University of Amsterdam

MSc Mathematics
Track: Stochastics
MSc Econometrics
Track: Complexity and Economic Behaviour

Master Thesis

Optimizing Parking Capacities in Urban Areas

Author:
Tygo Nijsten

Supervisor:
dr. J.L. Dorsman
prof. dr. M.R.H. Mandjes
dr. M. Snelder
dr. ir. E.M.P. Walraven

Examination date:
August 25, 2023

Korteweg-de Vries Institute for Mathematics

Amsterdam School of Economics

TNO: Sustainable Urban Mobility & Safety

XCARCITY

Abstract

This thesis aims to optimize parking capacities in an urban area with three goals in mind: minimize fossil fuel emissions, minimize travel times, and minimize the space occupied by parking facilities. The method to achieve this goal can be separated into two components. Firstly, a model is defined to simulate traffic and parking in an urban area. Secondly, using this model, evolutionary algorithms are applied to find parking capacities with the aforementioned objectives.

Initial experiments involve applying the developed method to various small-scale scenarios. The results of these experiments reveal promising outcomes, as the proposed method positively contributes to achieving the three objectives.

Furthermore, we show that the method can be applied to a real-world urban area: the city of Delft in the Netherlands. The application to Delft not only shows the adaptability of the method to real-world urban scenarios but also highlights its potential to provide benefits to urban planning.

Title: Optimizing Parking Capacities in Urban Areas
Author: Tygo Nijsten, t.g.c.nijsten@uva.nl, 12366838
Supervisor: dr. J.L. Dorsman, prof. dr. M.R.H. Mandjes, dr. M. Snelder, dr. ir. E.M.P. Walraven
Second Examiner: dr. N.P.A. van Giersbergen, prof. dr. R. Núñez Queija
Examination date: August 25, 2023

Korteweg-de Vries Institute for Mathematics
University of Amsterdam
Science Park 105-107, 1098 XG Amsterdam
`http://kdvi.uva.nl`

Amsterdam School of Economics
University of Amsterdam
Roetersstraat 11, 1018 WB Amsterdam
`https://ase.uva.nl`

Sustainable Urban Mobility & Safety
TNO
Anna van Buerenplein 1, 2509 JE Den Haag
`https://www.tno.nl`

Statement of Originality

This document is written by Tygo Nijsten who declares to take full responsibility for the contents of this document. I declare that the text and the work presented in this document is original and that no sources other than those mentioned in the text and its references have been used in creating it. The Faculty of Economics and Business, Faculty of Science, and TNO are responsible solely for the supervision of completion of the work, not for the contents.

Contents

1. Introduction

1.1. Research Motivation

According to a study conducted by CBS (2021), the most used mode of travel in the Netherlands is the car, accounting for 43 percent of all trips. The utilization of cars goes paired with fossil fuel emissions, damaging the environment. On top of this, the large number of cars on the roads causes a lot of congestion. Research conducted in the middle of 2023 by ANWB Verkeersinformatie (2023) showed that traffic congestion in the Netherlands in the first half of 2023 has increased by 15 percent compared to the first half of 2019.

Parking plays a crucial role when it comes to the subject of cars. As claimed by Bonsall and Palmer (2004), up to 40 percent of the total travel time when traveling to central urban areas, is used to find a parking space. Also, parking facilities require a lot of land. According to Kuys (2022), in the Netherlands, more space is occupied by parking facilities than by housing.

That is why it is important to consider how to optimize parking in urban areas. A method to achieve this is to optimize the number of parking spaces offered at every parking facility. In this thesis, we focus on optimizing parking capacities with the following objectives in mind.

- **Minimize emissions:** The usage of cars to travel cause a lot of greenhouse gas emissions and air pollution in urban areas. By optimizing parking capacities, we aim to minimize emissions.
- **Minimize travel times**: Traffic congestion and longer travel times are a big problem for car users. It is important for an urban area to still be accessible. By optimizing parking capacities, we try to reduce overall traffic congestion and thus, decrease overall travel times.
- **Efficient land utilization**: The limited availability of land in urban areas asks for efficient use to meet all the needs of inhabitants. However, parking facilities often make up large amounts of valuable urban space. By optimizing parking capacities, we aim to make more land available for other purposes.

Simultaneously addressing these goals can be difficult. For example, if the number of parking spots is reduced, finding an available spot can take significantly longer. So while we may limit the space parking facilities utilize, travel times will increase. These kinds of trade-offs make this such a complex problem to solve.

Hence, our aim is to minimize some weighted average of these three variables. The weights depend on the importance assigned to each objective. This importance is for example chosen by a policymaker in an urban area.

1.2. Brief Overview of Past Literature

Several papers have conducted research on the optimization of parking with different objectives in mind. We divide this research into three categories: the maximization of parking capacities in a parking facility, optimizing parking locations, and optimizing parking prices.

- **Maximizing parking capacities:** Abdelfatah and Taha (2014) presented a method to determine the optimal parking angle to maximize parking capacities in a parking facility. However, in this case, the goal is to maximize the number of parking spaces in a certain parking facility by looking at the design. Our goal is not to maximize parking capacities in a parking facility, but our goal is to choose optimal parking capacities for different parking facilities with the three objectives mentioned before. So although this is in the literature referred to as the optimization of parking capacities, it is fundamentally different research.

- **Optimizing parking locations:** Chen et al. (2001) proposed a method to choose parking locations in a city to minimize the total walking distance from a parking space to a destination. Austin and Lee (1973) considered parking cost and travel distance on top of walking time as factors in their method of optimizing parking locations. Shen, Hua, and Liu (2019) presented a model to minimize emissions by optimizing parking locations. Ruan et al. (2016) considered two cases of parking capacities in their model to optimize parking locations with the objective of maximizing accessibility.

- **Optimizing parking prices:** D'Acierno, Gallo, and Montella (2006) introduced a model to set parking fees with the goal of increasing the use of other modes of transport than the car. Pierce, Willson, and Shoup (2015) designed a method to choose parking fees such that the occupancy of parking facilities is high, but still enough empty parking spaces are available at all times.

However, all these papers do not address the optimization of parking capacities as proposed. By considering parking capacities, parking locations are also implicitly included. In this research, we aim to come up with a model to optimize parking capacities with the three objectives previously mentioned. One of these objectives is minimizing the space used by parking facilities. This objective has not been considered yet in the studies we have discussed.

1.3. Research Questions

This thesis aims to develop a method to optimize parking capacities to minimize a weighted average of emissions, travel time, and the use of land for parking, which leads to the following research question:

> **Main Research Question.** How can we optimize parking capacities to minimize a weighted average of emissions, travel time, and land used for parking?

To answer this research question, we introduce two sub-questions. We present these questions and discuss how these questions contribute to the main research question.

> **Sub-question 1.** How can we simulate traffic and parking in an urban area?

In principle, there are two approaches to help answer the research question.

Firstly, we could use data about emissions, travel times, and the land occupied by parking facilities and see what implications certain changes in parking capacities have on these variables. By examining this data, we can determine the parking capacities that result in the smallest weighted average of emissions, travel times, and utilization of land for parking space. However, due to the unavailability of such data and the high cost and low efficiency of implementing parking capacity changes in real urban areas for experimentation, an alternative approach is necessary.

A second approach is to simulate traffic and parking in an urban area. Through this simulation, we can analyze the effects of different parking capacities on emissions, travel times, and land used for parking. By evaluating the simulation outputs, we can select the parking capacities that minimize the weighted average of emissions, travel times, and land occupied by parking. This approach provides a cost-effective and efficient method of analyzing the implications of changing parking capacities in an urban area.

> **Sub-question 2.** Which techniques can be used to optimize parking capacities and how can these methods be applied exactly?

When we can simulate traffic and parking, selecting the parking capacities that minimize this weighted average of emissions, travel times, and the usage of land for parking is not that straightforward. Two problems make this question difficult to answer. Firstly, it will turn out that this weighted average is not just some smooth function. Secondly, the solution space of this optimization problem is very large. It is therefore not guaranteed that we find a method that finds an exact optimum.

1.4. Outline

Theoretical background about simulating traffic is introduced in Chapter 2. A mathematical analysis is given, while also a numerical approach is shown. This chapter helps to answer sub-question 1.

Chapter 3 presents theoretical background about optimization techniques. In this chapter, we describe existing techniques in general, without applying them to traffic and parking yet. This chapter helps us provide an answer to sub-question 2.

In Chapter 4, the methodology is presented. In Section 4.1 the addition of parking to the simulation of traffic is addressed and an answer is given to sub-question 1. Applying optimization techniques to the traffic and parking simulation to find optimal parking capacities is investigated in Section 4.2, which answers sub-question 2.

Finally, in Chapter 5 we apply our model of optimizing parking capacities on some examples. We start with small examples, which allows us to make a complete analysis of the results. We also show that our model can be used in practice by applying it to the city of Delft in the Netherlands.

1.5. Acknowledgements

Firstly I want to thank my supervisors Jan-Pieter Dorsman, Michel Mandjes, Maaike Snelder, and Erwin Walraven for providing me their help whenever I needed it and for their valuable feedback. Furthermore, I want to show my gratitude to all my colleagues at TNO for welcoming me into the department and for their assistance throughout my internship. Lastly, I am grateful to my family and friends for supporting me during this thesis. A special thanks goes to Sara el Ela-Morcy, Sliem el Ela, Saaly Alsaadi, Sanae Azzouzi, Ali Moin, Ince Wright, and Floor Clarisse for accompanying me on many days while I was working on this thesis.

2. Road Traffic Models

We start by looking at basic traffic models. In this chapter, we define basic road traffic networks and how drivers behave in these road networks. In principle, drivers try to minimize their travel times, and under certain circumstances, it will turn out that this leads to an equilibrium. The background information in this chapter is mainly based on Chapter 4 of Kelly and Yudovina (2014) and Chapter 3 to 5 of Sheffi (1985).

2.1. Defining a Road Network

Formally, a road network can be represented by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{J})$, where $\mathcal{V}$ are the vertices and where $\mathcal{J}$ are the directed edges of the graph (roads can be two-way, by introducing two links for both directions). The (weighted) edges represent the roads in the network, while the vertices represent the intersections between these roads. The weight of an edge can be seen as the cost for a driver when choosing that road. A natural choice would be to take the length of the road as cost. An example of such a graph is given in Figure 2.1.

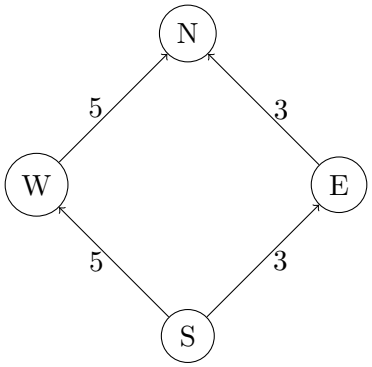

Figure 2.1.: Example of a basic traffic network

Suppose a driver would want to travel from the south (S) to the north (N). They would choose to travel through the east (E), as this would cost them 6 units while traveling through the west (W) would cost them 10 units. In this network, they would always choose the path with the lowest cost. To find such a shortest path, we can use Dijkstra's shortest path algorithm, which is described in Section 7.20 of Miller and Ranum (2011).

The problem with this basic model is that it does not take into account how many drivers use a road, which we call the *flow*. In practice, when too many drivers use a road at the same time, a negative externality arises, namely traffic congestion. In urban

areas, as roads are scarce, this is a reoccurring problem. In the example of Figure 2.1, every driver in the network would use the road going from the south (S) to the east (E) and the road going from the east (E) to the north (N) to minimize their travel time. In practice, this will most likely cause these roads to become congested, which would increase travel times considerably. Therefore, it may be more attractive for some drivers to avoid this by choosing to drive through the west (W) instead. As a result, when determining which road to choose, it is important to take flows on roads into account.

Let y denote the flow on a road. An example that considers the flows on the roads is given in Figure 2.2. Again, assume drivers want to go from the south (S) to the north (N).

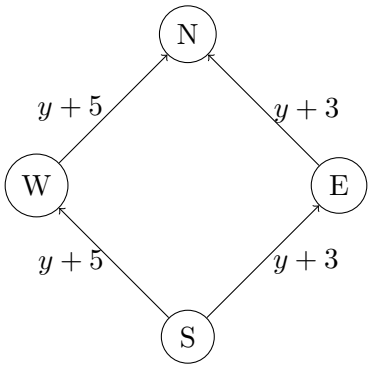

Figure 2.2.: Example of a basic traffic network with flows

In an empty network, where all flows are equal to 0, a driver would still choose to travel through the east (E). However, suppose 10 cars want to travel from the south (S) to the north (N). If all cars choose to travel through the east (E), the cost of this route would be 26 units. In contrast, traveling through the west (W) would still cost 10 units. So now traveling through the west (W) is more attractive. We say this network is in equilibrium when no driver has the incentive to choose an alternative route. In this example, that would be when six drivers would choose to travel through the east (E) and four drivers would choose to travel through the west (W). This way, both routes would have a cost of 18 units. In the next section, we will formalize this equilibrium, which is known as the *Wardrop equilibrium*.

2.2. Wardrop Equilibrium

To introduce the notion of the Wardrop equilibrium, we need to discuss the concept of routes and flows on routes. In addition to the set of (directed) edges $\mathcal{J}$, we define the set of possible routes $\mathcal{R} \subset 2^{\mathcal{J}}$; each route is a subset of edges. We define the link-route incidence matrix A so that $A_{jr} = 1$ if $j \in r$ for some link j and route r, and $A_{jr} = 0$ otherwise.

Let x_r denote the flow on route r. Then let $x = (x_r, r \in \mathcal{R})$ represent the vector of

flows on the routes. The flow on a single edge j is then given by

$$y_j = \sum_{r \in \mathcal{R}} A_{jr} x_r, \quad j \in \mathcal{J},$$

or equivalently: $y = Ax$. Here $y = (y_j, j \in \mathcal{J})$ represents the vector of flows on the roads.

The cost or delay that a driver experiences on a single edge j is given by a function $D_j(y_j)$. We assume that this function is continuously differentiable and increasing. To compute the cost of a route, we simply take the sum of the delays of the edges in this route.

We assume that drivers only want to travel from A to B in the shortest possible time, but do not have any other preferences for the route they take. Let $\mathcal{S} \subset \mathcal{J} \times \mathcal{J}$ be the set of OD pairs (origin-destination pairs). For every OD pair, there is a set of routes that serve it. So for the examples in Figures 2.1 and 2.2 we only had (S, N) as an OD pair. For this OD pair, we considered two routes to serve it: one route through the west (W) and the other route through the east (E). In general, we write H to denote the incidence matrix where $H_{sr} = 1$ if the OD pair s is served by route r, and $H_{sr} = 0$ otherwise. Note that we do not have to include every possible route for an OD pair, we could also choose to omit some possible routes. We write $s(r)$ for the OD pair corresponding to route r. The flow f_s on an OD pair is then given by

$$f_s = \sum_{r \in \mathcal{R}} H_{sr} x_r, \quad s \in \mathcal{S},$$

or equivalently: $f = Hx$. Here $f = (f_s, s \in \mathcal{S})$ denotes the vector of flows on an OD pair.

2.2.1. User-equilibrium Formulation

In equilibrium, no driver has the incentive to deviate from the route they are taking on a certain OD pair. Formally, this means:

$$x_r > 0 \implies \sum_{j \in \mathcal{J}} D_j(y_j) A_{jr} \leq \sum_{j \in \mathcal{J}} D_j(y_j) A_{jr'}, \quad \forall r' \in s(r).$$

In words, this states that a flow on a particular route r can be non-zero, only when the delay experienced on this road is less than or equal to the delay experienced on any other route with the same OD pair. There are two cases: either the flow for a route is zero or the cost is equal to the minimum cost of the corresponding OD pair. This equilibrium is known as the Wardrop equilibrium:

Definition 2.3. *A* Wardrop equilibrium *is a vector of flows on the routes,* $x = (x_r, r \in \mathcal{R})$ *such that*

$$x_r > 0 \implies \sum_{j \in \mathcal{J}} D_j(y_j) A_{jr} = \min_{r' \in s(r)} \sum_{j \in \mathcal{J}} D_j(y_j) A_{jr'},$$

where $y = Ax$.

We will now show that such a Wardrop equilibrium exists, given the assumptions we made.

Theorem 2.4. *Suppose that there is at least one path between every OD pair specified in some set of OD pairs $\mathcal{S}$. Also, assume that the delay functions $D_j(y_j)$ are continuously differentiable and increasing for every link $j \in \mathcal{J}$. Then there exists a Wardrop equilibrium.*

Proof. We consider the following minimization problem:

$$\begin{aligned} &\text{minimize} && \sum_{j\in\mathcal{J}} \int_0^{y_j} D_j(u)\,du \\ &\text{subject to} && Hx = f, Ax = y \\ &\text{over} && x \geq 0, y. \end{aligned}$$

By definition, as $Ax = y$ and as A only contains 0 or 1, we know that $y \geq 0$ as well. The region we minimize over is convex and compact. As we defined D_j to be continuously differentiable and increasing, the objective function is differentiable and convex. Therefore, the first-order conditions for a minimum are necessary and sufficient. To find these first-order conditions for a minimum, the Lagrange multiplier method can be applied. The Lagrangian is given by

$$L(x, y; \lambda, \mu) = \sum_{j\in\mathcal{J}} \int_0^{y_j} D_j(u)\,du + \lambda \cdot (f - Hx) - \mu(y - Ax).$$

To solve this, we differentiate with respect to x and with respect to y:

$$\begin{aligned} \frac{\partial L(x, y; \lambda, \mu)}{\partial y_j} &= D_j(y_j) - \mu_j, \\ \frac{\partial L(x, y; \lambda, \mu)}{\partial x_r} &= -\lambda_{s(r)} + \sum_{j\in\mathcal{J}} \mu_j A_{jr}. \end{aligned}$$

As we minimize over $y_j \in \mathbb{R}$, we know that the minimum is attained when the derivative with respect to y_j is equal to 0. So we obtain

$$D_j(y_j) - \mu_j = 0 \iff \mu_j = D_j(y_j).$$

As we minimize over $x_r \geq 0$, we know the minimum is attained for $x_r = 0$ when the derivative is nonnegative and for $x_r > 0$ when the derivative with respect to x_r is equal to 0. So we obtain

$$\begin{cases} -\lambda_{s(r)} + \sum_{j\in\mathcal{J}} \mu_j A_{jr} = 0, & x_r > 0, \\ -\lambda_{s(r)} + \sum_{j\in\mathcal{J}} \mu_j A_{jr} \geq 0, & x_r = 0, \end{cases} \iff \begin{cases} \lambda_{s(r)} = \sum_{j\in\mathcal{J}} \mu_j A_{jr}, & x_r > 0, \\ \lambda_{s(r)} \leq \sum_{j\in\mathcal{J}} \mu_j A_{jr} & x_r = 0. \end{cases}$$

We can interpret $\lambda_{(s(r)}$ as the minimal cost corresponding to the OD pair $s(r)$. In conclusion, solutions to this minimization, are the solutions of the Wardrop equilibria. □

If we would assume that the delay functions D_j are strictly increasing, the objective function would become strictly convex. We know if an optimum exists for a strictly convex function over a convex set, that it is unique. As existence is proven by Theorem 2.4, in this case, the Wardrop equilibrium would be unique.

Connection with Nash Equilibrium

There is a clear connection between the Wardrop equilibrium in traffic assignment and the *Nash equilibrium* in game theory. The Nash equilibrium is the solution to a non-cooperative game with multiple players. The players are assumed to know the equilibrium strategies of other players, which is minimizing travel time in the case of traffic. The Nash equilibrium is then defined as the solution in which players do not have the incentive to change their strategy, which is their route in the case of traffic. In principle, this definition is equivalent to the definition of the Wardrop equilibrium in traffic assignment. However, in the case of traffic assignment, there are many players, which makes it more complex. The Wardrop equilibrium is therefore also referred to as the Nash equilibrium for games with a continuum of players. A more extensive analysis of the relationship between these two equilibria is provided by Haurie and Marcotte (1985).

2.2.2. Braess's Paradox

As we stated in Chapter 1, two variables we would want to minimize are total delay and total distance. It would be logical to assume that simply adding roads would decrease the total delay. However, this is not necessarily the case.

We know from game theory that the Nash equilibrium is often not the *social optimum*. Similarly, the Wardrop equilibrium is often not the social optimum. This is because drivers only want to minimize their own travel time, which does not mean that the overall travel time is minimized. In the social optimum, the sum of travel times is minimized. In this situation, there is often a possibility for drivers to decrease their individual travel times by deviating from their chosen route, resulting in a deviation from the social optimum under the assumptions we made.

To illustrate this, we analyze the examples in Figure 2.5. Suppose again, that 10 drivers want to travel from the south (S) to the north (N). In the first example, the network structure is the same as in Figure 2.2, a driver could choose to travel through the west (W) or the east (E). However, in the second example, a road is added between the west (W) and the east (E). This way, a new route arises.

In red, the flows are shown for which the networks are in a Waldrop equilibrium. In the first example, the resulting total travel times for both routes are equal to 60. In the second example, the resulting total costs for the three routes are equal to 64. The total delay in the equilibrium has indeed increased by adding a road. Clearly, the Wardrop equilibrium in the second example is not the social optimum, as the same flows as in the first example would achieve a lower total travel time. This phenomenon is called *Braess's paradox*.

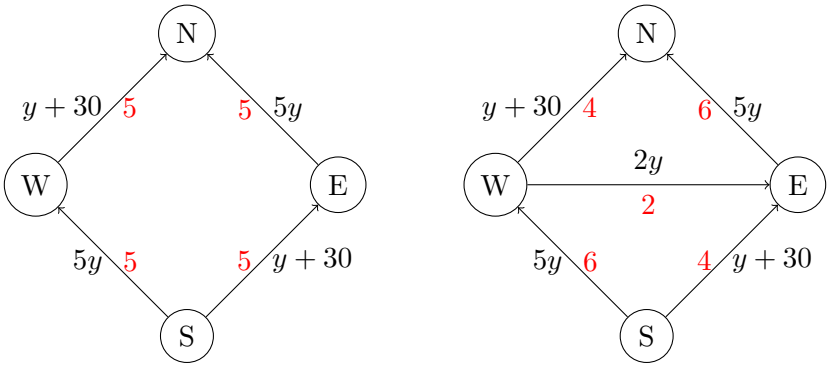

Figure 2.5.: An example showing Braess's paradox

2.2.3. System-optimization Formulation

In practice, drivers always tend to minimize their own costs. It is interesting to analyze if there is a method to let the Wardrop equilibrium align with the social optimum. In particular, is it possible, for example by enforcing some policy, to force the Wardrop equilibrium to be the same as the social optimum, or at least to force the Wardrop equilibrium to get closer to the social optimum?

First, let us rewrite the minimization problem such that the solution aligns with the social optimum. To achieve this, we can use the cumulative total delay $\sum_{j\in\mathcal{J}} y_j D_j(u)$ as our objective function:

$$\begin{aligned} &\text{minimize} && \sum_{j\in\mathcal{J}} y_j D_j(u) \\ &\text{subject to} && Hx = f, Ax = y \\ &\text{over} && x \geq 0, y. \end{aligned}$$

This problem can again be solved in the same fashion as Theorem 2.4. We define the Lagrangian

$$L(x, y; \lambda, \mu) = \sum_{j\in\mathcal{J}} \sum_{j\in\mathcal{J}} y_j D_j(u) + \lambda \cdot (f - Hx) - \mu(y - Ax).$$

To solve this, we differentiate with respect to x and with respect to y:

$$\begin{aligned} \frac{\partial L(x, y; \lambda, \mu)}{\partial y_j} &= D_j(y_j) + y_j D_j'(y_j) - \mu_j, \\ \frac{\partial L(x, y; \lambda, \mu)}{\partial x_r} &= -\lambda_{s(r)} + \sum_{j\in\mathcal{J}} \mu_j A_{jr}. \end{aligned}$$

We again obtain

$$\begin{cases} \lambda_{s(r)} = \sum_{j\in\mathcal{J}} \mu_j A_{jr}, & x_r > 0, \\ \lambda_{s(r)} \leq \sum_{j\in\mathcal{J}} \mu_j A_{jr} & x_r = 0. \end{cases}$$

However, for μ_j we obtain

$$\mu_j = D_j(y_j) + y_j D_j'(y_j).$$

This suggests that by adding some toll $T_j(y_j) = y_j D_j'(y_j)$ on the roads, we could encourage drivers to move to the social optimum. For example, for the edge from the south (S) to the west (W) in Figure 2.5, $D_j(y_j) = 5y_j$, and thus the toll would be $T_j(y_j) = y_j D_j'(y_j) = 5y_j$. Note that this toll depends on the flow, so in reality, this toll would have to be dynamic, depending on how many drivers are on the road. This may be a bit impractical to implement, so an alternative would be to compute the flows in the Wardrop equilibrium at rush hour and use these flows.

2.3. Numerical Methods

It is not always feasible to find the Wardrop equilibrium, for example when the network gets too large and too complex. When this is the case, we can use numerical methods to approximate the Wardrop equilibrium.

2.3.1. Naive Method

A naive approach would be to first start with an all-or-nothing assignment in an empty network (flows equal to 0). In the *all-or-nothing assignment*, all drivers choose the shortest path without considering congestion effects, so without taking into account the change of flows such that delays are static. We have seen earlier that in that case, Dijkstra's shortest path algorithm can be applied. Then based on the flows after this assignment, delays can be updated. Consequently, a new all-or-nothing assignment can be assigned. We can iterate this until there is convergence. In this case, by convergence, we mean that the maximal difference in flows between iterations becomes small, let's say smaller than some $\kappa > 0$. Now, let D_j^n denote the delay on edge j and y_j^n denote the flow on edge j in the n-th iteration. Then the mathematical formulation of this naive approach is given by Algorithm 2.6.

1 Set $D_j^0 := D_j(0)$ for all $j \in \mathcal{J}$.
2 Obtain $\{y_j^0\}$ by all-or-nothing assignment and set iteration counter $n := 1$.
3 Set $D_j^n := D_j(y_j^{n-1})$ for all $j \in \mathcal{J}$.
4 Obtain $\{y_j^n\}$ by all-or-nothing assignment.
5 **if** $\max_j\{|y_j^n - y_j^{n-1}|\} \leq \kappa$ **then**
6 | Terminate
7 **else**
8 | Set $n := n + 1$ and return to step 3.
9 **end**

Algorithm 2.6: Naive algorithm to approximate the Wardrop equilibrium

The problem with this algorithm is that it does not necessarily converge. If we again consider the example in Figure 2.2, this algorithm will continue to give full flow to one of the two routes alternatingly. A remedy could be to stop the algorithm after N iterations and take the final flow assignment as the average of the last $n \leq N$ flow assignments. However, it is useful to consider an algorithm that provides convergence.

2.3.2. Frank-Wolfe algorithm

An example of a method that is guaranteed to provide convergence is the Frank-Wolfe algorithm, which was introduced in Frank and Wolfe (1956). First, we look at the Frank-Wolfe algorithm in general, and then we will specify how to use it for finding or approximating a Wardrop equilibrium. Let f be a convex and continuously differentiable function, and let $\mathcal{D}$ be a compact convex set. Additionally, let the gradient of f, ∇f be L-Lipschitz continuous. Suppose we want to solve the minimization problem

$$\min_{y \in \mathcal{D}} f(y).$$

Then the Frank-Wolfe algorithm is given by Algorithm 2.7.

1 Let $y^0 \in \mathcal{D}$ be a feasible solution and set $n := 0$.
2 Obtain $z^n = \arg\min_{z \in \mathcal{D}} \nabla f(y^n)^T z$
3 Set $y^{n+1} = y^n + \alpha_n(z^n - y^n)$ for $\alpha_n = \frac{2}{n+2}$.
4 **if** $\max_j\{|y_j^n - y_j^{n-1}|\} \leq \kappa$ **then**
5 | Terminate
6 **else**
7 | Let $n := n + 1$ and return to step 2
8 **end**

Algorithm 2.7: Frank-Wolfe algorithm

We continue by providing a proof of the convergence of the Frank-Wolfe algorithm, which is mainly based on Jaggi (2013). We start by defining the *curvature constant* C_f of a convex and continuously differentiable function f (on a set $\mathcal{D}$):

$$C_f := \sup_{\substack{y,z \in \mathcal{D} \\ \alpha \in [0,1] \\ w = y + \alpha(z-y)}} \frac{2}{\alpha^2} \left(f(w) - f(y) - \langle w - y, \nabla f(y) \rangle \right).$$

Given our assumptions that f is continuously differentiable and has L-Lipschitz continuous gradient we can apply Theorem A.3 to conclude that C_f is in this case positive and bounded.

To help us prove convergence, we continue by defining a function g:

$$g(y) = \max_{z \in \mathcal{D}} \nabla f(y)^T (y - z).$$

Note that this maximum is achieved for $z = \arg\min_{z \in \mathcal{D}} \nabla f(y)^T z$. The following lemma then presents the main inequality we need to show the convergence of the Frank-Wolfe algorithm.

Lemma 2.8. *Let f be a convex and continuously differentiable function defined on a compact convex set $\mathcal{D}$. Additionally, let ∇f be L-Lipschitz continuous and let y^n, z^n and α_n be as defined in Algorithm 2.7. Lastly, let C_f be the curvature constant of f. Then*

$$f(y^{n+1}) \leq f(y^n) - \alpha_n g(y^n) + \frac{\alpha_n^2}{2} C_f.$$

Proof. We see that

$$\begin{aligned}
f(y^{n+1}) &\overset{(1)}{=} f(y^n + \alpha_n(z^n - y^n)) \\
&= f(y^n) + \langle \alpha_n(z^n - y^n), \nabla f(y^n) \rangle \\
&\quad + \frac{\alpha_n^2}{2} \left(\frac{2}{\alpha_n^2} \left(f(y^n + \alpha_n(z^n - y^n)) - f(y^n) - \langle \alpha_n(z^n - y^n), \nabla f(y^n) \rangle \right) \right) \\
&\overset{(2)}{\leq} f(y^n) + \langle \alpha_n(z^n - y^n), \nabla f(y^n) \rangle + \frac{\alpha_n^2}{2} C_f \\
&= f(y^n) - \alpha_n \langle (y^n - z^n), \nabla f(y^n) \rangle + \frac{\alpha_n^2}{2} C_f \\
&\overset{(3)}{=} f(y^n) - \alpha_n g(y^n) + \frac{\alpha_n^2}{2} C_f,
\end{aligned}$$

where (1) follows from the definition of y^n, (2) follows from the definition of C_f, and (3) follows from the definition of z^n and the maximality of g. □

In the next theorem, we formalize the convergence of the Frank-Wolfe algorithm.

Theorem 2.9. *Let f be a convex and continuously differentiable function. Additionally, let ∇f be L-Lipschitz continuous and let y^n, z^n and α_n be as defined in Algorithm 2.7. Lastly, let C_f be the curvature constant of f. Then for $n \geq 1$, we have:*

$$f(y^n) - f^* := f(y^n) - \min_{y \in \mathcal{D}} f(y) \leq \frac{2C_f}{n+2}.$$

Proof. We find

$$\begin{aligned}
f(y^{n+1}) - f^* &\overset{(1)}{\leq} f(y^n) - f^* - \alpha_n g(y^n) + \frac{\alpha_n^2}{2} C_f \\
&\overset{(2)}{\leq} f(y^n) - f^* - \alpha_n (f(y^n) - f^*)) + \frac{\alpha_n^2}{2} C_f \\
&= (1 - \alpha_n)(f(y^n) - f^*) + \frac{\alpha_n^2}{2} C_f,
\end{aligned}$$

where (1) follows from Lemma 2.8, and (2) follows from the maximality of g. Now to find the desired upper bound, we use induction. We start with the base case. Note that $\alpha_0 = 1$ and so that the above tells us $f(y^1) - f^* \leq \frac{1}{2}C_f \leq \frac{2}{3}C_f$, as $C_f \geq 0$. Furthermore, when we fill in the value of α_n, we see that

$$\begin{aligned}
f(y^{n+1}) - f^* &\leq (1-\alpha_n)(f(y^n) - f^*) + \frac{\alpha_n^2}{2}C_f \\
&= \left(1 - \frac{2}{n+2}\right)(f(y^n) - f^*) + \frac{4}{2(n+2)^2}C_f \\
&\overset{(3)}{\leq} \left(1 - \frac{2}{n+2}\right)\frac{2C_f}{n+2} + \frac{4}{2(n+2)^2}C_f \\
&= \frac{2C_f}{n+2}\left(1 - \frac{1}{n+2}\right) \\
&= \frac{2C_f}{n+2}\frac{n+1}{n+2} \\
&\leq \frac{2C_f}{n+2}\frac{n+2}{n+3} \\
&= \frac{2C_f}{(n+1)+2},
\end{aligned}$$

where (3) follows from the induction hypothesis. □

Indeed, when C_f is bounded, we see that $f(y^n) - f^* \to 0$ as $n \to \infty$, and thus, the Frank-Wolfe algorithm provides convergence with rate $\mathcal{O}\left(\frac{1}{n}\right)$.

We now have the rather strong assumption that the gradient, $\nabla f(y)$, is L-Lipschitz continuous. However, it is also enough to consider the weaker assumption that C_f is finite. We could even argue that C_f can be anything sublinear i.e. $C_f = \mathcal{O}(n^\beta)$ for $\beta \in [0, 1)$ would be sufficient.

Choice of Step Size α_n

The step size we have chosen in step 3 of Algorithm 2.7, $\alpha_n = \frac{2}{n+2}$, seems quite arbitrary. The idea is that when we use a decreasing step size, the solution moves less and less in the direction of the linearization minimizer as the algorithm continues. Furthermore, a step size should always be between 0 and 1, otherwise, it could happen that the new solution, y^{n+1}, ends up outside of the solution space. Obviously, there are a lot more possible step sizes with the same properties though.

An entirely different type of choice for the step size is

$$\alpha_n = \arg\min_{\alpha_n \in [0,1]} f(y^n + \alpha_n(z^n - y^n)).$$

The Frank-Wolfe algorithm with this step size is also known as the *line search method*, which is described in Chapter 3 of Nocedal and Wright (2006). However, it turns out that this also provides convergence of rate $\mathcal{O}\left(\frac{1}{n}\right)$.

Canon and Cullum (1968) have actually proven that the rate of convergence of the Frank-Wolfe algorithm is no better than $\mathcal{O}\left(\frac{1}{n}\right)$ no matter which α_n is chosen. Theorem 2.9 shows that the choice $\alpha_n = \frac{2}{n+2}$ provides a convergence of rate $\mathcal{O}\left(\frac{1}{n}\right)$. Thus, as we can do no better, it is not needed to consider other step sizes.

Translation to Traffic

Now we move on to translating the Frank-Wolfe algorithm to be applied to traffic assignment. Remember that to obtain the Wardrop equilibrium, the function we want to minimize is given by

$$f(x) = \sum_{j \in \mathcal{J}} \int_0^{y_j} D_j(u)\, du.$$

Also, remember that the region we minimize over is convex and compact. Then to apply Frank-Wolfe, we first compute the gradient

$$\begin{aligned} \nabla f(x) &= \nabla \left(\sum_{j \in \mathcal{J}} \int_0^{y_j} D_j(u)\, du \right) \\ &= (D_1(y_1), \ldots, D_J(y_J)), \end{aligned}$$

where $J = |\mathcal{J}|$. Then the second rule of Algorithm 2.7 translates to

$$z^n = \arg\min_{z \in \mathcal{D}} \sum_{j \in \mathcal{J}} D_j(y_j) z_j.$$

This z^n corresponds to the flow that minimizes the total delay of the network. Note that weights are not dependent on z, so the minimizer is the all-or-nothing assignment. In conclusion, we can apply the Frank-Wolfe algorithm as given by Algorithm 2.10 to approximate the Wardrop equilibrium. To obtain convergence of the algorithm, we have to assume that $\nabla f(x) = (D_1(y_1), \ldots, D_J(y_J))$ is L-Lipschitz continuous.

1 Set $D_j^0 := D_j(0)$ for all $j \in \mathcal{J}$.
2 Obtain $\{y_j^0\}$ by all-or-nothing assignment and set iteration counter $n := 0$.
3 Set $D_j^n := D_j(y_j^n)$ for all $j \in \mathcal{J}$.
4 Obtain (auxiliary) $\{z_j^n\}$ by all-or-nothing assignment.
5 Set $y_j^{n+1} = y_j^n + \alpha_n(z_j^n - y_j^n)$ for $\alpha_n = \frac{2}{n+2}$.
6 **if** $\max_j\{|y_j^n - y_j^{n-1}|\} \leq \kappa$ **then**
7 | Terminate
8 **else**
9 | Let $n := n + 1$ and return to step 3
10 **end**

Algorithm 2.10: Frank-Wolfe algorithm to find Wardrop equilibrium

In Table 2.11, the Frank-Wolfe algorithm applied to the left network of Figure 2.5 is shown. Note that the all-or-nothing flows z_j^n also correspond to the naive method. Indeed, in this case, the naive algorithm does not converge as it just assigns a full flow of 10 to either the route $S \to W \to N$ or the route $S \to E \to N$. However, the flows y_j^n corresponding to the Frank-Wolfe algorithm indeed converge to the Wardrop equilibrium in which the flow on every edge is equal to 5. Obviously, in the cases of Figure 2.5, it is easy to compute the Wardrop equilibrium by hand, but in cases where the network is much larger and much more complex, the Frank-Wolfe algorithm is a necessity.

n	SW	SE	WN	EN	Step Size
0	$y_{SW}^0 = 10$	$y_{SE}^0 = 0$	$y_{WN}^0 = 10$	$y_{EN}^0 = 0$	
1	$z_{SW}^0 = 0$ $y_{SW}^1 = 0$	$z_{SE}^0 = 10$ $y_{SE}^1 = 10$	$z_{WN}^0 = 0$ $y_{WN}^1 = 0$	$z_{EN}^0 = 10$ $y_{EN}^1 = 10$	$\alpha_0 = 1$
2	$z_{SW}^1 = 10$ $y_{SW}^2 \approx 6.667$	$z_{SE}^1 = 0$ $y_{SE}^2 \approx 3.333$	$z_{WN}^1 = 10$ $y_{WN}^2 \approx 6.667$	$z_{EN}^1 = 0$ $y_{EN}^2 \approx 3.333$	$\alpha_1 \approx 0.667$
3	$z_{SW}^2 = 0$ $y_{SW}^3 \approx 3.333$	$z_{SE}^2 = 10$ $y_{SE}^3 \approx 6.667$	$z_{WN}^2 = 0$ $y_{WN}^3 \approx 3.33$	$z_{EN}^2 = 10$ $y_{EN}^3 \approx 6.667$	$\alpha_2 = 0.5$
4	$z_{SW}^3 = 10$ $y_{SW}^4 \approx 6$	$z_{SE}^3 = 0$ $y_{SE}^4 \approx 4$	$z_{WN}^3 = 10$ $y_{WN}^4 \approx 6$	$z_{EN}^3 = 0$ $y_{EN}^4 \approx 4$	$\alpha_3 = 0.4$
5	$z_{SW}^4 = 0$ $y_{SW}^5 \approx 4$	$z_{SE}^4 = 10$ $y_{SE}^5 \approx 6$	$z_{WN}^4 = 0$ $y_{WN}^5 \approx 4$	$z_{EN}^4 = 10$ $y_{EN}^5 \approx 6$	$\alpha_4 \approx 0.33$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1000	$z_{SW}^{999} = 10$ $y_{SW}^{1000} \approx 5.005$	$z_{SE}^{999} = 0$ $y_{SE}^{1000} \approx 4.995$	$z_{WN}^{999} = 10$ $y_{WN}^{1000} \approx 5.005$	$z_{EN}^{999} = 0$ $y_{EN}^{1000} \approx 4.995$	$\alpha_{999} \approx 0.002$

Table 2.11.: The flows obtained by 1000 iterations of the Frank-Wolfe algorithm applied on the left network of Example 2.5

2.4. Practical Considerations

We have seen some small examples of traffic networks in this chapter. However, if we would want to apply traffic models in practice, there are several things we need to consider.

2.4.1. Delay Functions

In the examples in this chapter, we have so far worked with simple delay functions like linear functions. In reality, delay functions are not that simple. When the flow is not too large, cars can more or less freely drive (without exceeding the maximum speed). However, when the flow gets too large, there will be congestion and delay will increase dramatically. The function that is often used to achieve this, is the BPR function. The BPR function is developed by the Federal Highway Administration (FHWA), formerly

known as the Bureau of Public Roads (BPR), which was introduced by Ryan (1979). The BPR function is given by

$$D_j(y_j) = t_j \cdot \left(1 + \alpha_j \left(\frac{y_j}{c_j}\right)^{\beta_j}\right), \tag{2.1}$$

where

- t_j is the free flow travel time: the time it would take on edge j when there would be no flow. It holds that $t_j = \frac{s_j}{v_j}$, where s_j is the length of road j, and v_j is the speed limit on road j.
- c_j is the capacity of edge j.
- $\alpha_j > 0$ and $\beta_j > 0$ are edge specific constants. Ryan (1979) proposes to put $\alpha_j = 0.15$ and $\beta_j = 4$, but this is calibrated for American highways.

In urban areas, typically the value of α_j is larger. Roads have fewer lanes, so congestion is more severe. This leads to small changes in the traffic flow to have large effects on the travel time. Muijlwijk (2012) for example proposes $\alpha_j = 2$ for urban areas. In Figure 2.12, two BPR functions with different values for α_j are shown.

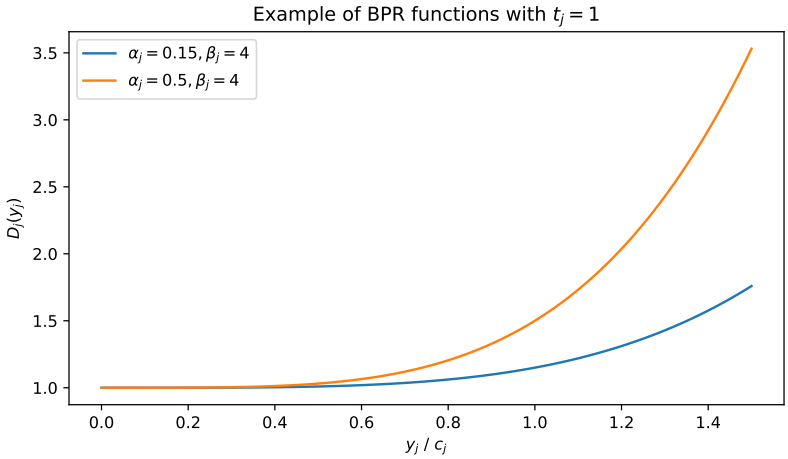

Figure 2.12.: Graph of two BPR functions

Note that the BPR function is not Lipschitz-continuous. However, if we restrict its domain to some bounded set $\mathcal{D}$, it is L-Lipschitz continuous with $L = \sup_{y \in \mathcal{D}} ||D_j'(y)||$. In practice, this Lipschitz condition is therefore irrelevant, as we always work with finite flows. The larger the L, the larger the curvature constant C_f. This means that convergence is slower for larger L, but it is still of order $\mathcal{O}\left(\frac{1}{n}\right)$, so in the long run this makes no difference. In conclusion, we can still apply Theorem 2.9, and convergence to the Wardrop equilibrium is assured when using the Frank-Wolfe algorithm.

2.4.2. Behavior of the Road User

Up to this point, we have only made the assumption that drivers try to minimize their own travel time. In this subsection, we briefly discuss more aspects of the behavior of road users.

Route Choice

Until now we have only considered travel time as the sole explanatory variable for route choice. However, there could be more explanatory variables for route choice. Bekhor, Ben-Akiva, and Ramming (2006) and Cascetta et al. (2002) propose more possible explanatory variables. Some other possible explanatory variables these two papers mention are:

- Road capacity, length, and type;
- Socio-economic variables i.e. age, gender, income;
- Distance between OD pair (relative to the distance as the crow flies).

Note that the BPR function depends on road capacity and length. It also can depend, through the parameters α_j and β_j in Equation 2.1, on road type. As the travel time depends on the delay functions, when choosing delay functions as BPR functions, we implicitly take road capacity, length, and type as explanatory variables already. For the other possible explanatory variables either their significance is too small, or there is not sufficient data. Therefore we argue that it is sufficient to indeed only consider travel time as the sole (explicit) explanatory variable for route choice.

In practice a population can be heterogeneous: individuals can perceive travel times differently. For example, one driver could only be interested in minimizing their travel time, while another driver wants to choose a route with nice scenery and assigns less importance to travel time. In this case, we can model the perceived delay of each route as a random variable distributed across the population of drivers. We can achieve this by adding some random noise term. To be more precise, let D_r^{od} be the delay a driver encounters on route r between origin o and destination d, then the perceived delay is given by

$$D_r^{'od} = D_r^{od} + \epsilon_r^{od}, \quad \epsilon_r^{od} \sim (0, \sigma^2).$$

Route choice now basically reduces to a multinomial discrete choice model. In Chapters 10, 11, and 12 of Sheffi (1985), this is described in more detail and the author shows there still exists an equilibrium in this case: the stochastic user equilibrium. When $\sigma^2 = 0$, the stochastic user equilibrium is the same as the Wardrop equilibrium. In Chapter 12, the author argues that the Wardrop equilibrium is in general a very good approximation of the stochastic user equilibrium for congested networks. Thus, we think that it is sufficient to only consider the Wardrop equilibrium.

Parking Choice

Aside from choosing a route, drivers also have to choose a parking facility close to their destination. Possible explanatory variables for parking choice are:

- Travel time;
- Park search time;
- Parking charge;
- Walking time from the parking facility to the destination;
- Capacity of the parking facility.

The total travel time for a driver from origin to destination is the sum of travel time, park search time, and the walking time from the parking facility to the destination. It will turn out that parking capacity and parking charges can implicitly be taken into account when computing park search times. Therefore, we only take travel time, park search time, and walking time to the destination as the (explicit) explanatory variables for parking choice.

3. Optimization

The goal of this thesis is to optimize parking capacities to decrease emissions, reduce travel time, and limit the space used by parking facilities. Finding a solution to this optimization problem analytically is computationally not feasible. Due to the underlying traffic network, it is impossible to construct some differentiable objective function that takes into account our three objectives.

Therefore, we need to make use of numerical optimization methods and metaheuristics. In this chapter, we discuss these numerical optimization methods and metaheuristics. We purely discuss optimization techniques in general, so not specifically applied to parking capacities. Thus, suppose that our goal is to maximize some function $f(x)$ over some solution space $\mathcal{X}$.

3.1. Grid Search

A naive numerical optimization method is grid search: define a grid of possible solutions, try every solution on this grid, and choose the solution x^* which maximizes $f(x)$. Restricted to this grid, grid search returns the exact optimizer, so as long as the solution space $\mathcal{X}$ is finite, grid search can return the exact optimizer.

Unfortunately, if the grid contains too many points, grid search often becomes computationally infeasible as well. In general, it is not possible to obtain an exact solution to large optimization problems, even restricted to some bounded set. Often, algorithms can only get a sufficiently good solution to an optimization problem in a reasonable time. We call such methods *metaheuristics*.

3.2. Evolutionary Algorithms

In this section, we describe such a class of metaheuristics: *evolutionary algorithms*. This chapter is mainly based on Chapters 3-6 of Negnevitsky (2011) and Chapter 7 of Eiben and Smith (2003).

Evolutionary algorithms are based on biological evolution: there is a population with limited resources, causing only the fittest to survive. This way, the fitness of the population increases. Given a so-called fitness function that has to be maximized, we randomly initialize a population of feasible solutions. Then based on these values, candidates with high fitness are selected to create the next population. This is done by applying crossover (which can be seen as parents getting children) and mutation (which can be seen as a mutation in the DNA of an individual) to them. Generally, the population size is kept equal over time. So unless the size of the offspring is equal to the size of the population,

individuals from the old population also must be selected to survive. This process can be repeated until a population containing a solution with sufficient fitness is found.

Formally, this leads to a general scheme for evolutionary algorithms, given by Algorithm 3.1.

```
1  Randomly initialize a population with feasible solutions
2  Compute the fitness of each candidate
3  Select parents
4  Let crossover take place on the parents
5  Mutate the resulting children
6  Compute the fitness of the new candidates
7  Select candidates for the next population
8  if Termination criterion satisfied then
9  |  Terminate
10 else
11 |  Return to step 2
12 end
```

Algorithm 3.1: General scheme for evolutionary algorithms

We cannot always directly apply evolutionary algorithms to an optimization problem. Sometimes, the solution space must be represented by some other space to make evolutionary algorithms applicable. In the case of parking capacities, it will turn out later that we can represent solutions by a sequence of positive integers. So without loss of generality, we assume that the solution space is $\mathbb{N}_{\geq 0}^k$ where k is some integer specifying the dimension of the solution space.

3.2.1. Parent Selection

There are several techniques to select parents, we discuss the two most widely used ones: selection proportional to fitness and selection based on rank.

Fitness Proportional Selection

When applying fitness proportional selection, individuals are chosen based on their fitness value compared to the fitness values of other individuals. Concretely this means that individual x_i is chosen from a population of size N with probability

$$\frac{f(x_i)}{\sum_{j=1}^{N} f(x_j)}.$$

There are some problems with this selection method:

- Firstly, individuals with very high fitness relative to other individuals, take over the entire population very quickly. This can cause the algorithm to get stuck in a sub-optimal solution, this is also known as *premature convergence*.

- When fitness values are very similar, the selection is almost entirely uniformly distributed. Having higher fitness then barely has any effect. So then although convergence may take place, the mean population fitness only increases very slowly.
- The fitness proportional method is sensitive to translations. This is illustrated in Table 3.2, which shows a population of two individuals, x_1 and x_2, with fitness $f(x_1) = 1$ and $f(x_2) = 9$. Adding a constant to the fitness function does not change the location of the optimum, but it does change the selection probabilities drastically. In the original case, x_2 is chosen with a much higher probability than x_1. However, when adding 100 to the fitness values, both have almost equal probability to be chosen.

	$f(x_i)$	sel. prob.	$f(x_i)+10$	sel. prob.	$f(x_i)+100$	sel. prob.
x_1	1	0.1	11	0.367	101	0.481
x_2	9	0.9	19	0.633	109	0.519

Table 3.2.: Adding a constant to the fitness function changes selection probabilities

A remedy often used to solve the last two problems, is a method called *windowing*. The differences in fitness are kept by subtracting a (time-dependent) value β_t from the fitness values. The most simple choice is

$$\beta_t = \min_{x \in P_t} f(x),$$

where P_t is the population at time t. Note when all fitness values are equal, we don't have to apply this windowing, as we would have uniformly distributed selection probabilities regardless.

Ranking Selection

Rank-based selection is another method that was inspired by the problems that occur when the fitness proportional selection method is used. This method sorts the population based on their fitness values and assigns selection probabilities to individuals based on their rank. Assume that the best solution in a population has rank $N-1$, and the worst solution has rank 0, where N is again the population size.

In practice, we usually then compute selection probabilities based on a linear ranking scheme. Explicitly, given a rank r of an individual and some parameter $1 < s \leq 2$, the selection probability is given by

$$P^s_{lin}(r) = \frac{2-s}{N} + \frac{2r(s-1)}{N(N-1)}.$$

The parameter s tells us how much emphasis we want to put on selecting individuals with high fitness. This is called *selection pressure*. When s is close to 1, there is almost no selection pressure: selection probabilities are close to uniform. When s is large, there is more selection pressure: individuals with high rank have a higher selection probability.

The selection pressure of a linear ranking scheme is limited, as s cannot be larger than 2. When s is larger than 2, the worst individual would have a negative selection probability. That is why another ranking scheme is proposed: an exponential ranking scheme, given by

$$P_{exp}(r) = \frac{1 - e^{-r}}{c},$$

where c is some normalizing constant.

In conclusion, ranking selection should be considered when fitness values are very close to each other or very far away from each other, which the fitness proportional selection cannot deal with properly as discussed before. A disadvantage of the ranking selection scheme compared to the fitness proportional selection scheme, is that the selection pressure may not be appropriate, but the exponential ranking scheme could provide a solution.

3.2.2. Crossover

The most used crossover technique is the one-point crossover. Let $x_1 = (x_1^1, \ldots, x_1^k)$ and $x_2 = (x_2^1, \ldots, x_2^k)$ be two individuals. Let crossover take place with probability p_c. If crossover takes place, a random number $r \in \{1, \ldots, k-1\}$ is chosen to split both parents at that point to create two children by exchanging the tails. This results in offset $x_1' = (x_1^1 \ldots x_1^r, x_2^{r+1}, \ldots, x_2^k)$ and $x_2' = (x_2^1 \ldots x_2^r, x_1^{r+1}, \ldots, x_1^k)$. An example of this crossover is shown in Figure 3.3.

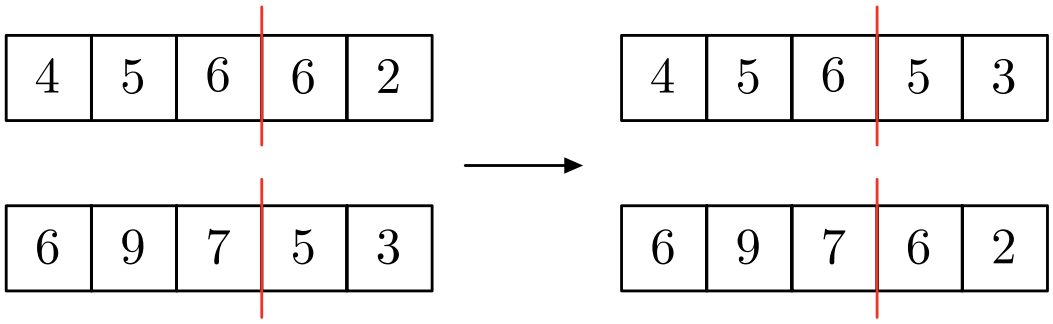

Figure 3.3.: Example of one-point crossover with integer representation

3.2.3. Mutation

Mutation takes place as follows: in each position independently, with some mutation probability p_m, a new integer is chosen at random such that the mutated individual is still part of the solution space. An example of such a mutation is shown in Figure 3.4.

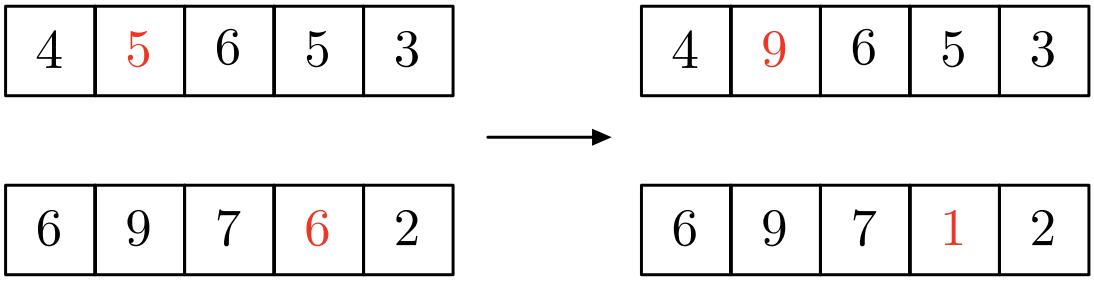

Figure 3.4.: Example of mutation with integer representation

3.2.4. Survivor Selection

In principle, the methods that are applied for parent selection can be used for survivor selection. However, in practice, as opposed to parent selection, mostly deterministic methods are used for survivor selection. There are two flavors: selection based on 'age' and selection based on fitness.

Age-Based Selection

The principle of selection based on age is that individuals survive in a population for a fixed amount of iterations. For this method of selection, fitness is completely irrelevant. Consequently, the mean and even the best fitness can decrease after an iteration. Nonetheless, in the long run, this could even be beneficial: if the algorithm is stuck at a local optimum, it can escape. It is however important that there is enough selection pressure when selecting the parents.

Fitness-Based Selection

Contrary to age-based selection, the fitness-based selection makes use of the fitness of each individual. Several different fitness-based selection methods are used in practice:

- **Replace worst**: In this method, we simply remove the individuals with the worst fitness. This can lead to premature convergence though, as it tends to only select the fittest individuals.
- (μ, λ)**-selection**: In this method, $\lambda \geq \mu$ children are born from μ (in our case N) parents. This method is a mixture of age-based and fitness-based selection. The age-based component implies that parents are removed and thus that every individual is considered only for one iteration. The fitness-based component implies that the best μ children out of the λ children are chosen to be part of the new population.

3.2.5. Genetic Algorithm

The most widely used evolutionary algorithm is the genetic algorithm, which was introduced by Holland (1975). In the genetic algorithm, the offspring is the same size as the population. This also means that survivor selection is not necessary for the genetic algorithm. In the simple genetic algorithm, fitness proportional selection is used to select parents. Formally, this leads to the scheme given by Algorithm 3.5.

1 Randomly generate an initial population of size N of chromosomes: $x_1, \ldots, x_N$
2 Compute the fitness of each chromosome: $f(x_1), \ldots, f(x_N)$
3 Select a pair of chromosomes (y, z) from the population with
$\mathbb{P}(y = x_i) = \mathbb{P}(z = x_i) = \frac{f(x_i)}{\sum_{j=1}^{N} f(x_j)}$
4 With probability p_c apply crossover to (y, z) to create an offspring (y', z')
5 With probability p_m apply mutation to (y', z') to create an offspring (y'', z'')
and add this to the new population
6 **if** $\#$*new population* $= N$ **then**
7 | Continue
8 **else**
9 | Return to step 3
10 **end**
11 Replace the population with the new population
12 **if** *Termination criterion satisfied* **then**
13 | Terminate
14 **else**
15 | Return to step 2
16 **end**

Algorithm 3.5: Simple Genetic Algorithm

The crossover probability p_c is often chosen around 0.7, while the mutation probability p_m is often chosen around $1/N$, where N is the population size. Other popular choices for the mutation probability are 0.001 and 0.01. In general, it turns out that it can be difficult to choose a suitable (fixed) mutation probability. That is why dynamic mutation probabilities have been proposed, for example by Bäck (1992), where the mutation rates are incorporated into the genetic representation of the individuals.

Lastly, as we have seen in Section 3.2, using the fitness proportional selection method has some flaws. That is why variations on fitness proportional selection or other selection methods could be considered as well.

3.3. Brief Overview of Other Methods

Aside from evolutionary algorithms, there are several other metaheuristics. In this section, we briefly describe the most used ones and discuss the differences with evolutionary algorithms.

Hill Climbing

One of the most basic metaheuristics is hill climbing. The algorithm starts with some random solution and iteratively tries to find a better solution by making a small incremental change to the solution. If no increment can be made, the algorithm terminates, and it returns a solution. Depending on the initial solution, most likely, this is a local optimum and not a global optimum. Evolutionary algorithms on the other hand can give an approximation of the global maximum.

Tabu Search

Another very popular metaheuristic is tabu search, introduced by Glover (1986). Tabu search repeatedly moves from one solution to another solution in its neighborhood. A neighborhood of a solution can be defined in different ways, for example when we work with binary strings, a neighbor of a certain bitstring could be another bitstring with one bit flipped. Generally, we choose a new solution in the neighborhood of a solution if it outperforms all other solutions in this neighborhood. This means that it does not have to outperform the current solution, contrary to hill climbing. This means tabu search allows discovering new parts of the solution space, which makes it possible to jump out of a local optimum. There is also a so-called tabu list with solutions that have been chosen recently. These solutions cannot be chosen again for a certain amount of iterations, which ensures that the algorithm does not jump back and forth between two solutions.

Simulated Annealing

A more sophisticated form of tabu search is simulated annealing, which was proposed by Kirkpatrick, Gelatt, and Vecchi (1983). This metaheuristic is inspired by annealing in condensed matter physics. According to Wikipedia (2023b), in condensed matter physics, annealing involves heating and controlled cooling to improve the physical properties of a material. When heating, atoms can move freely (move from solution to solution freely), while when cooling the structure becomes more rigid (not every change in the solution is allowed). In every iteration, the current solution is replaced by a neighboring solution with some probability based on the difference in the objective function values and the temperature T.

Initially, the temperature T is high and new solutions are chosen with high probability. After every iteration, T is reduced. If the temperature decreases, the probability of 'bad' solutions (solutions with relatively low objective function value) getting accepted, decreases.

Comparison with Evolutionary Algorithms

Tabu search and simulated annealing only look at neighboring solutions every iteration, while evolutionary algorithms consider a way more diverse possible offspring of solutions. However, as the so-called No Free Lunch (NFL) theorems stated in Wolpert and

Macready (1997) tell us, which method is the most accurate, depends on the problem we consider. Some comparisons between metaheuristics have been made in past literature, see for example Manikas and Cain (1996), but those results only hold for very specific types of problems. For optimization regarding parking capacities, comparisons have not been made. On top of this, which method performs best can also depend on the particular traffic network we consider.

A disadvantage of evolutionary algorithms is that it considers an entire population at every iteration, which makes it computationally slower. This can be solved by parallelizing evolutionary algorithms though.

In the end, we cannot determine which algorithm is better based on theory. Evolutionary algorithms are expected to cover more of the solution space as they consider an entire population of solutions, and do not restrict themselves only to so-called neighboring solutions. The computational disadvantages could be bypassed by parallel programming.

Thus, in this research, we stick with the evolutionary algorithm. However, we note that it is not possible based on theory to determine if this is the best method. If it turns out that the evolutionary algorithm does not provide the desired results, other methods should be considered.

4. Methodology

As stated in Chapter 1, we want to find a method to optimize parking capacities with three main objectives: minimize emissions, minimize travel times, and minimize the use of land for parking purposes. With the current model of traffic, we only obtain delays and flows on every edge of a network. We can quantify each of the three objectives in the following way:

- **Distance:** According to Csikós, Tettamanti, and Varga (2015), emissions can be modeled based on the total distance covered by cars and the average speed. With the current traffic model, we cannot compute the average speed. However, distances can be computed using flows and lengths on every edge. Therefore, we propose to use distance as a proxy for emissions.
- **Travel times:** Using the flows and delays on every edge, we can derive travel times directly.
- **Parking capacities:** We take the total parking capacities as a proxy for land used with the purpose of parking: the lower the parking capacities, the less space parking facilities use.

We combine these three objectives into a single objective function, which we will give explicitly in Section 4.2.

The general scheme of the methodology can then be seen in Figure 4.1. There are two main components: the traffic and parking model and the optimization component.

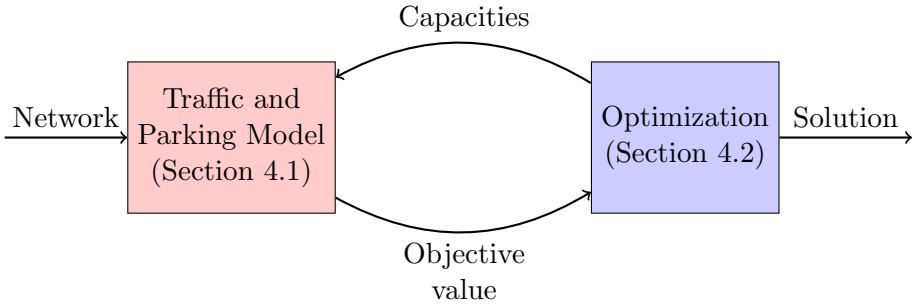

Figure 4.1.: General Scheme Methodology

In Section 4.1, we discuss the addition of parking to the current traffic model. The idea is that the traffic and parking model is provided with some network. It returns both the total delay and distance, then together with the total capacities, these can be used

to compute the corresponding objective function. An optimization layer tries to find parking capacities, such that the objective function is maximized. This optimization layer is treated in Section 4.2.

4.1. Parking in Traffic Models

We have defined the basic traffic network and we have described how to solve a basic traffic assignment problem both analytically and numerically in Chapter 2. In this section, we will describe how we incorporate parking into the traffic model. We also discuss the implications this has for solving the traffic assignment problem. This section serves as an answer to sub-question 1 in Chapter 1.

4.1.1. Extending the Network with Parking Nodes and Links

We already defined a road network in Section 2.1. We will now extend these road networks by adding parking facilities to the network. We said the road network can be represented by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{J})$, where $\mathcal{V}$ are the vertices and where $\mathcal{J}$ are the directed edges of the graph. We said that the edges represent the roads in the network, while the vertices represent the road intersections. For this reason, we now refer to $\mathcal{V}$ as the set of intersection nodes.

We can extend this definition by adding a set of parking nodes $\mathcal{V}_p$ to $\mathcal{V}$. Each parking node belongs to one specific intersection node and represents a parking facility at this intersection. Thus, using this definition, a parking node can only coexist with a corresponding intersection node.

We also add the set of parking links $\mathcal{J}_p$ to $\mathcal{J}$. These edges connect a regular intersection node with the corresponding parking node.

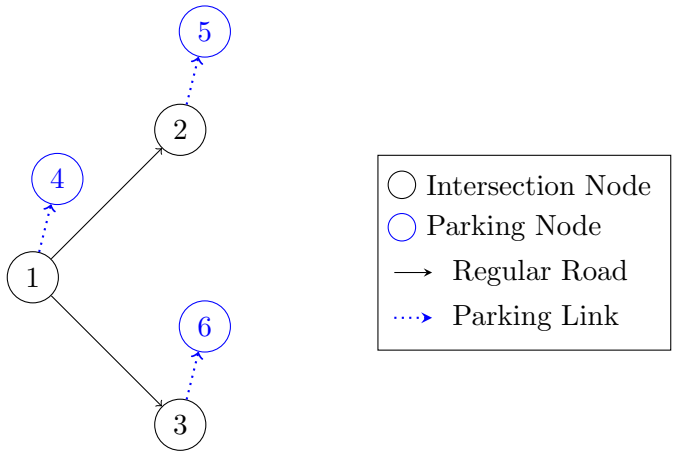

Figure 4.2.: Example of an (unweighted) road network with parking facilities incorporated

An example of such an extended network is given in Figure 4.2. Now if a driver wants

to travel from node 1 to node 2, they have to park their car at node 2 as well. This is represented by the corresponding parking node, node 5. The link between node 2 and node 5 corresponds to the time it takes them to park their car at node 2, so it does not correspond to travel time. We sometimes also call this the *park search time*.

4.1.2. Extending the Network with Walking Links

In reality, some parking facilities might be always full. In this case, it could be an option for a driver to park their car at another node, and walk from that node to their destination. To make this possible, we extend the network further.

We do this by adding a set of walking links $\mathcal{J}_w$ to $\mathcal{J}$. These edges connect parking facilities and represent walking routes between the corresponding intersection nodes. In Figure 4.3, such an extension of the previous network is shown.

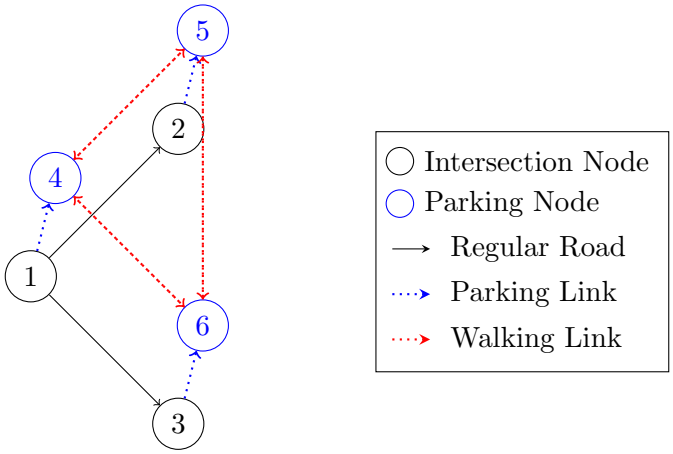

Figure 4.3.: Example of an (unweighted) road network with parking facilities and walking links incorporated

To summarize, we can represent a road network, with parking included, by a graph $\mathcal{G}^+ = (\mathcal{V}^+, \mathcal{J}^+)$, where $\mathcal{V}^+ = \mathcal{V} \cup \mathcal{V}_p$ consists of the intersection nodes and corresponding parking nodes and where $\mathcal{J}^+ = \mathcal{J} \cup \mathcal{J}_p \cup \mathcal{J}_w$ consists of road links, parking links, and walking links.

When we think about road networks in practice, it is not logical to put walking links between every possible pair of parking nodes. Some nodes may be too far away from each other. Therefore, we introduce a walking limit, wl. This walking limit is the maximum length a walking link is allowed to be. The length is determined by the Euclidean distance between two parking nodes, which is technically the Euclidean distance between the two intersection nodes they correspond with.

We also want to reduce the number of drivers parking their cars somewhere and then walking several kilometers by using multiple walking links. We can do this by not adding a walking link between parking nodes where both facilities have a capacity equal to 0. To avoid this phenomenon altogether, a change has to be made in computing the route

with the shortest travel time. However, Dijkstra's algorithm, which we use to compute the shortest paths, is only able to take (static) weights into account.

4.1.3. Park Search Time and Walking Delay

We have defined both parking and walking links, but we have not covered the delays we impose on these links. We continue by discussing the delays on parking and walking links.

Park Search Time

We have defined park search time as the delay on a parking link. This park search time can be defined in several ways. We choose the method proposed by Lam et al. (2006), where the authors used the BPR functions we already use as road delays to model park search time. However, the parameter α_j as in Equation 2.1, is typically drastically different compared to those used for road delays. When a parking facility is full, a car has to wait for another car to leave the facility. On a road, when the capacity is reached, there may be a traffic jam, but traffic normally still slowly continues. This means that the parameter α_j is typically much higher for the delay on parking links than for the delay on road links.

Some researchers, for example Pel and Chaniotakis (2017), argue that it is necessary to account for uncertainty in parking occupation by adding some stochastic elements. With current technologies though, it is possible to retrieve live data of parking occupation. An example is Gemeente Amsterdam (2023) which is an interactive map that shows the occupation of parking facilities in the municipality of Amsterdam, the Netherlands. That is why we would argue the inclusion of uncertainty is not necessary anymore, definitely considering the fact that technologies will only improve in the (near) future, which will only extend the knowledge drivers have about parking occupations.

Walking Delay

We also defined walking links. On these walking links, we assume the delay to be constant, as in practice it does not depend on the flow. An obvious choice would be to choose

$$D_j(y_j) = \frac{s_j}{v_j},$$

where s_j is the length of the walking link and v_j is some constant walking speed.

4.1.4. Wardrop Equilibrium and Frank-Wolfe Algorithm

Mathematically, we have only added nodes and links to the road network defined in Section 2.1. As we use BPR functions for delays on both the road and parking links and constants for the delays on walking links, delays are continuously differentiable and increasing. For that reason, we can still apply Theorem 2.4 to conclude that a Wardrop equilibrium exists.

Furthermore, as we already argue in Section 2.4, we can restrict BPR functions to some finite domain to make them L-Lipschitz continuous for some constant L. Moreover, constant functions are 0-Lipschitz continuous. Therefore, we can still apply Theorem 2.9 to conclude that the Frank-Wolfe algorithm provides convergence to the Wardrop equilibrium, even when parking is added to the traffic model.

4.2. Optimization of Parking Capacities

We have introduced optimization methods in general in Chapter 3. We continue by discussing how to apply these methods to parking capacities to answer sub-question 2 stated in Chapter 1.

4.2.1. Formal Problem Statement

To solve this optimization problem, we will use variants of the evolutionary algorithms which are described in Section 3.2. Later, we specify which variants we use exactly. Remember that evolutionary algorithms are used to maximize some function, while we now want to minimize three variables instead. Also, note that we want to compute weights based on the fitness values when applying the evolutionary algorithm, That is why it is important that the objective function values are always non-negative.

Suppose we have some network with parking as described in 4.1 which consists of n nodes (without parking nodes). Then we can assign some capacities $c = (c^1, \dots, c^n)$ to every parking node, which will be used in the delay function of the corresponding parking link. Consequently, we can obtain the Wardrop equilibrium using the Frank-Wolfe algorithm, and we can compute both the total delay, say t_c, and the total distance covered by the car, say d_c when the network is in equilibrium. This leads to the following objective function:

$$f(c) = \frac{1}{w_1 \cdot \sum_{j=1}^{n} c^j + w_2 \cdot t_c + w_3 \cdot d_c},$$

where w_1, w_2, w_3 are some (positive) weights that can be chosen based on the importance of every quantity for for example a policymaker wanting to optimize parking capacities in a certain area.

Note that maximizing this function is indeed equivalent to minimizing a weighted average of the three variables, which was exactly our goal. Another way this is usually achieved is to put a minus in front of the variables we want to minimize, but again, we want to avoid negative objective function values. In general, an objective function of the following type would work:

$$f(c) = g\left(w_1 \cdot \sum_{j=1}^{n} c^j + w_2 \cdot t_c + w_3 \cdot d_c\right),$$

where g is a decreasing and positive function. We have now used $g(x) = \frac{1}{x}$, but other choices include $g(x) = e^{-x}$ or $g(x) = x^\beta$ for any $\beta < 0$. The choice of $g(x) = \frac{1}{x}$ is the

easiest to interpret, as it is an inverted weighted average of the variables of interest. If it turns out that our choice is not suitable, we can try different decreasing and positive functions.

It is obvious that this objective function is not easily differentiable in c as the variables t_c and d_c depend on c through the underlying network structure. This indeed shows the need for metaheuristics like evolutionary algorithms.

In practice, capacities are always positive and finite, so we assume that there is some maximal capacity $0 < c_{max} < \infty$. Furthermore, in practice, the number of parking facilities is usually not equal to the number of intersection nodes n. So we also assume there can be $0 \leq n_{max} \leq n$ parking facilities. It could also be the case that it is not possible to have a parking facility at some of the intersection nodes. Therefore, we introduce some subset $B \subset \{1, \ldots, n\}$, which tells us at which intersection nodes it is possible to add parking facilities. The exact optimization problem then becomes

$$\begin{aligned} \text{maximize} \quad & \frac{1}{w_1 \cdot \sum_{j=1}^{n} c^j + w_2 \cdot t_c + w_3 \cdot d_c} \\ \text{subject to} \quad & w_1, w_2, w_3 \geq 0, \quad w_1 + w_2 + w_3 > 0, \\ & |\{j : c^j > 0\}| \leq n_{max}, \\ & c^j > 0 \implies j \in B, \\ \text{over} \quad & 0 \leq c^j \leq c_{max}. \end{aligned} \tag{4.1}$$

4.2.2. Evolutionary Algorithms for Parking Capacities

As mentioned earlier, we will use variants of the evolutionary algorithms to approximate the solution of the maximization problem in Equation 4.1.

Representation

Independent of the choices we make for the evolutionary algorithm variants, the representation of the solutions is the same for each method. We already said in Section 3.2 that we represent solutions as a sequence of integers. Specifically, a solution x_i is represented as follows:

$$\begin{aligned} x_i &= (x_i^1, \ldots, x_i^n, x_i^{n+1}, \ldots, x_i^{n+n_{max}}) \\ &:= (c_i^{*1}, \ldots, c_i^{*n}, l_i^1, \ldots, l_i^{n_{max}}), \end{aligned}$$

where $0 \leq c_i^{*j} \leq c_{max}$ represents the parking capacity we would assign to a parking facility corresponding to intersection node j and where $l_i^k \in B$ represents that parking facility l_i^k is actually considered. This can then be translated to the corresponding capacities $c_i = (c_i^1, \ldots, c_i^n)$ with

$$c_i^j = \begin{cases} c_i^{*j}, & \text{if } l_i^k = j \text{ for some } k \in \{1, \ldots, n_{max}\}, \\ 0, & \text{otherwise.} \end{cases}$$

So for example, if we have $n = 4$, $n_{max} = 3$, $B = \{1, 2, 3, 4\}$ and $c_{max} = 5$, a possible solution would be

$$(5, 3, 4, 1, 1, 2, 4),$$

such that parking facilities at intersection nodes 1, 2, and 4 are considered with parking capacities 5, 3, and 1. This also means that there is no parking facility at intersection node 3. This is then translated to the capacity vector

$$c = (5, 3, 0, 1).$$

Parent Selection

We consider both the fitness proportional selection and ranking selection as described in Subsection 3.2.1 to select parents. For ranking selection we now propose to set the parameter $s = \frac{3}{2}$, so exactly between the two extreme values of 1 and 2. Beforehand, it is not possible to establish which s works the best. Computationally, it is rather demanding to consider several values of s. If it turns out this s does not provide good performance, we can potentially consider other values.

Crossover

Crossover happens exactly as described in Subsection 3.2.2. We choose $p_c = 0.7$, which is the same crossover probability that is often chosen in the special case of the genetic algorithm.

Mutation

Mutation happens exactly as in Subsection 3.2.3. Note that for some (mutated) solution x_i to be in the solution space, capacities c_i^{*j} should still be between 0 and c_{max} and that locations l_i^j should still be in B. We choose $p_m = 1/N$, where N is the population size. This is the same mutation probability that is often chosen for the genetic algorithm.

Survivor Selection

For the survival selection, we combine fitness-based and age-based selection similar to (μ, λ)-selection. Let p_r be the proportion of the population that is replaced every iteration. Then the best $1 - p_r$ part of the population survives to the next iteration, while the other part of the population is replaced by offspring created by parent selection, crossover, and mutation. When applying the genetic algorithm as described in Algorithm 3.5, fitness proportional parent selection is used and $p_r = 1$ is chosen. On the other hand, when $p_r = 0$, all solutions survive every iteration, and nothing happens.

We consider three values for p_r. Firstly, we choose $p_r = 1$ because this is the value of p_r in the genetic algorithm. We also choose a value of p_r close to 1, namely $\frac{19}{20}$. Lastly, we choose a more extreme value of p_r, namely $\frac{1}{2}$. In this case, we keep the best half of the solutions and replace the worst half in every iteration. As mentioned in Chapter

3, the NFL theorems stated in Wolpert and Macready (1997) tell us that we can only choose the best p_r for one specific problem. As we are not considering one particular urban area, we cannot choose one optimal value for p_r. Instead, the goal is to give an indication: does sticking to the genetic algorithm proposed by Holland (1975) provide the most accurate solutions, or does deviating from it provide the most accurate solutions? Can the same conclusion be made in different situations or does the conclusion vary in different situations?

Number of Iterations and Population Size

As Negnevitsky (2011) argues, both the number of iterations and the population size depend on the type of problem that needs to be solved. This depends mostly on the solution space size and the objective function considered. In practice, these parameters are chosen empirically. De Jong (1975) found out that a relatively small population size improves performance in the beginning while a relatively large population size improves long-term performance.

Eiben and Smith (2003) choose both the population size and the number of iterations equal to 100 in one of their examples. This is why we also choose both the population size and the number of iterations equal to 100. If it turns out that this choice causes results to be unacceptable, we can consider changing these values.

Overview

The general framework for the variants of the evolutionary algorithms we consider is given by Algorithm 4.4. A variant is completely determined by the selection method we choose in step 3 and the replacement proportion p_r.

```
1  Randomly generate an initial population x_1, ..., x_100
2  for i = 1 to 100 do
3  |  Compute the fitness of each chromosome: f(x_1), ..., f(x_100)
4  |  Select a pair of chromosomes (y, z) from the population using fitness
   |   proportional or ranking selection
5  |  With probability p_c apply crossover to (y, z) to create an offspring (y', z')
6  |  With probability p_m apply mutation to (y', z') to create an offspring (y'', z'')
   |   and add this to the new population
7  |  if #new population = p_r · N then
8  |  |  Continue
9  |  else
10 |  |  Return to step 3
11 |  end
12 |  Replace the worst p_r · N individuals of the old population with the new
   |   population.
13 end
```

Algorithm 4.4: Evolutionary algorithm framework

Table 4.5 summarizes the combinations of selection methods and replacement proportions we consider.

Label	Parent selection	p_r
fp_1	Fitness proportional	1
fp_19/20	Fitness proportional	19/20
fp_1/2	Fitness proportional	1/2
r_1	Ranking	1
r_19/20	Ranking	19/20
r_1/2	Ranking	1/2

Table 4.5.: Considered combinations evolutionary algorithm variants

4.2.3. Imposing Further Constraints

In practice, we may also want to assume an explicit limit on the total parking capacities in the network. Right now, the limit is given by the maximum for a single parking facility multiplied by the maximum number of parking facilities. In principle, a total capacity that is too large is taken care of by a low fitness value, but it could be that a policymaker already has an explicit global maximum in mind.

So suppose we want to impose some maximum, gm, on the sum of all capacities:

$$\sum_{i=1}^{n} c^i \leq gm, \quad gm > 0.$$

Unfortunately, the crossover and mutation operators that are used when applying evolutionary algorithms can cause solutions to violate this restriction. We illustrate this by looking at the following example. Consider a network with the following properties.

- Three intersection nodes, so $n = 3$;
- The maximal number of parking facilities, $n_{max} = 2$;
- The maximal capacity of every parking facility, $c_{max} = 10$;
- The subset of intersection nodes considered, $B = \{1, 2, 3\}$;
- A global maximum on the parking capacities, $gm = 15$.

Two solutions for this problem can then be $(10, 1, 1, 1, 2, 3)$ and $(1, 7, 7, 1, 2, 3)$. An example of offspring after crossover is then $(10, 1, 7, 1, 1, 2, 3)$ and $(1, 7, 1, 1, 2, 3)$. Now notice that for the first of these two solutions, capacities sum up to 18, which is larger than 15.

A remedy can be to give solutions that end up outside of our desired solution space fitness equal to 0. In general, this approach can be applied to impose further restrictions, so to assign fitness equal to 0 to solutions that do not satisfy these further restrictions.

In practice, assigning a fitness of 0 to a solution can cause complications when computing weights for the fitness proportional selection method. Therefore we assign a very small fitness value to these infeasible solutions, namely 10^{-14}. We choose this value because, in Python, floats are 64 bits, with 53 bits of precision. This means a precision of about 15 to 17 significant decimal digits. Hence leading to our choice of 10^{-14}, which consists of exactly 15 decimal digits. A more extensive explanation about the precision of floats is provided by Wikipedia (2023a). Obviously, we need to make sure fitness values are nowhere near this value though. Otherwise, a different type of objective function should be considered.

4.3. Implementation

We proceed with discussing the implementation of both the parking and traffic simulation and the optimization of parking capacities. The general scheme of the implementation is shown in Figure 4.6. All components are implemented in Python. We make a distinction between the two layers shown in Figure 4.1, namely the traffic and parking model and the optimization component.

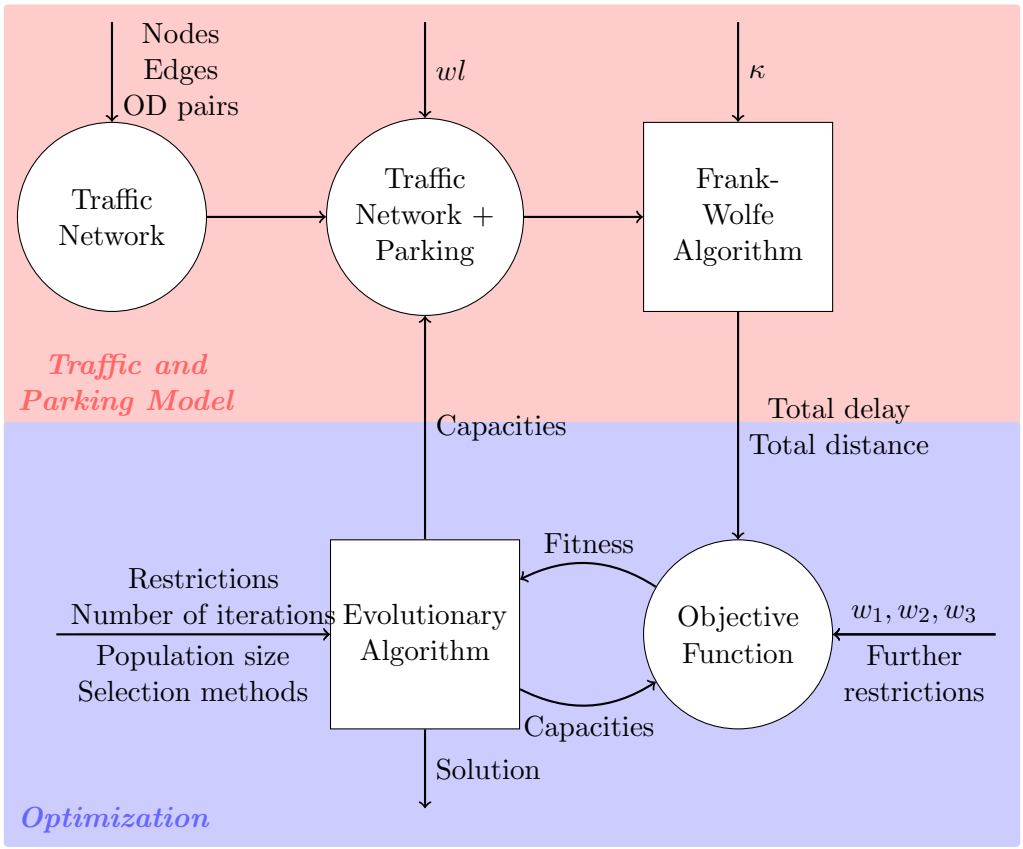

Figure 4.6.: Scheme of implementation

4.3.1. Traffic and Parking Model

In the traffic and parking model, first, a basic traffic network needs to be created. This is achieved by translating nodes, edges, and OD pairs into a graph. Thereafter, parking nodes, parking links, and walking links can be added via the procedure described in Section 4.1. To do this, we also provide the walking limit parameter, wl. Capacities of the parking nodes are provided by the optimization layer. Then to assign flows to every link, the Frank-Wolfe algorithm can be applied to compute the Wardrop equilibrium. When applying the Frank-Wolfe algorithm, it is necessary to define the parameter κ in Algorithm 2.7. Remember that this parameter is used to define a stopping criterion. We want to choose this value as small as possible to get as close to the actual Wardrop equilibrium as possible, but executing the Frank-Wolfe algorithm should still be computationally feasible. This depends on the size and complexity of the network that we apply the algorithm to.

Parallelizing the Frank-Wolfe Algorithm

As described in Section 2.3, when applying the Frank-Wolfe algorithm, in every iteration, an all-or-nothing assignment is determined. This is done by computing the shortest path for every OD pair. Computationally, this can be very demanding. To potentially speed these computations up, we can apply *parallel computing*. However, when we use Python, we are dealing with the Global Interpreter Lock (GIL). The GIL allows only one thread to be executed at a time. Instead of creating multiple threads, it is possible to create multiple Python processes by using the Multiprocessing library. Parallel computing is then still technically possible, but to really exploit the power of parallel programming, a different language like C++ should be considered.

To analyze if we can still gain performance by using the Multiprocessing library in Python, we implemented a parallel version of the Frank-Wolfe algorithm. We compared this parallel version with the basic version by considering three random networks of 50, 100, and 200 nodes. For these networks, we added 10, 20, and 40 OD pairs respectively. Also, we made sure that there was at least one route possible for every OD pair. We put κ equal to 0.5, to limit computation time. This only influences the number of iterations, while parallelization happens only in individual iterations. The computation times for both versions of the Frank-Wolfe algorithm are presented in Table 4.7.

Method	50 nodes/10 OD pairs	100 nodes/20 OD pairs	200 nodes/40 OD pairs
Parallel	9.37s	25.06s	296.4s
Basic	0.56s	4.43s	50.56s

Table 4.7.: Computation times of the parallel and basic Frank-Wolfe algorithm for the different network sizes

We see that the basic Frank-Wolfe algorithm is computationally faster. Of course, the more OD pairs, the more computations have to be made. So, we would expect that the performance of the parallel Frank-Wolfe algorithm improves relative to that of the basic

Frank-Wolfe algorithm as the number of OD pairs increases. However, at some point, as the number of processes that can run simultaneously is limited, this improvement will stop. It seems like the parallel algorithm does relatively better when going from 50 to 100 nodes. However, it does relatively the same when going from 100 to 200 nodes. Hence, it seems like parallelizing the Frank-Wolfe algorithm in Python does not provide us with any computational benefits no matter how large the network we consider.

In the remainder of this thesis, we keep $\kappa = 0.5$ to let computation time not get unreasonably high. When using the evolutionary algorithm, the Frank-Wolfe algorithm has to be applied $100 \cdot 100 = 10000$ times. For the example with 50 nodes this would then already take roughly 5000 seconds, so more than an hour.

4.3.2. Optimization

Using the flows obtained by applying the Frank-Wolfe algorithm, we can compute both total delay and total distance in the network. These two values, together with the total capacities, are used to compute the fitness. To define an objective function, we need to provide weights w_1, w_2, w_3. Moreover, there might be further restrictions we want to impose as described in Subsection 4.2.3 by setting the fitness equal to 10^{-14}. Also, it could happen that choosing certain capacities, particularly a lot of capacities equal to zero, causes routes between certain OD pairs to not exist. In this case, this solution is infeasible, and we set the fitness value equal to 10^{-14}.

The fitness is then passed along to the evolutionary algorithm to determine new capacities, that potentially yield a higher fitness. To apply the evolutionary algorithm, it is necessary to establish the restrictions on the solutions as in Equation 4.1, the number of iterations, the population size, and the selection methods for both parent and survivor selection as described in Subsection 4.2.2.

In conclusion, as illustrated by Figure 4.6, there are two interacting layers, the traffic and parking model, and the optimization layer. The components inside each layer have been treated separately, and in this section, we have discussed how to put them together.

5. Experiments

In this chapter, we apply the proposed methods displayed in Table 4.5 to different examples. We start by applying each method on a small example network of nine nodes. We continue by considering an even smaller network of seven nodes, such that it is possible to validate the results obtained by the methods by comparing them with the actual optimum obtained by grid search. We also consider several different situations to check if the methods are robust to changes in the weights of the objective function and in the optimization constraints. Lastly, we provide a case study of the city of Delft in the Netherlands.

5.1. Scenario 1: Small Example Network

We start by evaluating the different methods on the network given by Figure 5.1. The distance between neighboring nodes is chosen as 1 km. We put $wl = 1$, such that a walking link is possible between every pair of neighbors. As discussed in Subsection 4.1.2, a walking link can only be added when the capacity of one of the two neighboring parking facilities is non-zero. The network consists of three OD pairs shown in Table 5.2.

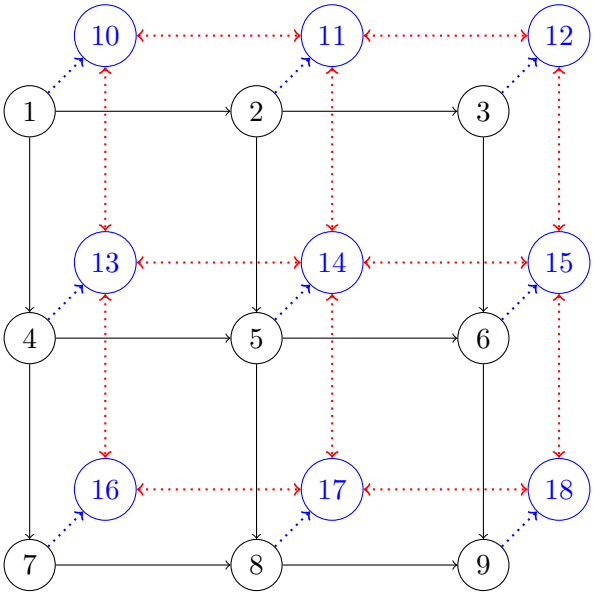

Figure 5.1.: Scenario 1: Grid network

As discussed in Section 2.4, we use a BPR function with parameters $\alpha_j = 2$ and $\beta_j = 4$ as the delay on road edges. Furthermore, we impose a maximum speed of 50 km/h on the roads. As the total demand is equal to 1000, we set the capacity of every road equal to 1000, such that all roads are equally as accessible for every driver.

In Subsection 4.1.3, we argued we can use a BPR function for the park search times as well. We choose a free-flow travel time equal to $\frac{1}{30}$ such that parking in an empty parking facility takes 2 minutes. We also argued that α_j is typically higher for parking links, so we choose $\alpha_j = 9$, while we keep $\beta_j = 4$. These parameter values are chosen arbitrarily. In practice, these parameters can be chosen for example by using available parking data of a certain urban area. The parking capacities are chosen by the evolutionary algorithm. Lastly, we choose the walking speed equal to 4 km/h, such that there is a constant delay of $\frac{1}{4}$ on the walking links. The delays on the different types of roads are presented in Table 5.3.

OD pair	Demand
(1, 3)	200
(1, 7)	200
(1, 9)	600

Table 5.2.: OD matrix for scenario 1

Type	$D_j(y_j)$
Road	$\frac{1}{50} \cdot \left(1 + 2\left(\frac{y_j}{1000}\right)^4\right)$
Parking	$\frac{1}{30} \cdot \left(1 + 9\left(\frac{y_j}{c_j}\right)^4\right)$
Walking	$\frac{1}{4}$

Table 5.3.: Delays on the different edges for scenario 1

We move on to the optimization layer. Weights are all chosen equal to 1. As we discussed before, weights only have meaning when a user is involved. We consider all nine parking nodes in our optimization. Lastly, we impose a maximum capacity on every parking facility of 500, and a global maximum of 4500 (which effectively is redundant). The optimization restrictions are shown in Table 5.4. We choose both the number of iterations and the population size equal to 100, as argued in Subsection 4.2.2.

	(w_1, w_2, w_3)	n_{max}	c_{max}	B	gm
Scenario 1	(1,1,1)	9	500	{1,2,3,4,5,6,7,8,9}	4500

Table 5.4.: Optimization restrictions for scenario 1

The resulting solutions for every method with their corresponding fitness values are given in Table 5.5. In Figures 5.6, 5.7, 5.8, 5.9, 5.10, and 5.11 the fitness, total capacity, total delay, and total distance for the best solution in every iteration are plotted.

We see that the solutions obtained by the methods fp_19/20, fp_1/2, r_19/20, and r_1/2 have similar fitness values. The fitness of the solution found using r_1 is also still close to the fitness values of the solutions found with these four methods, but for the method fp_1 the obtained fitness value is significantly smaller than for the other methods. This might be explained by the fact that the fitness values of individuals are close to each other, in which case fitness proportional parent selection is less appropriate. Apparently,

Method	Solution	Fitness
fp_1	(0, 418, 240, 9, 221, 144, 0, 0, 0)	0.000271
fp_19/20	(490, 229, 12, 223, 0, 0, 0, 0, 12)	0.000382
fp_1/2	(480, 25, 52, 341, 0, 0, 0, 17, 0)	0.000372
r_1	(205, 234, 0, 498, 0, 0, 0, 5, 0)	0.000356
r_19/20	(470, 2, 0, 476, 0, 0, 0, 3, 0)	0.000384
r_1/2	(489, 162, 0, 242, 16, 0, 0, 0, 2)	0.000383

Table 5.5.: Results for the different evolutionary algorithms

when keeping a fraction of the best solutions in every iteration for the next iteration (so $p_r \neq 1$), this difference between fitness proportional and rank selection does not occur.

Looking at the plots, we observe for the methods fp_1 and r_1 that the fitness value of the best solution in each iteration decreases in some cases. This is because every iteration, the entire population is replaced by the offspring. Furthermore, it seems like the fitness values for the solutions using fp_19/20 and r_19/20 increase faster than for the solutions using fp_1/2 and r_1/2 respectively.

At first glance, the solutions may seem a bit strange. Firstly, mostly the parking facilities close to the origin instead of the destinations are used. An explanation is that the distance covered by car significantly decreases as people have to walk more. This increases the travel time, but apparently, the decrease in distance is larger, which highlights the trade-off between the three variables. We observe this decrease in total distance and increase in total delay in the plots as well. Secondly, to make walking possible, at least the capacities for either the parking nodes corresponding to the neighboring nodes of the destinations, or for the parking nodes corresponding to the destinations themselves, should be chosen non-zero. Otherwise, for at least one of the OD pairs, there would not be a possible road, as there is no walking link between parking nodes that both have a capacity of 0. This gives some unwanted behavior as now capacities close to 0 are chosen, just to allow drivers to walk from a node close to the origin to the destination.

A naive and intuitive solution in this scenario would be to divide 1000 parking spots over the parking facilities corresponding to nodes 3, 7, and 9. Ideally, based on the OD matrix, we would want to assign 200 parking spots to the parking nodes corresponding to nodes 3 and 7 and 600 parking spots to the parking nodes corresponding to node 9. However, as there is a maximum of 500 parking spots, we could instead assign 250 and 600 parking spots respectively. This solution, (0, 0, 250, 0, 0, 0, 250, 0, 500), has a fitness of roughly 0.0002132, which is significantly smaller than the fitness values obtained by our proposed methods.

In conclusion, it seems like all the methods, except for fp_1, provide a method to find parking capacities yielding a high value of the objective function of Equation 4.1. However, we don't know how close the obtained solutions are to the actual optimal solution. On this small example network, it is already computationally infeasible to apply grid search to find the exact optimum. To be able to achieve this, a smaller network should be considered, which is done in the next section.

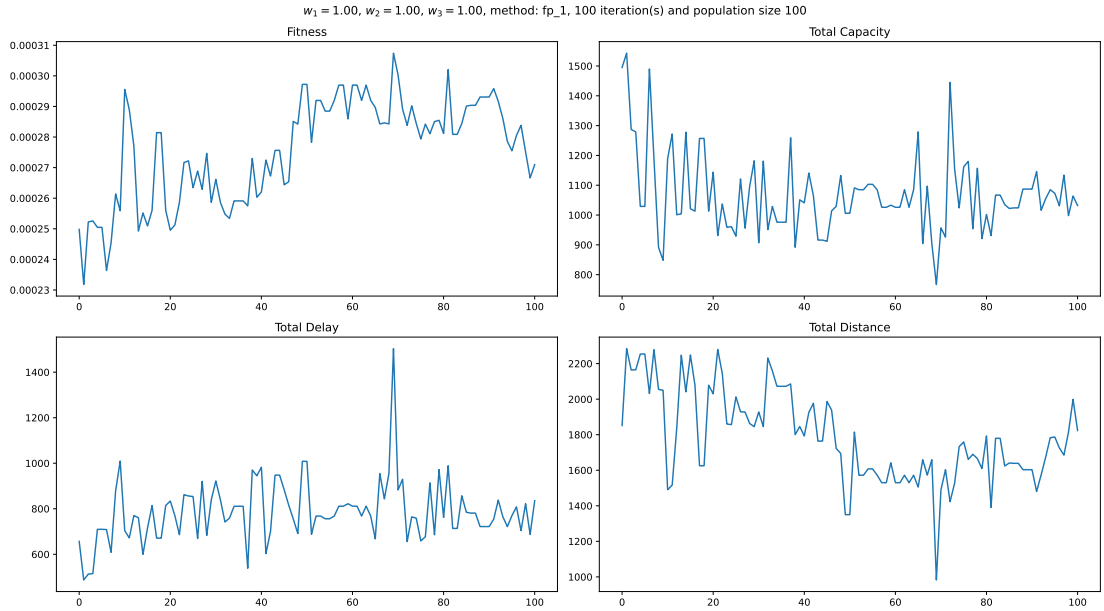

Figure 5.6.: Fitness, total capacity, total delay, and total distance for method fp_1

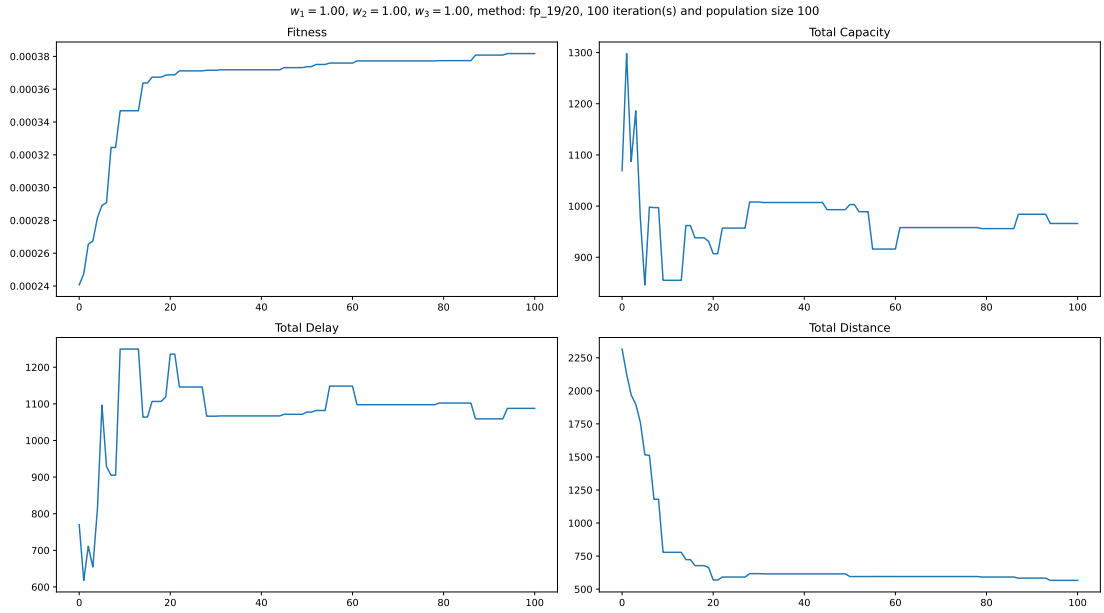

Figure 5.7.: Fitness, total capacity, total delay, and total distance for method fp_19/20

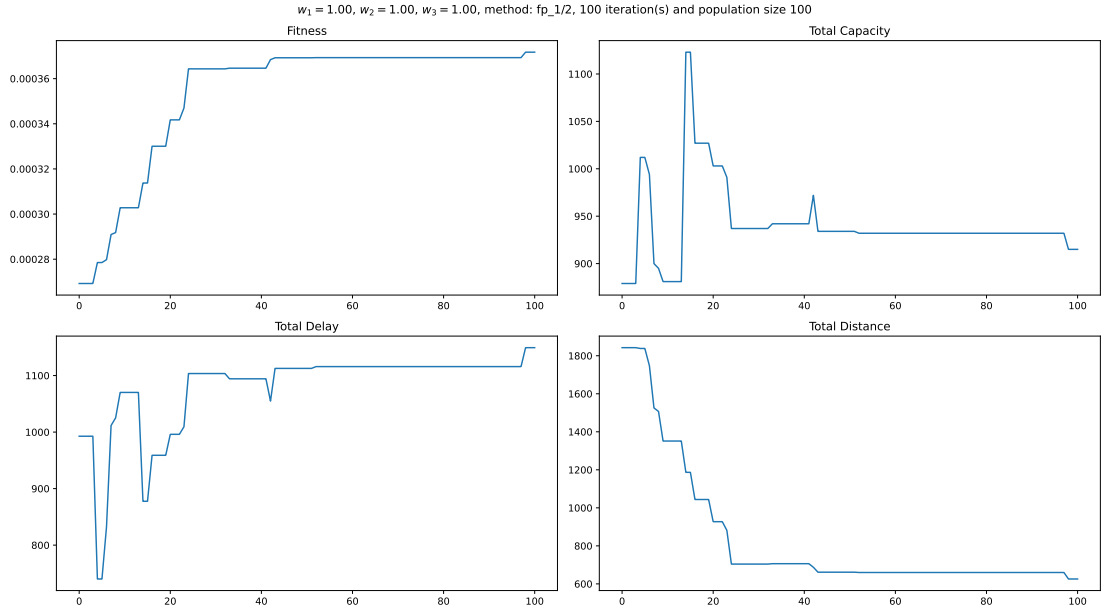

Figure 5.8.: Fitness, total capacity, total delay, and total distance for method fp_1/2

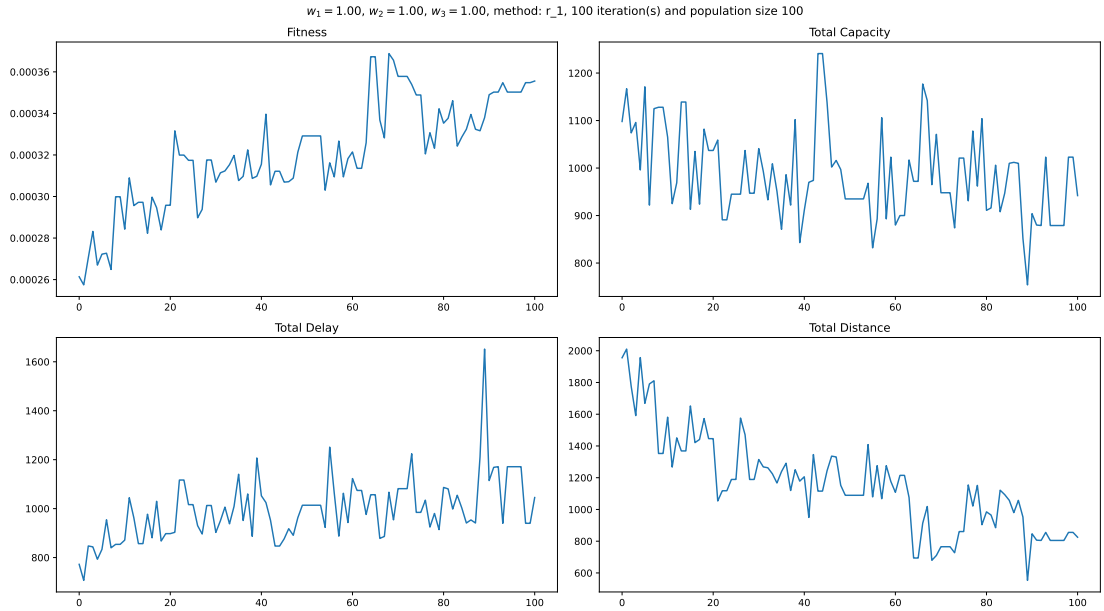

Figure 5.9.: Fitness, total capacity, total delay, and total distance for method r_1

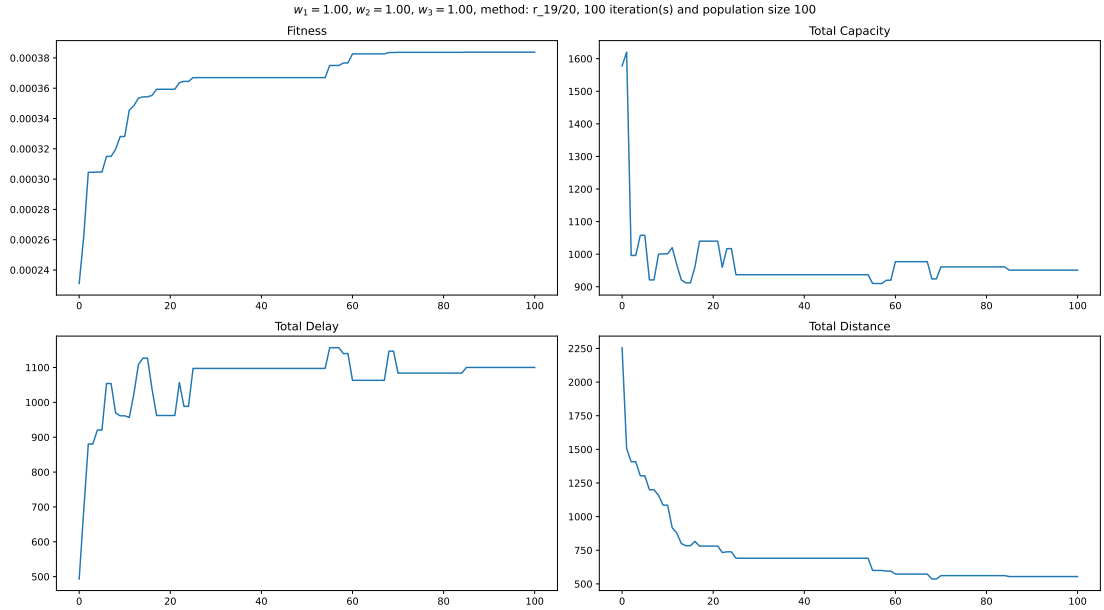

Figure 5.10.: Fitness, total capacity, total delay, and total distance for method r_19/20

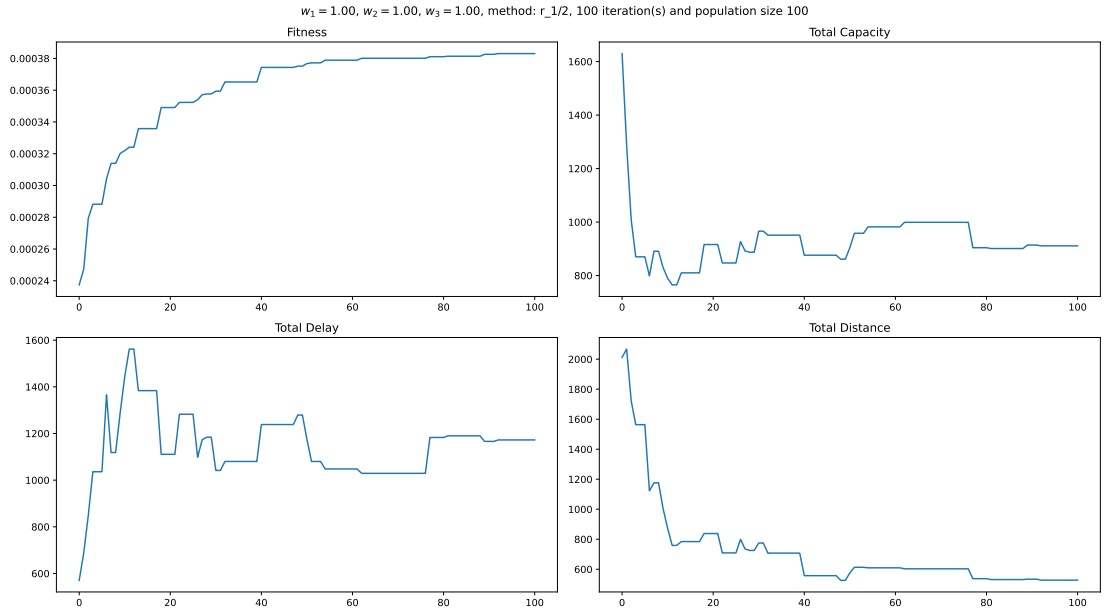

Figure 5.11.: Fitness, total capacity, total delay, and total distance for method r_1/2

5.2. Scenario 2: Validation and Robustness

We continue by applying our proposed methods as described in Table 4.5 to different scenarios. For each scenario, we compare the obtained solutions to the optimal solution. To be precise, we apply the proposed methods 50 times and then we compare the fitness values of the solutions obtained by these experiments with the optimal fitness values obtained using grid search. We do this by looking at the sample means and standard deviations and by looking at the sample quantiles using boxplots. As mentioned before, performing a grid search is computationally infeasible for the network shown in Figure 5.1. Therefore, we need to consider a smaller network which we will specify later.

We start by applying the methods in a baseline scenario. We then test if our proposed methods are robust to changes in the objective function weights. This robustness check is done to see how the proposed methods deal with the difficult trade-off between the three variables we want to minimize, namely total distance covered by cars, total travel times, and total capacities. Also, we check what happens when we change the optimization restrictions. In particular, when we enforce a global maximum on the total parking capacities. As described in Subsection 4.2.3, this is dealt with by setting fitness values of infeasible solutions equal to 10^{-14}. The question is if our proposed methods are able to deal with this 'trick'. Lastly, we check what happens when a particular parking facility is not considered. In particular, a parking facility that is normally included in the optimal solution. This leads to the following six different scenarios: one baseline scenario, three scenarios where we put more weight on one of the three variables in the objective function, one scenario where we impose a global maximum, and one scenario where we do not consider one particular parking facility. In all scenarios, just like in scenario 1, we choose both the number of iterations and the population size equal to 100.

As we discussed, to be able to validate the results obtained by the proposed methods, we consider a smaller network, which is given by Figure 5.12. The coordinates of every node are given in Table 5.13, from which the length of every road can be deduced. We set the walking limit $wl = 1$, such that walking links can only be added as in Figure 5.12. Again, when two neighboring parking facilities both have a capacity of 0, a walking link does not get added though. The network consists of three OD pairs shown in Table 5.14. The demands are very small, to make sure grid search is computationally feasible.

We choose the delays on the links almost exactly as in the previous section, but now the edges have different lengths instead of only length 1. Also, as the total demand is equal to 12 instead of 1000, we choose a capacity of 12 for every road. Again, this choice is made to ensure that all roads are equally as accessible for every driver. For clarity, the delays on every edge are presented again in Table 5.15, with the delays slightly altered compared to the previous section, where s_j is the length of edge j.

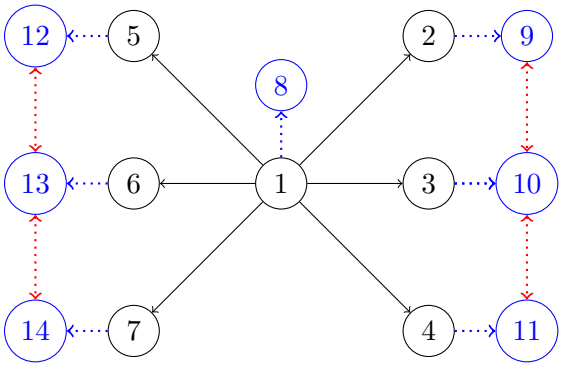

Figure 5.12.: Scenario 2: Star network

Node	Coordinate
1	(0, 0)
2	(2, 1)
3	(2, 0)
4	(2, -1)
5	(-2, 1)
6	(-2, 0)
7	(-2, -1)

Figure 5.13.: Coordinates of the nodes

OD pair	Demand
(1, 2)	2
(1, 4)	5
(1, 6)	5

Table 5.14.: OD matrix for scenario 2

Type	$D_j(y_j)$
Road	$\frac{s_j}{50} \cdot \left(1 + 2\left(\frac{y_j}{12}\right)^4\right)$
Parking	$\frac{1}{30} \cdot \left(1 + 9\left(\frac{y_j}{c_j}\right)^4\right)$
Walking	$\frac{s_j}{4}$

Table 5.15.: Delays on the different edges for scenario 2

In scenario 1 we concluded that all methods except the method fp_1 are able to find parking capacities to obtain high objective function values. We argued that this shortcoming of fp_1 is potentially caused by both the entire population being replaced by the offspring in every iteration and the fitness values of the individuals being very close to each other. We also saw that the method r_1 performs slightly worse compared to the other four methods. In the case of the method r_1, this is also potentially caused by the entire population being replaced by the offspring in every iteration. Therefore, we expect similar conclusions for scenario 2. However, as the considered network is smaller, fitness values should be larger, and thus fitness values may not be as close to each other as in scenario 1. So potentially, the method fp_1 may not perform as badly compared to the other methods as it did in scenario 1.

5.2.1. Scenario 2a: Baseline

We start with the baseline scenario. We set all weights equal to 1. We apply a maximum of three parking facilities with a maximum capacity of 10. The global maximum is equal to 30 (so technically redundant) and we consider every possible location for the parking facilities. The optimization restrictions are summarized in Table 5.16.

	(w_1, w_2, w_3)	n_{max}	c_{max}	B	gm
Scenario 2a	(1,1,1)	3	10	{1,2,3,4,5,6,7}	30

Table 5.16.: Optimization restrictions for scenario 2a

The sample means and standard deviations of the obtained fitness of every method are shown in Table 5.17. The boxplot showing the sample quantiles is given in Figure 5.18. We see that the mean fitness values obtained by all methods are close to the optimal fitness value. The boxplot also suggests that all methods can appropriately find a solution with fitness close to the optimum.

Method	Mean Fitness	Standard deviation
fp_1	0.02358	0.00015
fp_19/20	0.02366	0.00003
fp_1/2	0.02365	0.00004
r_1	0.02359	0.00018
r_19/20	0.02365	0.00006
r_1/2	0.02365	0.00005
Grid search	0.02370	-

Table 5.17.: Summary of the resulting fitness values for scenario 2a

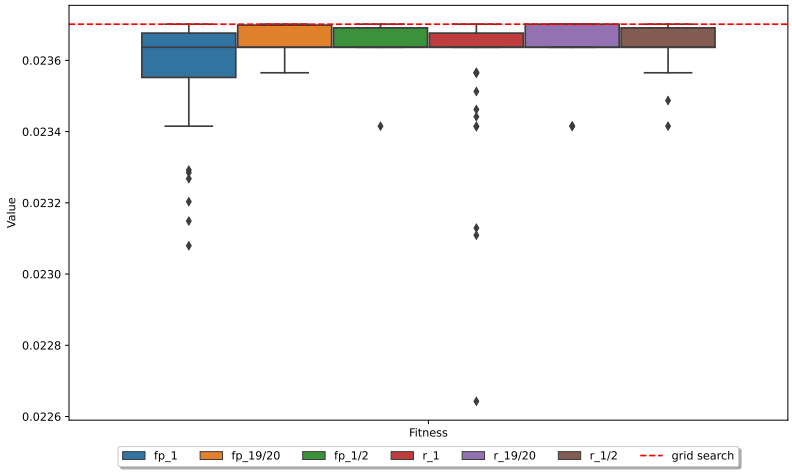

Figure 5.18.: Boxplot of the resulting fitness values for scenario 2a

5.2.2. Scenario 2b: More Importance on Capacities

In this scenario, we keep everything the same as in Scenario 2a, but now we increase the importance of the capacities by setting the corresponding weight equal to 3. The optimization restrictions are summarized in Table 5.19.

	(w_1, w_2, w_3)	n_{max}	c_{max}	B	gm
Scenario 2b	(3,1,1)	3	10	{1,2,3,4,5,6,7}	30

Table 5.19.: Optimization restrictions for scenario 2b

The sample means and standard deviations of the obtained fitness of every method are shown in Table 5.20. The boxplot showing the sample quantiles is given in Figure 5.21. Just as in the baseline scenario, these results suggest that all methods can appropriately find a solution with fitness close to the optimum. However, it is clear that the method fp_1 is the least appropriate method in this scenario.

Method	Mean Fitness	Standard deviation
fp_1	0.01537	0.00022
fp_19/20	0.01552	0.00010
fp_1/2	0.01546	0.00017
r_1	0.01550	0.00010
r_19/20	0.01550	0.00014
r_1/2	0.01548	0.00016
Grid search	0.01556	-

Table 5.20.: Summary of the resulting fitness values for scenario 2b

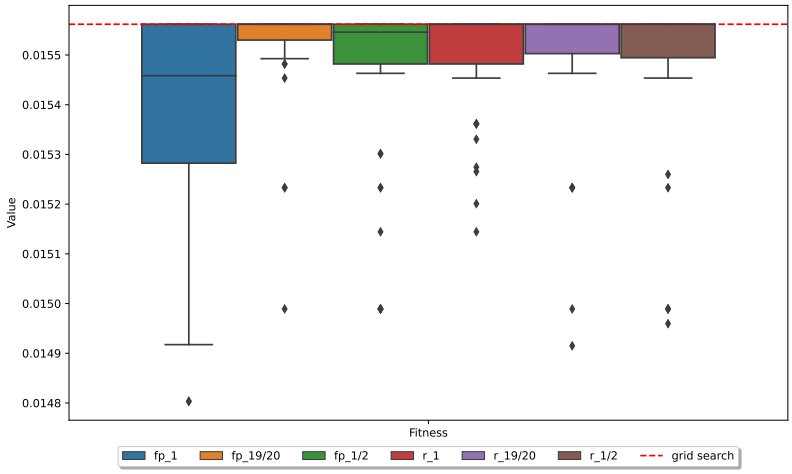

Figure 5.21.: Boxplot of the resulting fitness values for scenario 2b

5.2.3. Scenario 2c: More Importance on Travel Time

Instead of increasing the importance of the capacities, we now increase the importance of the total delay by assigning a weight of 3 to this variable. The optimization restrictions are summarized in Table 5.22.

	(w_1, w_2, w_3)	n_{max}	c_{max}	B	gm
Scenario 2c	(1,3,1)	3	10	{1,2,3,4,5,6,7}	30

Table 5.22.: Optimization restrictions for scenario 2c

The sample means and standard deviations of the obtained fitness of every method are shown in Table 5.23. The boxplot showing the sample quantiles is given in Figure 5.24. Once more, these results suggest that all methods are able to obtain a solution with a fitness value close to the optimum. The method fp_19/20 was even able to do so 50 out of 50 times. In this scenario, the methods fp_1 and r_1 clearly seem to be less appropriate than the other four methods.

Method	Mean Fitness	Standard deviation
fp_1	0.02025	0.00036
fp_19/20	0.02076	0
fp_1/2	0.02071	0.00015
r_1	0.02042	0.00043
r_19/20	0.02074	0.00009
r_1/2	0.02066	0.00023
Grid search	0.02076	-

Table 5.23.: Summary of the resulting fitness values for scenario 2c

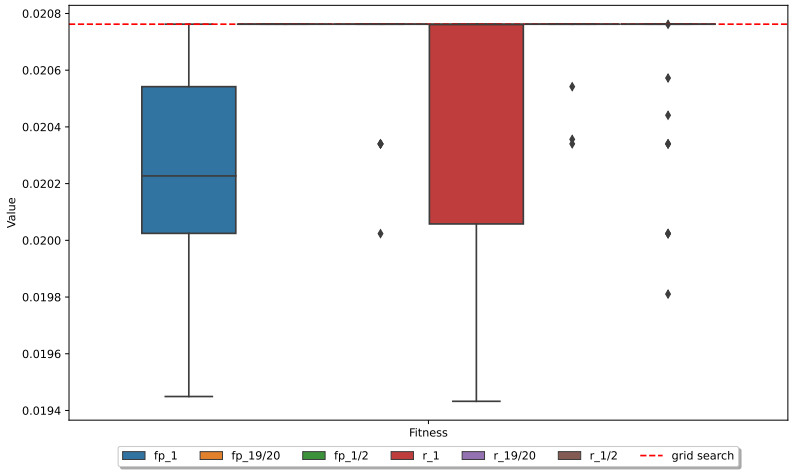

Figure 5.24.: Boxplot of the resulting fitness values for scenario 2c

5.2.4. Scenario 2d: More Importance on Distance

Instead of increasing the importance of the capacities or the importance of the total delay, we now increase the importance of the total distance by assigning a weight of 3 to this variable. The optimization restrictions are summarized in Table 5.25.

	(w_1, w_2, w_3)	n_{max}	c_{max}	B	gm
Scenario 2d	(1,1,3)	3	10	{1,2,3,4,5,6,7}	30

Table 5.25.: Optimization restrictions for scenario 2d

The sample means and standard deviations of the obtained fitness of every method are shown in Table 5.26. The boxplot showing the sample quantiles is given in Figure 5.27. We get a similar conclusion as in scenario 2b: from all methods a solution can be obtained with fitness close to the optimum, but it is clear that the method fp_1 is the least appropriate method in this scenario.

Method	Mean Fitness	Standard deviation
fp_1	0.01103	0.00006
fp_19/20	0.01107	0.00002
fp_1/2	0.01107	0.00002
r_1	0.01107	0.00004
r_19/20	0.01107	0.00001
r_1/2	0.01107	0.00002
Grid search	0.01107	-

Table 5.26.: Summary of the resulting fitness values for scenario 2d

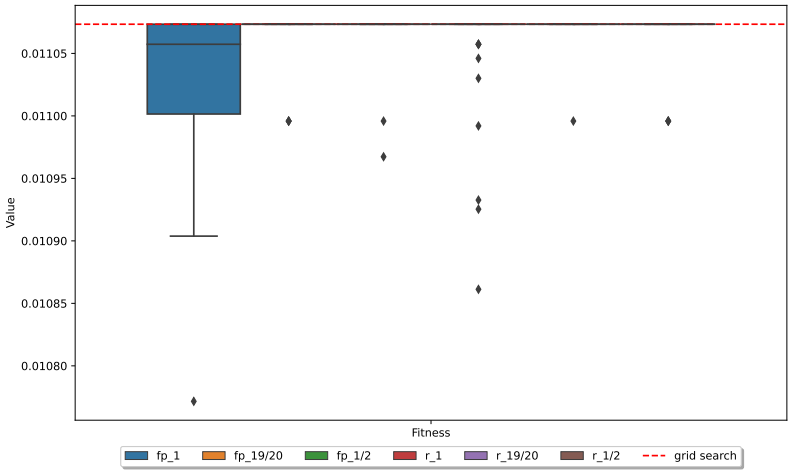

Figure 5.27.: Boxplot of the resulting fitness values for scenario 2d

5.2.5. Scenario 2e: Restricted Global Maximum

In this scenario, we keep everything the same as in the baseline example, except that the global maximum is put to 25. The optimization restrictions are summarized in Table 5.28.

	(w_1, w_2, w_3)	n_{max}	c_{max}	B	gm
Scenario 2e	(1,1,1)	3	10	{1,2,3,4,5,6,7}	25

Table 5.28.: Optimization restrictions for scenario 2e

The sample means and standard deviations of the obtained fitness of every method are shown in Table 5.29. The boxplot showing the sample quantiles is given in Figure 5.30. Note that the optimum is the same as in scenario 2a. We can draw the same conclusions as for scenario 2a: all methods can appropriately find a solution with fitness close to the optimum. However, the methods do perform slightly worse compared to scenario 2a.

Method	Mean Fitness	Standard deviation
fp_1	0.02353	0.00021
fp_19/20	0.02366	0.00005
fp_1/2	0.02363	0.00019
r_1	0.02361	0.00010
r_19/20	0.02365	0.00006
r_1/2	0.02364	0.00007
Grid search	0.02370	-

Table 5.29.: Summary of the resulting fitness values for scenario 2e

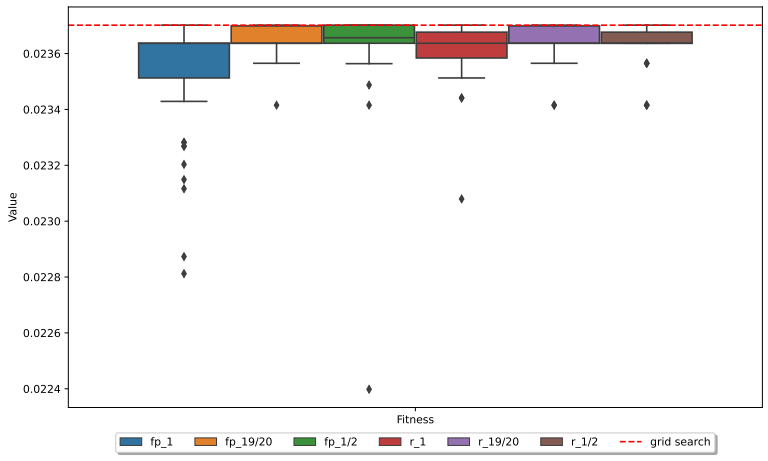

Figure 5.30.: Boxplot of the resulting fitness values for scenario 2e

5.2.6. Scenario 2f: Restrict to a Subset

Lastly, it turns out that in every scenario, the parking facility corresponding to node 6 gets assigned non-zero capacity in the optimum computed by grid search. That is why we also consider a case where we exclude this parking facility. The optimization restrictions are then presented in Table 5.31.

	(w_1, w_2, w_3)	n_{max}	c_{max}	B	gm
Scenario 2f	(1,1,1)	3	10	{1,2,3,4,5,7}	30

Table 5.31.: Optimization restrictions for scenario 2f

In this scenario, it turns out two optimal solutions exist, which makes it an interesting robustness check. The sample means and standard deviations of the obtained fitness of every method are shown in Table 5.32. The boxplot showing the sample quantiles is given in Figure 5.33. Also in this last scenario, we can conclude that all methods are able to produce solutions with fitness values close to the optimal fitness value.

Method	Mean Fitness	Standard deviation
fp_1	0.02233	0.00006
fp_19/20	0.02236	0.00003
fp_1/2	0.02236	0.00004
r_1	0.02234	0.00004
r_19/20	0.02236	0.00004
r_1/2	0.02236	0.00005
Grid search	0.02240	-

Table 5.32.: Summary of the resulting fitness values for scenario 2f

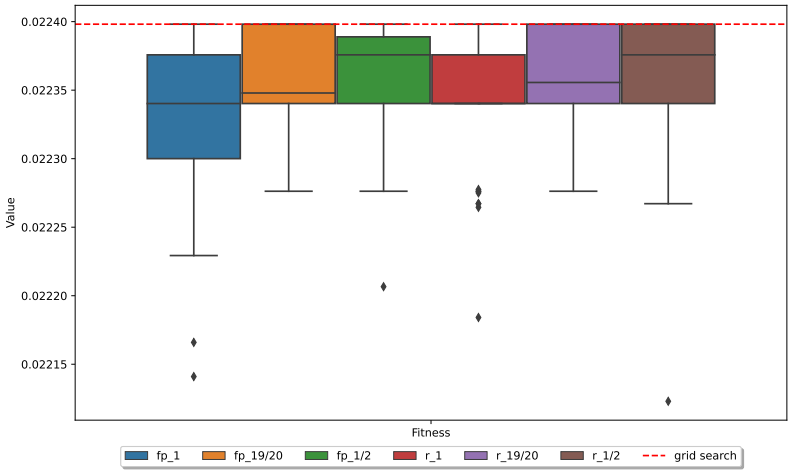

Figure 5.33.: Boxplot of the resulting fitness values for scenario 2f

5.2.7. Conclusions

By comparing with grid search, we can conclude that all methods can give solutions with fitness values close to the optimal fitness value in different scenarios of the simple network presented in Figure 5.12. It seems like in some scenarios the methods fp_1 and r_1 obtain solutions with on average smaller fitness values compared to the other four methods. Looking at the boxplots, we also see that this is not only due to some negative outlier, but that these two methods just perform worse than the other four methods in these scenarios.

Furthermore, there may not be a big difference in the resulting fitness values of the other four methods, but it does seem like the method fp_19/20 does the best job. In all scenarios, there is no method with a larger mean fitness value. Only in scenario 2d the method r_19/20 has a slightly smaller standard deviation, but in all other scenarios, the standard deviation of the fitness values obtained by the method fp_19/20 are the smallest.

As we predicted, the higher fitness values with potentially larger differences between the fitness values of individuals cause the method fp_1 to perform not as badly compared to the other method as in scenario 1. However, in scenarios 2b and 2d, where fitness values are significantly smaller than in the other scenarios, this difference in performance between fp_1 and the other methods is again clearly visible. The results in scenario 2c are a bit surprising though. Travel time is the variable that is subjected to change the most though, as a small change in parking capacities can have a large effect on the park search time and therefore also the travel time. So it could be possible that replacing the entire population with the offspring in every iteration is punished more severely. This then would cause the methods fp_1 and r_1 to perform significantly worse than the other four methods in this scenario.

In conclusion, the four methods fp_19/20, fp_1/2, r_19/20, and r_1/2 are all very suitable to optimize parking capacities in the case of this small example network. It seems like the method fp_19/20 is the most appropriate method based on the results on this small network.

However, the results seem to suggest that when fitness values become smaller, they also become closer to each other. For larger networks, fitness values will become smaller, as travel times, distances, and parking capacities generally increase. For this reason, it could be beneficial to use a ranking selection scheme for the parent selection instead, so then the method r_19/20 might be the most suitable. In the end though, as we have argued multiple times throughout this thesis, because of the NFL theorems, this does not mean by default that these methods can get similar performance on other networks.

5.3. Scenario 3: Case Study of Delft, the Netherlands

We end this chapter by applying (some of) our methods to a real-life example, namely the city of Delft in the Netherlands. We concluded when applying our methods to the smaller example networks that the methods fp_19/20 and r_19/20 were the most suitable. In this section, we want to find out if these methods can also successfully be applied to the real-life network of Delft.

As we discussed before, by using Python, we cannot successfully parallelize the Frank-Wolfe algorithm. Therefore, applying it to a network like the city of Delft would be computationally infeasible with our current implementation of the traffic and parking model.

For this reason, we use the implementation of the traffic and parking model of Delft provided by TNO. In other words, we replace our own implementation with the implementation of TNO. However, we do not change anything about the optimization part. This leads to an adjusted scheme for the methodology which is given by Figure 5.34.

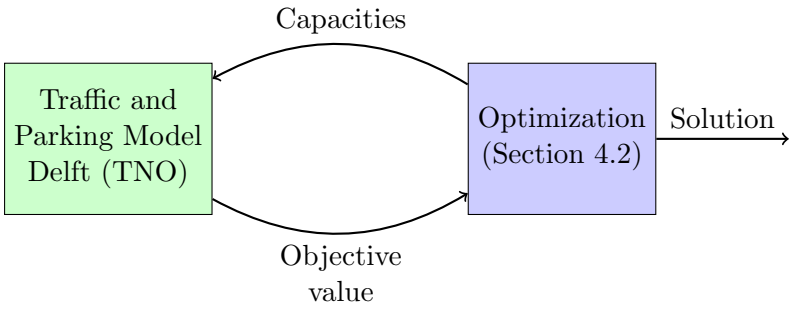

Figure 5.34.: Adjusted Scheme for Case Study

5.3.1. Traffic and Parking Model Delft (TNO)

The implementation of the traffic model of Delft by TNO does have a lot of similarities with our implementation. We briefly discuss this implementation and how it compares to ours. First, the network of Delft is translated into a graph. However, the implementation of TNO also takes into account the structure of the network, so for example turns. The resulting network consists of 1490 nodes and 2887 edges. The delay functions on the edges are still BPR functions but with different parameters specifically chosen for Delft. On top of this, 25 zones are defined and every zone gets assigned a certain node, which can be interpreted as the center of the zone. There are 625 OD pairs based on real data of travelers in Delft, but values are scaled, so do not represent reality directly. OD pairs consist of an origin zone and a destination zone.

Parking is also added to the traffic model of TNO, but this addition is still at an early stage. In every zone, a parking node gets added which represents the aggregate of parking facilities in that particular zone. Just as in our implementation, a BPR function is also used for the park search time, also with different parameters than for

the roads. These parameters are not calibrated yet specifically for Delft though. By default, the capacity of every parking facility is equal to 10000. Obviously, these are not the real parking capacities of the network of Delft, this is still a shortcoming of this implementation. Moreover, as the implementation is right now, it is only possible to change five parking capacities simultaneously.

Just like our implementation, the implementation of TNO does also allow walking links between parking facilities. Also in the implementation of TNO, a walking limit is imposed, and this limit is 1 km.

To compute the Wardrop equilibrium, the implementation of TNO uses the Frank-Wolfe algorithm just as our implementation uses. However, in the case of the implementation of TNO, the Wardrop equilibrium gets updated when certain aspects, like parking capacities, are adjusted. So in the beginning flows are computed for some base situation, and every time when parameters get updated, flows get adjusted with the current flows as a starting point. In our implementation, the Wardrop equilibrium is computed from scratch every time.

Although this implementation makes it possible for us to apply our optimization methods to the real-life example of Delft, there are some issues we need to consider:

- **Destination of travelers:** The overwhelming majority of drivers want to travel out of Delft. These drivers most likely want to travel to cities like Rotterdam, the Hague, or maybe even Amsterdam. Their destination is then a zone at the edge of Delft, but obviously, they would not have to park there. We would want to know the trip purpose of every driver and incorporate this into the model.

- **Limitations of parking model:** As we already mentioned, the addition of parking in the implementation of TNO is still at an early stage. We would prefer to use the actual parking capacities in a zone instead of the 10000 that is chosen by default now. Furthermore, we would want to be able to change all 25 parking capacities at the same time instead of only 5.

- **Handling invalid OD pairs:** When the parking capacity of a certain zone is equal to 0 and there is no walking link with another zone, a route to that zone does not exist. The implementation of TNO then ignores all the OD pairs with this zone as the destination and computes the Wardrop equilibrium without these. However, our current optimization method needs to know when this occurs.

- **Computational Limitations:** The way the Wardrop equilibrium gets computed does not allow us to parallelize the evolutionary algorithm. There is only one instance of the network and we cannot compute multiple equilibria simultaneously. This is not an issue right now as we have not parallelized our evolutionary algorithms, as Python does not allow us to. However, this can be an issue when we want to speed up our implementation by parallelizing it using C++ for example.

Regardless of these issues, we can still apply our implementation of parking capacity optimization to this traffic and parking model of Delft and find out if it can actually be applied to real-life cases. We move on to describing the actual case we will consider.

5.3.2. Case Description

As we discussed, we can only optimize over 5 out of the 25 zones. We consider 4 zones close to or inside the city center of Delft, namely zone 9, 10, 11, and 12. These zones are illustrated in Figure 5.35. Table 5.36 presents the (scaled) total demand for each zone as a destination, while Table 5.37 shows whether there is a walking link between zones or not. The position of zone 12 may suggest that it is a zone where drivers leave Delft. Yet, the drivers have to leave Delft through other zones where they can leave the ring road around Delft (zones 1, 3, 4, 6, 7, and 22). So if they use zone 12 to leave the center of Delft, they still have to go to one of these zones from there, and this would be their destination zone. Moreover, there are also zones on the right of zone 12. Therefore we argue that drivers with destination zone 12 actually have zone 12 as their real destination and want to park their car there.

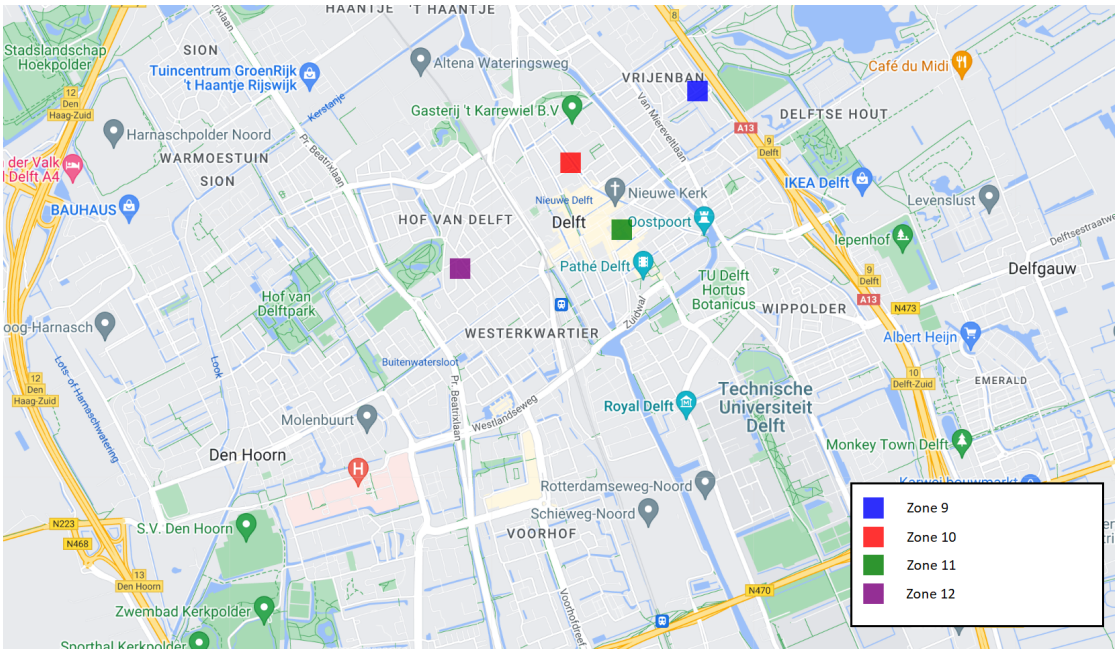

Figure 5.35.: Scenario 3: Delft, the Netherlands

Destination	Demand
9	2223
10	1887
11	1965
12	2673

Table 5.36.: (Scaled) demands for destination zones considered in scenario 3

	9	10	11	12
9	-	✓	✓	✗
10	✓	-	✓	✓
11	✓	✓	-	✓
12	✗	✓	✓	-

Table 5.37.: Walking links between zones considered in scenario 3

We continue by discussing the optimization part of this case study. Instead of considering all integers as possible parking capacities, we only consider multiples of 100 to keep the solution space somewhat limited.

Note that the total delay is computed for the entire network consisting of 25 zones. This is because changing parking capacities can change flows throughout the entire network. However, this does mean that capacities are more subject to change than the total delay. The parking capacities for all other zones are kept equal to 10000. Furthermore, only parking in 4 (central) zones out of 25 zones is considered, while the overwhelming majority of drivers want to travel out of Delft. For this reason, we assign a smaller weight to the total capacities and a larger weight to the total delay, namely 0.1 and 1 respectively. Furthermore, the implementation of TNO does not provide us with the total distance covered by cars, so by default, we have to assign a weight of 0 to the total distance.

We consider the following case. Suppose the city of Delft wants to remove parking facilities in at least one zone, so at most three zones are allowed to have non-zero parking capacities. On top of this, suppose the city of Delft has enough budget to double parking facilities such that the maximum capacity for every zone is equal to 20000. Just to be complete, we specify a global maximum of 60000 which is redundant in this case.

As we have discussed, the implementation of TNO simply ignores OD pairs when a route does not exist. However, as we also argued, in our current implementation, we want to assign a fitness value of 10^{-14} when this happens. To make sure this does not happen, we have to make sure there is at least one incoming walking link to every zone. In this case, it suffices to make sure at least two parking capacities are non-zero. We denote this minimum amount of parking capacities by n_{min}. Adding this restriction as a real restriction to Equation 4.1 gives the same complications as discussed for the global maximum in Subsection 4.2.3: the crossover and mutation operators can cause solutions to violate this restriction. Therefore, we also handle this minimum by assigning a fitness value of 10^{-14} to solutions having less than two non-zero parking capacities.

The optimization restrictions are summarized in Table 5.38. Just as in scenarios 1 and 2, we choose both the number of iterations and the population size equal to 100.

	(w_1, w_2, w_3)	n_{max}	c_{max}	B	gm	n_{min}
Scenario 3	(0.1,1,0)	3	20000	{9, 10, 11, 12}	60000	2

Table 5.38.: Optimization restrictions for scenario 3

In conclusion, we try to answer two questions regarding the optimization of parking capacities in Delft:

1. If we want to remove parking in at least one zone and in at most two zones among zones 9, 10, 11, and 12 in Delft (Figure 5.35), which zone(s) should we select to maximize the objective function with the chosen weights?

2. Which parking capacities should we choose for the remaining zones to maximize the objective function with the chosen weights?

5.3.3. Results

The resulting solutions for the two methods with their corresponding total delays and fitness values are given in Table 5.39. Also, the total delay and fitness value of the original situation, where all capacities are equal to 10000, are provided. In Figures 5.40 and 5.41 the fitness, total capacity, and total delay for the best solution in every iteration are plotted for both methods.

Method	Solution	Total delay	Fitness
fp_19/20	(0, 100, 0, 5900)	20573	$4.723 \cdot 10^{-5}$
r_19/20	(100, 0, 0, 5800)	20591	$4.721 \cdot 10^{-5}$
Baseline	(10000, 10000, 10000, 10000)	20443	$4.091 \cdot 10^{-5}$

Table 5.39.: Results for the two methods applied to Delft

The solutions of the two methods provide a significantly higher fitness value than the original solutions where all capacities are equal to 10000. The fitness values of the solutions obtained by the two methods are very similar to each other.

In both cases the solutions are also similar: only two parking capacities are non-zero. Moreover, in both cases, zone 12 is chosen to have a non-zero capacity. The other zone that has non-zero capacity differs, but in both cases, it gets assigned the minimum number of capacities, namely 100.

So in principle, it seems like only parking in zone 12 is necessary in terms of maximizing our objective function. However, as this would cause certain routes to be impossible, one of the other zones is required to have non-zero capacities. This is similar to what happened in scenario 1.

In both scenarios, the capacities for zone 12 are chosen approximately equal to 6000. However, in both cases, it turns out that only roughly 3500 drivers park their cars in zone 12. Moreover, less than 200 drivers want to park their cars in the other zone that gets assigned non-zero capacity. The rest of the drivers wanting to travel to zones 9, 10, 11, and 12 park their cars in another zone. This difference between capacity and parking intensity in zone 12 showcases the trade-off we face. As zones 10 and 11 have zero capacity and zone 13 has a constant capacity, decreasing the capacity in zone 12 implies an increase in total travel time. If this increase is too large, it is not beneficial anymore to further decrease capacities.

Looking at the plots, it also seems like the method r_19/20 provides a faster increase in the fitness value of the best solution. Already after less than 10 iterations the fitness value of the best solution is approximately $4.72 \cdot 10^{-5}$. For the method fp_19/20 this takes more than 50 iterations. However, after less than 10 iterations, the fitness of the best solution is still already $4.70 \cdot 10^{-5}$. This could confirm our hypothesis that r_19/20 is a more suitable method as fitness values become smaller. To really test this hypothesis, we would have to conduct more experiments though.

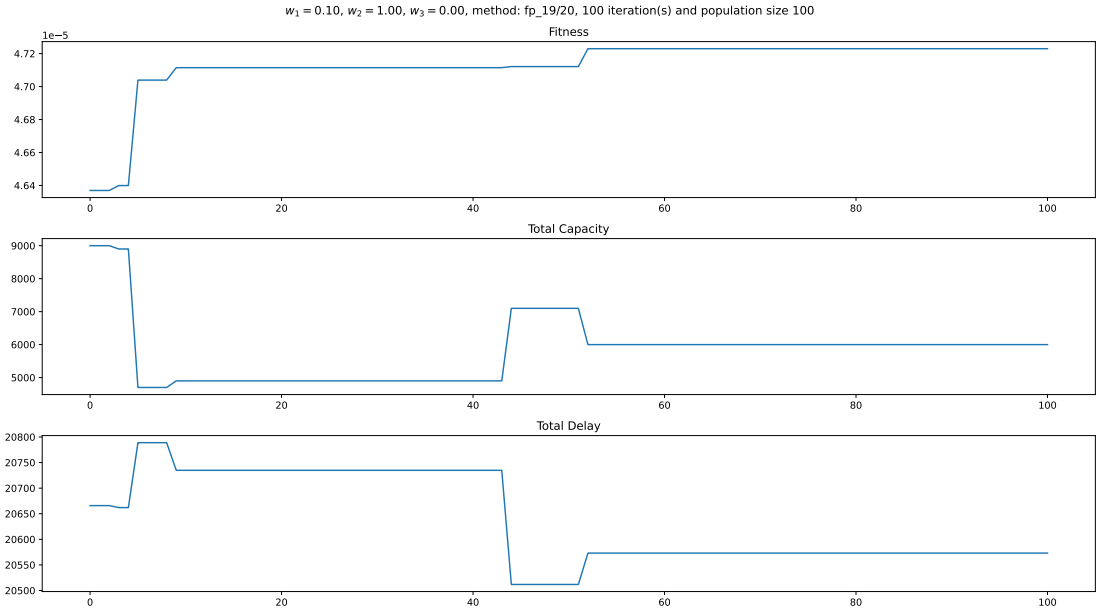

Figure 5.40.: Fitness, total capacity, and total delay for method fp_19/20

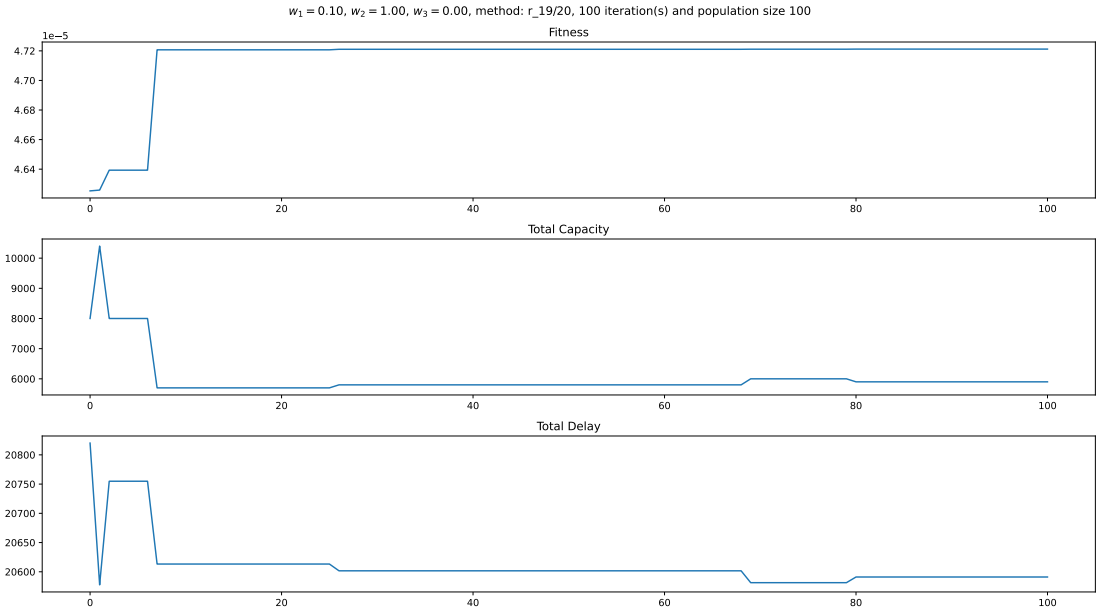

Figure 5.41.: Fitness, total capacity, and total delay for method r_19/20

5.3.4. Conclusions and Implications

In conclusion, we have provided an answer to the two questions we asked in this case study. The answer to the first question is that offering parking in zone 12 seems to be necessary to maximize the objective function. A reason could be that the demand as a destination for zone 12 is higher than for zone 9, 10, and 11. The answer to the second question is that the parking capacity of zone 12 should be chosen roughly equal to 6000, which is significantly larger than the number of cars parking there due to the mentioned trade-off between the capacities and the travel time.

We saw an increase in fitness from roughly $4.09 \cdot 10^{-5}$ to roughly $4.72 \cdot 10^{-5}$ for both methods we applied, which corresponds to an increase of approximately 15 %. The results also seem to imply that our hypothesis about parent selection is correct: when dealing with very small fitness values, r_19/20 outperforms fp_19/20. As we discussed though, to confirm this hypothesis, we would have to conduct more experiments.

It is difficult to directly quantify any concrete gains made by optimizing parking capacities in this case study as the traffic and parking model still lacks realistic parking data for the current situation in Delft. Nevertheless, this case study illustrates that we can apply our parking capacity optimization method to real-life scenarios. This could be achieved by creating our own traffic and parking model as we have discussed throughout this thesis. However, this case study has shown that we can use any traffic and parking model as long as it can be used as follows:

- **Input:** As input, we want to provide a vector or list of parking capacities corresponding to nodes or zones.
- **Output:** As output, we want to obtain total travel times and total distance covered by cars.
- **Handling invalid input:** As mentioned before, the implementation of TNO ignores OD pairs for which routes are impossible. For our parking capacity optimization method, we would want the traffic and parking model to explicitly tell us when parking capacities are invalid.

If these three conditions are satisfied, our method can be applied to optimize parking capacities, which also highlights its generalizability.

Conclusion

In this thesis, we have provided a method to optimize parking capacities to minimize a weighted average of emissions, travel time, and land used for parking. As we discussed in Chapter 1, no similar research has yet been conducted. We achieved this by answering two sub-questions we defined in the same chapter:

1. How can we simulate traffic and parking in an urban area?
2. Which techniques can be used to optimize parking capacities and how can these methods be applied exactly?

To provide an answer to the first question, we started in Chapter 2 by defining a road network without parking. Such a network exists of nodes representing intersections and edges representing roads. These edges have a corresponding delay function which is a function of the flow. We continued this chapter by defining the Wardrop equilibrium, a situation in which no driver in the network has an incentive to change their route. We also argued that it is infeasible to compute the Wardrop equilibrium by hand for larger networks and that it is necessary to consider a numerical method. We provided such a numerical method, namely the Frank-Wolfe algorithm. We also gave a proof that, under certain circumstances, the Frank-Wolfe algorithm converges. We finished this chapter with some practical considerations.

In Section 4.1, we proposed a way to add parking to the traffic network. We extended the traffic network by adding parking nodes to intersection nodes where the edge between them represents the park search time. On top of this, we also introduced walking links which make it possible for travelers to park their car at a different node than their destination and walk from there to their destination. We finished by discussing the implications this addition has on the Wardrop equilibrium and its computation.

To answer the second question, we began with a general introduction to optimization in Chapter 3. We argued that finding the exact optimum is computationally infeasible when the solution space is too large. Therefore, it is necessary to consider methods that are only able to provide a sufficiently good solution to an optimization problem, which we call a metaheuristic. We continued by extensively discussing such a class of metaheuristics, namely evolutionary algorithms. We finished this chapter by briefly talking through some other methods and how they compare with evolutionary algorithms.

In Section 4.2 we gave a mathematical formulation of our optimization problem and the corresponding objective function. In the same section, we also explained how we can apply evolutionary algorithms to this specific optimization problem. We proposed to try two different methods to select parents: fitness proportional selection and ranking selection. On top of this, we also proposed trying three different survivor selection

methods based on the fraction of the best solutions that survive to the next iteration. This results in six different methods. In Section 4.3 we discussed how we can implement everything by putting the optimization part together with the traffic and parking model.

We finished the main body of this thesis with some experiments in Chapter 5. We started by applying the six proposed methods to a simple example in Section 5.1. We concluded that except for the method fp_1, our proposed methods are able to find parking capacities that yield high objective function values. However, we don't know how close the obtained fitness values are to the optimal fitness value, as it was already computationally infeasible to compute the optimal value for this example.

In Section 5.2 we considered an even smaller network for which it was computationally feasible to compute the optimum using grid search. We also analyzed how sensitive our methods are to changes in the optimization framework. We concluded that except for two out of the six proposed methods, our methods are able to find parking capacities that yield an objective function value close to the optimum. The methods that were not able to do so were the two methods where we let the entire population be replaced by the offspring in every iteration, namely fp_1 and r_1. The method that seemed to provide us with solutions closest to the exact optimum in the examples of this section, was fp_19/20. However, it seemed like ranking selection may be preferred over fitness proportional parent selection when fitness values become very small. This happens when networks become very large. Therefore it is also good to consider the method r_19/20.

So from these two experiments, we indeed concluded that we have found multiple variants of evolutionary algorithms to optimize parking capacities to maximize our objective function. To achieve this, it turned out that it is essential to let part of the best solutions obtained in a certain iteration survive to the next iteration. This is a deviation from the popular genetic algorithm proposed by Holland (1975), where the entire population is replaced every iteration.

Finally, we applied the two methods fp_19/20 and r_19/20 in a case study of the city of Delft in the Netherlands. These methods were able to find parking capacities yielding a significantly higher fitness value than the fitness value of the original situation. We saw an increase of approximately 15% in the fitness value. Although both methods were able to provide solutions with similar fitness, it did seem like the method r_19/20 converges faster in this scenario. Therefore, also based on the results from the smaller examples, we would suggest using this method to optimize parking capacities. However, more research should be conducted. As we have mentioned, the current traffic and parking model of Delft still has its shortcomings and does not exactly represent reality yet. However, this case study highlights the adaptability of our proposed method to large real-life urban networks and thus shows it can provide benefits to urban planning. It also showcases the generalizability of our optimization method as it can be applied to other traffic and parking models.

Discussion and Future Work

We finish this thesis by discussing the results and conclusions we obtained. We also propose some future work based on this discussion.

We have discussed throughout this thesis how Python is not suitable for parallel programming. However, as we also mentioned, parallelizing the Frank-Wolfe algorithm can provide us with a lot of computational gains. Therefore it is a good idea to consider another programming language, for example, C++. We also noted that using the current implementation of the traffic and parking model of Delft by TNO, we cannot parallelize the evolutionary algorithm. If it turns out that in practice it is impossible to use a parallel implementation of the evolutionary algorithm, it may be necessary to consider one of the other methods we briefly discussed, like simulated annealing.

With the help of a parallel implementation of the Frank-Wolfe algorithm, computing the Wardrop equilibrium will take considerably less time. This means it is possible to look at larger networks than what is done in this thesis. In particular, grid search is also computationally feasible for larger networks than the one we have considered. This way, it is possible to validate our proposed methods on larger networks than what is done in Section 5.2.

Right now, it is difficult to give an interpretation of the weights we assign to every variable in the objective function. In every iteration, we could try to standardize every variable before computing the corresponding objective function values by subtracting the sample mean and dividing this by the sample variance. However, this means negative objective function values can occur, which we want to avoid. An alternative is normalizing variables by subtracting the minimum and dividing it by the difference between the maximum and minimum. Even so, the maximum and minimum of the variables change through different iterations, so it is questionable if this is appropriate. Alternative methods to better understand the choice of the weights in the objective function should probably be considered.

In Sections 5.1 and 5.3, it occurred that certain nodes were chosen to have non-zero parking capacity just for the sake of having a parking facility. This is because this allows a walking link to go through this node. As walking is not dependent on the flow and as it does not contribute to the total distance covered by car, letting travelers walk could help increase the objective function value. Therefore, it could be interesting to consider minimizing total walking distance as an objective, as proposed by Chen et al. (2001).

The traffic and parking model we have defined in this thesis can still be improved to fit better with reality. We think the following aspects should be considered first:

- **Include trip purpose:** As discussed in Section 5.3, it is necessary to exclude drivers that want to travel out of an urban area as these do not need to park.

Also, for example, people traveling to their own homes, often have assigned parking spaces and thus no park search time. This is also something that should be taken into account.

- **Make a distinction between off-street and on-street parking:** There is a big difference between off-street parking and on-street parking. In the case of off-street parking, drivers typically directly go to the parking facility. In the case of on-street parking, drivers typically drive randomly around their destination to find a parking space. This also yields very different park search times. Moreover, on-street parking is often way less space efficient than off-street parking. A method to make this distinction is for example provided in Pel and Chaniotakis (2017).
- **Extension to multi-modal traffic model:** The current model only considers cars. However, we could also include cycling and public transport. This way we could also analyze if changing parking capacities for example forces travelers to choose other, more sustainable, modes of traffic.

In conclusion, in this thesis, we have successfully provided a method to optimize parking capacities. Nonetheless, we have provided several suggestions for future research. These suggestions can be summarized as follows:

- Using another programming language, make use of parallel programming.
- Find a method to make it possible to interpret the weights of variables in the objective function.
- Add minimizing total walking distance as one of the objectives.
- Assign trip purposes to every driver.
- Make a distinction between off-street and on-street parking.
- Extend the traffic model to a multi-modal model.

We hope that these suggestions can help towards improving the optimization of parking capacities in urban areas even further.

Popular Summary

According to CBS (2021), the car is the most used mode of transport in the Netherlands. Car usage goes paired with fossil fuel emissions and traffic congestion. Parking plays a crucial role: according to Bonsall and Palmer (2004), up to 40 of the travel time when traveling to central urban areas is used to find a parking space. Furthermore, parking facilities take up a lot of space which can be used for other purposes.

The capacities of parking facilities play a crucial role: they are an important factor in the time it takes to park your car and they are an indication of the space occupied by parking facilities. The time it takes to park your car is an important factor in the route you choose, which determines your travel time and (partly) determines your emissions. The goal of this thesis is to design a method to find parking capacities in an urban area with three objectives:

- Minimize emissions;
- Minimize travel times;
- Minimize the amount of land occupied by parking spaces.

To achieve this, we start by discussing a way to translate traffic and parking in an urban area into mathematics. The idea is that we can represent roads and intersections using a graph. On every road, drivers experience a certain delay, which depends on how many other cars are on the same road. We represent this by adding weights to the edges as a function of the flow. An example is given by Figure 5.42.

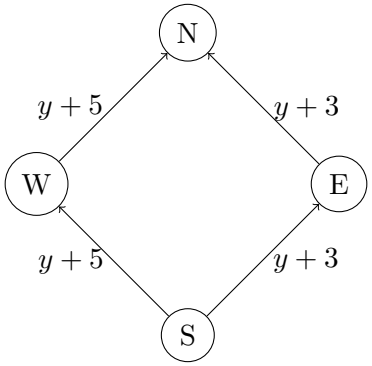

Figure 5.42.: Example of a basic traffic network with flows

An important assumption we make is that every driver tries to minimize the time they have to travel from their origin to their destination. Based on how many drivers

want to travel and where they want to travel, we can compute the number of drivers on each road. This results in the so-called *Wardrop equilibrium*. For readers who have some knowledge of game theory, this equilibrium is similar to the *Nash equilibrium*. On top of this, we also propose a method to extend traffic networks with parking, for which it turns out that the Wardrop equilibrium still exists.

The number of possible parking capacities we can choose is in practice often very large. Therefore, it is computationally infeasible to find the best capacities exactly, but it is still possible to find parking capacities close to the best capacities. The method we use to do this is the evolutionary algorithm, which is inspired by biological evolution.

In this thesis, we applied this algorithm to find parking capacities with the objectives mentioned before. We saw that this algorithm was indeed able to find parking capacities close to the best capacities. Finally, we also showed that our algorithm can be applied to real-life networks. We did this by applying it to the city of Delft in the Netherlands.

Bibliography

Abdelfatah, A.S. and Taha, M.A. (2014). "Parking Capacity Optimization Using Linear Programming". In: *Journal of Traffic and Logistics Engineering* 2, pp. 176–183.

ANWB Verkeersinformatie (2023). *Vijftien procent filegroei in eerste halfjaar 2023.* URL: `https://www.anwb.nl/verkeer/nieuws/nederland/2023/juli/filezwaarte-juli-2023` (visited on 08/05/2023).

Austin, T.W. and Lee, M.J. (1973). "Estimation of Potential Use of Peripheral Parking for Los Angeles CBD". In: *Highway Research Board* 444, pp. 20–26.

Bäck, T. (1992). "Self-Adaptation in Genetic Algorithms". In: *Proceedings of the 1st European Conference on Artificial Life*, pp. 263–271.

Bekhor, S., Ben-Akiva, M.E., and Ramming, M.S. (2006). "Evaluation of Choice Set Generation Algorithms for Route Choice Models". In: *Annals of Operations Research* 144.1, pp. 235–247.

Bonsall, P.W. and Palmer, I.A. (2004). "Modelling Drivers' Car Parking Behaviour Using Data from a Travel Choice Simulator". In: *Transportation Research Part C: Emerging Technologies* 12.5, pp. 321–347.

Canon, M.D. and Cullum, C.D. (1968). "A Tight Upper Bound on the Rate of Convergence of Frank-Wolfe Algorithm". In: *SIAM Journal on Control* 6.4, pp. 509–516.

Cascetta, E. et al. (2002). "A Model of Route Perception in Urban Road Networks". In: *Transportation Research Part B: Methodological* 36.7, pp. 577–592.

CBS (2021). *Hoeveel reisden inwoners van Nederland en hoe?* URL: `https://www.cbs.nl/nl-nl/visualisaties/verkeer-en-vervoer/personen/hoeveel-reisden-inwoners-van-nederland-en-hoe-` (visited on 06/16/2023).

Chen, J. et al. (2001). "Planning Method of Urban Parking Facilities' Locating Model with its Genetic Algorithm". In: *China Journal of Highway and Transport* 14.1, pp. 85–88.

Csikós, A., Tettamanti, T., and Varga, I. (2015). "Macroscopic Modeling and Control of Emission in Urban Road Traffic Networks". In: *Transport* 30.2, pp. 152–161.

D'Acierno, L., Gallo, M., and Montella, B. (2006). "Optimisation Models for the Urban Parking Pricing Problem". In: *Transport Policy* 13, pp. 34–48.

De Jong, K.A. (1975). "An Analysis of the Behavior of a Class of Genetic Adaptive Systems". PhD thesis. University of Michigan.

Eiben, A.E and Smith, J.E. (2003). *Introduction to Evolutionary Computing.* 1st ed. Natural Computing Series. Heidelberg: Springer Berlin.

Frank, M. and Wolfe, P. (1956). "An Algorithm for Quadratic Programming". In: *Naval Research Logistics Quarterly* 3.1-2, pp. 95–110.

Gemeente Amsterdam (2023). *Druktebeeld*. URL: https://druktebeeld.amsterdam.nl/ (visited on 05/23/2023).

Glover, F. (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence". In: *Computers & Operations Research* 13.5, pp. 533–549.

Haurie, A. and Marcotte, P. (1985). "On the Relationship between Nash-Cournot and Wardrop Equilibria". In: *Networks* 15.3, pp. 295–308.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. 1st ed. Ann Arbor: University of Michigan Press.

Jaggi, M. (2013). "Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization". In: *Proceedings of the 30th International Conference on Machine Learning*. Vol. 28. 1. PMLR, pp. 427–435.

Kelly, F.P. and Yudovina, E. (2014). *Stochastic Networks*. 1st ed. Cambridge: Cambridge University Press.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. (1983). "Optimization by Simulated Annealing". In: *Science* 220.4598, pp. 671–680.

Kuys, D. (2022). "Er is in Nederland meer parkeerruimte voor auto's dan woonruimte voor mensen". In: *VPRO*. URL: https://www.vpro.nl/programmas/tegenlicht/lees/artikelen/2022/fietsprofessor.html (visited on 06/20/2023).

Lam, W.H.K. et al. (2006). "Modeling Time-dependent Travel Choice Problems in Road Networks with Multiple User Classes and Multiple Parking Facilities". In: *Transportation Research Part B: Methodological* 40.5, pp. 368–395.

Manikas, T.W. and Cain, J.T. (1996). *Genetic Algorithms vs. Simulated Annealing: A Comparison of Approaches for Solving the Circuit Partitioning Problem*. Tech. rep. The University of Pittsburgh.

Miller, B.N. and Ranum, D.L. (2011). *Problem Solving with Algorithms and Data Structures Using Python*. 2nd ed. Portland: Franklin, Beedle & Associates.

Muijlwijk, H. (2012). "Static Traffic Assignment with Junction Modelling". MSc Thesis. University of Twente.

Negnevitsky, M. (2011). *Artificial Intelligence: A Guide to Intelligent Systems*. 2nd ed. Essex: Pearson Education Limited.

Nocedal, J. and Wright, S. (2006). *Numerical Optimization*. 2nd ed. Springer Series in Operations Research and Financial Engineering. New York: Springer Science+Business Media.

Pel, A.J. and Chaniotakis, E. (2017). "Stochastic User Equilibrium Traffic Assignment with Equilibrated Parking Search Routes". In: *Transportation Research Part B: Methodological* 101.C, pp. 123–139.

Pierce, G., Willson, H., and Shoup, D. (2015). "Optimizing the Use of Public Garages: Pricing Parking by Demand". In: *Transport Policy* 44, pp. 89–95.

Ruan, J.M. et al. (2016). "How Many and Where to Locate Parking Lots? A Space–time Accessibility-Maximization Modeling Framework for Special Event Traffic Management". In: *Urban Rail Transit* 2, pp. 59–70.

Ryan, J.M. (1979). *Urban Transportation Planning System (UTPS): The Community Aggregate Planning Model (CAPM) Users' Guide*. Report - Urban Mass Transporta-

tion Administration. Federal Highway Administration, [Office of Highway Planning], Urban Planning Division, Technical Branch.

Sheffi, Y. (1985). *Urban Transportation Networks: Equilibrium Analysis With Mathematical Programming Methods*. 1st ed. Englewood Cliffs: Prentice-Hall.

Shen, T., Hua, K., and Liu, J. (2019). "Optimized Public Parking Location Modelling for Green Intelligent Transportation System Using Genetic Algorithms". In: *IEEE Access* 7, pp. 1–1.

Wikipedia (2023a). *Double-precision Floating-point Format — Wikipedia, The Free Encyclopedia*. URL: `https://en.wikipedia.org/wiki/Double-precision_floating-point_format` (visited on 08/02/2023).

— (2023b). *Simulated Annealing — Wikipedia, The Free Encyclopedia*. URL: `http://en.wikipedia.org/w/index.php?title=Simulated%5C%20annealing&oldid=1154654876` (visited on 05/19/2023).

Wolpert, D.H. and Macready, W.G. (1997). "No Free Lunch Theorems for Optimization". In: *IEEE Transactions Evolutionary Computing* 1, pp. 67–82.

Zhou, X. (2018). "On the Fenchel Duality between Strong Convexity and Lipschitz Continuous Gradient." arXiv preprint arXiv:1803.06573.

A. Boundedness of Curvature Constant

In this appendix, we provide a proof of the boundedness of the curvature constant C_f in the context of the Frank-Wolfe algorithm applied to find the Wardrop equilibrium. We start with proving an equivalence of three statements. The proof of this equivalence is based on Zhou (2018).

Lemma A.1. *Let f be differentiable and let ∇f be L-Lipschitz continuous. Then the following three statements are equivalent:*

(i) $(\nabla f(y) - \nabla f(w))^T(y - w) \leq L||y - w||^2 \quad \forall y, w.$

(ii) The function $g(y) := \frac{1}{2}y^T y - f(y)$ *is convex.*

(iii) $f(w) - f(y) - f(y)^T(w - y) \leq \frac{L}{2}||w - y||^2 \quad \forall y, w.$

Proof.

- (i) $\iff$ (ii): Suppose (i) holds. We get

$$\begin{aligned}(\nabla f(y) - \nabla f(w))^T(y - w) \leq L||y - w||^2, \quad \forall y, w \iff \\ (L(y - w) - (\nabla f(y) - \nabla f(w))^T(y - w) \geq 0, \quad \forall y, w \iff \\ (\nabla g(y) - \nabla g(w))^T(y - w) \geq 0, \quad \forall y, w,\end{aligned}$$

  which is equivalent with $g(y) := \frac{1}{2}y^T y - f(y)$ being convex by the monotone gradient condition for convexity.

- (ii) $\iff$ (iii): Suppose now that $g(y) := \frac{1}{2}y^T y - f(y)$ is convex. Then using the first-order condition for convexity, this is equivalent to $g(w) \geq g(y) + \nabla g(y)^T(w - y)$ for all y, w. We then have

$$\begin{aligned}g(w) \geq g(y) + \nabla g(y)^T(w - y), \quad \forall y, w \iff \\ \frac{L}{2}w^T w - f(w) \geq \frac{L}{2}y^T y - f(y) + (Ly - \nabla f(y))^T(w - y), \quad \forall y, w \iff \\ f(w) - f(y) - \nabla f(y)^T(w - y) \leq \frac{1}{2}||w - y||^2, \quad \forall y, w,\end{aligned}$$

  which concludes the proof.

□

We can now prove a useful property of differentiable functions with L-Lipschitz continuous gradient.

Lemma A.2. *Let f be differentiable and let ∇f be L-Lipschitz continuous. Then it holds that $f(w) - f(y) - \nabla f(y)^T(w-y) \leq \frac{L}{2}||w-y||^2$.*

Proof. By the L-Lipschitz continuity of ∇f, we have

$$||\nabla f(y) - \nabla f(w)|| \leq L||y-w||^2, \quad \forall y, w.$$

Using the Cauchy-Schwarz inequality, we then obtain

$$(\nabla f(y) - \nabla f(w))^T (y-w) \leq L||y-w||^2, \quad \forall y, w.$$

We can now apply Lemma A.1 to conclude that

$$f(w) - f(y) - f(y)^T(w-y) \leq \frac{L}{2}||w-y||^2$$

□

Finally, we can prove that the curvature constant C_f is indeed bounded in the context of the Frank-Wolfe algorithm.

Theorem A.3. *Let f be a convex and continuously differentiable function defined on a compact convex set $\mathcal{D}$. Additionally, let ∇f be L-Lipschitz continuous. Then it holds that $0 \leq C_f \leq L \cdot diam(\mathcal{D}) < \infty$.*

Proof. First, note that by convexity of f, it holds that $f(w) - f(y) - \langle w-y, \nabla f(y)\rangle \geq 0$ for all $w, y \in \mathcal{D}$, so $C_f \geq 0$. As ∇f is L-Lipschitz continuous, we can apply Lemma A.2 and we then see that

$$\begin{aligned}
C_f &= \sup_{\substack{y,z\in\mathcal{D} \\ \alpha\in[0,1] \\ w=y+\alpha(z-y)}} \frac{2}{\alpha^2}\left(f(w) - f(y) - \langle w-y, \nabla f(y)\rangle\right) \\
&\leq \sup_{\substack{y,z\in\mathcal{D} \\ \alpha\in[0,1] \\ w=y+\alpha(z-y)}} \frac{2}{\alpha^2}\frac{L}{2}||w-y||^2 \\
&= \sup_{\substack{y,z\in\mathcal{D} \\ \alpha\in[0,1]}} \frac{L}{\alpha^2}||\alpha(z-y)||^2 \\
&= \sup_{y,z\in\mathcal{D}} L||z-y||^2 \\
&= L \cdot \mathrm{diam}(\mathcal{D})
\end{aligned}$$

We assumed $\mathcal{D}$ to be compact, so by the Heine-Borel Theorem, we know that $\mathcal{D}$ is bounded. It then follows immediately that C_f is indeed bounded as well. □