Technische Universität München

Fakultät für Mathematik
Lehrstuhl für Angewandte Geometrie und Diskrete Mathematik

Traffic Network Optimization

an Approach Combining Genetic Algorithms and Nonlinear Programming

Master's Thesis by Lucia Fogelstaller

Examiner:	Prof. Dr. Peter Gritzmann
Advisor:	Dr. Michael Ritter
Submission Date:	30.09.2014

I hereby confirm that this is my own work, and that I used only the cited sources and materials.

München, 30.09.2014

Lucia Fogelstaller

Danksagung

Ich möchte mich vor allem bei Dr. Michael Ritter für die umfangreiche Betreuung meiner Masterarbeit bedanken. Danke, dass du dir stets Zeit für mich genommen hast um meine vielen Fragen ausführlich zu beantworten, mir dabei immer konstruktive Kritik mitgegeben hast und mir außerdem immer „Rückendeckung“ bei der Benutzung des Servers gegeben hast.

Vielen Dank an die ganze MBA Gruppe, insbesondere an Prof. Gröger und Dr. Leuschner, die mir stets hilfreich zur Seite standen. Ihr Interesse an meiner Masterarbeit hat mich sehr gefreut und der Austausch mit Ihnen hat mir immer sehr gut getan. Danke dafür.

Ein großer Dank geht außerdem an meine Eltern, die mir dieses Studium erst ermöglicht haben. Ihr habt mich nicht nur finanziell, sondern vor allem moralisch unterstützt und mir immer den Rücken gestärkt wenn es mal nicht so gut lief. Ohne euch hätte ich das nicht geschafft. Vielen, vielen Dank.

Abstract

Traffic network optimization aims to improve a given road network by maximizing the utility for traffic participants. The present thesis investigates a solution-based approach for tackling the traffic network optimization problem by modifying an established metaheuristic—the genetic algorithm—in such a way that mutation does not appear at random but with the help of a local optimization procedure. In comparison to the standard genetic algorithm, the approach using local optimization in form of a neighbourhood search should result in increased solutions. Primarily, we have to make a statement regarding how well a network performs with respect to traffic participants. One opportunity might be to approximate the utility based on the travel time of road users. This evaluation forms the basis of the neighbourhood search. Finally, by means of practical data we compare and assess the aforementioned enhanced approach with the standard genetic algorithm in terms of quality and development.

Zusammenfassung

Die Optimierung von Verkehrsnetzen hat das Ziel, ein gegebenes Straßennetz dahingehend zu verbessern, dass der Nutzen der Verkehrsteilnehmer in diesem Straßennetz maximiert wird. Die vorliegende Arbeit beschäftigt sich mit einem Lösungsansatz für das Problem der Verkehrsnetz-Optimierung, indem eine bewährte Metaheuristik – der Genetische Algorithmus – so modifiziert wird, dass die Mutationsphase nicht zufällig geschieht, sondern mittels lokaler Optimierung. Im Vergleich zu einem allgemeinen Genetischen Algorithmus soll dieser Ansatz der lokalen Optimierung in Form einer Nachbarschaftssuche zu besseren Lösungen führen. Dabei muss zunächst eine Aussage darüber getroffen werden, inwiefern ein Netzwerk für Verkehrsteilnehmer als nützlich angesehen werden kann. Eine Möglichkeit besteht darin, den Nutzen durch die Fahrzeit im Verkehrsnetz zu approximieren. Diese Bewertung stellt dann die Basis der Nachbarschaftssuche dar. Abschließend wird dieser erweiterte Ansatz mit dem allgemeinen Genetischen Algorithmus hinsichtlich Qualität und Entwicklung der Lösungen anhand von Daten aus der Praxis verglichen und bewertet.

Contents

Chapter 1

Problem Description

Almost everyone knows the feeling: While driving to work in the morning—a time when it is already hectic—you only get slowly ahead or even find traffic coming to a standstill. In such an annoying situation, some of us may wonder why the streets are not extended or why no additional streets are being built even though they are sorely needed. Though it might be obvious to us where the new streets are most urgently needed, the decision concerning where they should be built is by no means an easy task. The difficulty lies in understanding traffic behaviour. Although every road user decides by him-/herself which route to take to work in the morning, this must be estimated and included in the final determination of traffic routing, which is the job of Munich's Planning and Building Department in the aforementioned city. However, before starting with the construction of new roads, it is difficult for them to understand the effects of the planned enhancements. For this reason, we discuss and develop approaches that are able to forecast traffic behaviour after road construction to provide decision-making support. Let us examine the concrete research question. Our objective is to ascertain which new roads should be built and which streets should be expanded to ensure a smooth flow of traffic in the future. Thus, we are interested in finding optimal construction plans with respect to traffic flow in the considered area. In the following, we will refer to this issue as 'traffic network optimization'.

In this thesis, we present a technique to optimize traffic networks using genetic algorithms combined with nonlinear programming. Since we deal with traffic flow on roads, we first discuss how traffic behaves in networks. In order to measure how useful a network is, we solve a nonlinear program whereby the objective reflects the travel time of the traffic participants. This concept will provide a basis for exploring more advanced methods, such as those that simulate traffic behaviour. In order to ensure better understanding of traffic flows, these simulation tools additionally consider the day schedules of traffic participants and, thus, their objective is to measure the network utility for traffic participants. The approach developed in this thesis works with algorithms based on evolution; hence, we

will discuss genetic algorithms as a special type of evolutionary algorithms. A simulation tool will serve as an evaluation function. In order to exploit the knowledge of traffic flows from above, we aim to modify the genetic algorithm whereby we combine it with local optimization to improve its performance.

The remainder of this thesis is structured as follows. In chapter 1, we introduce the task of traffic network optimization and sketch the approach to solve it while also explaining the general structure of the thesis. In chapter 2, we mainly focus on modelling the problem of traffic network optimization in a mathematical way. First, we introduce the basic notations and definitions that are necessary for the rest of the thesis. Subsequently, in section 2.2, the aforementioned problem of traffic assignment is formulated as an integer optimization program. Since problems dealing with traffic in networks are usually modelled as network flow problems, we present the main concepts of such network flows in section 2.3. The study of network flows is a well-developed field in operations research since a multitude of problems relating to economics, logistics and informatics underlie a network structure. In particular, flow problems with linear objective functions, such as maximum flow and minimum cost flow problems, have been studied extensively, along with the various algorithms used to solve them. An overview of this topic is provided by Ahuja, Magnata and Orlin [AMO93], while standard reference works in combinatorial optimization, such as Korte and Vygen [KV08] and Schrijver [Sch03], are also dedicated to network flow problems. Special treatment is necessary for problems where we cannot optimize from a central perspective, precisely where actors in the network decide by themselves which route to take. On these grounds, we chiefly follow a more recent book called *Selfish Routing and the Price of Anarchy* from Roughgarden and Tardos [Rou05] while introducing the topic of network flows. Nevertheless, the issue was developed much earlier from Pigou [Pig20] in 1920 and was later taken up in 1952 by Wardrop [WCEGB52], who also gives his name to a mathematical concept called "Wardrop equilibrium". In section 2.3.3, we use the results from the previous sections and derive concrete formulations for our purposes and subsequently approach a preliminary optimization method for the traffic network optimization problem. Furthermore, section 2.3.4 briefly introduces a tool called MATSim [Rie14], which simulates traffic flows in networks and thus uses the schedules of traffic participants (also called agents) that reflect the day plans of road users.

In chapter 3, we focus on solution methods for the traffic network optimization problem. We concentrate on metaheuristics classifying optimization algorithms, which are mostly applied to problems that are NP hard by the theory of computational complexity. A handbook of metaheuristics that covers methods like local search, simulated annealing, tabu search and evolutionary algorithms is provided by Glover [GK03]. In the field of metaheuristics, we find the naturally inspired class of evolutionary algorithms, to which genetic algorithms also belong. These methods use the evolution phenomena and transfer

it to optimization problems. The concept of genetic algorithms was invented by Holland and published in *Adaptation in Natural and Artificial Systems* [Hol75]. A recommendable introduction into the topic of genetic algorithms is provided by Mitchell [Mit96] as well as by Sivanandam and Deepa [SD07]. The course of genetic algorithms in chapter 3 is fairly standard, whereby we cover the different phases of evolutionary algorithms, including selection, mutation, crossover and replacement. We essentially emphasize methods used in the explicit implementation of the program, while also referring to some other techniques advocated in the literature.

Chapter 4 serves to present an implementation of the traffic network optimization problem solved by using two different approaches. First, a standard genetic algorithm will be applied to the problem, which we subsequently alter in the phase of mutation and take the idea of traffic flows from section 2.3 into account. Therefore, we modify the standard genetic algorithm in a way that solves a nonlinear network flow problem to evaluate the current network and its neighbours, before mutating the network into the best one in that neighbourhood. Through this enhanced genetic algorithm, we hope to improve the solution, or rather, improve the iteration steps needed to achieve an eligible solution. Therefore, the algorithm for the further developed method is simply distinguished in one step—namely, the mutation step. The implemented programs are illustrated from two perspectives: The user documentation in section 4.2 provides instructions for handling the program, while we explain the structure from a developer's perspective in section 4.3.

Even though we will exert ourselves to generalize the study of traffic network optimization in chapters 1–4 independent from a specific network, for the comparison of the two implemented algorithms we restrict ourselves to the road network of Perlach in chapter 5, which operates based on real data provided by the Department of Public Order of Munich. In section 5.1, we introduce the data for which we test our algorithms. Section 5.2 first presents the settings of the test runs, and especially parameters for the genetic algorithm, such as the number of iterations and the probabilities of mutation and crossover. Subsequently, we collect and appraise the results from the test runs. We primarily present the solutions with a focus on the differences between the standard genetic algorithm and the further developed algorithm, where we use nonlinear programming to evaluate networks. Thereafter, we concentrate on the algorithm's performance, including a consideration of running times.

Finally, we conclude the underlying work in chapter 6. In order to evaluate the outcome of the thesis, we first provide a short summary of the main results. Moreover, we will offer ideas for future prospects on this topic and ascertain which parts are worth further investigation.

Let us now proceed in more detail with the task of traffic network optimization. We

expect that the utility depends upon the network configuration. On this basis, we consider different scenarios of road construction, with several available projects having this purpose. A single project describes one or more streets that can be either newly built or expanded. Consider that a project schedules a street to be expanded; subsequently, this leads, for example, to building a new lane, whereby we deem that, following the street expansion, the volume of traffic that can pass through will double. It may also happen that in order to build a new road, some other roads have to be reconstructed in such a way that they become narrower, and thus the number of possible vehicles on this road decreases. In addition, every project has some project costs needed to construct the streets under this project. Furthermore, a limited budget is given and must be adhered to. Obviously, the solution might be a realization of a combination of projects, whereby the sum of the respective projects' costs shall not exceed the given budget. In the following thesis, we discuss the task of traffic network optimization and evolve an approach to find eligible solutions.

Chapter 2

Mathematical Modelling

This chapter forms the cornerstone of the thesis. Here, we study the traffic network optimization problem described in chapter 1 from a mathematical point of view. In section 2.1 we introduce some basic notations and definitions needed for the remainder of the thesis. The formulation of the optimization problem as an integer program will be presented in section 2.2 and will constitute the fundament for solving the traffic network optimization problem. Section 2.3 develops techniques for handling traffic flows, whereby we proceed as follows: First, we analyse the "system optimum problem" in section 2.3.1 by acting as a route controller and determining flows that are optimal for the system as a whole. Subsequently, in section 2.3.2, we consider flows that appear if traffic participants make their own choices in such a way that they navigate on their own most favoured route. In section 2.3.3, we apply the knowledge of the previous sections to derive a first approach to the traffic network optimization problem. Finally, section 2.3.4 presents the simulation tool MATSim that tries to simulate real-world behaviour by taking the day's schedules and more constraints into account.

2.1 Basic Definitions

For this section, we operate in the following way. First, we introduce the mathematical definition and notation and, afterwards, we think about the objects to be represented using this mathematical model. Since we deal with a road network, as in fig. 2.1, we have to find a tool to save information about streets, crossroads and connections in this road network.

Definition 2.1
A graph is a pair $\mathcal{G} = (V, E)$ and we call elements of the set V, *vertices* (often *nodes*), and elements of the set E, *edges* (often *links*).

(a) Road network **(b)** Graph

Figure 2.1: Road network and graph representing Perlach

In this thesis, we always assume a finite number of nodes and links. An edge is a pair of vertices $(v_i, v_j) \in E, i \neq j$, which connects $v_i \in V$ with $v_j \in V$. If two vertices v_i and v_j are connected by an edge, we also say that v_i and v_j are *adjacent*. An edge e and a vertex v are *incident* if e connects v with another vertex. For the sake of simplicity, we only enumerate the vertices from $1, \dots, n$ and denote the edges by a pair of numbers corresponding to the nodes, e. g. $(i, j) \in E$. So, we represent streets in a road network by edges and, crossroads, where two or more streets intersect, by vertices. Whenever we talk about graphs in this thesis, we mean directed graphs, i. e. $(i, j) \neq (j, i)$. This facilitates modelling one-way streets in a road network. A sequence of nodes and edges $v_1, e_2, \dots, e_n, v_n$ with $e_i = (v_{i-1}, v_i)$ is called a *walk*; if we additionally claim that all edges must differ, it is called a *path*. We call a $v_1 - v_n$ path a path from v_1 to v_n without specifying the sequence of edges and v_1 is referred to as the *start node* and v_n as the *end node*.

Having introduced the basic definition of a graph, we would now like to extend that definition by a few properties and call it a *network graph*.

Definition 2.2
A network graph is a tuple $\tilde{\mathcal{G}} = (\mathcal{G}, cap, t^0)$ with a graph $\mathcal{G}$, *capacities* $cap : E \to \mathbb{R}_{\geq 0}$ and *free-flow travel times* $t^0 : E \to \mathbb{R}_{\geq 0}$.

As per the above definition, a network graph consists of a graph $\mathcal{G}$ in the sense of Definition 2.1. In addition, we associate a positive number $cap_{ij} \in \mathbb{R}_{\geq 0}$ with every edge (i, j). The capacity cap_{ij} of an edge (i, j) must be understood as follows: If the amount of flow exceeds the capacity, smooth flow can no longer be guaranteed, i. e. capacity is not a hard limit for the number of traffic participants on this road; it is rather that one will get into a traffic jam if capacity is reached. The *free-flow travel time* $t^0_{ij} \in \mathbb{R}_{\geq 0}$ of

an edge (i, j) is the time needed to travel from i to j with a so-called free-speed. The free-speed denotes the possible velocity if no one else travels on the same road.

We now introduce the concept of *networks* that will come up repeatedly and will be necessary for the entire thesis.

Definition 2.3
A network is a tuple $\mathcal{N} = (\tilde{\mathcal{G}}, \mathcal{C}, q, s, d, t)$, where $\tilde{\mathcal{G}}$ represents a network graph as in Definition 2.2, $\mathcal{C}$ a set of commodities, $q, s : \mathcal{C} \to V$ a source and a sink respectively, $d : V \times \mathcal{C} \to \mathbb{R}_{\geq 0}$ the demand and $t : E \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ a travel time function.

Let us clarify Definition 2.3 by considering its elements. Since we are studying traffic participants navigating from one location to another, there are two special vertices in a network called *source* (start node) and *sink* (end node). Such a source-sink pair (q, s) of two vertices belongs to a *commodity*. A network with only one source and one sink we call a *single-commodity network*. In contrast, within a *multi-commodity network* we have a set of commodities $\mathcal{C} = \{1, \ldots, K\}$ and each commodity $k \in \mathcal{C}$ is associated with a source q^k and a sink s^k. In order to travel from q^k to s^k there exist various routes and we name such a route an $q^k - s^k$ path. The demand $d_i^k \in \mathbb{R}_{\geq 0}$ of a commodity $k \in \mathcal{C}$ defines the amount of flow entering or leaving vertex i. Moreover, travelling on roads causes costs—namely, travel time. For this, we define *travel time functions* $t_{ij} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ for all edges. The travel time of an edge is dependent on the flow on this edge. We will closely examine the characteristics of travel time functions in the next section.

2.2 The Traffic Network Optimization Problem

In the first chapter, we introduced the issue of traffic network optimization as an assignment of road planning and road construction. Our aim for this section is to find expressions that describe this real-world problem mathematically. Given a network $\mathcal{N}$ as described in Definition 2.3, we need to consider—for the incorporation of road building—a network graph $\tilde{\mathcal{A}} = (\mathcal{A}, cap, t^0)$, which represents a set of potentially new streets A within a graph $\mathcal{A} = (V, A)$ and gives information about the free-flow travel time t^0 and the capacity *cap* of the new edges.

As already discussed in chapter 1, different projects are available for the road construction. We denote the set of n different projects with $\mathcal{P} = \{p_1, \ldots, p_n\}$ and define a project $p \in \mathcal{P}$ as a tuple $(a, \bar{a}, r)$. The set a denotes a subset of $A \cup E$—more precisely, a is a set of streets that must be newly built or expanded within this project, whereby $\bar{a} \subset E$

denotes the set of streets that has to be reduced in this project. Thus, we differentiate between three different types of edges: An edge e is

an expansion edge	if $e \in a \cap E$	or
a new edge	if $e \in a \cap A$	or
a reduction edge	if $e \in \bar{a}$.	

In concrete terms, we increase the capacity by a specific factor if e is an expansion edge and scale down the capacity by a specific factor if e is a reduction edge. The positive real number r defines the realization cost of project $p \in \mathcal{P}$. Furthermore, a limited budget $b \in \mathbb{R}_{\geq 0}$ is needed as input information.

We are now able to define an instance of the traffic network optimization problem explicitly:

Definition 2.4
An instance of the traffic network optimization problem is a tuple $\mathcal{I} = (\mathcal{N}, \tilde{\mathcal{A}}, \mathcal{P}, b)$ with a network $\mathcal{N}$, a network graph $\tilde{\mathcal{A}}$, a set of projects $\mathcal{P}$ and a limited budget b.

As we now know an instance of the problem, we will devote our full attention to the modelling of the problem itself. Essentially, we are searching for an extended network, that

1. can be realized within a set of given projects,

2. does not exceed the given budget and,

3. reaches maximal utility of the traffic participants.

Thus, given an instance $\mathcal{I} = (\mathcal{N}, \tilde{\mathcal{A}}, \mathcal{P}, b)$, we search for a network $\mathcal{N}^*$, which can be realized with the initial network $\mathcal{N}$ and additional streets represented by a combination of projects within the set of given projects $\mathcal{P}$. Let $P = \{p_i : i \in I\}$ with $P \subset \mathcal{P}$ and $I \subset \{1, \ldots, n\}$ be such a project combination. For the set of projects P, the overall costs must stay within the budget. This can be expressed by the following inequality:

$$\sum_{i \in I} r_i \leq b, \tag{2.1}$$

Moreover, we are not interested in arbitrary solutions, but rather in the one that additionally maximizes the utility for traffic participants. The phrase, “to maximize utility”, is not that easy to transfer as we do not know exactly how to measure the

utility. But surely, the utility depends on the network configuration and, thus, on the specific project combination P. However, the relation between a project and its resulting utility is certainly not linear; rather, it is complex, since the measurement of the utility is a problem in itself. Thus, for the present, we simplify this dependency by letting u_P be the utility u attainable with project combination P without specifying a concrete determination of the utility. Hence, the objective yields

$$\underset{P \subset \mathcal{P}}{\text{maximize}} \quad u_P, \tag{2.2}$$

where $\mathcal{P}$ denotes the set of available projects and P a subset of $\mathcal{P}$.

To formulate the traffic network optimization problem as an integer optimization program, we introduce the decision variables x. Since the task is to decide which combination of projects should be realized, an obvious approach might be to model x as a binary vector where x_i becomes 1 if project p_i will be realized and 0 otherwise.

$$x_i = \begin{cases} 1 & \text{if project } p_i \text{ is realized} \\ 0 & \text{otherwise} \end{cases}$$

Due to the limited budget, we are merely interested in solutions where the sum of the project cost will not exceed the budget. Thus, we can now rewrite the restriction (2.1) as the sum of all projects:

$$\sum_{i=1}^{n} r_i x_i \leq b.$$

We remember the goal of increasing convenience for traffic participants as well as increasing their utility (2.2). Nevertheless, we content ourselves with a general formulation of the project-dependent utility without specifying a concrete function. This yields the objective:

$$\max \quad u(x)$$

We focus on concrete utility functions later in this work. In section 2.3.1 and section 2.3.2 we assume that the total travel time of all network participants mostly influences the utility and, therefore, it can be approximated by the minimization of travel time; where, in section 2.3.4, the utility is calculated by simulation tools that use multiple influencing factors for the calculation of a network utility.

Using the above definitions we formulate the traffic network optimization problem (TNOP) as an integer program:

$$\begin{aligned} \text{(TNOP)} \qquad & \max && u(x) \\ & \text{s.t.} && \sum_{i=1}^{n} r_i x_i \leq b && (2.3) \\ & && x \in \{0,1\}^n && (2.4) \end{aligned}$$

The first constraint (2.3) guarantees that the realized projects abide by budget b and condition (2.4) is the restriction to binary variables since we do not consider a fractional realization of projects. Consequently, project i can either be realized completely ($x_i = 1$) or not at all ($x_i = 0$). We quickly determine that in case of a linear objective function

$$u(x) = \sum_{i=1}^{n} u_i x_i$$

the above program is exactly what we call a knapsack problem, where $u_1, \ldots, u_n$ define the knapsack values and the project costs $r_1, \ldots, r_n$ equal the weights of the knapsack. The knapsack problem is NP-hard and as long as $P \neq NP$, there can be no exact polynomial-time algorithm [KV08]. But the knapsack problem can be solved exactly in pseudo-polynomial time using dynamic programming. There even exists a fully polynomial time approximation scheme (FPTAS) that uses dynamic programming as a subroutine. A FPTAS is an algorithm that solves a problem with a correctness of factor $1 - \epsilon$ of the optimal solution and is polynomial in $\frac{1}{\epsilon}$ and in the size of the problem instance [KV08].

However, the network utility and the associated travel time depend on the amount of traffic navigating through this network, and as a conclusion, we derive a more complex objective function; hence, we are not blessed with such well-performing algorithms. To experience the impact of road users' interactions in the given network, we focus on network flows in the following chapters.

2.3 Flow of Traffic

The study of traffic flows seeks to describe the interactions between vehicles, traffic participants and the network's infrastructure. In section 2.3.1, we consider optimal traffic flows from a somewhat central viewpoint, but also learn that traffic participants act in a selfish way. Hence, a different model for traffic flows is required and studied in section 2.3.2. In order to measure the loss from selfish routing to the system optimal flow model, we obtain a worst-possible ratio. Afterwards, we apply this acquired knowledge to the traffic network optimization problem in section 2.3.3. Finally, we explore how it is possible to simulate traffic flows, which we primarily explain with the help of the simulation tool MATSim.

A *flow* f is a function $f : E \to \mathbb{R}_{\geq 0}$ that represents the volume of traffic. Thus, a *flow on edge* is a nonnegative real vector $f_{ij} \in \mathbb{R}_{\geq 0}$ representing the amount of flow on edge

(i, j). We say a flow is feasible, if it satisfies the *flow conservation constraints*. For this, consider a setting of navigating $|d|$ units from source q to sink s.

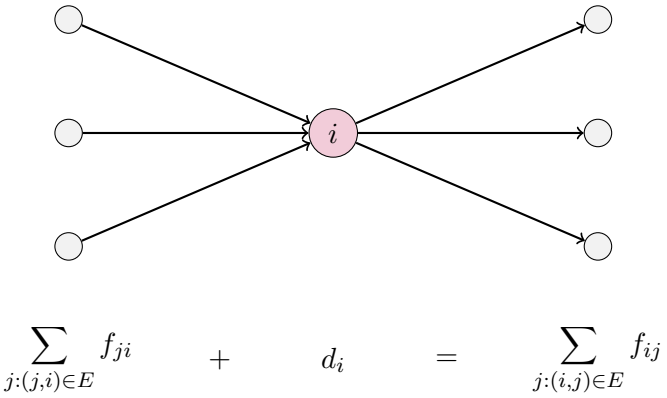


Figure 2.2: Flow conservation constraints

In fig. 2.2, we differentiate between three cases. For all nodes except source and sink $(i \neq q, s)$, it holds: Flow in is flow out, i. e. $d_i = 0$. Meanwhile, the special nodes $i = q$ and $i = s$ represent connections to the environment surrounding. Hence there is a net gain of flow, i. e. $d_i > 0$ and respectively a net loss of flow, i. e. $d_i < 0$.

2.3.1 System Optimal Flow

So far, we have introduced a mathematical model to represent the traffic network optimization problem. However, we have left unanswered the question of how to measure a network's benefit. One option for the objective function is to minimize the travel time that road users spent while navigating in the network. It matches our experience that travel time is dependent on the traffic flow on this road. The more vehicles are on the street the more travel time we have to expect. Thus, the cost function of a road is defined dependent on the flow on it. We call the flow-dependent cost function $t_{ij} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ as the *travel time function* of edge $(i, j) \in E$. In the literature, this is also referred to as the *link performance function* or *latency function*. Let us closely examine the properties of such travel time functions. Usually, they are expected to be nonlinear, convex and monotonously increasing. In order to express the travel time function explicitly, we use a function developed by the *Bureau of Public Roads*, [PR64] which reads as:

$$t_{ij}(f_{ij}) = t_{ij}^0 \left(1 + \alpha \left(\frac{f_{ij}}{cap_{ij}}\right)^\beta\right) \tag{2.5}$$

The real number t_{ij}^0 denotes the free-flow travel time introduced in section 2.1. The travel time $t_{ij}(\cdot)$ grows with an increasing amount of flow and the capacity influences the travel time in such a way that in case of exceeded capacity, road users end up in a traffic jam and their travel time becomes overwhelming. A graphical representation of the travel time function is provided in fig. 2.3. The horizontal axis represents the amount of flow and the vertical axis the associated costs (travel time).

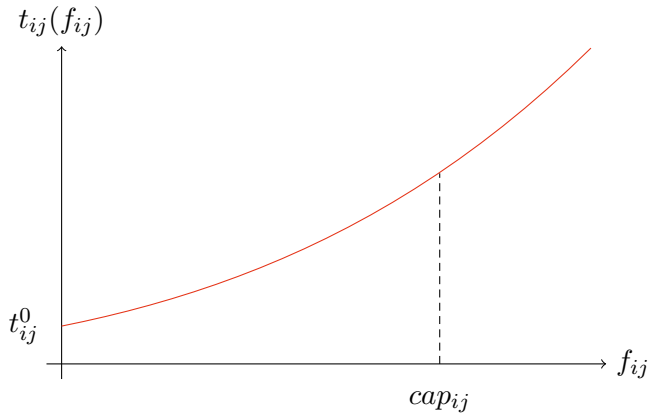


Figure 2.3: Travel time function $t_{ij}(\cdot)$

After developing the travel time function, we focus on the objective. Since we want to minimize the total travel time $C(f)$ in the given network, the target function can be expressed as the sum of travel time (dependent on the amount of flow) times the flow over all edges:

$$C(f) = \sum_{(i,j)\in E} t_{ij}(f_{ij}) f_{ij} = \sum_{(i,j)\in E} c_{ij}(f_{ij}). \tag{2.6}$$

We denote (2.6) also as cost of a flow f in a network $\mathcal{N}$ and thereby are able to compare different flows in the same network. Since we are interested in only feasible flows, we add the flow conservation constraints (fig. 2.2) which guarantee that no flow leaves or enters the nodes, except at the source and the sink. The assignment of flows to edges that minimizes the total travel time of all road users is called *system optimum* (SO) and we define the system optimum problem (SOP) as follows:

$$\text{(SOP)} \qquad \min \quad C(f) = \sum_{(i,j)\in E} t_{ij}(f_{ij}) \cdot f_{ij} = \sum_{(i,j)\in E} c_{ij}(f_{ij}) \tag{2.7}$$

$$\text{s.t.} \qquad \sum_{j:(i,j)\in E} f_{ij} - \sum_{j:(j,i)\in E} f_{ji} = d_i \quad \forall i \in V \tag{2.8}$$

$$f_{ij} \geq 0 \quad \forall (i,j) \in E$$

Note that we formulated (SOP) without specifying capacity constraints; capacity is considered indirectly in the travel time function. The problem above has at least one feasible solution as the objective $C(\cdot)$ is continuous and the space of all feasible flows is a bounded and closed subspace of the Euclidean space [Rou05]. To distinguish flows from those in the next chapter, we denote a system optimal flow as a solution of (SOP) with f^*.

We now devote ourselves to a closer inspection of (SOP). We have already ascertained that the travel time function is convex and, thus, $C(f)$ is a convex nonlinear objective function under linear constraints. For this kind of problem, we know that every local optimum is a global optimum as well and we can formulate the following theorem [UU12].

Theorem 2.5
A flow f^* is optimal for a convex program (SOP) and for continuously differentiable travel time functions if and only if

$$\sum_{(i,j)\in E} c'_{ij}(f^*_{ij}) f^*_{ij} \leq \sum_{(i,j)\in E} c'_{ij}(f^*_{ij}) f_{ij} \tag{2.9}$$

for every feasible flow f.

Sketch of Proof. Apply the KKT conditions from Karush [Kar39] and Kuhn and Tucker [KT50] along with the sufficient optimality conditions to the system optimum problem to get the above statement. A proof of the general necessary and sufficient optimality conditions can be found in [UU12]. □

In this statement, we used the derivative of c_{ij}, which is defined as $c'_{ij} := \frac{d}{dx} c_{ij}(x)$. By applying the product rule for differentiation for the specific expression

$$t_{ij}(f_{ij}) \cdot f_{ij} = c_{ij}(f_{ij})$$

it follows that

$$c'_{ij}(f_{ij}) = t_{ij}(f_{ij}) + t'_{ij}(f_{ij}) f_{ij}.$$

Since we know that $t_{ij}(f_{ij})$ is the travel time, we can interpret $t'_{ij}(f_{ij})$ as the cost entailed for road users of edge (i, j) when a single traffic participant also decides to use that road.

The optimal solution f^* of (SOP) is optimal for the system as a whole, but might be unfair to some of the traffic participants. Could some of the single road users increase their own travel time by switching to a faster path? And will the selfish behaviour of

the road users disturb the socially optimal outcome? To answer these questions, we will consider a seminal example developed by Pigou [Pig20] in 1920.

Pigou's Example

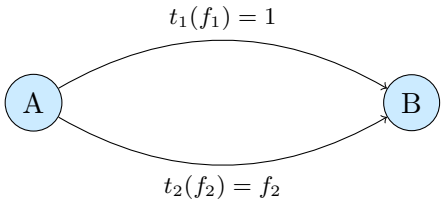


Figure 2.4: Network in Pigou's Example

Consider two locations A and B connected by two roads as in fig. 2.4. We have to navigate one traffic unit from A to B. The travel time function for the upper route is constant at $t_1(f_1) = 1$ regardless of whether other traffic participants are using the same road. For practical understanding, we think of a long and wide highway. For the lower route, the time that drivers need is dependent on the overall traffic, say a narrow route through the city. Here, the travel time is exactly the fraction of the drivers who chose the lower road with respect to the overall traffic $t_2(f_2) = f_2$, i. e. travel time is dependent on the amount of flow on this route. We dedicate ourselves to finding an optimal flow from A to B. We follow the above-mentioned instructions and solve the system optimum problem for the stated travel time functions under the assumption of navigating one traffic unit from A to B, i. e. we are left with the task of solving the following problem:

$$
\begin{aligned}
\min \quad C(f) = \quad & \underbrace{t_1(f_1)f_1}_{c_1(f_1)} + \underbrace{t_2(f_2)f_2}_{c_2(f_2)} = f_1 + f_2^2 \\
\text{s.t.} \quad & f_1 + f_2 = 1 \\
& f_1, f_2 \geq 0
\end{aligned}
$$

We can quickly find the optimal flow $f_1^* = f_2^* = \frac{1}{2}$ and the associated objective value of $\frac{3}{4}$.

Now consider the case where the drivers decide by themselves which path to take. With the assumption that every driver has the aim of minimizing his own travel time, traffic always uses the lower route. He does so because when choosing the lower road, the driver needs at most the time that he needs for the upper one. As a result, all traffic participants will reach B in not less than one hour.

Remembering the objective value (i. e. travel time) of $C(f^*) = \frac{3}{4}$ from the solution above, we discover that the selfish routing of the drivers leads to a worse travel time of $C(f) = 1$.

Taking that selfish behaviour of the traffic participants into account in the next chapter, we approach a model where traffic status is deemed to be at an equilibrium.

2.3.2 User Equilibrium

As we have suspected in the previous section and even calculated in Pigou's example, the solution of the system optimum problem does not reflect real-world behaviour since some of the traffic participants can improve their travel time by switching to another road and thus destroy the optimal flow. We use this as an occasion and assume now a *selfish routing* network, i. e. every traffic participant can choose his route in a selfish way. In doing so, every single user has full information and knows the travel paths of all the other users. Although the topic was discovered much earlier from Pigou [Pig20] in 1920 and Wardrop [WCEGB52] in 1952, the term "selfish routing" is due to Roughgarden and Tardos [RT02] in 2000.

Wardrop's first principle stated that a flow f is at equilibrium if "all journey times on all the routes actually used are equal, and less than those which would be experienced by a single vehicle on any unused route" [WCEGB52]. In other words, we say that a flow is at equilibrium, if none of the traffic participants can improve their own travel time by changing to a different road. We call that system state a *user equilibrium* (UE) and Definition 2.6 states the mathematical formulation for the (UE):

Definition 2.6
A flow f is at user equilibrium (often Nash equilibrium) if and only if

$$\sum_{(i,j)\in E} t_{ij}(f_{ij}) f_{ij} \leq \sum_{(i,j)\in E} t_{ij}(f_{ij}) \tilde{f_{ij}} \tag{2.10}$$

for all feasible flows $\tilde{f}$.

It should be mentioned that in the literature we often find the expression *Wardrop equilibrium*, which is basically the same for a feasible flow f and continuous monotonous functions $t_{ij} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. Once we find a flow f at user equilibrium, all $q^k - s^k$ paths have the same travel time $t^k(f)$ and since the flow is feasible, the cost amounts to

$$C(f) = \sum_{k=1}^{K} t^k(f) d^k.$$

There exist mathematical statements for the existence and the uniqueness of flows at user equilibrium cf. [Rou05]. Thus, every instance admits a flow at user equilibrium and all these flows have equal costs $C(f)$.

With a closer look at the definition of a user equilibrium (2.10), one may recognize the similarity to the optimality conditions (2.9) of the system optimum problem in section 2.3.1. It turns out that flows at user equilibrium and system optimal flows are the same but with different travel time functions. Therefore, we can formalize the relationship between (SO) and (UE) as in the following statement provided by Beckmann, McGuire and Winston [BMW56]:

Theorem 2.7
A flow f, feasible for $\mathcal{N} = (\tilde{\mathcal{G}}, \mathcal{C}, q, s, d, t)$, is system optimal if and only if f is at user equilibrium for a network $\mathcal{N}^* = (\tilde{\mathcal{G}}, \mathcal{C}, q, s, d, t^*)$, where t are convex, continuously differentiable and monotonously increasing travel time functions and t^* the corresponding marginal travel time functions with $t^* = \frac{d}{dx}(x \cdot t(x))$.

The proof of Theorem 2.7 can be found in [BMW56]. The above theorem delivers an especially useful statement: In order to calculate a flow f in such a way that the status of the traffic is at user equilibrium, we just need to solve (SOP) using the modified travel time function. Based on these new findings, we tackle Pigou's example once again.

Pigou's Example Revisited

If we are interested in finding a user equilibrium in Pigou's example, we adapt the travel time functions in the following way:

$$t(f) = \frac{d}{dx}(x \cdot \tilde{t}(x))(f)$$

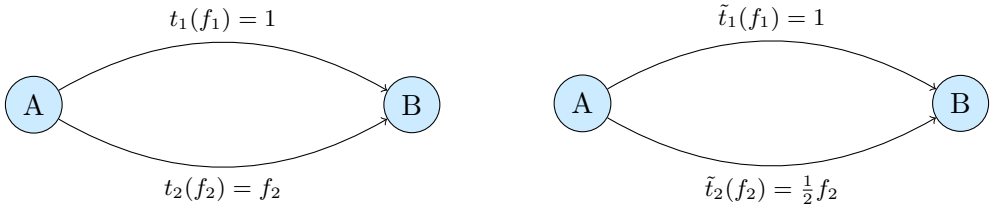
Resolved by $\tilde{t}$ we get the modified travel time functions:

$$\tilde{t}(f) = \frac{1}{f}\left(\int_0^f t(x)dx\right)$$

The original network and the modified network with its new travel times are illustrated in fig. 2.5. Thus, we are able to reach a user equilibrium in fig. 2.5a by solving the following modified (SOP):

$$\begin{array}{lll} \min & C(f) = & \tilde{t}_1(f_1)f_1 + \tilde{t}_2(f_2)f_2 = f_1 + 0.5f_2^2 \\ \text{s.t.} & & f_1 + f_2 = 1 \\ & & f_1, f_2 \geq 0 \end{array}$$

The optimal flow for the network in fig. 2.5b are $f_1^* = 0$ and $f_2^* = 1$ and it is equal to the user equilibrium in fig. 2.5a, as per our assumptions in the first consideration of Pigou's example.



(a) Original network with travel time function $t(\cdot)$

(b) Modified network with travel time function $\tilde{t}(\cdot)$

Figure 2.5: (SO) and (UE) in Pigou's Example

Vice versa, a user equilibrium in a network with marginal travel time

$$t^*(f) = \left(\frac{d}{dx}(x \cdot t(x))\right)(f) = \begin{cases} 1 & \text{for the upper route} \\ 2f_2 & \text{for the lower route} \end{cases}$$

delivers the system optimal flow for fig. 2.5a. A closer consideration of the aforementioned example can be found in [Rou05].

We have so far achieved the calculation of a system optimal flow (SO) for an instance of a network $\mathcal{N}$ in section 2.3.1 and of flows at user equilibrium (UE) in the current section. Obviously, there is a gap between the value of the *system optimum* and the selfish *user equilibrium*. We follow Roughgarden [Rou05] and quantify that gap by

$$\rho(\mathcal{N}) = \frac{C_{(UE)}}{C_{(SO)}} = \frac{C(f)}{C(f^*)} \tag{2.11}$$

and name it the *price of anarchy*. The price of anarchy describes the worst ratio between the cost at user equilibrium $C(f)$ and the cost of a system optimal flow $C(f^*)$.

Returning to Pigou's example where the cost of system optimal flows were $C(f^*) = \frac{3}{4}$ and the cost of flows at user equilibrium $C(f) = 1$, we get for the price of anarchy

$$\rho = \frac{C(f)}{C(f^*)} = \frac{4}{3}. \tag{2.12}$$

As in [Rou05], one can show that for linear travel time functions, the price of anarchy is always less than $\frac{4}{3}$. The more the degree increases, the worse the price of anarchy, e. g. for travel time functions of degree 2 the price of anarchy yields to $\rho \approx 1.626$. For a closer examination of bounds of the price of anarchy, refer to [Rou05].

2.3.3 Application in Traffic Network Optimization

This section concerns itself with the application of flows to the traffic network optimization problem. Our aim is to evaluate the utility of networks by minimizing total travel time and taking the objective as a comparative value. Let us first consider the idea from the previous section. Since we experienced in section 2.3.2 that a system state at equilibrium can be reached by solving (SOP) using a modified cost function, we now want to develop explicit travel time functions $t(f)$. Let us recapitulate the main statement of the previous section. Theorem 2.7 stated that the system optimal flow of a network $\mathcal{N}$ is at user equilibrium for the network $\mathcal{N}^*$ for marginal travel time functions

$$t^* = \frac{d}{dx}(x \cdot t(x)) \tag{2.13}$$

and, accordingly, for a flow at user equilibrium, we can formulate the following:

> A flow at user equilibrium in a network $\mathcal{N} = (\tilde{\mathcal{G}}, \mathcal{C}, q, s, d, t)$ is a system optimal flow for $\tilde{\mathcal{N}} = (\tilde{\mathcal{G}}, \mathcal{C}, q, s, d, \tilde{t})$ with
>
> $$\tilde{t}(f) = \frac{1}{f}\left(\int_0^f t(x)dx\right).$$

This statement forms the basis for our further examinations. Note that we are restricted to continuously differentiable travel time functions if we wish to apply the above statement. Let $t(f)$ be an explicit travel time function, e. g. eq. (2.5) with the parameters $\alpha = 1$ and $\beta = 2$:

$$t(f) = t^0\left(1 + \alpha\left(\frac{f}{cap}\right)^\beta\right) = t^0\left(1 + \frac{f^2}{cap^2}\right)$$

As $t(f)$ is continuously differentiable we can determine the modified travel time function $\tilde{t}(f)$ for the system optimum problem in the following way:

$$\tilde{t}(f) = \frac{1}{f}\left(\int_0^f t(x)dx\right) = \frac{1}{f}\left(\int_0^f t^0\left(1 + \frac{x^2}{cap^2}\right)dx\right) = t^0\left(1 + \frac{f^2}{3cap^2}\right) \tag{2.14}$$

By plugging these modified travel time functions in the objective of (SOP), we can calculate a user equilibrium by minimizing total cost:

$$C(f) = \sum_{(i,j)\in E} \tilde{t}_{ij}(f_{ij})f_{ij} = \sum_{(i,j)\in E} t^0_{ij} f_{ij}\left(1 + \frac{f^3_{ij}}{3cap^2_{ij}}\right)$$

So far, we have assumed only a single commodity. In order to formalize the optimization problem, we extend the objective as well as the side conditions by taking multi-commodities $k \in \mathcal{C}$ into account. Altogether, we get an optimization program with a nonlinear objective function of third degree under linear constraints:

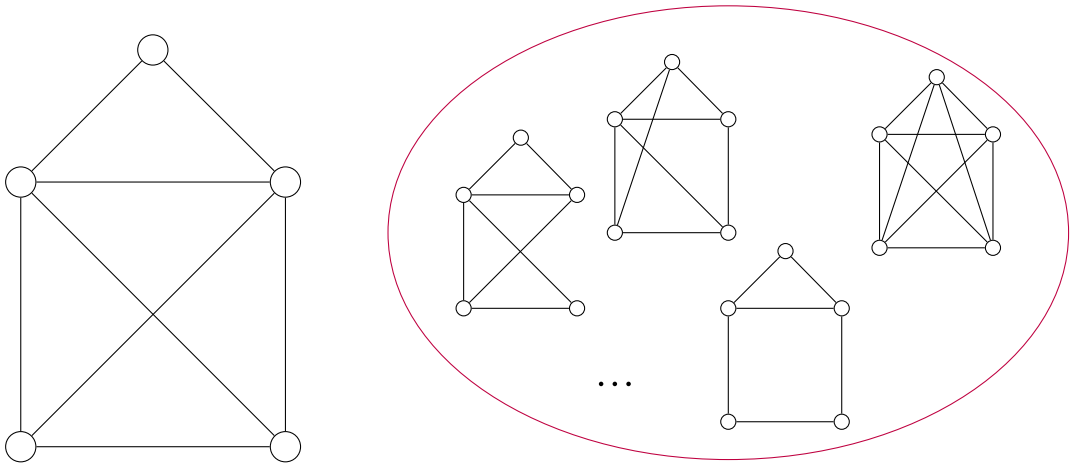
$$
\begin{aligned}
\text{(NLP)} \qquad \min \quad C(f) = & \sum_{(i,j)\in E} \left(\tilde{t}_{ij}(\sum_{k\in\mathcal{C}} f_{ij}^k) \sum_{k\in\mathcal{C}} f_{ij}^k \right) \\
& \sum_{j:(i,j)\in E} f_{ij}^k - \sum_{j:(j,i)\in E} f_{ji}^k = d_i^k \qquad \forall i \in V \quad \forall k \in \mathcal{C} \\
& f_{ij}^k \geq 0 \qquad\qquad\qquad\qquad\qquad\quad \forall (i,j) \in E \quad \forall k \in \mathcal{C}
\end{aligned}
$$

with explicit travel time functions (2.14). For (NLP), we are left with $k|E|$ variables, $k|V|$ flow conservation constraints and $k|E|$ nonnegativity constraints.

By solving (NLP) the objective value defines the total travel time in a network $\mathcal{N}$. In order to compare different networks, we just take $C(f)$ as an evaluation value and determine that the smaller the objective, the more useful the network.

We exploit this evaluation of networks and apply it to the traffic network optimization problem. In order to provide solutions to $(TNOP)$, we choose an approach using local optimization. The idea behind this is to optimize a given network within its "neighbourhood". We specify a neighbour of a network $\mathcal{N}$ as a network that differs in at most k edges, compared to the initial network $\mathcal{N}$. Figure 2.6 shows an example of $k = 2$. Hence, we are left with three different border cases: First, we remove two edges from the initial network, second, we add two edges and, third, we add one edge and additionally remove one. See fig. 2.6 for a practical explanation.

Since there are quantities of neighbours for a given network and not all of them suit our interest, we adapt the concept of a neighbourhood to the traffic network optimization problem. For a given instance $\mathcal{I}$ of $(TNOP)$, we are restricted to a set of projects, i. e. not every neighbour of $\mathcal{N}$ is part of a project. Thus, we just consider only those neighbours that are within some given projects. Furthermore, we consider only those networks that are feasible for the instance $\mathcal{I}$. Both together lead to a significant decrease



(a) Initial network $\mathcal{N}$ **(b)** Neighbourhood of $\mathcal{N}$

Figure 2.6: Example of a neighbourhood for $k = 2$

in the size of a neighbourhood. The optimization procedure then is as follows:

Algorithm 1: Local neighbourhood search

input : an instance $\mathcal{I} = (\mathcal{N}, A, \mathcal{P}, b)$
output : best network $\mathcal{N}^*$ in the neighbourhood of $\mathcal{N}$
1 Calculate all feasible, project-viable neighbours of $\mathcal{N}$;
2 **foreach** *network in neighbourhood* **do**
3 Solve the corresponding (NLP);
4 Save its objective $C(f)$ as an evaluation value;
5 **end**
6 Return network with best evaluation (smallest objective);

This is a simple procedure of local optimization, in chapter 3 we become acquainted with a more sophisticated method—the genetic algorithms.

2.3.4 Simulation of Traffic Flow

So far, we have gained a model that measures the network utility based on a road user's total travel time in this network. Various existing simulation tools are based upon this solid groundwork. However, they further develop the evaluation of the networks by taking

more factors into consideration. One established tool that is available as open source is MATSim (**M**ulti-**A**gent **T**ransport **Sim**ulation [Rie14])—a framework that provides a tool for implementing large-scale transport simulations. Besides MATSim, there exist plenty of simulation software such as SUMO (Simulation of Urban Mobility [Kra+12]) and TraNS (Traffic and Network Simulation Environment [Pi08]). For a brief overview we refer to [Ngu+12].

MATSim belongs to the *agent-based methods*, where small intelligent units—called agents—interact autonomously. Thus, the behaviour is not dictated by the system level but by results from the behaviour of the single agents. Similarly to [Rie14] we will give a short overview of the key components of MATSim to ensure understanding of the process behind it and will also refer to the MATSim user guide [Rie14] for a more detailed illustration of the concept. The five major stages of the simulation are initial demand, execution, scoring, replanning and analysis. Based on fig. 2.7 that represent these stages and their order, we will clarify the main MATSim operations.

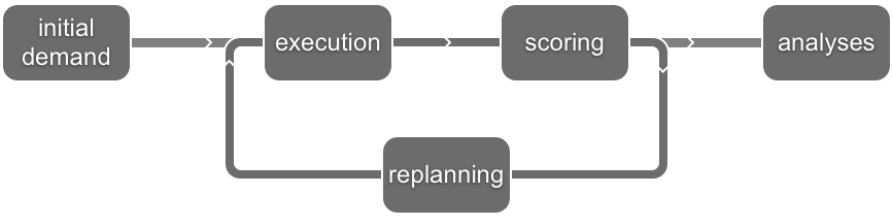


Figure 2.7: Stages of a MATSim Simulation [Rie14]

The first stage forms the **initial demand**, which contains a list of agents with their day plans and activities and thus describes the mobility behaviour to be simulated. Note that an agent's day plan describes only the intention of the agent and, under certain circumstances, it may happen that the day plan cannot be realized in the simulation.

The **execution** stage is often called mobility simulation—short mobsim—and represents the real motion of the agents in the network. During the execution, the agents influence each other and thus traffic jams can occur. The execution process is iterative and thereby the navigation of the agents changes over iterations.

The **scoring** is responsible for the evaluation of the agents' execution. As a rule, the time an agent spent in activities increases and the time spent travelling decreases its score. The scoring process is incremental and, hence, the agent's scoring increases with every iteration. We consider in detail the utility function of MATSim, which is also called the *Charypar-Nagel scoring function* [Rie14]. Imagine the typical course of an agent's day:

The agent is at home in the morning, then travels to work, at midday he is out for lunch and in the evening he travels back home. The Charypar-Nagel function is drafted in such a way that the agent increases his utility while being at home, at work or having lunch and he gets a negative utility for travelling between those locations. The mathematical formulation of the scoring function is the following:

$$V = \sum_i (V_i^{perf} + V_i^{late}) + \sum_j V_j^{leg}$$

V_i^{perf} defines the benefit from performing activity i, which is normally positive. The penalties V_i^{late} for arriving late and V_j^{leg} for travelling affect the scoring in a negative way. Behind those functions are many more calculations. However, for our purposes, it is sufficient to keep in mind which factors manipulate the scoring.

The **replanning** phase modifies the agents' plans in a way that bad scores—for example, if the agent got stuck in a jam—will be avoided in the next mobility simulation. This optimization follows the principle of evolutionary algorithms. For the usage of MATSim, we need not go into detail now, but the concept of such algorithms will be outlined in chapter 3 since we use these principles for our program as well. The replanning stage performs iteratively, too, and because it influences the agents' plans, the following execution and corresponding scoring of the agents will vary.

Once the maximum number of iterations is reached, some performance values are provided in the **analysis** stage. Important data that we need for further processing are the average trip duration and the development of the utility values (scoring values) of the agents over iterations. The following scoring values are available [MAT11]:

- utility (executed): average score of the executed plan of each agent
- utility (worst): average score of the worst plan of each agent
- utility (average): average of the average score plan per agent
- utility (best): average score of the best plan of each agent
- average trip duration

Moreover, travel distances as well as information about the executed plans are available.

Once we use MATSim for an evaluation of networks, an important question of how many iterations we should use, has to be answered. On these grounds, we look at the development of the average score values of an agent over time as shown in fig. 2.8.

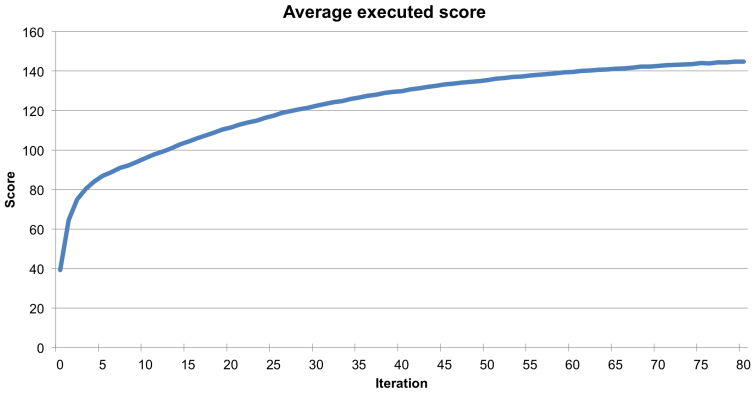


Figure 2.8: Typical development of the average score [Rie14]

In the first few iterations, there is an enormous potential to increase the average score, but the improvement is very slight in the later iterations. For our purposes, a fixed iteration number of 60 seems to be a good choice, but it is important to really fix this number of iterations; otherwise, the utility values are not comparable.

Chapter 3

Genetic Algorithms

This chapter focuses on genetic algorithms, their processes and detailed steps. It starts with the development of evolutionary algorithms in general and their inspiration from nature. In section 3.2, we first present the general concept and sketch a pseudo code for the standard genetic algorithm. Whereas, in section 3.2.1 to section 3.2.5, we conduct a detailed examination of the basic steps: evaluation of the individuals, selection according to fitness, crossover to generate new offspring, mutation of offspring and replacement of the population. In section 3.3, we will not only summarize the advantages but also become aware of the limitations of genetic algorithms.

3.1 Motivation of Evolutionary Algorithms

The term "evolution" refers to a gradual process of development and change in all forms of organisms over generations. Usually, the traits of a population will be copied from one generation to the next. Hereby, mutation in genes can appear randomly and thus biological diversity arises. Evolutionary theory is the study and formulation of an explanatory model of how evolution occurs. One of the most formative evolutionary theorists was Charles Darwin. About 1836, he started to formulate his idea of evolution and, in the subsequent years, systematically collected evidence about his concept of "natural selection" [Kut08]. In 1859, he published his collections in the book *On the Origin of Species*. The principle of "natural selection" is based on the respective probability of the survival of an individual and, thus, its opportunity to pass on a hereditary disposition to its offspring. Those individuals that are more adaptable to their environment and are more resistant against diseases have higher reproductive success than do others. One can also say that they have a higher fitness level. In passing a hereditary disposition to offspring, mutation can occur. In evolution theory, mutation describes a transformation in the genome [Kut08]. Evolutionary algorithms pick up these evolutionary phenomena

of mutations and the "survival of the fittest" and, therefore, attempts to simulate the behaviour of nature. In the field of optimization, evolutionary algorithms belong to the class of metaheuristic optimization methods, which means that they are determining enhanced solutions to the underlying problem within the framework of a search process but, in contrast to heuristics, are not restricted to a specific problem.

3.2 General Concept of Genetic Algorithms

As a special type of evolutionary algorithms, the genetic algorithms belong to the population-based methods. Genetic algorithms (GA) were invented by Holland [Hol75] in 1975. The main idea behind them is to consider not just a single solution but rather to search for an effective one from among a large number of possible solutions. This set of candidate solutions is called a *population* and each solution is called an *individual* of this population. In the following, we will denote the length of an individual by n and the size of the population by m. A critical problem in applying genetic algorithms is to find a suitable representation of the solutions in the problem domain as individuals. This process is called *encoding* and can be performed using different types or objects such as bits, arrays, lists, or numbers. The first practised encoding [Hol75] and even the most common way is a binary string with 1s and 0s. Nevertheless, there exist other encodings, such as *tree encoding*, *octal encoding* or *value encoding*; for a closer examination, we refer to [Mit96] and [SD07]. In this chapter, we will not further the topic of encoding since we are blessed with "natural" encoding as we are dealing with a binary optimization problem; for further guidance see chapter 4.

The aim of the application of a genetic algorithm is to augment the population in respect to the fittest individual iteratively by using evolutionary mechanics, such as elitism, mutation, and crossover. The procedure is repeated until some criteria are met. A maximum number of iterations as well as the percentage of equal individuals in the population could form such criteria. We now want to formulate the concept of genetic algorithms mathematically. Translating the process of the iterative generation of populations into code, we derive a simple pseudo code for the standard genetic algorithm (see Algorithm 2).

We start by generating a random population of m individuals and as long as the stop criteria are not met, a new population will be generated. In doing so, we first evaluate all individuals in the present population and then take the evaluation values as a decision-making basis for determining which individuals should have the chance to reproduce. Subsequently, the selected individuals go through mutation and crossover; afterwards,

Algorithm 2: Pseudo Code for a standard GA

```
1  Randomly generate a population of m individuals;
2  while stop criterion is not fulfilled do
3  |  Evaluation: Score all individuals of population;
4  |  Create offspring by:
5  |  |  Selection:  Determine survival probability acc. to individual's fitness;
6  |  |  Crossover:  Randomly choose parents and perform crossover;
7  |  |  Mutation:   Mutate some of the offspring;
8  |  Replacement: Replace current population by new population;
9  end
10 Return best individual in present population;
```

the current population will be replaced by them. Once the abort criterion is fulfilled, the algorithm returns the best individual that has occurred until that point. Naturally, there are various implementations of the different steps. In the following, we investigate closely several phases of Algorithm 2.

3.2.1 Evaluation and Fitness Function

Once we establish an initial population, we are interested in how well the individuals are performing. On that account, the evaluation of all individuals in the current population forms the first step of the algorithm. Obviously, all the individuals have different benefits for the underlying problem. Hence, the optimization criteria state the evaluation function of the individuals, and the evaluation value appraises the profit for the optimization problem. In the literature, the terms "evaluation" and "fitness" are not consistent. We often find that the evaluation function is equal to the fitness function itself; obviously, they are related to each other. However, for our purposes we differentiate between them as the fitness function measures the chance for an individual to reproduce, while the evaluation delivers the benefit for the optimization problem. What the fitness function will look like will be examined in the next section.

3.2.2 Selection

Selection is paramount since we decide which individuals in the current population will form the basis for creating new individuals. As a deciding factor, we use the fitness

function. Generally, individuals with higher scores are more likely to be selected than those with poor fitness. In the literature, there are plenty of selection techniques, such as tournament selection or rank selection [Mit96]. Within the scope of this work, we take a closer look at the *roulette wheel method*, since this technique was developed very early in 1989 by Goldberg [Gol89] and is hence well tested. The roulette wheel selection belongs to the fitness-proportional selection methods, which already suggest its advantage; the selection takes into account not only the order of fitness values, as for instance, the rank selection but also matters related to the proportions of the next fittest individual. Another advantage that we intend to leverage is the following: Even the worst evaluated individual has a chance to reproduce even if it is not particularly large; thus, we do not exclude some individuals at a too early stage. Obviously, fitter individuals tend to have greater chance to survive than do weaker ones. We define an individual's fitness as the fraction of its evaluation value compared to the other evaluation values:

$$\text{fitness}(i) = \frac{\text{evaluation}(i)}{\sum_i \text{evaluation}(i)}$$

As the name suggests, we can illustrate the method by considering the game of roulette. See fig. 3.1 and assume that there are five individuals (represented by the five different colours) with varying fitness values. The size of the slice that an individual owns is proportional to its fitness value.

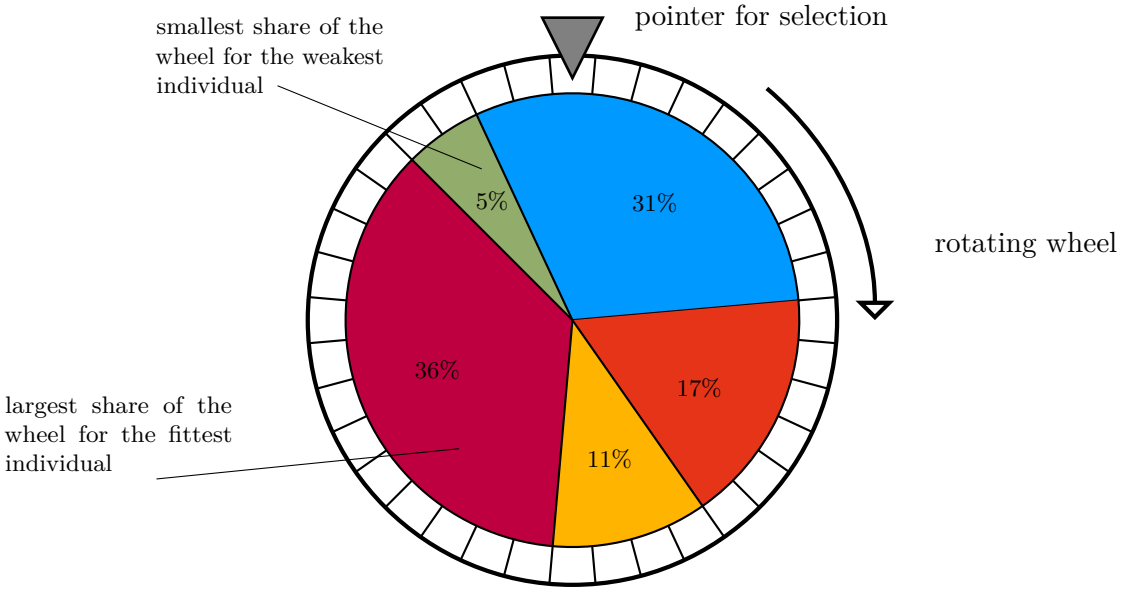


Figure 3.1: Roulette wheel selection

After allocating all individuals on the wheel, we select a random individual by rotating the roulette wheel. Wherever the pointer stops, we choose the corresponding individual. Obviously, we spin the wheel as many times as we need an individual—mostly, as per the size of the population. The selected individuals constitute the pool of parents for the next generation. In practice this method is implemented as follows:

```
1 Randomly generate a uniformly distributed number r ∈ [0, 1];
2 for i = 1, ..., m do
3 |   if Σ_{k=1}^{i} fitness(k) ≥ r then
4 |   |   Select individual i;
5 |   |   break;
6 |   end
7 end
```

3.2.3 Crossover

The crossover (often called *recombination*) is the process in which two individuals (*parents*) produce new individuals (*offspring*) by bequeathing parts of themselves to their offspring. This is implemented by first cutting the parents at specific positions; afterwards, they are put back together, crossbred. This specific positions are named the *crossover points*. Let n be the length of an individual. Then every integer between 1 and $n-1$ defines a possible crossover point. A crossover procedure with only one crossover point can be implemented as follows:

```
1 Select two individuals as parents;
2 Set offspring1 := individual1 and offspring2 := individual2;
3 Randomly generate a crossover point z ∈ {1, ..., n − 1};
4 for i = z + 1, ..., n do
5 |   value_offspring1(i) = value_individual2(i);
6 |   value_offspring2(i) = value_individual1(i);
7 end
```

An example illustrating the principal of the above procedure is provided in fig. 3.2. The instance of the two individuals form the parents and both of them have 8 bits. Possible crossover points could be every number from 1 to 7, e. g. for the crossover point $z = 3$, the two offspring are represented as shown in fig. 3.2.

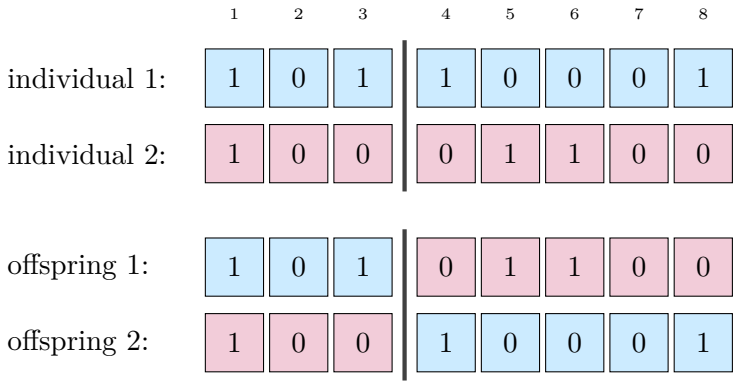


Figure 3.2: Parents and their offspring

If there is only a single crossover point as above, then we call it a *one-point crossover*. Obviously, we can consider more crossover points and get a *2-*, *3-* or *k-point crossover*. As the name already reveals, for a k-point crossover, we calculate k random crossover points and the offspring adopt the values alternating at the k points from their parents.

Another technique would be the *uniform crossover* schema, where each bit of the offspring is assigned either to individual 1 or individual 2, both with a probability of 0.5. In practice, this procedure is implemented as follows:

```
for i = 1, ..., n do
    Randomly generate a uniformly distributed number r ∈ [0, 1];
    if r < 0.5 then
        value_offspring1(i) = value_individual1(i);
        value_offspring2(i) = value_individual2(i);
    end
    else
        value_offspring1(i) = value_individual2(i);
        value_offspring2(i) = value_individual1(i);
    end
end
```

Using this approach, we get two offspring with approximately half of the values from each individual. The decision regarding which crossover scheme is suitable most often depends on the specific problem.

An important parameter for the crossover in general is the *crossover probability* p_c,

which decides how often the crossover will be performed. In the case of no crossover, the offspring of individuals are exact copies of their parents. Otherwise, we get new individuals for the next generation. The hope of performing a crossover is that the offspring will contain good parts of their ancestors, thus forming better individuals for the new generation. Typical values for the crossover probability are $p_c = 0.6$ to 0.9 [Gol89].

3.2.4 Mutation

In the phase of the GA named *mutation*, the structure of an offspring will be modified purposefully at one or more positions. A typical mutation process like the *flipping mutation* decides for each position of the individual whether it will be mutated and, if so, changes the value of that position. For the binary encoding, this can be realized by changing 1 into 0 and 0 into 1, respectively. We implement the flipping mutation as below:

```
for i = 1, ..., n do
    Randomly generate a uniformly distributed number r ∈ [0, 1];
    if r < p_m then
        Set value_individual(i) = 1 − value_individual(i);
    end
end
```

Figure 3.3 proposes an example where an individual mutates at position $z = 6$. The resulting individual differs exactly in the value of position 6, i. e. value 0 flips to 1.

	1	2	3	4	5	6	7	8
individual:	1	0	1	1	0	0	0	1
individual (mutated):	1	0	1	1	0	1	0	1

Figure 3.3: Individual before and after mutation

There are numerous mutation techniques. The *reversing mutation*, for example, generates a random position and flips the values next to it [SD07]. Furthermore, there are techniques that mutate individuals only if this assures an improvement in the evaluation value. Such a mutation method is called *hill-climbing mutation* [SD07]. In chapter 4 we will get to know a concrete mutation procedure that can be considered as part of these hill-climbing methods.

Similarly to the crossover, a basic parameter is the mutation probability p_m. The mutation probability decides whether each bit of an individual will be mutated or not. Obviously, if there is no mutation, the offspring will enter the new population without any change. If mutation is performed, parts of the offspring will be changed. This might allow jumps in the solution space, preventing its getting stuck in a local optimum. Additionally, the mutation process helps the population to prevent its diversity, so that it does not become too homogeneous over time. In general, random mutation will not occur very often and a guideline for mutation probabilities are $p_m = 0.001$ to 0.01 per bit. However, it must be said that the setting of the mutation probability as well as the crossover probability is a world of its own and often dependent on the problem instance. Those who would like to immerse themselves within this topic could refer to [Gol89].

3.2.5 Replacement

As the GA repeatedly replaces an old population by a new generation, the *replacement scheme* will be responsible for deciding which individuals will form the next population. An obvious way would be to replace the current population by adopting only the generated offspring. This leads to the drawback that no guarantee of monotonously increasing population is given. Accordingly, there may be occasions where the succession population has worse fitness than the previous population. This holds for both, the average fitness of the population as well as the value of the fittest individual. To avoid discarding the best individuals in the population, the principle of *elitism* can be implemented, whereby a copy of the top-rated individuals will be guaranteed for the next generation. In the instance of keeping the size of the population constant, one may first copy the elitist individuals and afterwards fill up the rest of the population with the generated offspring. Another advantage of elitism is the increased performance of genetic algorithms, since always keeping the fittest individuals leads to fast homogenization of the population and thereby to a fast convergence of the entire algorithm. Of course, this is not always desirable; hence, a good selection of parameters like the number of elitist individuals may be a worthwhile but not an easy task.

3.3 Advantages and Limitations

We utilize this section for a critical reflection on the topic of genetic algorithms. We would like to demonstrate the advantages of GAs, outline drawbacks and present where the method reaches its limits.

Advantages

Far-reaching application: Genetic algorithms provide a solution method for a multitude of optimization problems. The only requirement is that the problem must have the property to describe its decision variables with a finite individual encoding. Since the technique of genetic algorithms is not dependent on the problem surface, it can be used to solve complex problems. Even if we deal with non-continuous or non-differential problems where derivative methods fail, we can use GAs since they do not mind objectives that are not smooth. Additionally, genetic algorithms also scale very well for multi-dimensional problems or multi-objective optimization problems.

Multiple solutions: As the genetic algorithm iterates over populations it does not just provide a single solution but a set of candidate solutions. This can often be used as a basis for decision-making. In practice, there are numerous deciding factors and not all of them incur in the problem modelling. Hence, genetic algorithms place several solutions at the user's disposal and can later help determine the most appropriate one.

Easy incorporation: Genetic algorithms do not require advanced mathematical knowledge since the main steps are conceptually simple. Therefore, a quick introduction to the topic of GAs is possible even for non-mathematicians.

Drawbacks

Costly evaluation: The identification of the right evaluation function is inherently not a minor task. The calculation of the evaluation value is often very time-consuming, which leads to a bad performance in terms of the running time of the algorithm.

Local optima vs. global optima: Since we denote a solution as "good" only in comparison to the other solutions, we cannot provide evidence regarding the global optimum. Thus, there is no assurance that genetic algorithms will always determine the global optimum. Sometimes, they sadly get stuck in a local optimum even if we try to avoid this via the crossover and mutation phase where we hope to widen the search space. As a consequence, the GA will converge prematurely before it reaches a global optimum.

Parameter settings: The right choice of parameters is often the key to getting good solutions, even though the configuration of the parameters is not straightforward. The crossover probability p_c and the mutation probability p_m as well as the calibration among them have a strong impact on the performance of the algorithm. In addition, setting the size of the population and the abort criterion is also a sophisticated task, one that is often combined with time-consuming meddling with the right parameters.

In conclusion, it can be said that the suitability of GAs is dependent on several different factors: The amount of knowledge of the optimization problem as well as the texture of the objective function and the surface of the problem domain need to be involved in the decision. Other optimization algorithms might be more efficient for specific problems and some approaches that specifically address the special problem will perform better in terms of convergence. Nonetheless, genetic algorithms are a powerful tool, especially for solving problems where no exact polynomial time algorithm exists.

Chapter 4

Implementation of a Standard and Enhanced GA

Having studied the traffic network optimization problem ($TNOP$) and introduced genetic algorithms as a metaheuristic, we now focus on the implementation of the standard genetic algorithm and an enhanced genetic algorithm, which uses a GA combined with nonlinear programming. The implementation of the two programs just distinguishes in one step. Therefore we explain the structure of the program by means of both algorithms, the standard GA and the enhanced GA, yet specify the differences if necessary. The chapter sections are organized as follows: With the help of a graphical sequence diagram, we explain the program's principle of operation in section 4.1. We can break down the program into four steps and each step will be explained in detail. In section 4.2, the handling of the program is provided; hence, no knowledge about the program's design is required. For those who want to gain a deeper insight of the program, section 4.3 will provide the inner structure.

4.1 Program Structure

Let us remember the traffic network optimization problem ($TNOP$) from section 2.2:

$$
\begin{array}{lll}
\text{(TNOP)} & \max & u(x) \\
& \text{s.t.} & \sum_{i=1}^{n} r_i x_i \leq b \\
& & x \in \{0,1\}^n
\end{array}
$$

We are interested in finding a combination of projects p_i, $i = 1, \ldots, n$ that ensures the most utility $u(x)$ for the traffic participants with regard to a limited budget b. Let us solve the above problem with the help of genetic algorithms. By applying genetic algorithms to our problem, we are obligated to find a suitable encoding (see chapter 3). As $(TNOP)$ allows already binary values for the decision variable x, an obvious encoding is to represent an individual as a binary string of length n. In practical terms, if an individual contains a 1 at position i, this represents $x_i = 1$; i. e. project p_i will be realized. Moreover, an evaluation function has to be defined for the GA. As an individual represents a combination of projects and these represent a configuration of a network, we use MATSim to evaluate this network. Thus, MATSim acts as an evaluation function for us. We now have the necessary tool to solve the traffic network optimization problem for an instance $\mathcal{I} = (\mathcal{N}, \tilde{\mathcal{A}}, \mathcal{P}, b)$.

For a first impression of the program we look at the graphical sequence diagram in fig. 4.1. It depicts the entire time flow of the program and shows the program steps in detail.

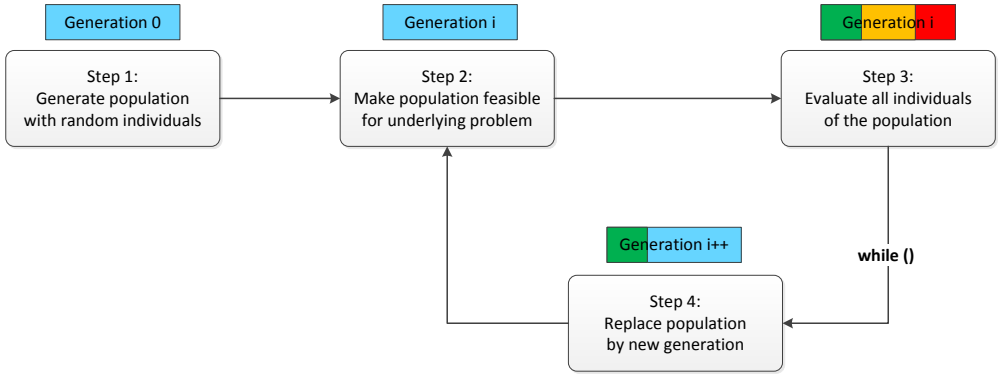
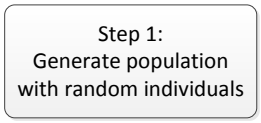


Figure 4.1: Graphical sequence diagram

Before getting into further detail, let us have a short overview of the topic. Typical for the genetic algorithm, a random population is generated, which forms Generation 0. At this stage, neither information about the evaluation nor about the feasibility of the individuals is available. This is followed by a step that is not typical for the genetic algorithm—namely, the validation of the individuals. Since we deal with some feasibility constraints, which warrant that the project cost will not exceed the budget, it is important to check whether the possible solutions are feasible or not. Therefore, step 2 of the program modifies all individuals in such a way that they later fulfil the feasibility

constraints. In order to evaluate all feasible individuals, the simulation tool MATSim will come into play. MATSim scores all individuals according to different measures and returns the corresponding scoring values (to review the several MATSim scoring, see section 2.3.4). The average utility will form the evaluation value of the genetic algorithm. One part of the evaluated population will be adopted unmodified into the next generation; this is the application of the elitism method. The remaining individuals will perform mutation and crossover. The offspring, together with the elite, will then form the new generation. With the recent population, the process starts once again and runs for as long as one of the abort criteria is fulfilled. We will now discuss steps 1–4 of the flow chart in detail.

Step 1: Initial Generation



The individuals of *Generation 0* are generated randomly. A first version of the program is implemented by chosen individuals randomly from a discrete uniform distribution. Consequently, every position of an individual is 0 or 1 with the same probability of 0.5. As a result each individual will consist of nearly the same quantity of 1s and 0s. This leads to the question of whether this is always meaningful for the various problems. Practical tests have revealed that generating individuals uniformly leads to combinations of more projects than can ever be feasible for the traffic network optimization problem. This leads to the idea of restricting the maximum number of 1s while generating the random individuals. The difficulty in doing so is to decide how many 1s we really want to produce. Hence, we resolved it by calculating an individual with the help of a normal distribution. It was found that the choice of a normally distributed number seems to be appropriated.

Obviously, the setting of the mean and the standard deviation is dependent on the specific problem instance $\mathcal{I}$. For example, a setting with a mean $\mu = 15$ and a standard deviation $\sigma = 3$, as shown in fig. 4.2, calculates between 9 and 21 ones with a probability of 95%.

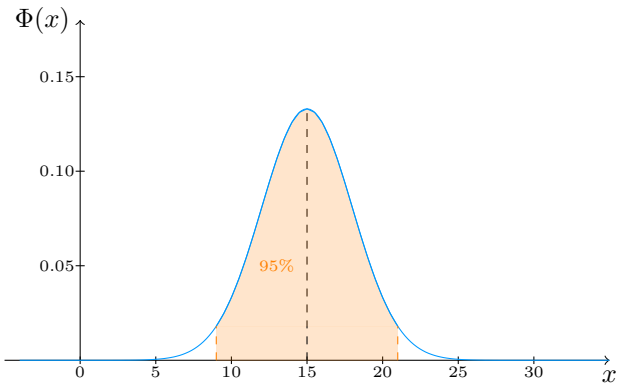
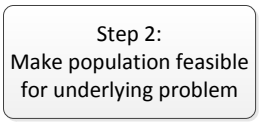
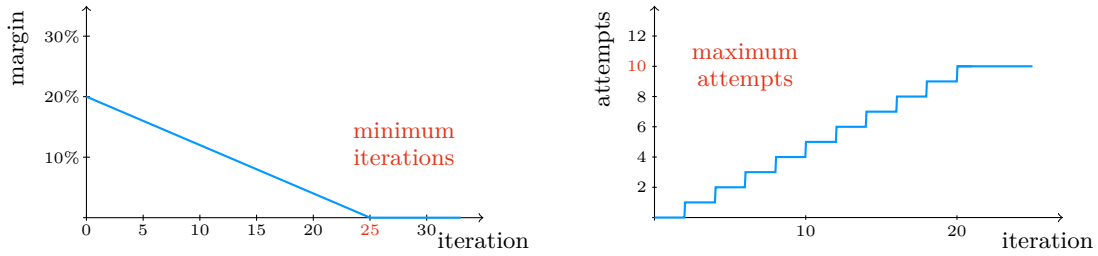


Figure 4.2: Probability density function: $\mathcal{N}(15, 9)$

Step 2: Feasibility



The feasibility of the individuals does not affect the further course of the genetic algorithm but is essential in such a way that we are just interested in potentially accessible solutions for the road expansions. Hence, before evaluating all individuals later on we want to assure the feasibility of all of them. When do we call an individual "feasible"? A first thought would be that an individual is feasible if its costs do not exceed the maximum budget. We additionally discover that, for our purpose, it could be interesting to handle individuals as feasible if they exceed the maximum budget "a little". An individual that is not feasible for the underlying problem can, however, be attractive for reproduction, for example, in the crossover phase. Nevertheless, in the end, we have to ensure that the maximum budget is observed. So, on the one hand, we appreciate individuals that are almost feasible; on the other hand, however, we want to guarantee feasibility in the end. An obvious solution and the one that is implemented here, is to introduce a weak budget condition. This condition appears as follows: In the beginning, we define a margin (percentage) within which we expect the limit to be violated and shrink that margin over time. As shown in fig. 4.3a, a linear reduction of the margin has been chosen. The margin decreases towards zero until the minimum number of algorithm iterations (in example: 25) is reached and from then on the maximum budget will be respected. The initial margin in that example is 20%, which means that the maximum budget can be exceeded by 20%; of course, any other margin can also be defined.



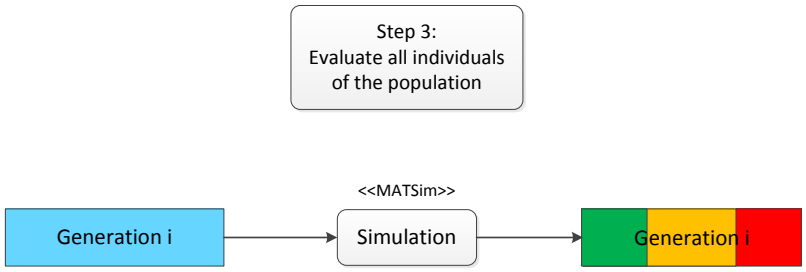
(a) linear reduction of margin over time

(b) number of attempts to add projects to an individual

Figure 4.3: Functions of step 2: Feasibility

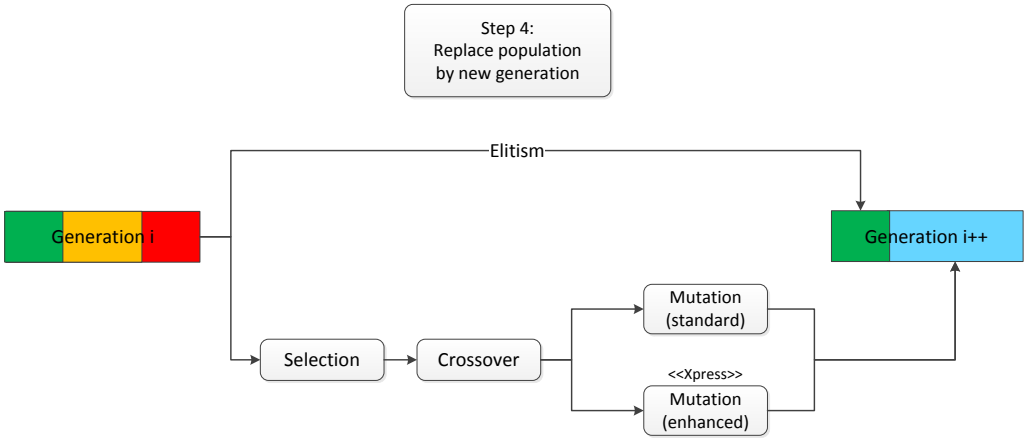
Another idea that emerged while testing the algorithm is the following: There are feasible individuals that are far away from the given budget limit and they most likely do not suit our interest for a solution. Hence, we force the individual to be close to the maximum number in the way that we add projects as long as it still remains feasible. As we represent the realizations of projects by an individual with binary values, this operation involves changing some values from 0 into 1. Similar to the weak budget condition, the number of attempts for performing this "adding" operation increases over time. More precisely, the number of attempts to fill up an individual with 1s increases linearly with the number of iterations. In fig. 4.3b we used $\#\text{attempts} = \lfloor 0.5 \cdot \text{iteration} \rfloor$ until the maximum number of attempts is reached. Since the number of attempts must be an integer, we round down the result. Thus, we get a step function that remains constant for a given number of maximum attempts. Again, this number of maximum attempts is instance-oriented and must be defined beforehand.

Step 3: Evaluation



The simulation tool MATSim is responsible for the evaluation and, for this purpose, it provides four different evaluation values. The one that will be used here is the average score of the agents since it seems to be the best measure for our problem. Also possible are the best value, the worst value, and the executed value. The rule here is, that the larger the values, the better it is for the agents. Another attribute that MATSim calculates is the average trip duration. This value does not influence the algorithm but will be saved for comparison with the average trip duration of the original network. For different adjustments in algorithms that MATSim uses and that consequently also affect the output values, one should refer to the MATSim tutorial [MAT11] and the MATSim user guide [Rie14]. The option to run the MATSim simulation for all individuals in the population parallel to each other is given, but then we have to accept a 2–3 % aberration from the original evaluation values. It is up to the user to decide how exact the solutions should be in the end.

Step 4: Replacement



The replacement is the step where the creation of a new population is processed. It consists of various sub-steps that we already defined purposefully in chapter 3.

1. *Elitism:* The implemented program uses the method of strong elitism, where the best individuals will be adopted into the upcoming generation without undergoing any change. This leads to the guarantee of monotonously increasing evaluation values over generations. How many individuals will form the elite can be adjusted in advance, but in order to guarantee a monotonously increasing evaluation of the fittest individual, a maximum number of one elitist individual is enforced.

2. *Selection:* The type of the selection method is switchable. The first method just takes the best individuals in the population as a basis for the reproduction. This method, which in the literature is often called *elitist selection*, might not be the best choice since those individuals with poor evaluation die out but might nonetheless be a good parent for reproduction. To manage this drawback, the roulette wheel method is implemented. We introduced this selection method already in section 3.2.2 but will have to change it a bit, since MATSim also produces negative score values if the number of iterations is adjusted too low in the configuration. Moreover, there is no fixed range for the evaluation value. For that reason, whenever there appears at least one negative evaluation value, we modify the method in the following way: We calculate the range from the worst to the best evaluation value and assume the *MATSim range* of evaluation values to be 10% greater then the calculated interval. By extending the range, we get a *roulette wheel bound*. Shifting this lower bound to zero now, we get strictly positive fitness values and can continue with the usual roulette wheel procedure. For a better understanding, consider fig. 4.4 below. Among all the blue evaluation values, a smallest value of -13.0 and a largest value

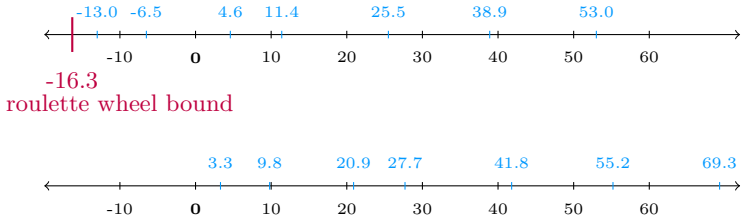


Figure 4.4: Adjustments for the roulette wheel selection

of 53.0 give us a range of size 66.0. Extending the range by 10% yields a roulette wheel lower bound of -16.3. Note that the 10% extension is assigned half to the lower side and half to the upper. This leads to the roulette wheel lower bound being exactly 5% smaller than the worst evaluation value with respect to the whole interval. On shifting the lower bound to zero, all subsequent evaluation values will be positive. One may ask why we chose exactly 10%? To put it simply, this gave us the best results while testing a couple of different values. It might occur that, for another instance, the range has to be adjusted. Observe that if no negative values appear, we apply the usual roulette wheel selection, as in section 3.2.2.

3. *Crossover:* A one-point crossover will be performed according to the initially specified probability p_c. In this case, the crossover point is a randomly calculated position between 1 and $n-1$ with equal probability for each position. An interesting thought would be an intelligent crossover that may appear as follows; try to measure sub-networks and, if they seem to be good, keep them together in the crossover

phase. In all likelihood, this is beyond the scope of this thesis, especially considering the dimension of the present data, but could be worth investigating with some effort later.

4. *Mutation:* We have already mentioned that the genetic algorithm is implemented in two different ways. Until this point, the standard GA and the enhanced GA have worked in exactly the same way. The phase of mutation is now the point where these genetic algorithms differ. On the one hand, we apply a random flipping mutation as described in section 3.2.4 but do not decide for each position of an individual whether it will be mutated or not. Instead, we randomly select a position which will then be flipped. We differ here a little to be able to compare the standard GA with the enhanced one. Recapitulate the mutation procedure of the standard GA below:

   1 Select an individual corresponding to p_m;
   2 Randomly generate an integer $z \in \{1, \ldots, n\}$;
   3 Set $\text{value}_{\text{individual}}(z) = 1 - \text{value}_{\text{individual}}(z)$;

   As in the crossover phase, there is a previously set percentage rate p_m that tells us how often mutation should be applied. It should be noticed that the mutation probability p_m must be adapted, since in the flipping mutation the probability refers to each bit whereby the probability in the aforementioned method refers to each individual as a whole.

   For the enhanced GA, we now discuss a different mutation approach that changes values meaningfully rather than at random. We can classify this procedure among the hill-climbing mutation methods. The objective is to change each individual in a way that the mutated individual is the best one among all its neighbours. This procedure is implemented with the help of the local optimization mentioned in section 2.3.3:

   1 Select an individual corresponding to p_m;
   2 Decode individual, i. e. individual $\rightsquigarrow \mathcal{N}$;
   3 Determine best network $\mathcal{N}^*$ within the neighbourhood of $\mathcal{N}$;
   4 Encode network, i. e. $\mathcal{N}^* \rightsquigarrow \text{individual}_{new}$;
   5 Set individual = individual_{new};

   Assume that we select an individual which shall be mutated now. We are mutating this individual by applying Algorithm 1 of section 2.3.3. Thus, we calculate its “neighbourhood”, which is defined as a network that differs by at most k

edges compared to the initial network (represented by the current individual). Subsequently, we compute an evaluation value of each network in this neighbourhood. For the evaluation, we solve (NLP) in the sense of section 2.3.3, where the idea is to calculate a user equilibrium by solving a system optimum flow problem. The objective value that reflects the travel time will thus deliver the evaluation value. We then mutate an individual in the way that we replace it by the best evaluated network in its neighbourhood (encoded again as individual).

4.2 User Documentation

We now dedicate ourselves to the handling of the program. It is not necessary to know the exact structure of the program in order to use it. We just interact with the input and output data, the program handles the rest by itself. Let us take a closer look at the input.

We are confronted with different kinds of input data. Some of them concern the specific problem and belong to the data defining an *instance* $\mathcal{I} = (\mathcal{N}, \tilde{\mathcal{A}}, \mathcal{P}, b)$ of the traffic network optimization problem. Other inputs are destined for the algorithm itself these are *parameters* that help to fine-tune the algorithm and still others are just needed for *external* programs.

Input: instance

The *networkOriginal.xml* file, which describes the currently existing network graph $\tilde{\mathcal{G}}$, is self-explanatory. In the same manner, the *newStreets.xml* file defines a network graph $\tilde{\mathcal{A}}$ corresponding to the possible new streets. The structure and setting of these files are fixed, because MATSim will operate on them. We find the set of projects $\mathcal{P}$ in *project.txt*, where we list all the projects. Moreover, every project is assigned a project ID (fixed integer) and the realization costs. By implication, the number of projects will schedule the length of an individual later on. In *budget.txt*, a positive real number is saved and it represents the overall budget b.

Input: parameters

All relevant parameters will be specified in the *parameter.xml* file. These include parameters such as the size of the population, the mutation and the crossover probability, the number of MATSim iterations, and the stop criteria. If we do switch between different

problem instances, this input file will be the only one we really work on.

Input: external

As mentioned already, MATSim needs different types of input data. There are the *config.xml*, the *counts.xml*, the *facilities.xml* and the *plans.xml*. For the exact design instructions of those files, we refer to the MATSim tutorial [MAT11]. For solving the nonlinear program in the enhanced approach, we need to deposit the files for the optimization program, *optimization.mos* and *optimization.bim*, where the network flow problem is implemented.

After declaration of the input data, we save the data in a folder called *input*. It is now time to execute the program, either in our favourite development environment or through a command in the console. After termination of the program, the output data is ready. We find the output data in a folder called "run" and a sub-folder named by the date and time we started the program. Within that folder, all the necessary output files are archived. Different types of output can be found here.

Output: parameters and information

The *parameter.xml* is merely a copy of the parameter file from the input folder. It conduces to be able to follow the constellation we used for the specific test run after trying different input parameters.

In *infos.txt* the program collects information about start time, end time and running time as well as algorithm details like the number of calculated generations and the number of equal individuals within the last population. It sheds light on how well the test run was and which factors might need cosmetic changes.

Output: solution and development

Probably the most interesting file is the *solution.txt*, in which the best calculated construction plan is displayed. Useful facts such as the overall costs, the budget gap and improvement to the original network can also be found here. The solution network $\mathcal{N}^*$ with stored nodes and edges can be found in *solutionNetwork.xml*.

To not just accept the presented solution but rather to understand the process of iteratively generating solutions for the underlying problem, the *evaluation.txt* will help. It reports the systematically increasing evaluation values—for example, experience regarding whether

the output solution was generated just in the last iterations or maybe long before the algorithm actually terminates.

Output: other

Additionally, some files for visualization of the solution network in Matlab can be found in the folder *dataMatlab*. To be able to compare the results to the input network, we find the evaluation values of the initial network in the folder *original*.

4.3 Developer Documentation

This section explains the design and composition of the program. The program has been implemented in Java (version 1.7.0_65) whereby the IDE (Integrated Development Environment) Eclipse (version 4.3.0) has been used. There are interfaces to the simulation tool MATSim (version 0.5.0) and to the optimization solver FICO Xpress (version 7.7); the nonlinear optimization program is implemented in Mosel (version 3.6.0) using the environment Xpress-IVE (version 1.24.04) [Xpra] [Xprb].

Essentially, there are four blocks working absolutely independent of each other; in Eclipse, these blocks are represented as packages. First of all, there is the *domain* block that comprehends everything relating to networks, projects, and extensions. Here, one finds all the information about the real problem flow in one place. The characteristics of the original network, as well as the street extensions and possible new streets are processed here. The second is the *genalgo* block. It is easy to assume that this part handles the whole genetic algorithm, which ranges from generating random populations to implementing mutation and crossover. Note that the genetic algorithm deals only with sequences of zeros and ones, and knows nothing about networks or projects and their costs. Last but not the least are the *matsim* block and the *xpress* block, which are crucial for evaluating the network. In these packages the transferral of the required data to the external programs takes place. Since all blocks work autonomously, the solver package brings them all together, which means that it interacts with all packages and ensures the coherence of the whole program. In table 4.1, the design and the aforementioned structure of the program is visualized. The used colours follow Eclipse, where classes are coloured green, interfaces violet and packages brown.

In each package, different classes represent the objects we deal with; in the following, major key classes will be introduced.

solver			

TrafficOptimizer (main){
1. Initialize relevant data.
2. Start Genetic Algorithm.
3. Display solution.
}
ToptParameter
MatSimScorer implements Evaluator
NetworkFeasibilityChecker implements FeasibilityChecker
MutationRandom and MutationOptimizer implements Mutator			

domain	genalgo	matsim	xpress
Project Construction Network NetworkExtension	Individual Population GenAlgoFunctions GenAlgo Evaluator FeasibilityChecker Mutator	MatSimController ScoreValues UtilityValues	XpressController MoselGraph MoselEvaluation

Table 4.1: Program design

Package: domain

As mentioned before, the *domain* package is responsible for managing the network itself. An object of class Project is unique in its project ID, consists of the project's realization costs and contains a set of street IDs. On the other hand, the class Construction consists of a set of projects and reflects a possible construction plan as a solution to the underlying problem. Every network with its links and nodes will be represented by an object of class Network. Moreover, the class NetworkExtension is able to transform a set of IDs into a construction plan, which means that it maps a specific integer to the right project. Additionally, it decides if a certain extension of the original network is feasible or not.

Package: genalgo

In *genalgo*, an object of class Individual contains, besides a 0-1-array, the evaluation value and the fitness value of an individual. Clearly, the class Population consists of a

list of individuals. GenAlgoFunctions involve static functions whose tasks are to modify individuals and populations through mutation, replacement and crossover. In GenAlgo, the proper process takes place. Verifying the feasibility of an individual, evaluating it with MATSim and mutating individuals in different ways would represent an interaction with the other blocks (*domain*, *matsim* and *xpress*); thus, they are implemented as interfaces.

Package: matsim

In the package *matsim*, we find the MATSim controller, where changes can be made in the configurations and the actual MATSim run takes place. Moreover, a class ScoreValues is introduced and is appropriate for the MATSim results. One object of the class ScoreValues captures the four utility values and the average trip duration of one MATSim run.

Package: xpress

All classes corresponding to the optimization solver Xpress are part of the package called *xpress*. As we deal with a multi-commodity flow problem, we define a special class named *MoselGraph* representing a graph $\mathcal{G}$ in a network $\mathcal{N}$. Settings pertaining to Xpress we handle in the class *XpressController*. Once we run Xpress, we save the corresponding evaluation values of that graph in *MoselEvaluation*.

Package: solver

In the *solver* package, the main method, which controls the entire traffic optimization process, is implemented. As mentioned earlier, the solver package manages interaction between the four areas. The important aspect is the following: None of the packages knows what the solver package is or does but the solver package itself is well aware of the others. This allows communication between all of them and passes on relevant information without violating their autonomy. Whenever some communication between those four packages is necessary, the solver package mediates between them. To realize this, we implement the interfaces in the solver package. Furthermore, within the main method, one can define the kind of mutation. The Class MutationRandom represents the standard GA with random mutation, whereas the class MutationOptimizer represents the mutation of the enhanced GA.

Chapter 5

Results

So far, we have described the traffic network optimization problem as well as introduced the implementation of two algorithms: the standard genetic algorithm and a further developed algorithm, which arose from the former one. In order to proceed with the structure of the current chapter, let us quickly recapitulate the essential property of the enhanced algorithm compared to the standard one. We developed an enhanced GA that uses kind of a hill-climbing mutation. Hereby, the mutation procedure has been implemented with the help of a nonlinear optimization program whose objective represents the evaluation value of a given network. Based on this evaluation process, the algorithm performs mutation by changing an individual into the best one in its neighbourhood. Therefore, this chapter aims to compare the results of the enhanced algorithm with those of the standard algorithm, i. e. with a random mutation procedure. The sections in this chapter are organized as follows: In section 5.1, we introduce the test data that is used to appraise both algorithms. Section 5.2 captures the parameter setting of the test runs as well as their outcome. In order to assess the performance of the algorithms, we focus on analysing the algorithms' running times in section 5.3.

5.1 Test Data

We now introduce the practical data for the algorithms' test runs. To apply the developed algorithms, we need test data in terms of an instance $\mathcal{I} = (\mathcal{N}, \tilde{\mathcal{A}}, \mathcal{P}, b)$ of $(TNOP)$. We call the specific instance of our test data $\mathcal{I}_{Perlach}$, since all test data originate from Munich's Department of Public Order and pertain to the district "Perlach". Most of the data have been edited afterwards by a research group at the Technical University of Munich (comprising Johanna Brandstetter, Lisa Gerstner, Saskia Schiele, and Maria Theresia von Soden).

For a review of the different objects of an instance $\mathcal{I}$ we give a brief subsumption in a hierarchical order but for detailed definitions of the used terms we refer to chapter 2:

Problem instance $\mathcal{I} = (\mathcal{N}, \tilde{\mathcal{A}}, \mathcal{P}, b)$
Network $\mathcal{N} = (\tilde{\mathcal{G}}, \mathcal{C}, q, s, d, t)$
Network graph $\tilde{\mathcal{G}} = (\mathcal{G}, cap, t^0)$ with graph $\mathcal{G} = (V, E)$ (original)
Network graph $\tilde{\mathcal{A}} = (\mathcal{A}, cap, t^0)$ with graph $\mathcal{A} = (V, A)$ (extension)
Projects $\mathcal{P}$
Budget b

The required data concern different types of information and, thus, can be divided into three categories with regard to their content:

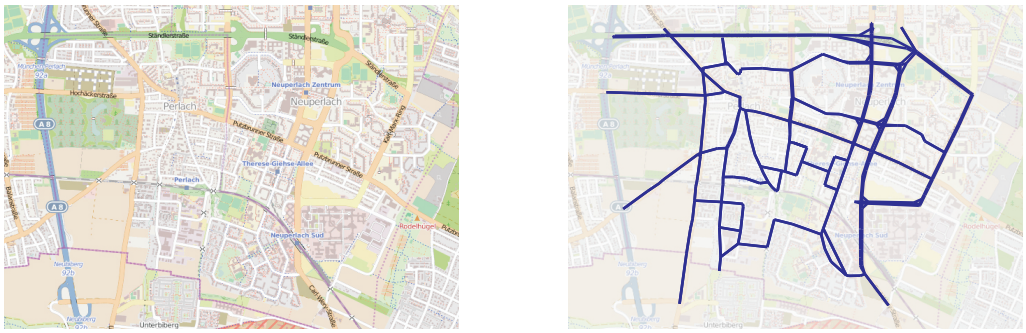
1. Data about the considered area (network environment) $\tilde{\mathcal{G}}$
2. Data about traffic flows (network behaviour) $\mathcal{C}, q, s, d, t$
3. Data about construction projects (network development) $\tilde{\mathcal{A}}, \mathcal{P}, b$

We will now explain in detail the three types of data provided within the scope of Perlach's traffic network optimization.

Network Environment

As mentioned already, the area covered by all of the data is Munich's district "Perlach"; therefore, the network environment provides data about the road network of Perlach. See fig. 5.1a for a street map of the considered district.

Primarily, street information is provided by *OpenStreetMap* (OSM) [Ope04], a platform that offers geographic data and is freely accessible and editable. The abstraction of Perlach's streets into a graph $\mathcal{G} = (V, E)$ as well as some edge information such as the free-flow travel times t^0 and the capacities *cap* have been amended by the above-mentioned research group. Figure 5.1b illustrates the resulting graph $\mathcal{G}$.



(a) Map of Perlach **(b)** Network representing Perlach

Figure 5.1: Munich's district Perlach

Network Behaviour

But data about the environment is not the only thing that is required. We learnt that in order to evaluate networks, we need information about traffic flows. Hence, to estimate traffic flows in Perlach, data regarding road users' travel behaviour must be collected. The simulation tool MATSim even requires timely day schedules of road users. For this purpose, the considered area is examined with consideration of the facilities where high levels of traffic are expected. Such facilities are, for example, educational facilities like schools, day-care centres and nurseries as well as super markets, pharmacies, large companies and hospitals. In the areas around these facilities, we expect high volume of traffic at certain times; subsequently, we take this information as a source to generate knowledge about the day plans of agents, i. e. activities of Perlach's residents. All agents' schedules were issued by the research group. Based on these schedules, MATSim calculates agents' utility values.

Since we use a different evaluation of networks in the mutation phase of the enhanced algorithm, we need to abstract these day plans into general traffic flows. Hence, we do not use the day schedules of agents that travel from an arbitrary point in the network to another. Instead, we consider traffic flows only during two periods and, in addition, allow them only between certain sectors. For the classification of the sectors, a simple division into four parts is used (see also fig. 5.2). The source q and sink s of a commodity are special vertices in the corresponding sector. An important aspect is to assure that these vertices are incident with several edges, since complete traffic rates emanate from them. This is often implemented by inserting a *supersource* or a *supersink* respectively. These supersources/supersinks are connected to a couple of nodes in the corresponding sector by a zero-cost edge. Specifically, this means an edge (q, i) from source q to some

nodes $i \in V$ with capacity $cap_{qi} = \infty$ and free-flow travel time $t^0_{qi} = 0$. The same applies for sinks. Both the supersource and the supersink of a commodity are naturally added to the existing graph and are treated like all of the other nodes. After introducing such supersources and supersinks to each of the four sectors, we can associate 12 commodities between these sectors. Each source and each sink of a commodity consists of demands $d \neq 0$, while for all other nodes $d = 0$ applies. Consider fig. 5.2 for the exemplary view of a commodity from sector 1 to sector 2.

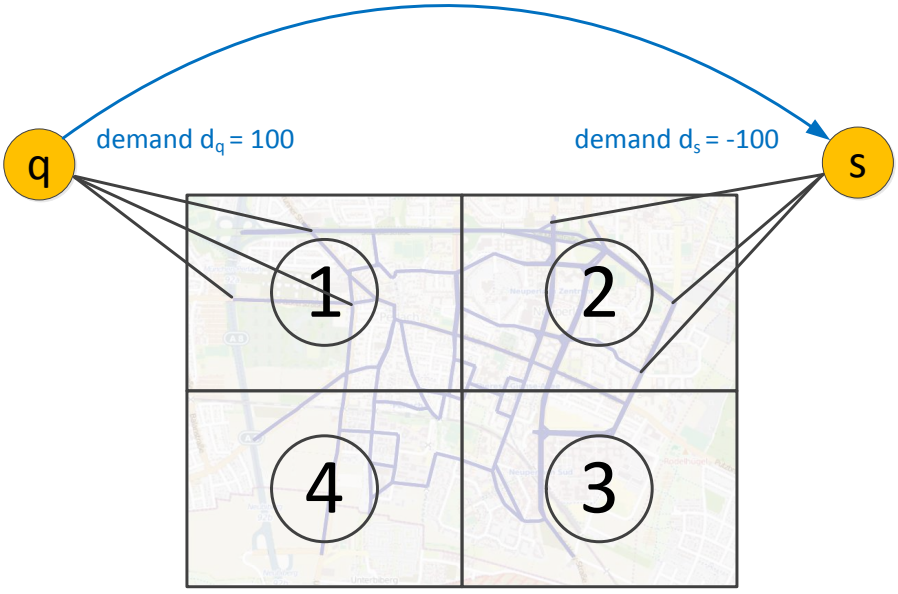


Figure 5.2: Abstraction of day plans into commodities

In addition, however, we consider the traffic rates of the commodities for different day times. The two considered periods are from midnight to 1pm (morning) and from 1pm to midnight (afternoon/evening). Due to these considerations are revealed $|\mathcal{C}| = 12$ commodities from any sector (1, 2, 3, 4) to another, all considered at the morning in the one case and at the evening in the other. Each source and each sink consists of demands $d \neq 0$; we extract these demands from the day schedules of the agents and, afterwards, merge them together.

Furthermore, as a travel time function, we use the one recommended by the Bureau of Public Roads [PR64] with constants $\alpha = 1$ and $\beta = 2$:

$$t_{ij}(f_{ij}) = t^0_{ij}\left(1 + \frac{f^2_{ij}}{cap^2_{ij}}\right)$$

These constants were chosen due to mathematical convenience, since quadratic objects are easier to handle than those for $\beta > 2$.

Network Development

Since the goal of traffic network optimization is to ensure smooth flow of traffic in the future, we will modify the street configuration in Perlach. Naturally, this will not be done in a haphazard manner; rather it will be handled by considering different street expansions and street constructions. To describe possible new streets, the Department of Public Order scheduled a list of these roads with corresponding street information; this represents the network graph $\tilde{\mathcal{A}} = (\mathcal{A}, cap, t^0)$.

The set of expansion projects $\mathcal{P}$ carried out by the Department of Public Order is defined by $|\mathcal{P}| = 110$ projects. A single project comprises between one and 18 streets, that can either be expanded, newly-constructed or reduced. Each project $p \in \mathcal{P}$ contains project cost $r \geq 0$—i. e. costs that are estimated by the department corresponding to the realization of this project. The available budget b amounts to 18,401,146, i. e. there is around 18.4 million Euro for the realization of construction projects in Perlach.

To get a feel for the dimension of the problem, note that for the given 110 projects, there exist $2^{110} = 1.3 \cdot 10^{33}$ project combinations. Obviously, not all of them are feasible—more precisely, there never can be realized more than 20 projects in order to adhere to the budget; but then we are still left with 1,048,576 possible project combinations.

5.2 Parameter Settings and Test Runs

After introducing the data taken for further consideration, this section surveys the predefined settings for the different test runs as well as appraises the outcome of these runs. We focus on test runs that are able to compare the two algorithms of this thesis: First, a standard genetic algorithm (standard GA) that performs mutation randomly and, second, a further developed algorithm (enhanced GA) that mutates individuals meaningfully into better evaluated ones. In order to match these two algorithms, we run them with the same setting of parameters. These parameter settings as well as the amplitude of considered test runs are presented in section 5.2.1. In section 5.2.2, we analyse the outcome of the run under several aspects. The first consideration focuses on the final solution network itself and evaluates it without involving its development. Subsequently, we take precisely this evolution into account and observe, for example, if the final solution is achieved with the last iterations or long before the algorithm is

actually terminated. In the end, a general view of the practical outcome is taken, since our target was to help in decision-making and, thus, we will focus on the realized projects themselves.

5.2.1 Parameter Setting

Since we consider two different algorithms, with the parameter setting we have to ensure their comparability afterwards. All parameters have been ascertained in plenty of runs beforehand and the adjusted values were rated "good" according to the given problem instance $\mathcal{I}_{Perlach}$. We find the setting of all necessary parameters in table 5.1.

Genetic algorithm	
population size	15
mutation probability	0.2
crossover probability	0.75
percentage of elite	0.2
maximum percentage of equal individuals	0.8
minimum number of iterations	50
maximum number of iterations	100
Others	
MATSim iterations	60
MATSim evaluation value	utility (average)
Xpress evaluation value	average (am/pm)
edge difference of neighbour	2
percentage of margin	0.2

Table 5.1: Parameter setting

With the above parameter setting of the genetic algorithm, we run the program by a consideration of $m = 15$ individuals. The length of an individual is specified automatically by the number of available projects, i. e. $n = |\mathcal{P}|$. With the above-mentioned percentage, three out of 15 individuals will be handled as elite, whereby the remaining individuals perform crossover with a probability $p_c = 0.75$ and mutation with a probability of $p_m = 0.2$. The mutation probability refers to the whole individual and is tantamount to a per-bit mutation probability of approximately 0.0018. The algorithm will terminate if one of the abort criteria is fulfilled. This may be a homogeneous population where more than 12 individuals are the same on the one hand, or if a maximum number of 100 iterations are reached on the other. The former stop criterion will not be met as

long as the minimum number of 50 iterations is achieved. We ascertain 60 MATSim iterations (for a motivation of this number see section 2.3.4). The average scoring of an agent over the 60 MATSim iterations serves as an evaluation value for the further course of the genetic algorithm. As mentioned in section 2.3.4, also available are the best, the worst and the executed values. Since we calculate an evaluation value within the phase of mutation according to two periods, we take the average travel time of traffic flows in the morning and of those in the evening. Furthermore, we use $k = 2$ edges difference for the definition of a network neighbourhood. Obviously, there is no need for this information when running the standard genetic algorithm. Finally, the percentage of margin indicates the extent to which one is allowed to exceed the budget until the minimum number of iterations is reached. Parameters outside those in table 5.1 occur as mentioned in chapter 4.

5.2.2 Test Runs and Results

All test runs were undertaken on the compute server of TU Munich (Dell PowerEdge R815-System), equipped with 2.2 Gigahertz 4x AMD Opteron 6174 processors, 48 cores and a main memory of 128 GB RAM. The simulation is taken on by MATSim and, as an optimization solver for the enhanced algorithm, we use FICO Xpress. We started both algorithms with the parameter setting in table 5.1, whereby we run each algorithm ten times. Since randomness plays a major role especially in generating an initial population, we generally take the average of the algorithm's outcome into consideration.

To evaluate the performance of both algorithms, we specify three types of outcomes:

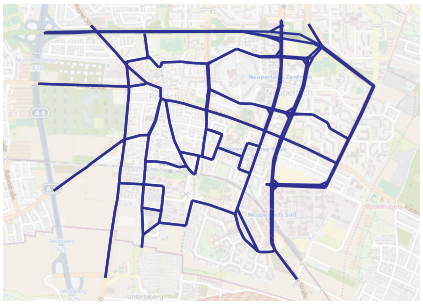
- Quality of resulting solutions
- Development of resulting solutions
- Characteristics of resulting solutions

The quality of the final solution of all test runs will be primarily measured by means of MATSim's average utility value, since it represents the evaluation of the network. The consideration of the solution development serves as an indicator of slow/fast convergence to culmination. For this purpose, we use the average utility value over all runs as a measurement. Furthermore, we additionally consider the evolution of population's trip duration since the enhanced algorithm take travel times not utility values as a basis for the evaluation. Finally, a breakdown of the chosen projects provides practical, decision-making support. Here, we try to recognize some patterns in the observed results. Are

there some valuable projects that are within most of the solutions or does any solution provide completely different project combinations?

Quality of Resulting Solutions

Before starting to evaluate the final solution, we examine the performance of the original network $\mathcal{N}$ of instance $\mathcal{I}_{Perlach}$ since this forms the basis of appraising the algorithm's solution. For the key data of $\mathcal{N}$, see fig. 5.3.



MATSim data of the original network

utility (executed)	102.829
utility (worst)	44.722
utility (average)	89.373
utility (best)	116.081
average trip duration	493.700

Figure 5.3: Original network $\mathcal{N}$

Obviously we run the optimization algorithms to reach a better utility; especially for the average utility values, we expect eligible improvements, i. e. higher values. However, the average trip duration should decrease as well. We mainly focus on these two values. The final values of the ten test runs are summarized in table 5.2.

	standard genetic algorithm		enhanced genetic algorithm	
	average utility	trip duration (sec)	average utility	trip duration (sec)
run 1	113.320	271.96	111.945	291.51
run 2	114.022	264.83	110.033	328.63
run 3	115.047	298.62	110.297	281.12
run 4	111.473	315.70	112.856	292.28
run 5	113.179	283.60	112.624	293.57
run 6	113.405	289.99	110.248	291.15
run 7	113.115	307.29	115.160	300.05
run 8	110.673	343.11	113.338	315.92
run 9	115.002	288.69	111.663	297.36
run 10	115.203	298.92	113.783	316.40

Table 5.2: Final results of each of the ten test runs

In the following, the outcome of the standard genetic algorithm is coloured blue by default and the enhanced one is coloured red.

The final solution is the best individual of the generated populations and represents a combination of projects; this, in turn, is a specific network. Therefore, we closely examine the average utility value, since we use this one as an evaluation value in both algorithms. Since a graphical interpretation of the values in table 5.2 leads to get a better appreciation of the outcome, we illustrate the outcome in the form of a diagram. Figure 5.4 shows the average utility value of the final network for each of the ten runs. The blue bars belong to the standard genetic algorithm, the red ones to the enhanced genetic algorithm, and the green line represents the value of the original network (compare to fig. 5.3).

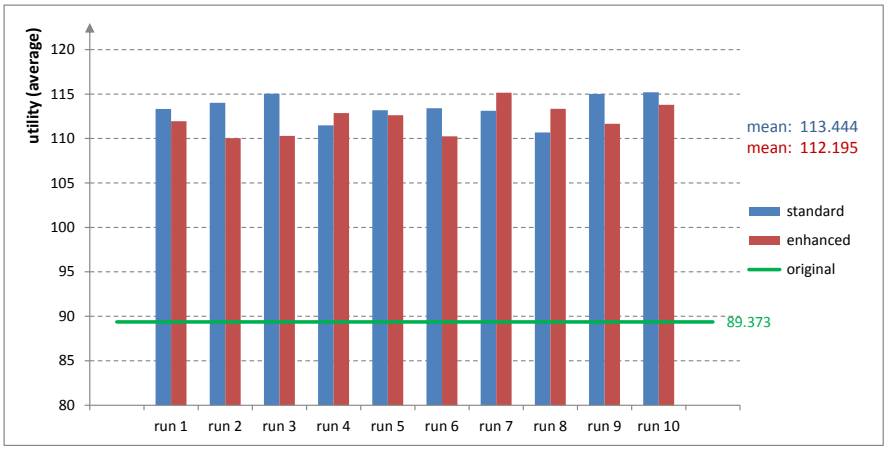


Figure 5.4: Best utility value of runs 1–10

We can clearly identify that both algorithms deliver utility values that are considerably better than the utility value of the original network. Another observation is that the two algorithms received similarly high values. The highest utility value of the standard algorithm amounts to 115.203 and the highest value of the enhanced one amounts to 115.160. The same applies for the lowest utility value (standard: 110.673, enhanced: 110.033). Nevertheless, on average, the standard algorithm (blue mean) reaches a slightly higher utility than the enhanced algorithm (red mean). As a result, if we look at only the final solution, we find that the enhanced algorithm brings no benefits over the standard one with random mutation. Note that initial generation is completely random; thus, the consideration of just the final utility values does not provide information about the performance of the algorithms.

We can say that both algorithms perform well with respect to the final solution but

presently there is still no indication that the further developed algorithm performs better.

Development of Resulting Solutions

So far, we have concentrated only on the obtained solution in the form of the best network. However, this is not the only thing to take into account while evaluating the algorithm. It is also highly relevant to consider the behaviour of the algorithm. In practical terms, this means taking into account the development of the considered values. Figure 5.5 thus shows the evolution of the best individual within a generation over iterations. In this consideration, the average utility value over all runs is used as a yardstick. The blue curve represents the utility values of the standard genetic algorithm whereas the red curve depicts the enhanced genetic algorithm.

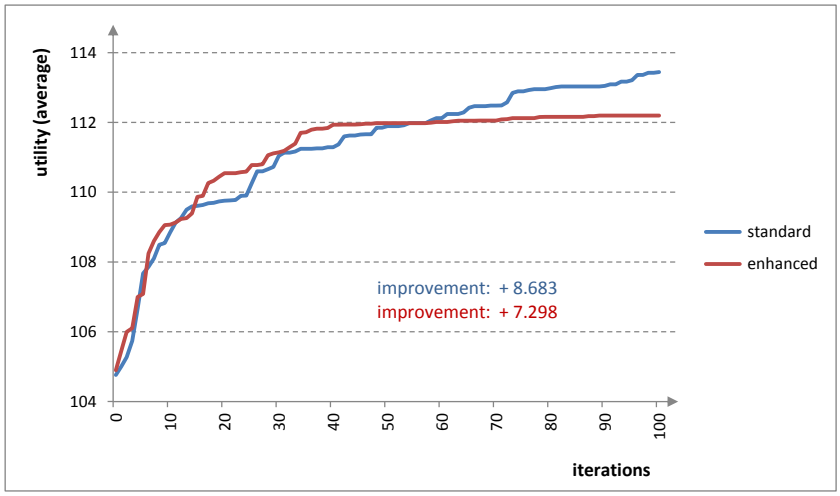


Figure 5.5: Development of the utility value (type: average)

It is seen that both algorithms significantly improve their initial utility values; however, there are considerable differences in the curve progression. They start with almost the same initial utility value, although Generation 0 is created at random. The development of both curves does not differ very much in the first 50 iterations. Then, the following takes place: While the utility values of the blue curve (i. e. the curve of the standard algorithm) is growing consistently until iteration 100, the red curve (i. e. the curve of the enhanced algorithm) increases only slowly until the maximum number is reached. In total, the standard genetic algorithm could improve the utility values by more than 19% in comparison to the enhanced algorithm. What is the reason for this? Since the enhanced algorithm performs a local optimization in the mutation step instead of

randomly mutating an individual, the stagnant curve, beyond half of the iterations, can be an indicator of being trapped into a local optimum. Consequently, a local search within the neighbourhood of a network achieves no higher utility, whereas the random mutation could have led to a jump in the search space, enabling the algorithm to further improve its utility values.

An alternative hypothesis to explain the poor performance of the enhanced algorithm in the later iterations is that "optimized" individuals will not be evaluated better in the simulation. Note that the local optimization is performed with the help of a different evaluation method that does not use simulation. Therefore, it is possible that the two evaluation methods are too diverse. Since the evaluation in the neighbourhood search is based on computing total travel times in a network, we should consider the development of the trip duration to get concrete evidence. Figure 5.6 is structured identically to the earlier figure where the blue curve belongs to the standard genetic algorithm and the red curve to the enhanced one. We now consider the evolution of the trip duration. For this, we take the mean of all individuals of the current generation and again observe the average over all runs.

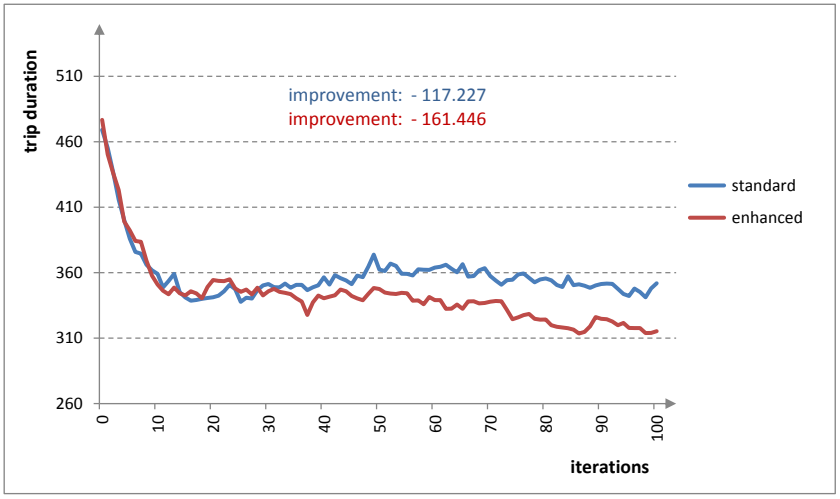


Figure 5.6: Development of the average trip duration

Again, both algorithms decrease their trip durations, even if these values are not included in the evaluation function. For the present, this observation leads one to assume a dependence of the trip duration and the agents' utility. Furthermore, it clearly states our aforementioned hypothesis. While the standard algorithm only decreases the trip duration by only 117.227, the enhanced algorithm excels with far higher improvement of 161.446, which corresponds to a better result of more than 37% . In concrete terms,

the enhanced algorithm heightens the travel time of single individuals of a population, i. e. in average the population yields better values with respect to the trip duration (cf. fig. 5.6). However, this improvement of the trip duration does not influence the utility value to such a degree, that there was great implications for the utility.

Overall, one can say that the enhanced algorithm does not lead to better utility values. However, this might rather be due to the deviation of the evaluation methods than to the failure of the algorithm. Quite the opposite, fig. 5.6 demonstrates a strong influence on the trip duration. Thus, in future, more time should be invested in levelling out the discrepancies of the two evaluation methods.

Characteristics of Resulting Solutions

Another aspect that we have already mentioned above is to examine closely the solutions' properties. Since we are not just interested in finding the "one" best solution, but rather a pool of eligible solutions to provide decision-making support. One such guidance for a decision might be to consider the most valuable projects that are suggested by the algorithms. Do they vary completely or are some projects strongly recommended? To answer this question, we count the number of projects selected from the different test runs. It turns out that especially one project seems to be very valuable. Besides some other projects, Project 84 was selected seven times out of ten in the standard algorithm and eight times in the enhanced algorithm. This suggests that by a realization of Project 84, the network utility increases significantly. Meanwhile, Projects 67 and 104 were chosen 11 times and nine times respectively for both algorithms together. See fig. 5.7 for a practical implementation of the recommended Projects 84, 67 and 104.

Another interesting point is that 45 of the 110 projects were not present in any of the solutions of the 20 test runs. This might be an indicator for a poor cost-performance ratio of those projects, rather, they are simply not needed. Since the final realization of projects may not exceed the given budget of 18 million Euro, we would like to examine the extent to which this budget has been exploited. It has been ascertained that, from the overall 20 test runs, the budget has been exhausted by at least 96.82% and at most 99.98%. However, no correlation between a good utility value and a small remaining budget was found.

Finally, after a close examination of the results from different perspectives, one can mention that the algorithm developed in this thesis performs very well with respect to the traffic network optimization problem. Even though there was no improvement of the final utility values on using the enhanced genetic algorithm, we ascertain that especially the increase of the trip duration in contrast to the standard genetic algorithm

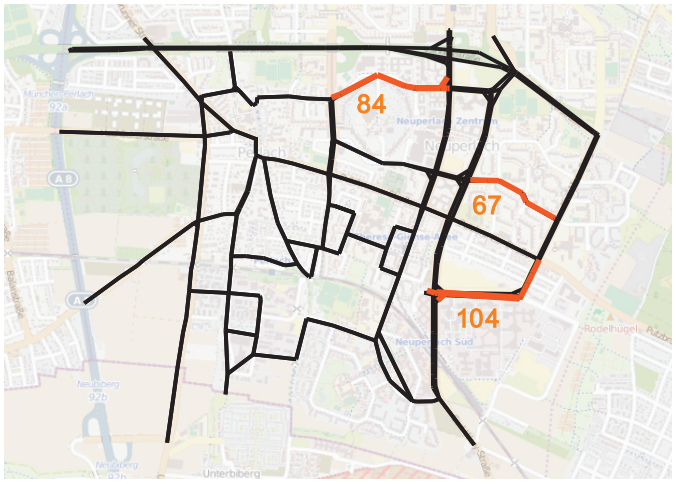


Figure 5.7: Projects recommended by both algorithms

offered convincing evidence about the potential of the algorithm. However, in order to arrive at more precise statements and deeper insights into the performance of the enhanced genetic algorithm, additional test runs might be necessary, mainly because randomness plays a big role in genetic algorithms. For example, the initial population is generated randomly and chance plays a crucial part in the roulette wheel selection. Also crossover and mutation are performed only up to a certain percentage. Subsequently, after undertaking further runs, one can eliminate weak points of the enhanced algorithm; some preliminary ideas for this will be given in chapter 6.

5.3 Running Times

This section examines the two algorithms with respect to their running times. The most time-consuming procedure in both algorithms is the evaluation of networks. In comparison, all other calculations are negligible in terms of their running times. Let us measure how costly is one such arbitrary network evaluation of the order of Perlach's graph (i. e. $|V| = 287$, $|E| = 533$). Since we have different alternatives for evaluating a network, we need to differentiate between the two evaluation methods - MATSim simulation and Xpress optimization.

Computing times	
MATSim	≈ 180 sec per evaluation
Xpress	≈ 30 sec per evaluation

Table 5.3: Computing times of evaluating one network

Note that the values are based on 60 MATSim iterations and the Xpress calculations involve traffic flows during two time periods (morning/afternoon). As table 5.3 shows, a network evaluation using the simulation method needs six times longer than an evaluation that comprises solving a nonlinear program with Xpress. To specify total running time, we first consider the MATSim calculations, since both algorithms make use of the simulation in their evaluation phase. In order to supply statements about absolute running times, let us review the benchmark data in table 5.4.

size of population	$m = 15$
size of individual	$n = 110$
mutation probability	$p_m = 0.2$
maximum number of iterations	$max_{iteration} = 100$

Table 5.4: Benchmark data for the computation of running times

Since we expect that a MATSim network computation needs 180 seconds on average, with the above assumptions of a population size of 15 and a maximum of 100 iterations, it yields a running time (rt)

$$rt_{\text{MATSim}} = max_{iteration} \cdot m \cdot 180sec = 270.000sec = 75h$$

i. e. the program needs more than three days just for the network evaluation. In most cases, this is not acceptable, a possible remedy here is to perform simulation of each individual in the current population simultaneously. With the given resources of 48 cores we are able to speed up running time significantly with the help of parallelization:

$$rt_{\text{MATSim}} = max_{iteration} \cdot 320sec = 32.000sec \approx 8.8h$$

Obviously, this is an enormous increase of running time. Let us now consider the computing time needed for the optimization of the nonlinear program in the enhanced genetic algorithm. We use the evaluation in the phase of mutation but only with a percentage of $p_m = 0.2$. Actually, the number of neighbours is not limited in the algorithm; however, in all of the test runs, there never occurred more than 49 neighbours. Therefore, we assume a maximum number of $max_{neighbours} = 50$ possible network neighbours. A network evaluation using Xpress needs approximately 30 seconds pursuant to table 5.3; thus, mutation in the enhanced algorithm has the following running time:

$$rt_{\text{Xpress}} = max_{iteration} \cdot p_m \cdot m \cdot max_{neighbours} \cdot 30sec = 450.000sec \approx 125h$$

This means in a worst case scenario, the calculation of all mutations requires more than five days. Naturally, we can increase running time as in the above to parallelize all computations of neighbours. Indeed, this might prove indispensable for future studies, but in the scope of this work, we must put up with the poor performance of the enhanced algorithm.

The actual running time strongly dependent on the utilization of the server. For the standard GA the running time has been varying from a minimum of 8 hours 5 minutes to a maximum of 10 hours 28 minutes, while for the enhance GA we have observed a running time from 41 hours 30 minutes up to 69 hours 7 minutes.

Chapter 6

Conclusion

We would like to utilize this chapter to retrospect on the task of traffic network optimization, the developed approach and its performance in contrast to the standard genetic algorithm. Moreover, we review the thesis in terms of future prospects by focusing on the weaker points of the approach. Subsequently, we discuss possible practical solutions in order to remedy this weakness in future studies.

The optimization of traffic networks aims to improve a given road network in terms of maximizing the road user's utility in this network. We abstracted this task by modelling an integer optimization program to represent it. The objective was to maximize the achieved utility of the traffic participants. Thereby, we were faced with the challenge of measuring this utility. Hence, we devoted ourselves to discussing different methods of measurements and, in a first approach, represented the utility by the road users' travel time. For this purpose, a nonlinear program has been solved in order to use the objective (total travel time) as an evaluation for a given network. The second approach for evaluating networks involved the use of professional simulation tools. These simulations not only take the travel time into account but also additional factors and timely day schedules; therefore, they can be used for a more authentic assessment. After providing methods to evaluate networks, we focused on the actual assignment—namely, advancing a given network. Since the integer program ($TNOP$), which represents the traffic network optimization problem, is already NP-hard for a linear objective function, and we still have a nonlinear cost function, a common way to solve such kind of problems is to use heuristics or metaheuristics respectively. Effective and widely used methods include, for example, genetic algorithms. These generate a set of solution candidates and iteratively increase this set according to the candidates' fitness. Since a solution (in this context called individual) represents a specific network configuration, we use the aforementioned techniques for appraising the evaluation of such an individual. A general genetic algorithm consists of different phases; the selection phase, where the decision of which individuals get a chance to reproduce is made; the crossover phase, where selected

individuals generate offspring by passing on parts of themselves; and the mutation phase, where parts of the individual are modified randomly. In order to provide a solution for the traffic network optimization problem, a standard GA and an enhanced GA have been implemented. In the latter case, the mutation will be performed in a goal-oriented manner by changing an individual into the best one in its neighbourhood. Therefore, a local optimization method is used: The neighbourhood of a network is characterized by all networks that have at most k edges difference to the given one. With the help of the above-mentioned nonlinear program, all networks are evaluated and the given network mutates into the best-rated one. The enhanced method is subsequently compared to the standard genetic algorithm by applying them in several test runs. The test network is Munich's district "Perlach". By comparing the quality of the solution as well as its evolution over iterations, it turned out that the enhanced GA performs well with respect to the trip duration. However, no significant improvement pertaining to the objective (i. e. utility) has been achieved. At this point, we would like to highlight the weakness of this algorithm and offer ideas for eliminating these vulnerabilities.

The performance of the enhanced algorithm is not convincing in all respects. These unsatisfactory aspects of the enhanced algorithm could be due to various factors. One can categorize these factors according to the algorithm's speed on the one hand and the algorithm's design on the other. However, the latter is often reasonable only if some speed-ups are implemented beforehand.

Speed-ups

A common problem—not just for this algorithm—are the very long computing times. In the case of the considered algorithms in this thesis, the evaluation is the most time-expensive computation regardless of whether we use the simulation or the slightly faster nonlinear optimization. The simulation procedures have already been made quicker by the parallelization of all networks in the population, cf. section 5.3. The same could be applied for the evaluation of all networks within the neighbourhood; this hastening is highly recommended for ensuring continued operations.

Another idea for the increase of the running time might be to limit the number of neighbours. Thereby, it is necessary to choose a strategy that selects neighbours wisely; otherwise, potentially suitable networks can be lost.

Design Changes

Once we have the ability to speed up the evaluations, a more improved version of a neighbourhood search can be implemented. This can be done by iterating over the neighbourhood several times, i. e. once the best network is found, one starts the procedure once again by generating the neighbourhood of this new network and then finding the best individual in this neighbourhood and so on. This leads to an expansion of the search space before the final mutation is performed; thus, it might increase the upcoming utility value of the simulation. Additionally, this could help to rectify the local optimum problem.

Another crucial reason why the outcome of the enhanced algorithm is particularly unsatisfactory might be the usage of different evaluation methods. For the general evaluation in the genetic algorithm, we take the simulation values as a basis, whereas the evaluation in the neighbourhood search of the mutation phase avails itself by solving a nonlinear program whose objective represents the total travel time in the given network. As an assumption for the functioning, these values need to be proportional to each other or at least to correlate with each other. An investigation of this point showed a correlation between the two evaluation methods of $r \approx 0.35$, which can be seen as a "moderate" correlation. A key issue for future studies lies in increasingly sharing common values. One thought for managing this is a more meaningful abstraction of the agents' day plans into traffic flows. A simple classification into four equally sized sectors and two time periods (cf. section 5.1) leads to major differences in both evaluations. Thus, a different sector allocation where the texture of the considered area is involved might result in a better correlation. In order to manage the above drawback of poor correlation, the simulation could be responsible for all evaluations. However, this requires a way to fasten the evaluation procedure since the simulation is the most time-consuming part in the algorithm.

In conclusion, it can be said that the developed algorithms in this thesis perform well with respect to the traffic network optimization problem and it is worthwhile to further study more advanced approaches to remedy the discussed weaknesses.

List of Figures

List of Tables

Bibliography

[AMO93] R. Ahuja, T. Magnanti, and J. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[BMW56] M. J. Beckmann, C McGuire, and C Winsten. *Studies in the economics of transportation*. Yale University Press, 1956.

[GK03] F. Glover and G. Kochenberger. *Handbook of Metaheuristics*. International series in operations research & management science. Kluwer Academic Publishers, 2003.

[Gol89] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Artificial Intelligence. Addison-Wesley, 1989.

[Hol75] J. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[Kar39] W. Karush. "Minima of Functions of Several Variables with Inequalities as Side Constraints". MA thesis. Dept. of Mathematics, Univ. of Chicago, 1939.

[Kra+12] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker. "Recent Development and Applications of SUMO - Simulation of Urban MObility". In: *International Journal On Advances in Systems and Measurements* 5.3&4 (2012), pp. 128–138.

[KT50] H. W. Kuhn and A. W. Tucker. "Nonlinear Programming". In: *Proceedings of the 2nd Berkeley Symposium on Mathematical Statistics and Probability*. Ed. by J. Neyman. 1950, pp. 481–492.

[Kut08] U. Kutschera. *Evolutionsbiologie*. UTB / UTB. UTB GmbH, 2008.

[KV08] B. Korte and J. Vygen. *Kombinatorische Optimierung*. Springer, 2008.

[MAT11] MATSim. *Tutorial: Learning MATSim in 8 Lessons*. 2011. URL: http://matsim.org/docs/tutorials/8lessons/ (visited on 09/24/2014).

[Mit96] M. Mitchell. *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1996.

[Ngu+12] M. H. Nguyen, T.-V. Ho, M. S. Nguyen, T. H. P. Phan, T. H. Phan, and V. A. Trinh. "An Agent-Based Model for Simulation of Traffic Network Status". In: *SEAL*. Ed. by L. T. Bui, Y.-S. Ong, N. X. Hoai, H. Ishibuchi, and P. N. Suganthan. Vol. 7673. Lecture Notes in Computer Science. Springer, 2012, pp. 218–227.

[Ope04] OpenStreetMap. *OpenStreetMap*. 2004. URL: //http://www.openstreetmap.org/ (visited on 09/07/2014).

[Pi08] M. Piórkowski, M. Raya, A. L. Lugo, P. Papadimitratos, M. Grossglauser, and J.-P. Hubaux. "TraNS: Realistic Joint Traffic and Network Simulator for VANETs". In: *SIGMOBILE Mob. Comput. Commun. Rev.* 12.1 (Jan. 2008), pp. 31–33.

[Pig20] A. Pigou. *The Economics of Welfare*. Macmillan, 1920.

[PR64] U. S. B. of Public Roads. *Traffic assignment manual for application with a large, high speed computer*. Traffic Assignment Manual for Application with a Large, High Speed Computer. 1964.

[Rie14] M. Rieser. *MATSim User Guide*. 2014.

[Rou05] T. Roughgarden. *Selfish Routing and the Price of Anarchy*. The MIT Press, 2005.

[RT02] T. Roughgarden and E. Tardos. "How Bad is Selfish Routing?" In: *J. ACM* 49.2 (Mar. 2002), pp. 236–259.

[Sch03] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics Bd. 1. Springer, 2003.

[SD07] S. Sivanandam and S. Deepa. *Introduction to Genetic Algorithms*. Springer, 2007.

[UU12] M. Ulbrich and S. Ulbrich. *Nichtlineare Optimierung*. Mathematik Kompakt. Springer Basel, 2012.

[WCEGB52] J. Wardrop and I. of Civil Engineers (Great Britain). *Some theoretical aspects of road traffic research*. Institution of Civil Engineers, 1952.

[Xpra] *Getting Started with Xpress*. 7.7. Fair Isaac Corporation, 2014.

[Xprb] *Xpress-Mosel. User guide*. 3.6. Fair Isaac Corporation, 2014.

Appendix

The following listed folders and files denote the data on the enclosed CD-R:

- Folder *binaries*:
  All jar-files needed for running the standard genetic algorithm ("TNOP_standard.jar") and the enhanced algorithm ("TNOP_enhanced.jar").

- Folder *eclipse*:
  Source code for the algorithms introduced in this thesis.

- Folder *input*:
  All necessary input files.

- Folder *results*:
  Summarized results of the test runs.

- File *thesis.pdf*:
  PDF file of master's thesis.