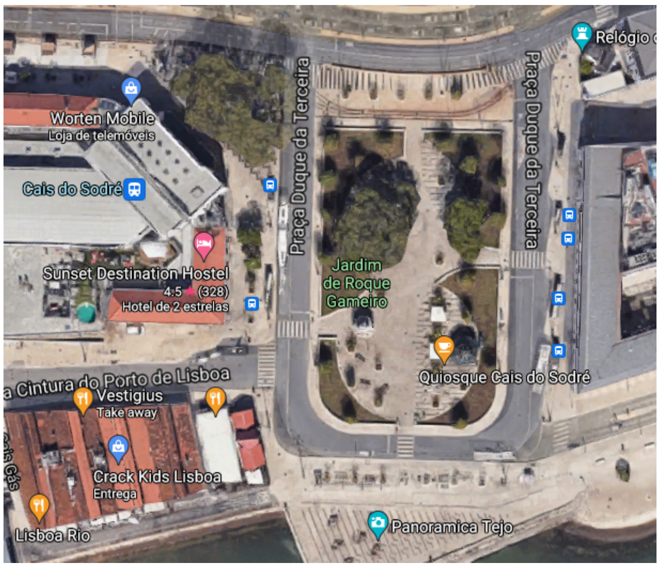


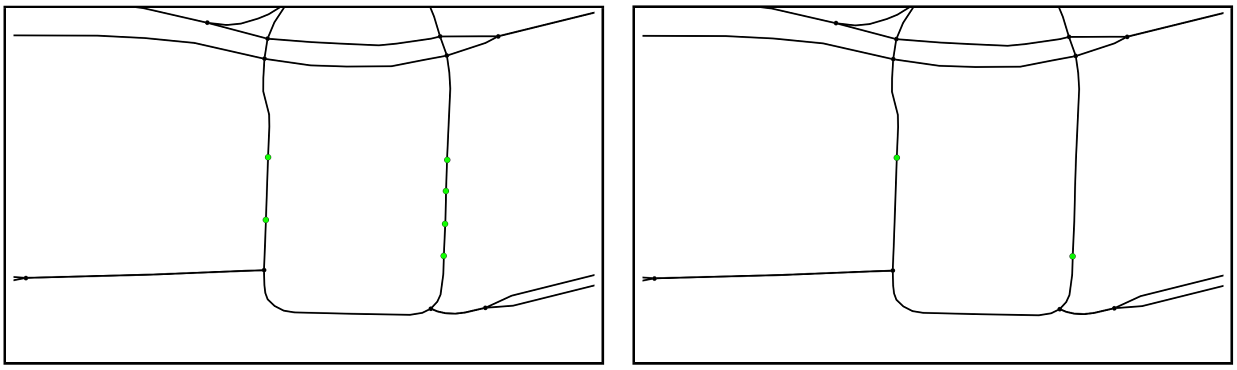
**Figure 4.11:** Redundant stops in Cais do Sodré.

## 4.3 Stop Clustering

There are stops that exist in close proximity to others and whose only purpose is to provide people with multiple boarding points so that stops do not become overcrowded. They are irrelevant to the optimization process as they serve different routes most of the time, leaving travel and waiting times unchanged if the stops were to be moved a few hundred meters along the road. As an example, in Cais do Sodré there are four stops all in the vicinity of each other, all serving different routes. From an optimization point of view, they are all equivalent.

We would like to cluster all equivalent stops into one, thus avoiding generating redundant routes and reducing the size of the bus network, making the network evaluation faster. To that end, two stops, $u$ and $v$ are clustered into one if the edge $(u, v)$ exists in the road network and has a cost of less than $100m$ and if $deg_{in}(v) = 1$. We force the edge $(u, v)$ to exist because, if a road node exists between the two stops, then that means that a route can go from $u$ to one other stop, $w$, without necessarily going through $v$, making it so that, if stops $u$ and $v$ are clustered under the location of $v$, that route would be way longer than it should as it would go from $v$ to $w$. We force $deg_{in}(v) = 1$ because if there is a route that goes from a third stop $w$ to $v$ and we cluster $v$ and $u$ under the location of $u$ then that route would have to go from $w$ to $u$, potentially making it much longer than what it, in fact, is. The $100m$ requirement is completely adjustable and it is there only so we do not cluster stops on different ends of the same avenue. If stops $u$ and $v$ can be joined and stop $v$ can be joined with a third stop $w$, then all the stops can be joined under the same cluster.

We started with $2193$ stops and managed to reduce this number to $1786$ using this criteria to join stops. The result for Cais do Sodré can be seen in Figure 4.12.



**(a)** Projected stops in Cais do Sodré before clustering. **(b)** Projected stops in Cais do Sodré after Clustering.

**Figure 4.12:** Impact of stop clustering in Cais do Sodré, downtown Lisbon. Black points are road junctions and green points are projected stops.

## 4.4 Relevant Graphs and Interactions

In this work, we use several different graphs to represent everything we need from the domain. It is then important to understand the different types of graphs, what is their purpose, and how they interact.

The first network, is the road network, $G_r$. In this graph, edges represent road segments and nodes represent road junctions, road ends or bus stops. Each node has an associated latitude and a longitude. The edges have several attributes, the length of the segment, the speed limit in that segment and, if the segment has complicated shapes, a geometry attribute, which is just a sequence of coordinate pairs, describing the form. In our case, the geometry attribute is used to compute projection candidates for bus stops, as described in Section 4.1, and for visualization purposes. The length and the speed limit allow us to estimate the time a bus takes to traverse a given segment.

The second network, and the only one with several instances, is the bus network, $G_b$. This is the object of optimization and, as such, a genetic algorithm population is filled with networks of this type, i.e. at each time step $t$, we will have a population $P_t = \{G_b^1, G_b^2, ..., G_b^n\}$. These networks are built from a set of routes. A route is just a sequence of stops. Those stops are made a part of the road network as described in Section 4.1, ergo we can compute optimal paths between consecutive stops in a route. In these networks, nodes correspond to stops and edges are connections between stops provided by a route. The cost of each edge is the time it takes for a bus to travel between the two stops through the optimal path. This is essentially the stop connection graph from Section 4.1, though now we need to take special care to properly compute optimal paths within the bus network. We need a node to exist for every stop a route belongs to and, as a consequence we need a different edge for every route in which a pair of consecutive stops exists. Every proxy of the same stop is connected by an edge whose cost is the transfer penalty, which, in our case, is $10min$. This simulates the users' preference to not transfer if the time penalty incurred by not changing bus is not very high. In Section 5.1 we discuss why we need to have multiple nodes per stop instead of using a multi-graph with multiple edges between the same nodes. All the tram routes are included in the bus networks but they are never interfered with during the optimization process.

Then we have the metro network, $G_m$. We want to assess what the use of the whole transportation network would be like with the bus network under evaluation, so the metro network must be present. It will not be changed and it is a singleton network. In terms of edges and nodes, it is very similar to the bus network. Two stations are connected if they appear consecutively in a line. In Lisbon, no two stops are connected by more that one line, so we do not have to have one node per station per line. If we were studying the city of Stockholm, for example, then we would have to have multiple nodes per station as there are multiple lines overlapping in some regions. For this network, we do not have geometry information for the edges, so the path between two stations is considered to be a straight line connecting them and the trains are considered to travel at $60Km/h$. The geographic points of the stations were not adjusted in any way. In Figure 4.13 we can see the metro network over the road network, for spacial reference.

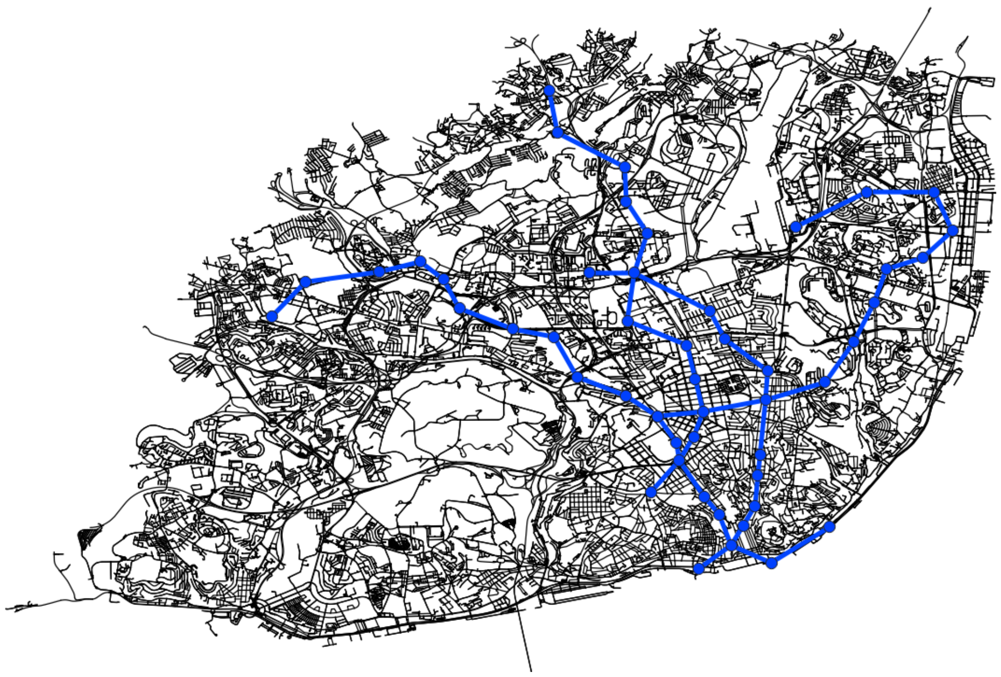


**Figure 4.13:** Metro network in the city of Lisbon.

The walking network, $G_w$, connects the bus network with the metro network and close bus stops. It is assumed that people are willing to walk up to $300m$ to transfer. The nodes are bus stops or metro stations and the edges represent possible walks between them. Even though metro stations have several exits, and the walking paths are complex, we do not have any information about the location of the exits so the walks are assumed to be a straight line from the origin to the destiny. OSM does provide walking paths along the road network if we so desire, but that would require us to have the metro exit locations. People are assumed to walk at $5Km/h$. This network is never changed. In Figure 4.14, we can see the resulting connections in the walking network in Cais do Sodré.

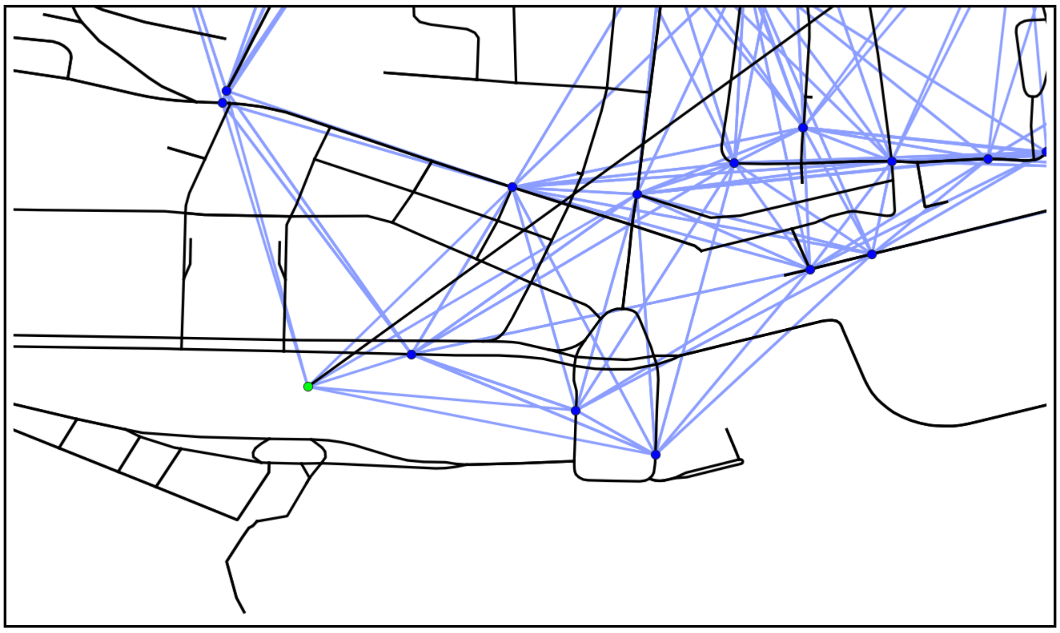


**Figure 4.14:** Walking network in Cais do Sodré. Black lines are road network edges and blue lines are walking network edges. Blue points are bus stops and green points are metro stations.

During optimization, we integrate the current bus network, under evaluation, with the walking network already connected to the metro network, creating a complete multimodal network, $G_c$, in which trips that involve walking, bus and metro can be planned. A small example of this interaction between the walking, bus and metro networks can be seen in Figure 4.15.

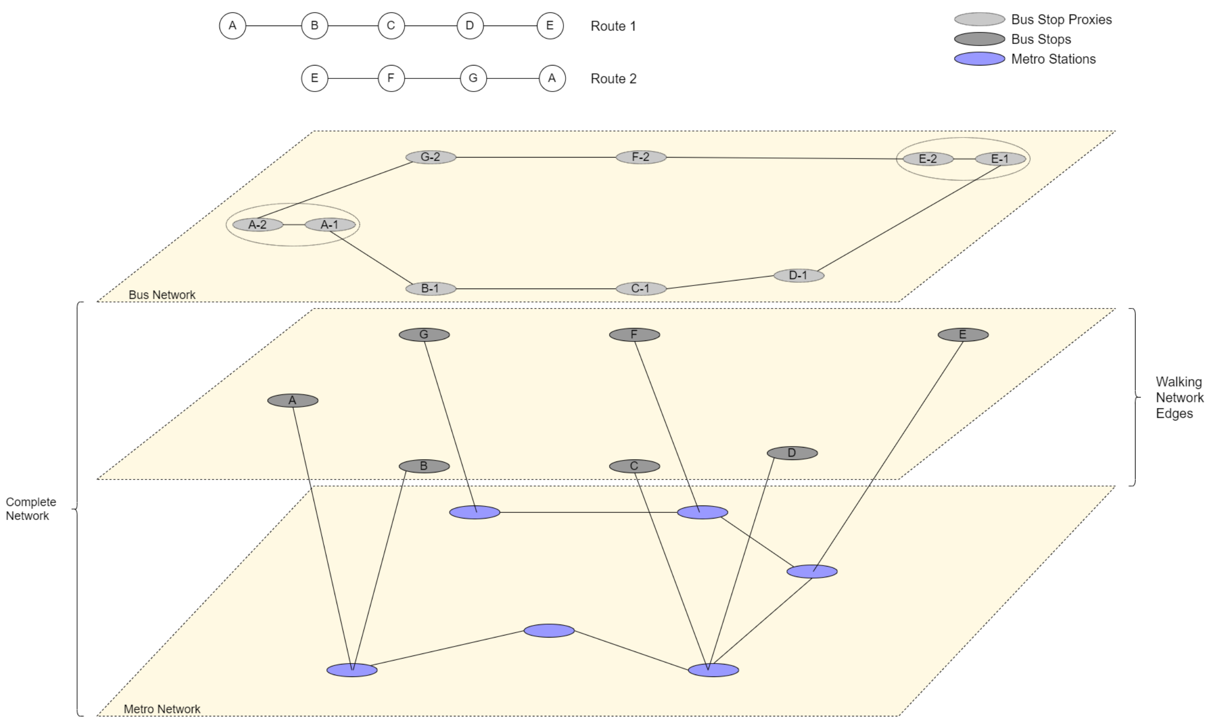


**Figure 4.15:** Interaction between the walking, bus and metro networks. The walking and bus networks are always the same. Bus networks are momentarily connected to them during optimization, creating a complete network.

# 5

# Network Evaluation

Contents

In this chapter we address the problem of efficiently evaluating a bus network. In Section 5.1 we go over how we simulate the way people use a bus network so that we can compute objectives based on the resulting trips. In Section 5.2 we discuss how we divided Lisbon into OD areas that allow the network evaluation to be faster.

## 5.1 Simulating Network Usage

In order to assess travel times, waiting times and all the other objectives we are striving to improve, we need to know how the public would use the transportation network under evaluation. Some authors, like Beltran et al. [14], use probabilistic models. In this work, however, we use a simple deterministic algorithm to predict how passengers would travel between two points in the city, using the available buses, the metro and, possibly, walking between the two.

In the deterministic approach, we face two options. We can do a real time simulation of the network being used or, we can compute everything in bulk, meaning, assuming that everyone traveling between the same points will always do it in the exact same way. Simulating the network allows the evaluation to be much more realistic as we could have passengers taking different paths due to prolonged waiting times or overcrowded buses and every objective would be computed more accurately. For example, to compute the loading factor of buses, in the simulation option, we would know exactly, at each point, how many people are inside each bus, while if we process everything in bulk, we have to assume a distribution of all the passengers that travel in a given segment and then compute the loading factor based on the total number of buses that will serve that segment during operation hours. Simulating is more accurate but it is also computationally expensive. Since the chosen method to predict network usage is part of the evaluation procedure which is executed many times (as it will be part of an optimization process), we opted for the bulk prediction method.

### 5.1.1 Time Optimal Paths

The algorithm that computes the paths that people take between ODs is a variant of Dijkstra's shortest-path algorithm. It computes time optimal paths and then introduces some penalties whenever the path has a transfer, thus ensuring minimization of transfers in the network under evaluation. Whenever a path goes through the walking network, a penalty of $10min$ is introduced because that means a transfer happened, the same happens whenever a path switches bus routes or metro lines without using the walking network. The path finding happens in the complete graph, $G_c$, which includes bus, metro and walking modes.

To compute shortest paths between all nodes in a graph, we considered three algorithms, one-to-all Dijkstra's Algorithm executed $|N|$ times, Johnson's Algorithm and the Floyd-Warshall Algorithm.

Johnson's Algorithm is a version of the Dijkstra's Algorithm to handle negative weighted edges. Since all the edges in the complete network have positive weights, we disregarded this option. A regular one-to-all Dijkstra has a temporal complexity of $\mathcal{O}(|E| + |V|log|V|)$, if we execute it $|V|$ times, one for each source node, we get $\mathcal{O}(|V||E| + |V|^2log|V|)$. The Floyd-Warshall Algorithm has a space complexity of $\mathcal{O}(|V|^2)$ and a temporal complexity of $\mathcal{O}(|V|^3)$. Since our graphs are sparse ($|E| \ll |V|$), the Dijkstra Algorithm is more efficient. There is another argument to chose Dijkstra over Floyd-Warshall, and that revolves around the fact that we are not really looking to compute paths between all nodes, but between some aggregation of the nodes. This aggregation is discussed in Section 5.2. This makes it so that the Dijkstra's complexity is brought down even further, because if we now have $K$ partitions, with $K \ll |V|$ and we want to compute shortest paths between them all, then Dijkstra's would have a complexity of $\mathcal{O}(K|E| + K|V|log|V|)$. The adaptation of the Dijkstra Algorithm to this partition of the graph is straight forward while it is not obvious how the Floyd-Warshall Algorithm can be adapted to this end. That was the final point we considered in our decision.

We introduced how we compute paths in the complete network. Now we can understand why we made the decision to have multiple nodes per real world stop instead of having a multi-graph. At first, it might seem like we do not need one node per route per stop if we have different edges for different routes connecting the same pair of stops, however, this would cause some routes to be discarded while looking for paths within the bus network. If we think about how Dijkstra's Algorithm works, when expanding a node, an edge is explored and the cost for the node it leads to would be noted. Then another edge, with the same cost, only corresponding to a different route, would be considered as an alternative, but because it does not provide an improvement, it would be disregarded. This makes it so that the computed path is completely dependent on the order in which edges are considered. Because the algorithm we are using to predict network usage is based on Dijkstra's Algorithm, we have to bypass this problem. Having multiple nodes for the same stop elegantly solves it.

### 5.1.2 Evaluation

Once we know how people travel between ODs, we can compute several quality metrics for the network. We will now present those metrics and then, in Chapter 6 we will utilize these metrics to formulate optimization processes that will allow us to have better networks.

First of all, we should introduce the concept of a trip. When a bus network is able to connect an origin and a destination, it does so through a trip, which is a path in the complete network, $G_c$. For further analysis of the trips, we can divide the trip into stages. There are three different kinds of stages, a bus stage, a walking stage and a metro stage. Practically speaking, a bus stage is a path, along a single route, in the bus network. A metro stage is a path, along a single line, in the metro network. A walking stage is a single edge path that connects two bus stages or a metro stage and a bus stage. This distinction is important because, every time there is a stage change, we apply a transfer penalty (unless the change is from a walking stage to a metro or bus stage, because we already apply the penalty whenever we transition to a walking stage) and the path cost increases.

Now, we establish the following notation regarding trips and bus networks:

- $\mathcal{T}$ - the set of all trips between all OD pairs. $\mathcal{T} = \{T(s,t) \mid s \in W^O, t \in W^D\}$;
- $T(s,t)$ - trip from $s$ to $t$, a sequence of stages $T(s,t) = (T^0(s,t), \dots, T^n(s,t))$, where $n \in \mathbb{N}_0^+$;
- $T^k(s,t)$ - the $k^{th}$ stage on the trip from $s$ to $t$. It is defined as sequence of triplets of the form $(u,v,r)$ where $u$ and $v$ are adjacent stops or stations and $r$ is the route/line connecting them or a special marker indicating that the path from $u$ to $v$ was made by foot;
- $T_{bus}(s,t)$ - bus stages in the trip from $s$ to $t$;
- $T_{metro}(s,t)$ - metro stages in the trip from $s$ to $t$;
- $T_{walk}(s,t)$ - walking stages in the trip from $s$ to $t$;
- $t(s,t)$ - travel time between nodes $s$ and $t$, i.e. $t(s,t) = t_{inv}(s,t) + t_{wai}(s,t) + t_{wal}(s,t)$;
- $t_{inv}(s,t)$ - in-vehicle time between nodes $s$ and $t$;
- $t_{wai}(s,t)$ - waiting time between nodes $s$ and $t$;
- $t_{wal}(s,t)$ - walking time between nodes $s$ and $t$;
- $f_r$ - frequency of service in route $r$ (in buses/hour);
- $t_r$ - time it takes for a bus to go from the starting station to the terminal station in a given route;
- $l_r$ - length of route $r$.

The total length, TL, of a bus network is computed as:

$$TL(G_b) = \sum_{r \in R} l_r \tag{5.1}$$

The Unsatisfied Demand, UD, of a bus network is computed as follows:

$$UD(G_b) = 1 - \frac{\sum_{(s,t) \in W} q(s,t) CO(G_b, s, t)}{\sum_{(s,t) \in W} q(s,t)}, \tag{5.2}$$

where $CO(G_b, s, t)$, the cover function, is one if the bus network $G_b$ provides, along with the rest of the transportation network, a connection between the origin $s$ and destiny $t$ within a number of transfers bellow the maximum allowed and zero otherwise.

The Required Fleet, which is the number of buses required to be in simultaneous circulation to assure the normal functioning of the network is given by:

$$RF(G_b) = \sum_{r\in R} t_r f_r. \tag{5.3}$$

We use the distance that will be covered by all the buses in operating hours as a proxy for what will be spent to keep the network working. So, if the network is active for $h$ hours, we have the Operator Costs, OC, given by:

$$OC(G_b) = \sum_{r\in R} f_r l_r h. \tag{5.4}$$

In vehicle time (IVT) is computed as follows:

$$IVT(G_b) = \sum_{(s,t)\in W} t_{inv}(s,t)\cdot q(s,t) \tag{5.5}$$

The Average Number of Transfers per passenger, ANT, is computed as follows:

$$ANT(G_b) = \frac{\sum_{(s,t)\in W}\Big(|T(s,t)| - |T_{walk}(s,t)| - 1\Big)q(s,t)}{\sum_{(s,t)\in W} q(s,t)} \tag{5.6}$$

The load factor is computed for connections between two adjacent bus stops for each route in which the stops are connected. This happens because different routes are served by different buses, therefore, the load is independent. We cannot compute this by stages because there can exist several trips that use the same connection within the same route but the stage does not necessarily coincide in its entirety. The Load Factor, LF, between stop $u$ and $v$, serving route $r$ then becomes:

$$LF(G_b,u,v,r) = \frac{pass(u,v,r)}{f_r\cdot h\cdot BUS\_CAP}, \tag{5.7}$$

where $BUS\_CAP$ is the capacity of each bus and $pass(u,v,r)$ is the number of passengers who traveled between stops $u$ and $v$ through route $r$ during the operation hours. This load factor calculation is assuming a uniform fleet and a uniform distribution of passenger flow over time. To formalize, $pass(u,v,r)$ can be defined as:

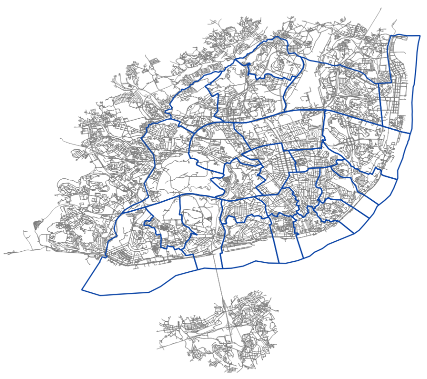
$$pass(u,v,r) = \sum_{(s,t)\in W} q(s,t)\cdot SIT(s,t,u,v,r), \tag{5.8}$$

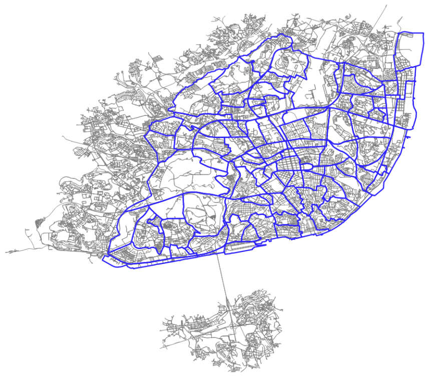
where $SIT(s,t,u,v,r)$ is the Segment in Trip function which is one if $\exists k$: $(u,v,r)\in T^k(s,t)$ and zero otherwise.

## 5.2 Map Partition

OD granularity is an important choice in these problems. A fine granularity will exponentially increase the OD pairs and the running time of the network evaluation routines. A coarse granularity, on the other hand, will not be faithfully representative of the users' reality, thus compromising the usefulness of our results. In Figure 5.1 we can see the several divisions we considered for this work on the map. An additional division is considered, which is bus stops and metro stations as the unit. This one would be the most accurate, but also the most expensive. We would have 1786 bus stops (after clustering) and 48 metro stations. On the other end of the spectrum, we have parishes which are too coarse. Also, Traffic Analysis Zone (TAZ) are still too large for the type of analysis we are doing. A last option is to divide the city into a squared grid. This allows us to control the size of the grid and thus, adjust the compromise between accuracy and computational complexity. We opted for a $30 \times 30$ grid, as shown in Figure 5.1c. All the squares in which there are no bus stops or metro stations are disregarded.

The paths are still computed in terms of the stops and stations, but we have aggregated origins and destinations. As our path finding algorithm is a modified Dijkstra, in the beginning, we place every stop and station from the origin square into the priority queue, and, once we expand a node in the destination square we terminate the search.


**(a)** Parishes


**(b)** Traffic Analysis Zones (TAZs)

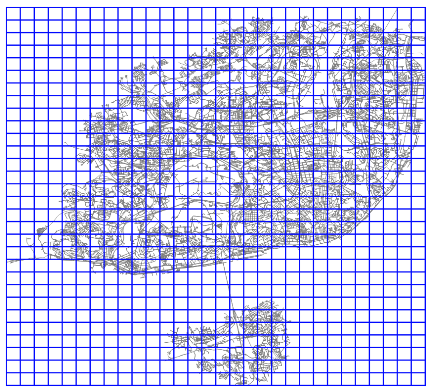

**(c)** $30 \times 30$ grid

**Figure 5.1:** Different OD granularities considered.

# 6

# Genetic Modeling

**Contents**

In this Chapter we go over how we model TNDP and TNFSP as optimization problems, and specifically, genetic problems. In Section 6.1, we talk about how we generate bus routes, which are the genes in TNDP. In Section 6.2, we show the problem formulation and discuss how to model the genetic operators for the problem at hands. In Section 6.3, we go into the transition from Multi to Single objective optimization using a human rating of the results from the Multi objective optimization processes.

## 6.1 Route Generation

In our formulation, Genetic Algorithms to solve TNDP will have to have a route pool as input, the algorithm will then decide the optimal set of routes from that pool. From the literature, Pattnaik et al. [19] computed shortest paths between bus stops, and then generated variants of each of those paths by temporarily removing links in those paths and recomputing the shortest paths between the stops. Fan and Machemehl [23] used Yen's Algorithm to compute the k-shortest paths between bus stops. These methods end up with a massive number of similar routes, maybe varying in some minor detours, this makes it so that many mutations will cause very small changes in the cost functions. We decided to compute the shortest paths between the main hubs in the network, generate what we call traversal routes and then include the original CARRIS routes in the pool. Examples of all these types of routes can be seen in Figure 6.1.

Traversal routes are routes intended to enable traveling between opposite ends of the city in a direct way. As the name indicates, they span almost the entirety of the city while obeying the constraints for maximal route size. There are essentially 4 distinct kinds of routes here, South-North, North-South, East-West and West-East. They are generated in similar ways, we start by dividing the city into strips, then there is a starting point per strip, and that is the most extreme point in terms of where we are looking to start, for example, if we are generating South-North routes, we divide the city into vertical strips and have as starting points, in each strip, the stop that is further to the South. Then we start building the routes by iteratively joining the closest stop in the general direction we are looking to go. We do this by looking for stops in an increasing radius centered around the latest stop. When we say closest stop, we are talking about the path through the road network, not the absolute distance between them.

## 6.2 Multi Objective Formulation

### 6.2.1 TNDP - First Optimization Phase

We will start the redesign of the network by deciding the best route composition for the users' needs without concerning ourselves with the route frequencies, in other words, we will be solving the Transit

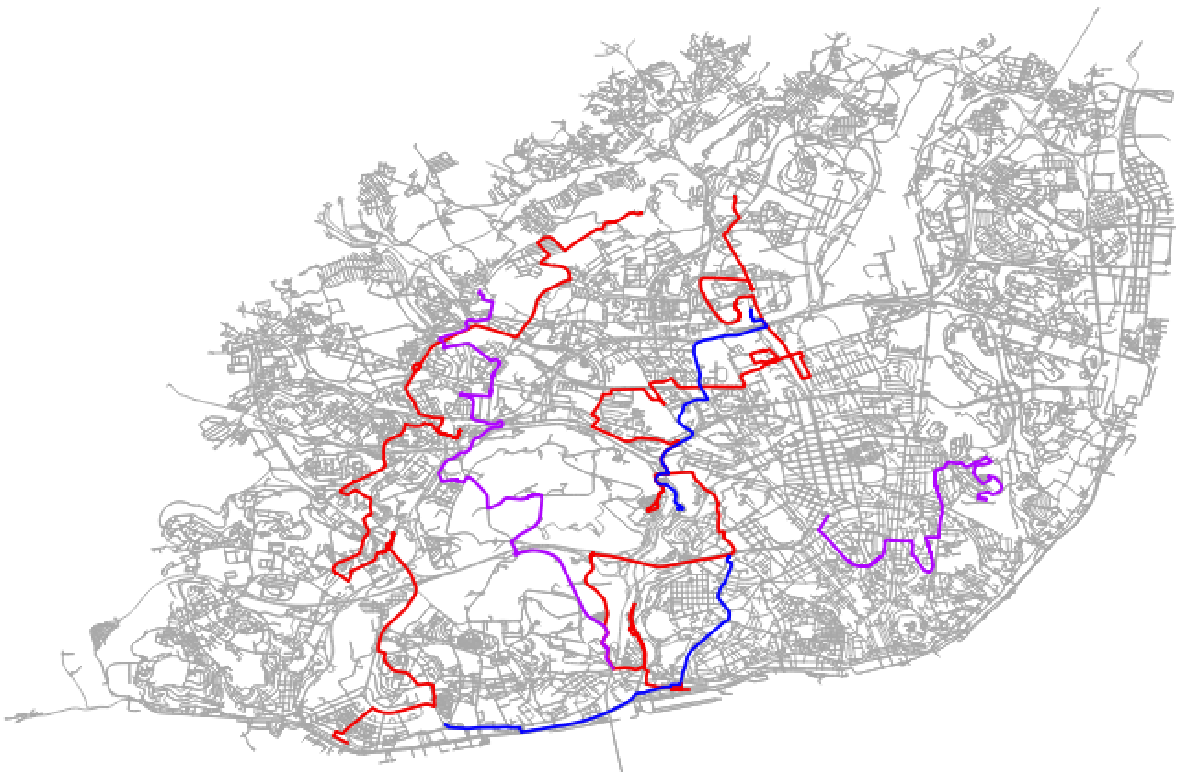


**Figure 6.1:** Example of routes in the route pool. Traversal routes in red (enabling travel from South to North), shortest paths between hubs in blue and original CARRIS routes in purple.

Network Design Problem. We will be looking to minimize the in-vehicle times, the transfers, the unsatisfied demand and, from the operator perspective, the total route length. We will be constraining the number of routes, the route length, and, because the network should be able to meet minimum quality of service requirements, the number of transfers.

TNDP is formalized as:

$$\begin{aligned} \underset{G_b}{\text{minimize}} \quad & TL(G_b), \quad UD(G_b), \quad IVT(G_b), \quad ANT(G_b), \\ \text{subject to} \quad & l_r \leq MAX\_ROUTE\_LEN \quad \forall r \in R, \\ & l_r \geq MIN\_ROUTE\_LEN \quad \forall r \in R, \\ & |R| \leq MAX\_NUMBER\_ROUTES, \\ & |R| \geq MIN\_NUMBER\_ROUTES. \end{aligned} \tag{6.1}$$

In terms of modeling, our proposal is similar to Pattnaik et al. [19] VSLC. We have a route pool which has the original routes found in the CARRIS network and a set of generated routes. Every network has all the tram routes that are in the original CARRIS network. Apart from that, the initial population is an array of randomly sized networks with routes randomly selected from the route pool. We never commit to a predefined size. Route insertion and deletion operators are introduced in the mutation process. Mutating, consists of swapping a random route in the network with a random route in the route pool. Single point crossover is used as shown in Section 2.2.4.D.

### 6.2.2 TNFSP - Second Optimization Phase

After we have our route composition decided, we want to optimize the frequencies of the routes in the network to better balance waiting times and operation cost of the network, in other words, we will be solving the TNFSP. We will be constrained by the available fleet, the load factor of the buses and the frequency values that are possible in a route.

We can formalize our TNFSP as:

$$\begin{aligned} \underset{G_b}{\text{minimize}} \quad & WT(G_b), \quad OC(G_b) \\ \text{subject to} \quad & RF(G_b) \leq AVAILABLE\_FLEET, \\ & LF(u,v,r) \leq MAX\_LOAD\_FACTOR \quad \forall (u,v,r) \in E_{G_b}, \\ & f_r \leq MAX\_FREQUENCY \quad \forall r \in R, \\ & f_r \geq MIN\_FREQUENCY \quad \forall r \in R, \end{aligned} \tag{6.2}$$

This optimization process is more conventional in the sense that it can easily be represented by a set of variables. We have a variable per route in the network whose frequencies we want to optimize. In this case, we also attempt to optimize the frequencies of all the tram routes. Single point crossover is used. Mutating a frequency set is just choosing a random route and give it a random frequency.

## 6.3 Single Objective Formulations

To go from a Multi objective to a Single objective setting, we usually consider a single objective function that is a sum of the objectives. We can do a uniform sum of the objectives or a weighted sum. In a uniform sum, objectives are considered equally important while in a weighted sum we can attribute relative importance to different objectives. It is also important to guarantee the different objectives share the same order of magnitude so objectives with higher values do not overshadow objectives with lower values. Attributing relevant importance to the objectives is a difficult task if we do not have a clear idea of the implications it may have on the results. Our approach to this problem does not require the objectives to share the same big-O notation and does not require the user to attribute a relative importance to the objectives. Instead, weights are inferred from a rating of the solutions resulting from Multi objective optimization. The idea is that after we assess the quality of the solutions that are given by the multi objective formulations, we can have a sense of what we are looking for in a network. After we know that, we can try to get as close as possible to a global optimum that represents the compromise we are looking for in a network. To this end, we will be rating networks generated by NSGA-II. Ratings can be gathered near experts, such as transport operators or the researchers on urban mobility. The rating will then serve as reference for a linear regression that will allow us to infer the weights for a weighted single objective formulation (introduced in Section 2.2) that stand for what we are looking for. We are trying to find:

$$\mathbf{w} = (w_0, w_1, w_2, \ldots, w_m)^T, \tag{6.3}$$

so that we can have the following objective function:

$$f(G_b) = w_0 + \sum_{i=1}^{m} w_i f_i(G_b) \tag{6.4}$$

In a Linear Regression Problem, we usually have a dataset of the form $\{t_i, x_{i1}, x_{i2}, \ldots, x_{ip}\}_{i=1}^{n}$, or simply $\{t_i, \mathbf{x}_i\}_{i=1}^{n}$. It is assumed that the relationship between the target and the vectors $\mathbf{x}_i$ is linear. We then try to find the best weight vector $\mathbf{w}$ such that a given error function is minimized. The error functions measure the difference between the target value in the dataset and the value given by:

$$y(\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{x} \tag{6.5}$$

A common error function is the Mean-Squared Error, given by:

$$E(\mathbf{w}) = \frac{1}{n} \cdot \sqrt{\sum_{i=1}^{n} \big(y(\mathbf{x}_i) - t_i\big)^2}, \tag{6.6}$$

or simply:

$$E(\mathbf{w}) = \frac{1}{n} \cdot \|\mathbf{y} - \mathbf{t}\|^2, \tag{6.7}$$

with $\mathbf{y} = \big(y(\mathbf{x}_1), y(\mathbf{x}_2), \ldots, y(\mathbf{x}_n)\big)^T$ and $\mathbf{t} = (t_1, t_2, \ldots, t_n)^T$. Minimizing the error is also an optimization problem, but in this case, we need only solve the following equation to obtain a closed-form solution:

$$\nabla E(\mathbf{w}) = 0, \tag{6.8}$$

which yields:

$$\mathbf{w} = (X^T \cdot X)^{-1} \cdot X^T \cdot \mathbf{t}. \tag{6.9}$$

This is the so called Least Squares Method. In this formula, $X$ is called the design matrix and it is defined as:

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}, \tag{6.10}$$

which allows us to write $\mathbf{y} = X \cdot \mathbf{w}$ and $E(\mathbf{w}) = \frac{1}{n} \cdot (X \cdot \mathbf{w} - \mathbf{t})^T \cdot (X \cdot \mathbf{w} - \mathbf{t})$.

Assuming that we have $n$ bus networks from the NSGA-II algorithm, we have:

$$X = \begin{bmatrix} 1 & TL(G_b^{(1)}) & UD(G_b^{(1)}) & IVT(G_b^{(1)}) & ANT(G_b^{(1)}) \\ 1 & TL(G_b^{(2)}) & UD(G_b^{(2)}) & IVT(G_b^{(2)}) & ANT(G_b^{(2)}) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & TL(G_b^{(n)}) & UD(G_b^{(n)}) & IVT(G_b^{(n)}) & ANT(G_b^{(n)}) \end{bmatrix}, \tag{6.11}$$

and:

$$\mathbf{t} = \left(MAX - r\left(G_b^{(1)}\right), MAX - r\left(G_b^{(2)}\right), \ldots, MAX - r\left(G_b^{(n)}\right)\right)^T, \tag{6.12}$$

where $r\left(G_b^{(i)}\right)$ is the average rating given to the $i^{th}$ network and $MAX$ is the maximum rating a network can be given. The target for each network is $MAX$ minus its rating because we want the objective function to be minimized and so, the better the rating, the lower the objective function value should be.

# Part III

# Results and Discussion

# 7

# Results

## Contents

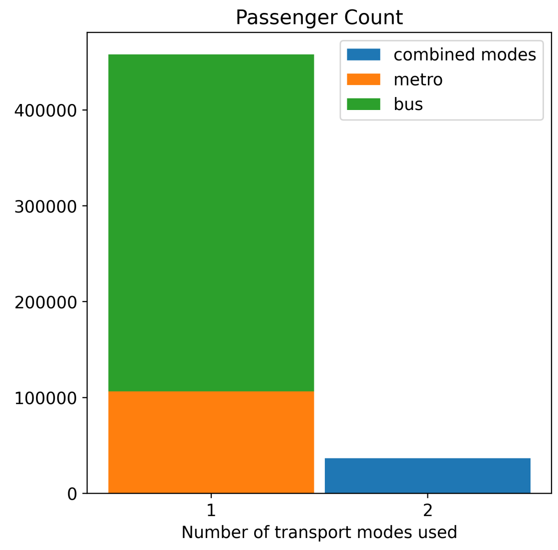
This Chapter presents the results obtained in this work, using the methodology described in Part II. We start by looking at the existing network in Section 7.1. In Section 7.2, we explore the networks that are in our approximation of the Pareto Front for the TNDP and compare them with the existing network. In Section 7.3, our attention will be drawn to the best network according to our single objective formulation of the TNDP. Frequencies for our top pick design will be set in Section 7.4 and a comparison to the existing network is provided. Lastly, in Section 7.5, we look into the GA parameters used and how they affect solutions.

## 7.1 Existing Network

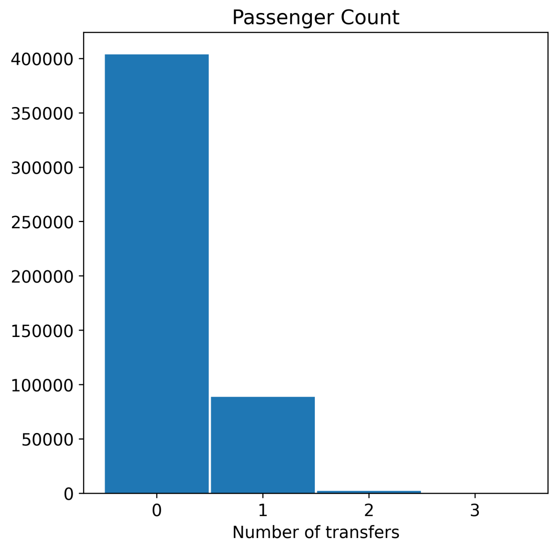
The existing CARRIS network has 309 routes, covering all 1786 stops (after clustering). According to our model of how passengers utilize the network, most passengers can get to their destination without the need to transfer but, there are a small number of passengers that have to transfer up to 3 times as depicted in Figure 7.1b. Most passengers use only the bus network while some use only the metro network and a small portion has to use a combination of modes between metro, walking and bus. This is depicted in Figure 7.1a. The big majority of passengers does not need to walk for more than 5 minutes but a small amount of passengers needs to walk up to 15 minutes. Figure 7.1c shows that traveling times follow a power law distribution, the majority of passengers gets to their destination very quickly and longer travels are not taken by many passengers. The routes in the network sum a total of $3561Km$ while leaving only $0.7\%$ of travels unsatisfied. All passengers together sum a traveling time of $4.1M$ minutes between all modes of transportation considered and average a number of $0.19$ transfers per trip.

## 7.2 Multi Objective Optimization

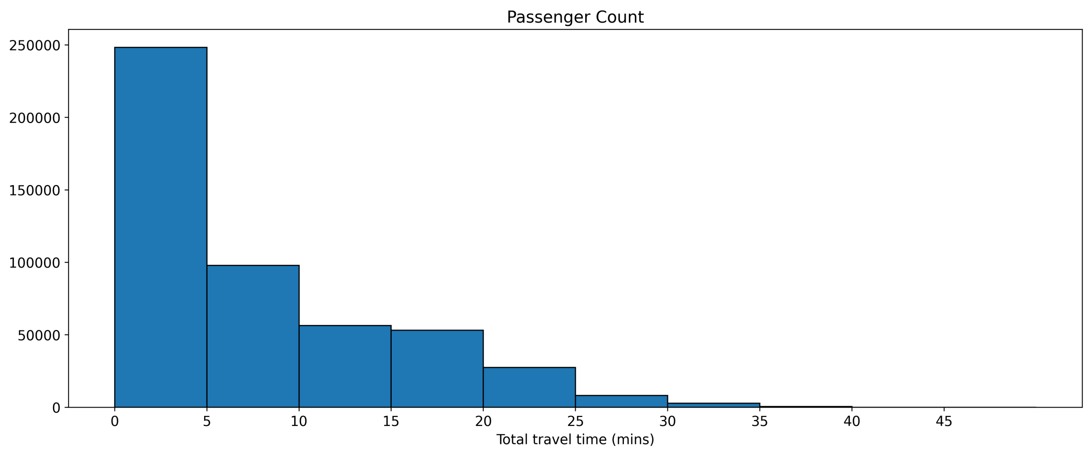
The networks that are discussed in this sections where obtained with the NSGA-II algorithm with a population of 200 networks, a mutation probability of 0.1 that ran during 300 iterations as these seem like the most appropriate, given our results from Section 7.5. The experiment was run in a computer with an Intel Core i5-8250U CPU and took 46 hours to finish. The evaluation of the networks is done in parallel using the 8 threads, but all the rest is executed in a single thread fashion. The time it takes to evaluate a network is a function of its size. The original network with 309 routes takes about 14 seconds to fully evaluate while a network with 200 routes takes about 5 seconds. The maximum allowable number of routes was 400 and the minimum was 200. A sample of 9 networks was chosen for a careful examination and their objective function values can be seen side by side in Figure 7.2. These will be rated on a scale of 1 to 10 so that we can infer the weights for a single objective optimization process. The networks are identified by their order in the crowd distance sorting. The number of routes in each



**(a)** Modes used by passengers in the original network



**(b)** Number of transfers per passengers in the original network



**(c)** Total traveling time per passenger (no waiting times).

**Figure 7.1:** Statistics on the existing CARRIS network.

network can be seen in Table 7.1.

| network | lisbon | $n0$ | $n24$ | $n49$ | $n74$ | $n99$ | $n124$ | $n149$ | $n174$ | $n199$ |
|---|---|---|---|---|---|---|---|---|---|---|
| route count | 309 | 201 | 200 | 200 | 209 | 200 | 207 | 232 | 200 | 200 |

**Table 7.1:** Number of routes of each network obtained with the NSGA-II algorithm.

The big majority of the networks has a number of routes very close to the minimum allowable number of routes. The network with the maximum number of routes is composed of 256 routes. We were expecting a more diverse set of networks when it comes to the number of routes, however the number of routes is not an objective, instead we opted to use the total length of the network as a measure of

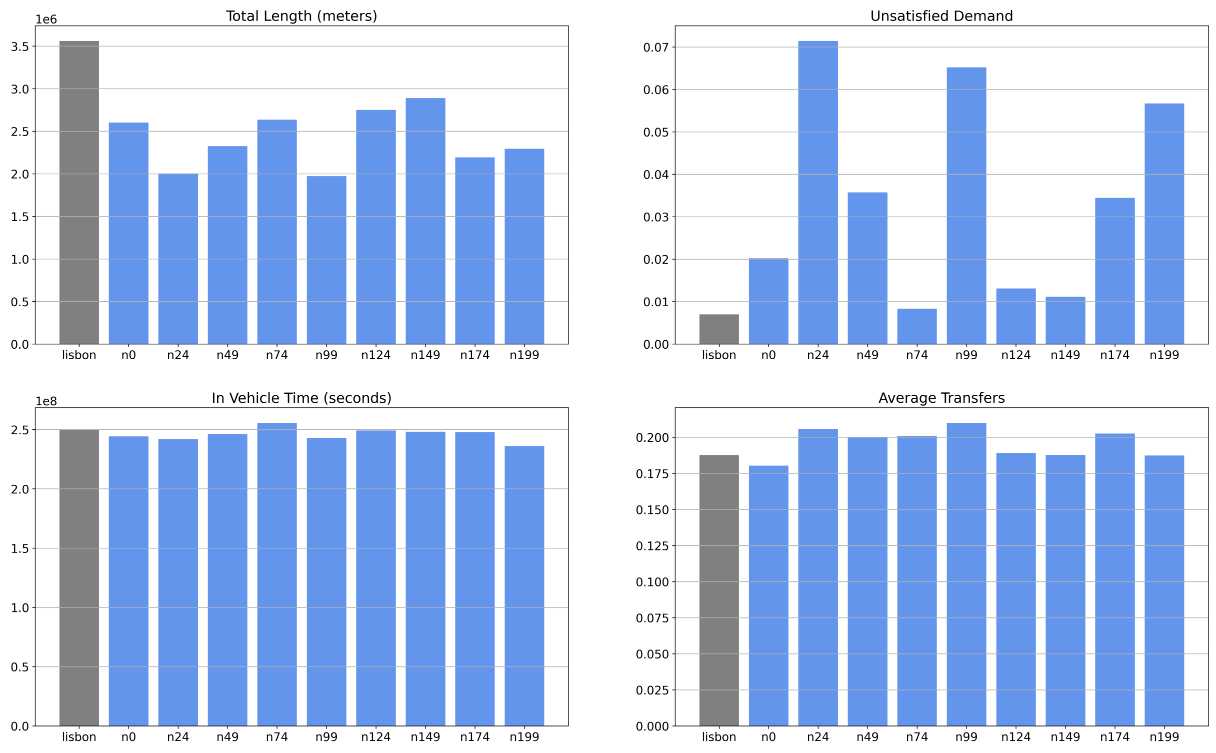


**Figure 7.2:** Comparison of objectives for networks obtained with the NSGA-II algorithm.

route efficiency so the diversity lies in the average length of the routes and not in the number of routes. The average route length varies from $9.5Km$ to $13.3Km$ and presents a bimodal distribution with peaks around the $10.5Km$ and $12.5Km$ marks.

All the networks present a total length bellow the original network but no network is capable of satisfying demand better. In terms of in-vehicle time and average transfers, there are networks capable of more direct and shorter trips but there are also networks that demand longer trips and a higher transfer rate. Looking at Figure 7.3, we can see that, generally speaking, total length is inversely correlated to the remaining objectives which makes sense, lengthy routes or an increased number of routes should cover a wider area and enable more direct trips which allow for a lower travel time and transfer rate. Unsatisfied demand is also inversely correlated to in-vehicle time and average transfers. This happens because OD pairs that are very far away imply a long time in-vehicle and there probably will not be a direct route connecting the two points which means that the average transfers also increase. The in-vehicle time and the average transfers are positively correlated. Increasing transfers, will decrease the directness of the path taken which, in turn, will increase the in-vehicle time.

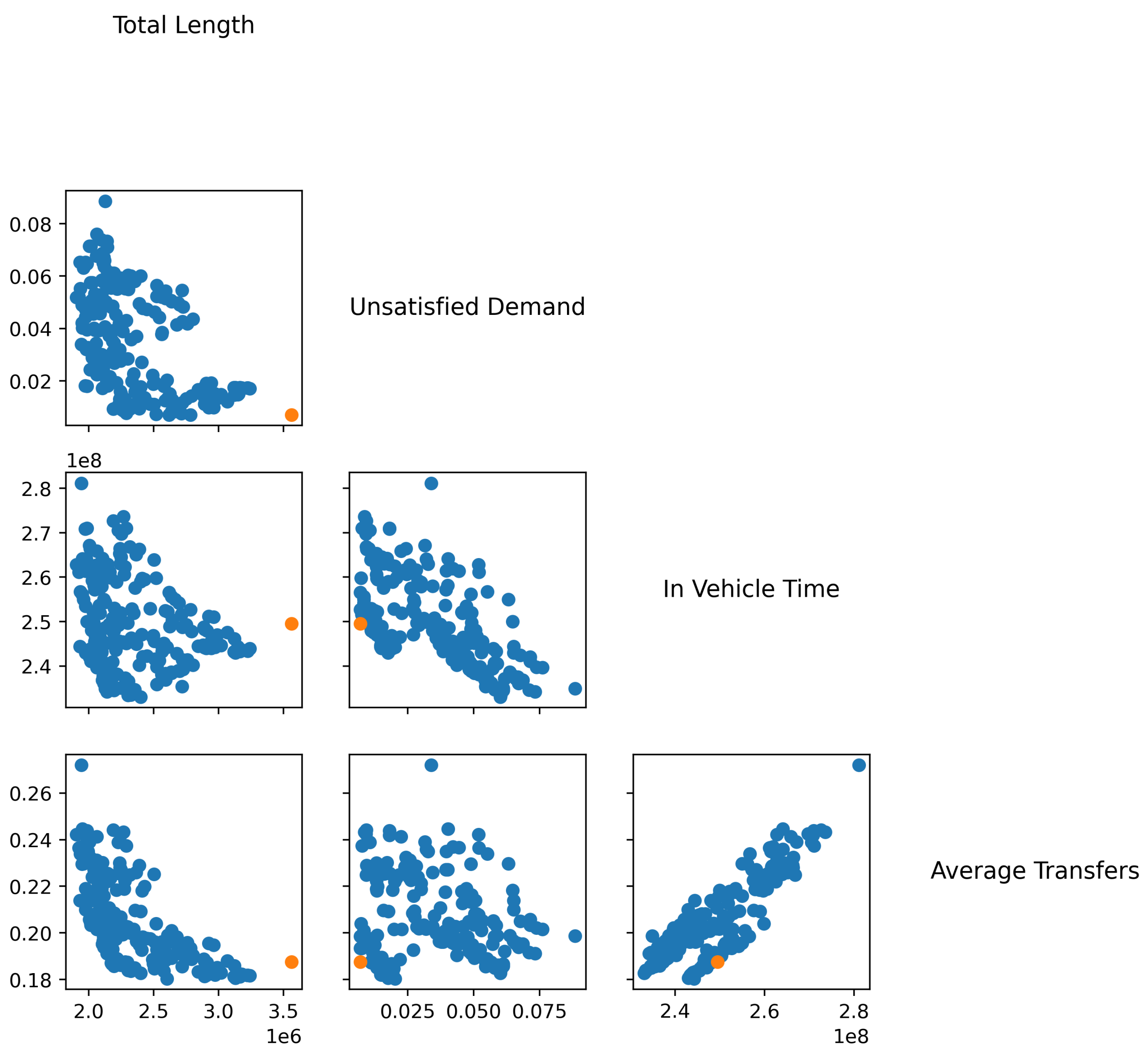


**Figure 7.3:** Scatter plots of objectives of all the networks obtained with the NSGA-II algorithm. The orange points correspond to the original CARRIS network.

## 7.3 Single Objective Optimization

Four people were asked to rate the networks selected in Section 7.2 from 1 to 10, yielding the following average rating for each network:

| network | $n0$ | $n24$ | $n49$ | $n74$ | $n99$ | $n124$ | $n149$ | $n174$ | $n199$ |
|---|---|---|---|---|---|---|---|---|---|
| rating | 7.5 | 3.5 | 5 | 5.75 | 3.25 | 6 | 7.5 | 5.5 | 4.5 |

**Table 7.2:** Average rating of each network obtained with the NSGA-II algorithm.

After the linear regression we get the following weights:

| $w0$ | $w1$ | $w2$ | $w3$ | $w4$ |
|---|---|---|---|---|
| $-15.85$ | $1.04 \times 10^{-06}$ | $53.13$ | $7.75 \times 10^{-09}$ | $72.38$ |

**Table 7.3:** Objective function weights after Linear Regression.

We ran a Classic GA with the weights in Table 7.3 with a population of $200$ individuals during $300$ iterations, with a mutation probability of $0.1$ and a crossover probability of $0.8$ because this is the best performing parameter combination according to our analysis in Section 7.5. We ran the Classic GA but with no weights, the same parameters and the individual objectives normalized with a min-max normalization so we could see the differences between the quality of the solutions when we emphasize different objectives versus when we do not. The objective function, in this case:

$$f(G_b) = \sum_{i=1}^{m} \frac{f_i(G_b) - \min(f_i)}{\max(f_i) - \min(f_i)}. \tag{7.1}$$

Both experiments take about 30 hours to complete in an Intel Core i5-8250U CPU. This is dependent on the crossover probability as fewer network evaluations have to be done if fewer new networks appear in new populations. The difference in time compared with the NSGA-II comes from the crossover probability and from the reduced amount of computation that comes from fitness computation versus the non-dominated sorting and crowd distance sorting of NSGA-II. The average weighted objective function value over the algorithm iterations can be seen in Figure 7.4 for the both experiments ran. According to the criteria we set when rating the networks, when we use the weighted objective function, a Single Objective GA is able to produce networks that are better than the best network we got from the Multi Objective Optimization after about 60 iterations. When we use a uniform weight distribution with normalized objectives we can reach a level of quality superior to the Multi Objective Optimization but that only happens after about 200 iterations.

In Figure 7.5 we can see the individual objectives of the networks obtained through Single Objective Optimization side by side with the ones selected from the Multi Objective Optimization and the original CARRIS network. Both networks obtained with Single Objective Optimization have $200$ routes. The highest rated network was network $0$ and, as we can see, the network obtained with the weights from the linear regression trumps that network in every objective. The network obtained with uniform weights and normalized objectives just does not achieve that in the number of transfers but it manages to have smaller total length than the network obtained with a weighted function.

The weighted objective function value can be seen for the networks in Figure 7.6. The Multi Objective Optimization produced three networks better than the original, according to this objective function. The Single Objective Optimizations produced the best networks of all with the weighted objective function

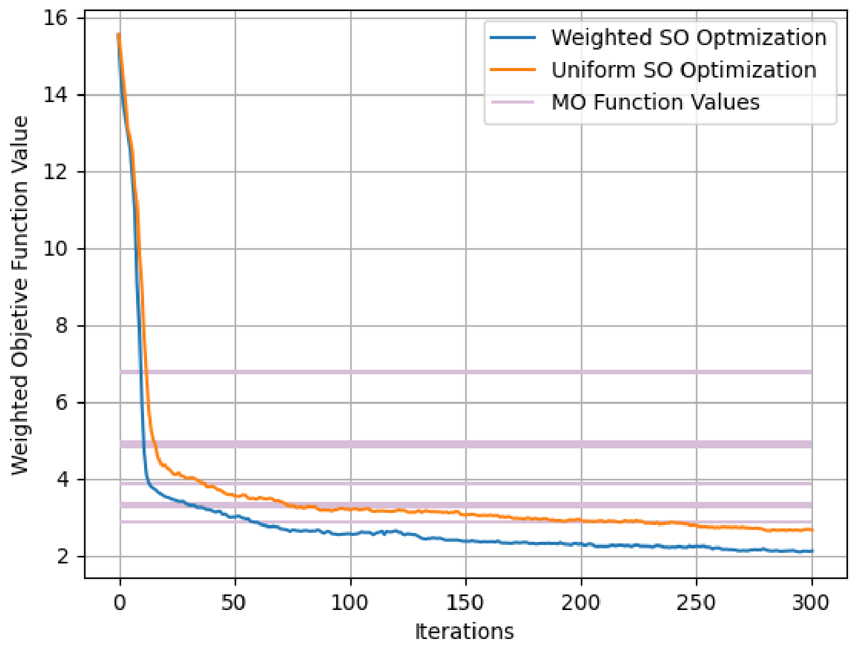


**Figure 7.4:** Average objective function during the iterations for the Single Objective GA considering Weighted function in blue and the Uniform function in orange. The displayed function is the weighted for both.

being able to surpass the uniform sum objective function.

The best network obtained through optimization has 200 routes, 109 less than the original network which translates in a difference of around $1300Km$ that no longer have to be traveled by the buses. In Figure 7.7 we can see the differences in travel times imposed by the new network. Most of the passengers has their traveling time unaffected (differences of up to $2.5$ are not considered) but the majority of trips in which there are significant changes were positively affected. The network also enabled more direct trips as seen in Figure 7.8. No passenger had their trips increased in more than one stage and the passengers that had stages cut off their trips are more than double of the ones that have to transfer one additional time now. Some even cut two transfers off their trips. In these histograms, we omitted the central bar because the total amount of passengers is $566K$ which would make the differences between the smaller bars imperceptible. For the big majority of passengers, travel times and transfers remained the same. This is due to the big majority of traffic happening in the center of Lisbon in which most trips are already direct with the existing network. The new network is not able to satisfy as many trips as the original network but the difference is marginal. In general, the new network is able to provide a similar service to the one provided in the original network, but does so in a more efficient way, using less routes. The roads covered by the original network and by the best network can be seen in Figure 7.9.

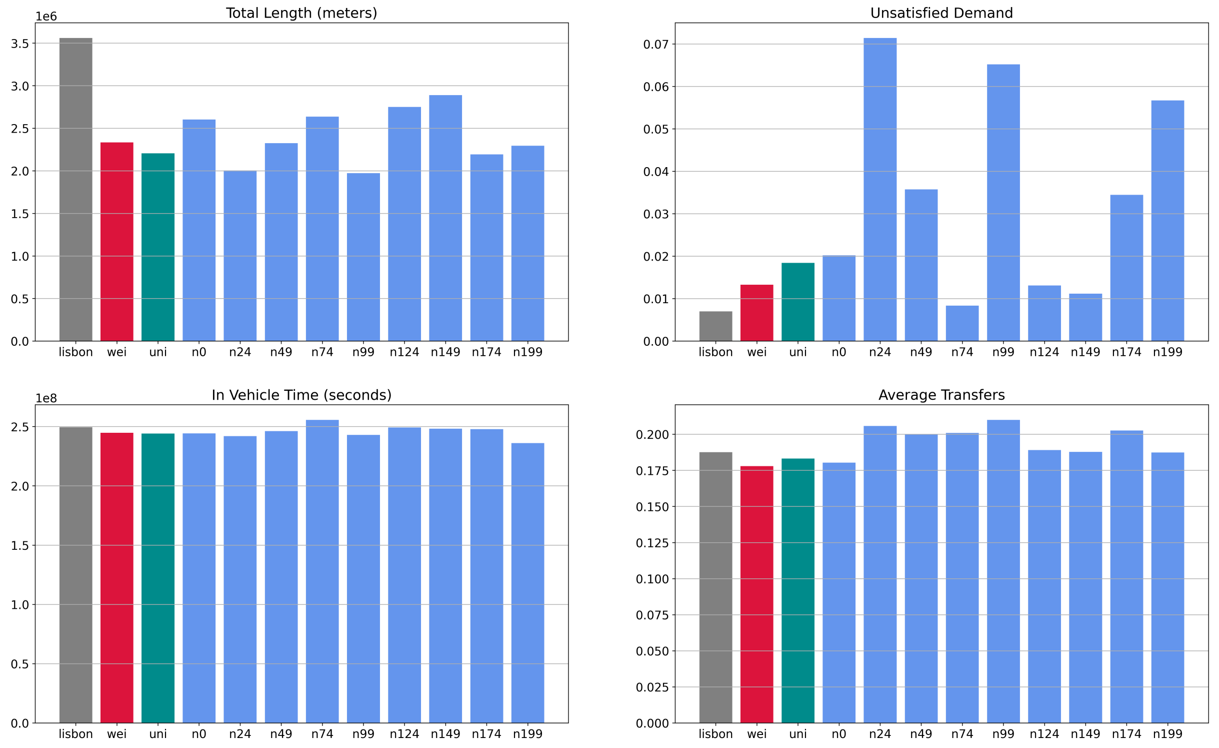


**Figure 7.5:** Comparing individual objectives of networks from Multi Objective Optimization and Single Objective Optimization. ”lisbon” is the original bus network. ”wei” was obtained in a single objective optimization considering the weighted function. ”uni” was obtained in a single objective optimization considering the uniform function. The remaining networks resulted from NSGA-II.

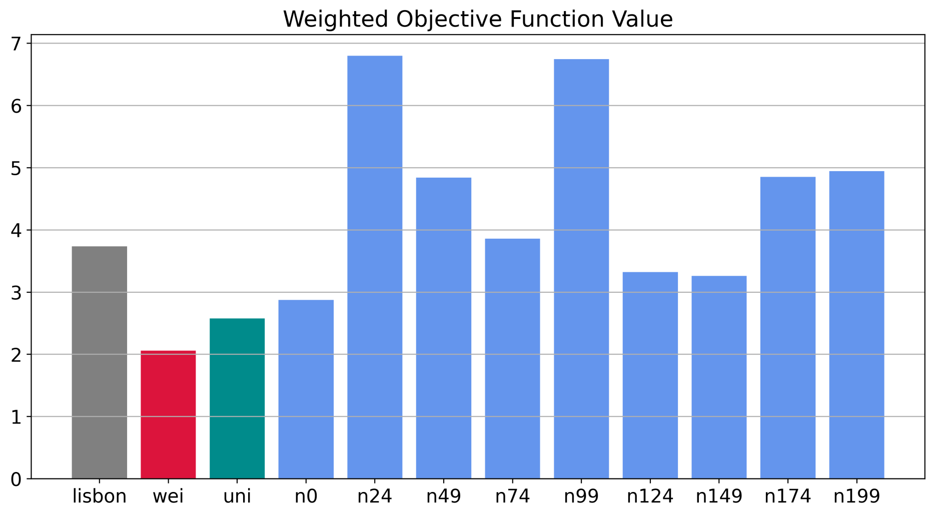


**Figure 7.6:** Weighted Objective Function value for the original network and the networks obtained through optimization.

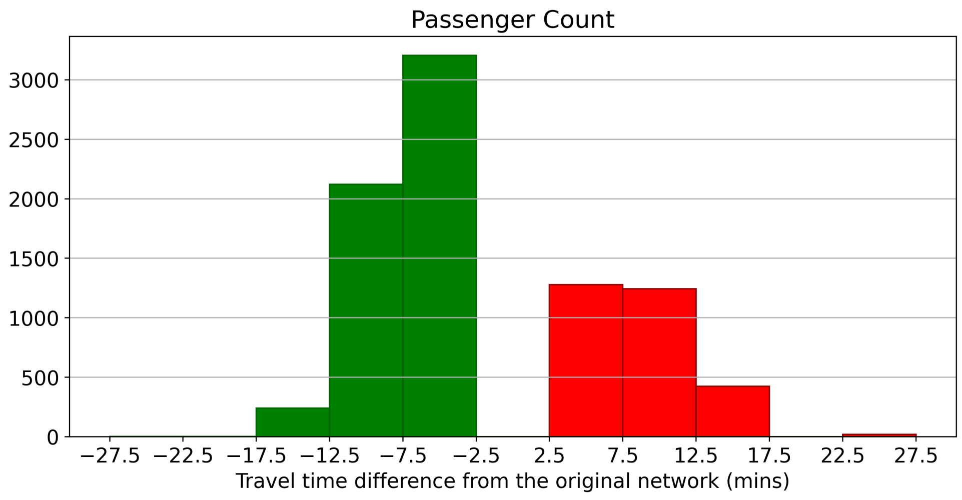


**Figure 7.7:** Differences in travel time from the original network to the best network.

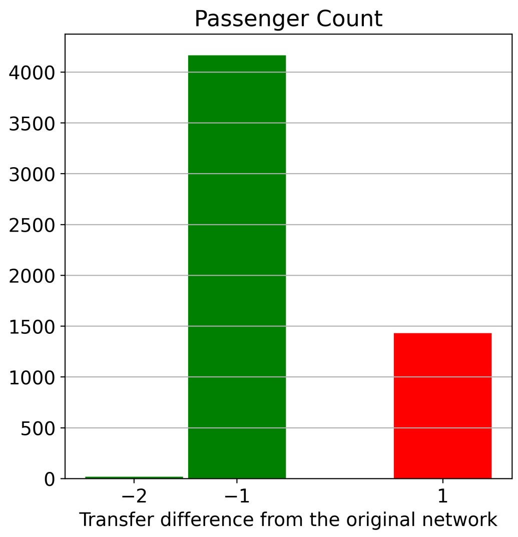


**Figure 7.8:** Differences in transfers from the original network to the best network.

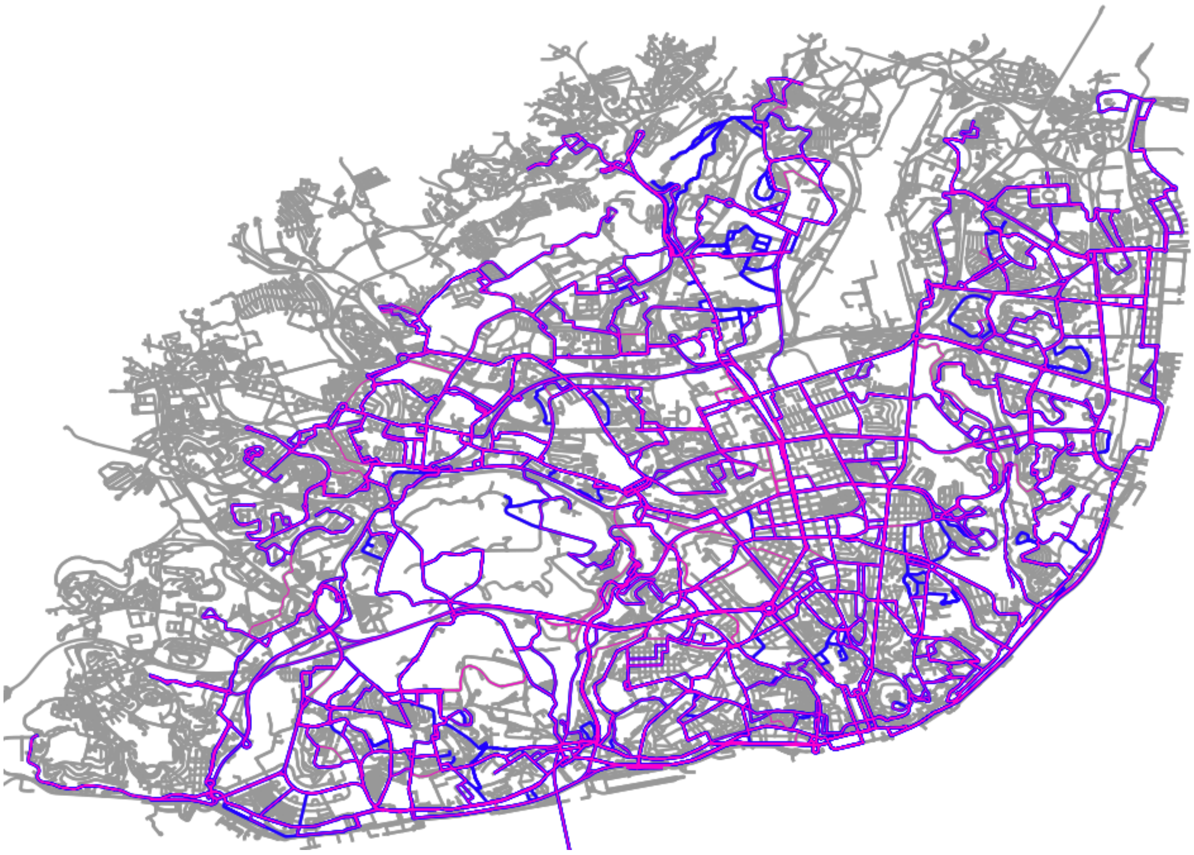


**Figure 7.9:** Roads covered by the original bus network (in blue) and the best network obtained through optimization (in pink).

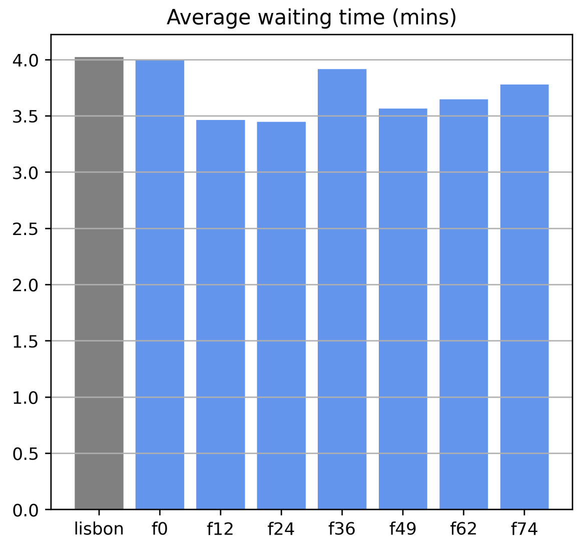
## 7.4 Frequency Setting

When it comes to frequency setting, we want to do the same transition from multi objective optimization to single objective optimization and assess the existence of any improvements. We also want to try optimizing frequencies for the whole day and optimizing frequencies for different times of the day to see if having different frequencies along the day can help improve the service and reduce operator costs. For the experiments shown, the parameters used were equal to the ones used in the Transit Network Design Problem however a new parameterization should be performed has the nature of the problem is different. The experiment took 2 hours in an Intel Core i5-8250U CPU machine. This time everything was ran in a single thread fashion. A frequency set takes about 75 milliseconds to evaluate.

We start by doing a multi objective experiment with the original CARRIS network because it provides us with a baseline to compare our results. The problem formulation is the one presented in Section 6.2.2. We assume a uniform fleet of $750$ buses with capacity for $80$ people, with a maximum load factor of $1.0$, a maximum frequency of $20$ buses per hour and a minimum frequency of $1$ bus per hour. It would be more accurate to have a non uniform fleet with different buses that can be attributed to different routes. That falls in the Transit Network Scheduling problem which is outside the scope of this work. The results for the experiment can be seen in Figure 7.10, once again, with the values of the CARRIS network presented for comparison. In general, all frequency sets performed better than the original, with some presenting an improvement of about $30$ seconds in the average waiting time and all reducing the total distance traveled by the buses during working hours in almost half. From the values we can see that the compromises between the two objectives are easier to understand this time, as the frequencies rise, so does the distance covered by the buses and the waiting times fall.

When rating the frequency sets and then running a linear regression so we could infer weights for a single objective optimization, we got negative weights for one of the objectives. This happened during TNDP as well but it disappeared as more people rated the networks. In this case, a single person rated the frequency sets. If more objectives existed, a correlation analysis could be performed and we could remove one of two positively correlated objectives, for example. For this particular case, we have no workaround. However, we still want to test how optimizing frequencies for different parts of the day can influence the quality of the frequency sets. To that end, we will use a single objective function that is the sum of both objectives normalized, similar to what is presented in Equation 7.3 but with frequency sets as the object of optimization.

For the case where each route has the same frequency throughout the day, a single experiment was done. For the other case, an optimization process was run for each part of the day. We divided the days into $5$ distinct times, the morning rush hour, from $07{:}00$ am to $10{:}00$ am, the middle of the day, from $10{:}00$ am to $04{:}00$ pm, the afternoon rush hour, from $04{:}00$ pm to $07{:}00$ pm, the happy hour, from $07{:}00$ pm to $11{:}00$ pm and the middle of the night, from $11{:}00$ pm to $07{:}00$ am. Because different normalizations



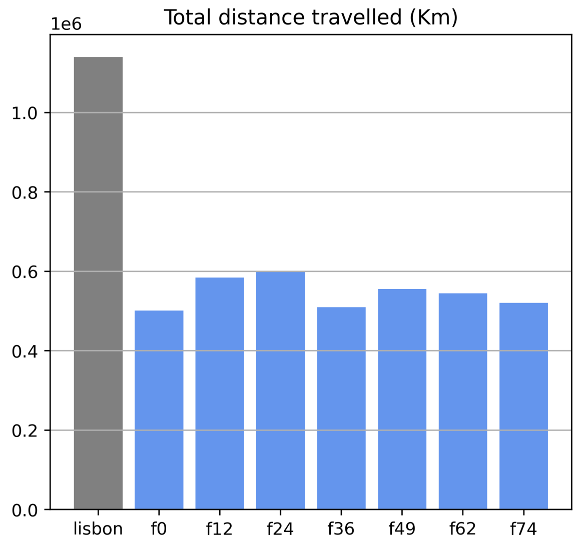


**Figure 7.10:** Frequency sets and respective objectives on the CARRIS original route plan, resulting from 300 iterations of NSGA-II with a population size of 200 and a mutation probability of $0.1$.

were made for the different parts of the day, as the total traveled distance varies a lot depending on the period of time under optimization, we do not present a comparison of the objective function values. However, a comparison of the evolution of the individual objectives can be seen in Figure 7.11. For the case in which separate optimizations are made, the waiting time is still presented as an average but the traveled distance is presented in a stacked fashion so we can compare it to the traveled distance when the routes have a fixed frequency for the total of the day. In both cases, we improve upon the original frequency set. When we go from optimizing to the whole day towards optimizing specific parts, we loose in distance traveled but gain in average waiting time. We were expecting an improvement in both objectives, however, the fact that we are not properly weighting the objectives is affecting the results and the fact that every sub-problem (for the different times of the day) is solved using the same parameters is also negatively affecting the results. For example, the night time is a big problem if we are demanding every route has at least one bus an hour. In the night time we probably should not even have buses working under the frequency paradigm, but scheduled to specific times which is what CARRIS does. The fact that everything is averaged is working against us in the night time because if only sporadic buses existed during the night, the waiting times would increase a lot. As we can see in Figure 7.11, the night time is actually the time of the day with the least average waiting time which does not contribute a lot to the average as there are less passengers at night but the distance traveled contributes to the whole and increases it a lot. This should either be run with different parameters or in a joint optimization process in which frequencies for different times of the day are optimized at the same time.

As a final experiment, we wanted to see how much improvement in terms of the TNFSP objectives a refined topology can enable. In Figure 7.12 we can see: the original network and original frequen-

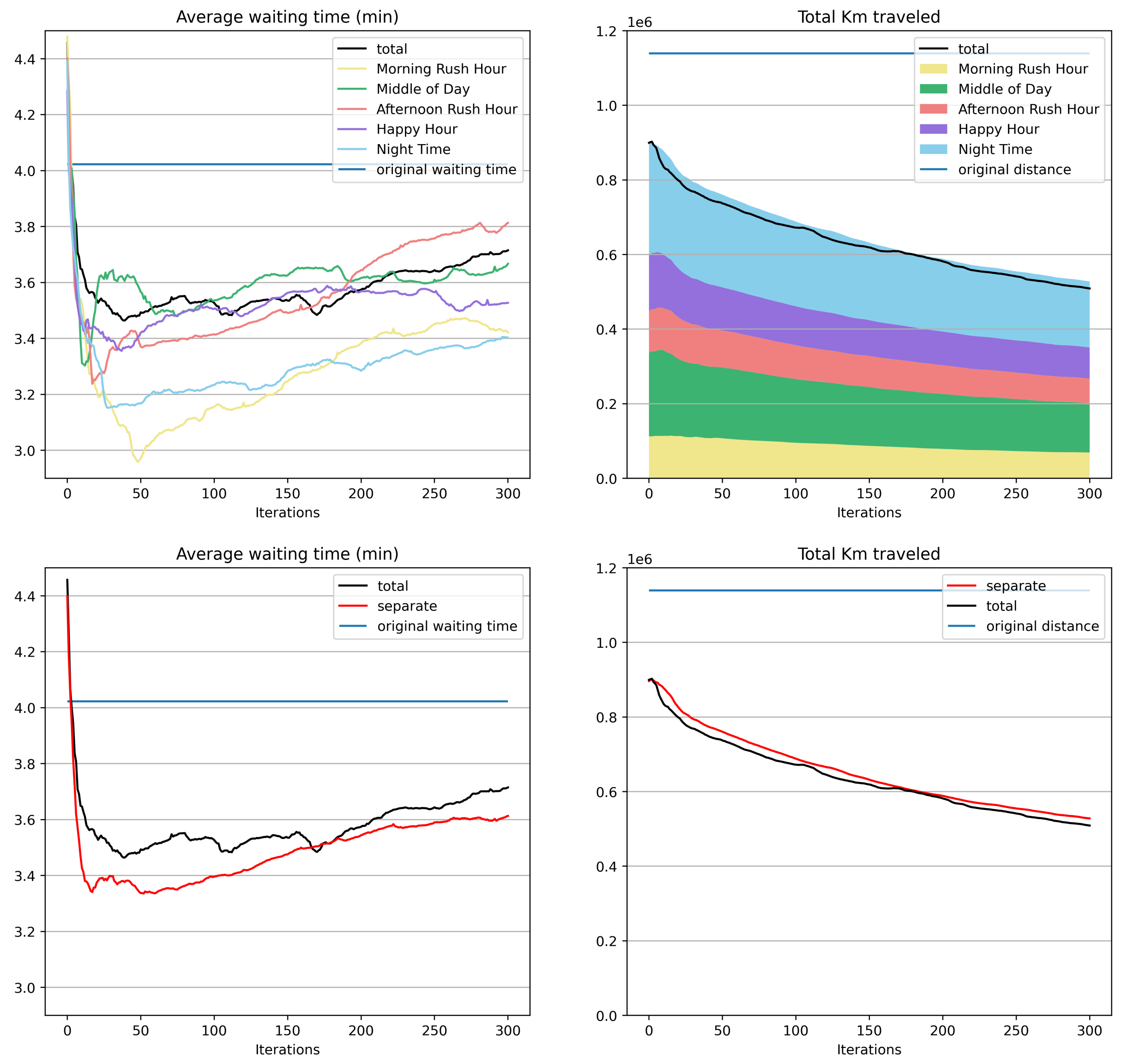
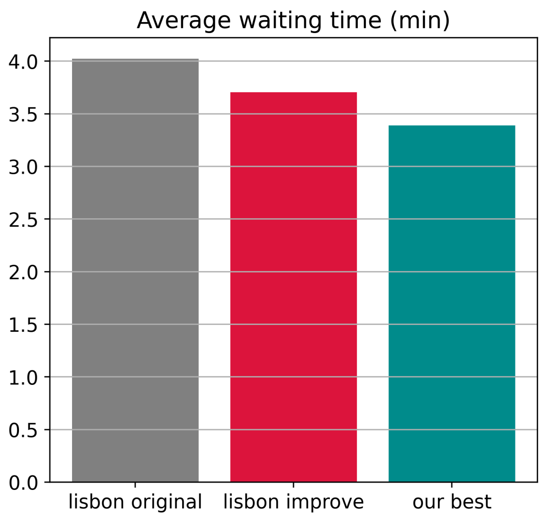


**Figure 7.11:** Objectives through the iterations of a single objective algorithm. Total (in black) is the optimization process for which a bus route as the same frequency throughout the whole day. Separate (in red) is an average or sum of the objectives for different optimizations for different times of the day.



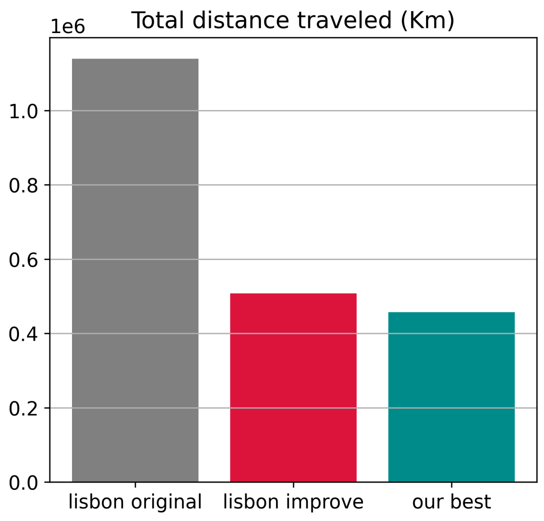


**Figure 7.12:** TNFSP objectives for the original CARRIS topology and frequencies (original lisbon), original CARRIS topology and improved frequencies (improve lisbon) and our best topology from Section 7.3 with optimized frequencies (our best).

cies; the original network with optimized frequencies; and our optimized topology from Section 7.3 with optimized frequencies. All the frequencies were set equal for the whole day, no individual optimizations for different times of day were made here. In terms of waiting time, the differences from the first case to the second is of 20 seconds and from the second to the third, 20 seconds again, which is not very significant. However from the first case to the second, the total distance traveled is reduced in half and from the second to the third cases the distance is reduced in $50\ 000$ Km.

## 7.5 Parameterization

The parameterization of the genetic algorithms was done with a smaller sample of trips. While the data used to run the optimization process, after parameter tuning, contained information relative to $566K$ trips, here we use a small sample of $51K$ trips. This translates to $45K$ OD pairs in the full dataset versus $15K$ OD pairs in this smaller sample. The differences in datasets should not affect the results as the nature of the problem remains the same. In terms of the path finding itself, the difference is not very noticeable, but the processing that is done after, to determine in-vehicle times, transfers and walking times, is heavily influenced as many more paths have to be processed after the path finding algorithm runs. This allowed us to save time in parameter tuning.

The way the hypervolume evolves with the different population sizes and mutation probabilities is depicted in Figure 7.13. Because of variability, multiple instances of the algorithm should be run with the same parameters, however, due to the time constraints we ran only one per parameter combination.

The best mutation probability was $10\%$. This seems to be a good balance between variability and

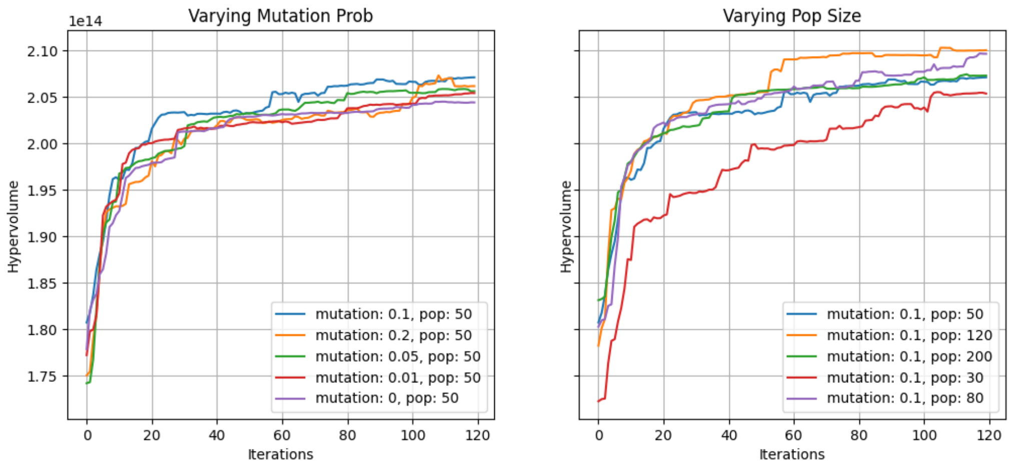


**Figure 7.13:** Hypervolume over NSGA-II iterations when varying population size and mutation probabilities.

avoiding early convergence. A mutation probability of $20\%$ introduces more variability and instability in the optimization process without consistently reaching the quality of solutions that a mutation probability of $10\%$ does. Having no mutations at all causes the algorithm to converge very early. Mutation probabilities of $5\%$ and $1\%$ where tested but performed worse under the hypervolume metric.

A population size of $120$ yielded the best results after $120$ iterations. However, a population size of $200$ presented a steadier progression of the hypervolume metric. As we wanted to run the algorithm with more iterations for the full dataset, we chose a population size of $200$ to avoid oscillations and to guarantee diversity within a population. Population sizes of $50$ and $80$ were also tested, yielding similar curves up to the $80$ iteration mark, after which the bigger population went in an upward trajectory and ended up achieving a similar hypervolume to that of the $120$ population size. A population size of $30$ is highly dependent on the initialization as it is not capable of carrying enough diversity to achieve better results through crossover. This was the worst performing population size.

We used the same population sizes and mutation probabilities for the Single Objective experiments we did. However, NSGA-II does not use crossover probability so we ran 5 Classic GA experiments with different crossover probabilities, a population size of $50$ and a mutation probability of $0.1$ for 120 iterations to assess the best crossover probabilities. The algorithm considered the weighted objective function. The average objective function values during the iterations can be seen in Figure 7.14. The crossover probabilities of $0.5$, $0.7$ and $0.9$ produced very similar results, producing considerably better results than a small crossover probability of $0.2$ and mildly worse results than the best performing $0.8$ probability. Variability is again a factor to consider, we ran the experiments only once per probability value, so the best result may not be as better as Figure 7.14 shows but we ran the longer experiments in Section 7.3 with a crossover probability of $0.8$.

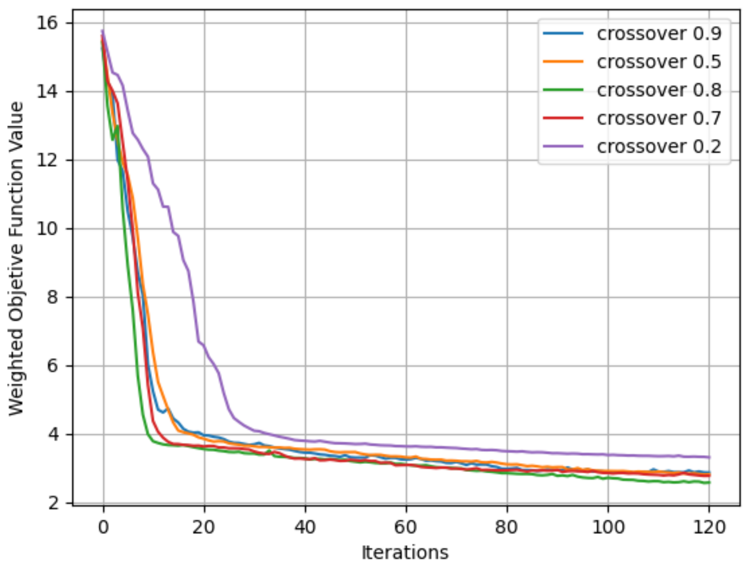


**Figure 7.14:** Average Weighted Objective Function over Classic GA iterations when varying crossover probabilities.

# 8

# Conclusion

**Contents**

In this work, we propose a system to redesign a bus network given the Origin-Destination demand in the city of Lisbon, Portugal. Two sub-problems are tackled, first the Transit Network Design Problem (TNDP) and then the Transit Network Frequencies Setting Problem (TNFSP). In the first problem, the system decides the set of routes that will define the bus network. In the second, bus frequencies are decided for each route in the route set. Both problems were modeled as an evolutionary problem and, firstly solved with a multi objective algorithm (NSGA-II). The results were then rated and weights for a single objective approach were inferred via a linear regression. Finally the problems were once more solved using a Classic Single Objective Genetic Algorithm. Overall, both problems managed to moderately improve the passenger related objectives while massively reducing operator related objectives.

In the TNDP, the four targeted objectives were the total length of the network, the unsatisfied demand, the in-vehicle time and the average transfers with restrictions on the number of routes. According to our criteria, the best network in the Pareto Front approximation reduces the original network objective function in $23.1\%$ and the network built through single objective optimization provides a $44.0\%$ reduction relative to the original network. This last reduction translates to a $34.6\%$ reduction in total network length (the network went from having 309 routes to having 200 routes), a $5.1\%$ reduction in average transfers, a slight reduction of in-vehicle time and an increase of unsatisfied demand from $0.7\%$ to $1.3\%$. The transition from Multi Objective to Single Objective optimization proved effective. With a Single Objective optimization considering the weights inferred through our rating process being able to produce better networks than the best one in the Pareto Front after 60 iterations, while a Single Objective optimization considering a uniform weighted sum of all objectives normalized achieved the same feat only after 200 iterations.

In the TNFSP, the two targeted objectives were the average waiting times and the total distance covered by buses during a whole day as a proxy for operator costs with constraints on the number of buses that can be in simultaneous circulation (fleet), the load factor of the buses and the frequencies. In the multi objective optimization, when trying to optimize frequencies for the original CARRIS topology, all the frequency sets improved on both objectives but once again, passenger costs were mildly reduced while operator costs were greatly reduced. In single objective optimization, we tried optimizing frequencies for the original CARRIS topology and for the best topology from the TNDP. The frequencies in the original topology provide an average decrease of 20 seconds in waiting time and a decrease of $55.4\%$ in total distance traveled while optimized frequencies in the optimized topology provide an average decrease of 40 seconds in waiting time and a decrease of $59.8\%$ in total distance traveled. Optimizing frequencies for different parts of the day as opposed to the whole day did not provide a significant increase because the separate optimization processes were not fine tuned to the different time frames.

## 8.1 Future Work

The optimization of frequencies for separate parts of the day can be further developed. Different parts of the day can consider different weight distributions, for example, in rush hours when there is a lot of people traveling to and from work, operator costs should not be the priority, but during the night, it is totally acceptable to prioritize operator costs as there are significantly less passengers and occasional services would be enough. Deciding the optimal time partitions in which traveling patterns are significantly different would also be an important data science contribution to this problem. It would also be relevant to test the use of exclusive routes in certain parts of the day to try to enable shorter trips under the presence of certain use patterns.

Building on this system, that already uses both buses and metro, it would be interesting to include more transportation systems, such as CP (Trains of Portugal) trains and GIRA bikes. Including more transports or even buses from other operators would allow us to widen the area under study and reveal opportunities or highlight the need to add some routes outside the current area of actuation. For instance, placing a lower limit on transfers in accordance with strategic goals.

Relocating the bus stops and then generating routes using the new stop locations could enable a much better network and allow easier transfers between different transportation modes as we could add proximity to other modes of transportation as an objective.

Running experiments with discrepant weights or particular constraints could help us study particular network features that help improve certain aspects of the network which could provide useful input for network design.

Other optimization algorithms should also be experimented with to see if we can find a better performing algorithm for our problem. For example, MOEA/D [27] could be tried in the multi objective setting and ACO in the single objective setting (similarly to Yu et al. [18]).

Finally, there should be reliable ways to go from these idealized models to the real world. Research on how to apply changes on existing networks from the knowledge attained from works similar to the present would be a step in the right direction.

## 8.2 Scientific Communication

The contributions of this thesis are submitted in a top tier scientific journal on Intelligent Transportation Systems.

# Bibliography

[1] J. Vuurstaek, G. Cich, L. Knapen, W. Ectors, T. Bellemans, D. Janssens *et al.*, “Gtfs bus stop mapping to the osm network,” *Future Generation Computer Systems*, vol. 110, pp. 393–406, 2018.

[2] R. H. Sofia Cerqueira, Elisabete Arsénio, “Integrative analysis of traffic and situational context data to support urban mobility planning,” *European Transport Conference (ETC), Milan, Italy*, 2020.

[3] I. Das and J. E. Dennis, “A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems,” *Structural optimization*, vol. 14, no. 1, pp. 63–69, 1997.

[4] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms*, 2nd ed. Luniver Press, 2008.

[5] G. F. Newell, “Some issues relating to the optimal design of bus routes,” *Transportation Science*, vol. 13, no. 1, pp. 20–35, 1979.

[6] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[7] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[8] F. Glover, “Future paths for integer programming and links to artificial intelligence,” *Computers operations research*, vol. 13, no. 5, pp. 533–549, 1986.

[9] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, 1st ed. MIT press, 1975.

[10] M. Dorigo, V. Maniezzo, and A. Colorni, “Ant system: optimization by a colony of cooperating agents,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[11] R. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” in *MHS’95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. IEEE, 1995, pp. 39–43.

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[13] R. Z. Farahani, E. Miandoabchi, W. Szeto, and H. Rashidi, “A review of urban transportation network design problems,” *European Journal of Operational Research*, vol. 229, no. 2, pp. 281–302, 2013.

[14] B. Beltran, S. Carrese, E. Cipriani, and M. Petrelli, “Transit network design with allocation of green vehicles: A genetic algorithm approach,” *Transportation Research Part C: Emerging Technologies*, vol. 17, no. 5, pp. 475–483, 2009.

[15] M. H. Baaj and H. S. Mahmassani, “An ai-based approach for transit route system planning and design,” *Journal of advanced transportation*, vol. 25, no. 2, pp. 187–209, 1991.

[16] Q. K. Wan and H. K. Lo, “A mixed integer formulation for multiple-route transit network design,” *Journal of Mathematical Modelling and Algorithms*, vol. 2, no. 4, pp. 299–308, 2003.

[17] G.-L. Jia, R.-G. Ma, and Z.-H. Hu, “Urban transit network properties evaluation and optimization based on complex network theory,” *Sustainability*, vol. 11, no. 7, p. 2007, 2019.

[18] B. Yu, Z. Yang, C. Cheng, and C. Liu, “Optimizing bus transit network with parallel ant colony algorithm,” *Proceedings of the Eastern Asia Society for Transportation Studies*, vol. 5, no. 1, pp. 374–389, 2005.

[19] S. Pattnaik, S. Mohan, and V. Tom, “Urban bus transit route network design using genetic algorithm,” *Journal of transportation engineering*, vol. 124, no. 4, pp. 368–375, 1998.

[20] M. H. Baaj and H. S. Mahmassani, “Trust: A lisp program for the analysis of transit route configurations,” *Transportation Research Record*, vol. 1283, pp. 125–135, 1990.

[21] S. Chien, Z. Yang, and E. Hou, “Genetic algorithm approach for transit route planning and design,” *Journal of transportation engineering*, vol. 127, no. 3, pp. 200–207, 2001.

[22] M. Bielli, M. Caramia, and P. Carotenuto, “Genetic algorithms in bus network optimization,” *Transportation Research Part C: Emerging Technologies*, vol. 10, no. 1, pp. 19–34, 2002.

[23] W. Fan and R. B. Machemehl, “Optimal transit route network design problem with variable transit demand: Genetic algorithm approach,” *Journal of Transportation Engineering-asce*, vol. 132, no. 1, pp. 40–51, 2006.

[24] H. M. Botee and E. Bonabeau, “Evolving ant colony optimization,” *Advances in complex systems*, vol. 1, no. 02n03, pp. 149–159, 1998.

[25] F. Zhao and A. Gan, “Optimization of transit network to minimize transfers,” Florida Department of Transportation, Center for Transportation Research, Florida International University, Tech. Rep. BD015-12, 2003.

[26] E. Miandoabchi, R. Z. Farahani, and W. Y. Szeto, “Bi-objective bimodal urban road network design using hybrid metaheuristics,” *Central European Journal of Operations Research*, vol. 20, no. 4, pp. 583–621, 2012.

[27] Q. Zhang and H. Li, “Moea/d: A multiobjective evolutionary algorithm based on decomposition,” *IEEE Transactions on evolutionary computation*, vol. 11, no. 6, pp. 712–731, 2007.