TECHNISCHE UNIVERSITÄT WIEN

DIPLOMARBEIT

Vehicle Routing Problem of the Car Distribution in Central Europe

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Statistik-Wirtschaftsmathematik

eingereicht von

Georg Schett
Matrikelnummer 01129268

ausgeführt am Institut für Stochastik und Wirtschaftsmathematik
der Fakultät für Mathematik und Geoinformation der Technischen Universität Wien

Betreuung
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Gernot Tragler

Wien, 21.09.2018

(Unterschrift Verfasser) (Unterschrift Betreuer)

Abstract

Today's freight transport market in Central Europe is characterized by a stagnating total transport volume combined with an increasing volatility of the transport requests. At the same time, in order to reduce the emission of greenhouse gases, more and more transports should be transferred from the road to more environmentally friendly transport modes. This, however, conflicts with the increasing volatility of the transport requests as a truck is more flexible in comparison to a train or a barge. In this thesis, on basis of the example of the car distribution in Central Europe we develop a model describing the underlying vehicle routing problem to be able to evaluate different fleet compositions of transport vehicles and to show up more environmentally friendly ways of transport. To solve the resulting mixed integer linear program, we present an exact optimization algorithm as well as a tabu-search heuristic. At the end of the thesis we apply both algorithms on an example case of real-world transportation data of the car distribution and compare the performance of the algorithms.

Contents

1 Introduction

The car transportation sector faces the problem that the needed transport capacities are hard to predict, even for the near future. The consequence is a short-termed decision making to account for the high degree of planning insecurity and as a result the freight transportation becomes ecologically and economically inefficient. To address this problem, the Austrian Ministry for Transport, Innovation and Technology financed the research project "**I**ntelligente Vernetzung von **P**rognose, **P**lanung und **O**ptimierung zur Gestaltung nachhaltiger Transportketten", or in short IPPO. Some of the project's results can be found in Brunnthaller and Stein (2017).
The companies involved in the project were the two application-oriented research organizations Fraunhofer Austria and RISC Software GmbH as well as the freight forwarder Hödlmayr International AG, who is specialized in vehicle logistics. I started working at Fraunhofer Austria besides my studies in December 2016 and worked there almost solely on the research project IPPO. My task was to find a way to estimate the distance traveled by different vehicle fleet compositions and soon it was clear that the only way to answer this question reasonably was by solving the underlying vehicle routing problem (VRP).
I developed an exact solution approach to solve the problem, but unfortunately it is unlikely that this approach is expedient for large problem instances. In this thesis, we present the results I derived in the course of the project IPPO and also introduce an alternative approach to be able to solve larger instances of the vehicle routing problem.

1.1 Problem Description

The problem we are going to focus on can be described as follows: On predefined transport relations a certain number of cars has to be transported from factories to turnover points within a month. Cars can in general be transported with three different types of vehicles, namely with a truck, a train or a barge, but not all transport modes are available for all transport relations. Each mode of transport has a fixed number of cars it can carry on a single tour, but it does not have to be the same for each relation as it largely depends on the type of cars that have to be transported. Furthermore, the travel costs and distance for each mode of transport and for each relation is known and each vehicle has a maximum number of kilometers it can travel within a month.
Now the task of the vehicle routing problem is to find the solution with the minimum total costs so that all transport demands are fulfilled, while the route of each transport vehicle is a closed walk, i.e., their route is a sequence of destinations with its end point being the same as its starting point. Furthermore, each vehicle must not exceed its maximum travel distance and can only travel on relations where the vehicle type is available.

1.2 Structure of the Thesis

In Chapter 2 we have a closer look at the challenges freight forwarders face in their daily business and motivate why it is essential to solve the vehicle routing problem to be able to find answers

to those questions. Chapter 3 introduces a basic notation and some basic models to formulate a VRP. Furthermore, we give a classification for different types of VRPs. In chapter 4 we present exact solution algorithms to find the optimal solution of mixed integer linear programs. Chapter 5 starts off with showing that VRPs are $\mathcal{NP}$-hard and after that we present different approaches to solve an optimization problem by using heuristic optimization methods. Finally, in Chapter 6 we introduce a model describing the problem of the car distribution in Central Europe together with an exact algorithm and a tabu-search algorithm to solve the corresponding VRP. We conclude the chapter by applying both algorithms to real-world transportation data. Chapter 7 summarizes the results and gives an outlook on potential extensions of the model.

2 Today's Challenges in Transport Logistics

2.1 Development of the Transport Demand

Since after the economic crises in 2009, the transport volume has steadily increased only with a short break in 2012. Reasons for that are the worldwide recovery of the economic crisis as well as an increase in global trade and industrial output. This trend of increasing transport volumes is expected to continue until 2019, but with a decreasing growth rate. The basis of this continuing growth forms a weak Euro and therefore an increase in exports, low raw material prices and a in comparison to previous years low oil price. Nevertheless, the growth rate is expected to decrease during the following years due to higher inflation rate, rising interest rates and a slight increase of the crude oil prices. In addition, the exchange of goods with the USA is expected to become more difficult and limitations in trade with the United Kingdom will become noticeable. On top of that, shortages in the availability of skilled workers in industry and the transport sector will become more and more a problem (SSP Consult (2017), p. 21).
As an example for one of Europe's leading economies, let us have a look at the German transport sector. The following figures (see Table 1) are based on a study commissioned by the German Federal Ministry of Transport and Digital Infrastructure (see SSP Consult (2017), p. 23-29).
The overall transport volume is expected to increase from 4275 million tons in 2016 to 4403 million tons in 2019. This corresponds to an average growth of about 1% per year. At the same time the payload-distance in ton kilometers is expected to increase at the rate of 1.5% per year, which indicates a continuation of the trend of longer transport distances.

	Million t resp. Billion tkm					**annual change in %**			
	2015	**2016**	**2017**	**2018**	**2019**	**15/16**	**16/17**	**17/18**	**18/19**
Transport volume									
Road Freight Transports	3539.2	3593.3	3662.5	3700.1	3722.9	1.5	1.9	1.0	0.6
Rail Freight Transports	367.3	363.5	362.7	361.8	361.6	-1.0	-0.2	-0.2	-0.1
of that: Comb. Transports	89.4	91.8	93.7	96.1	97.5	2.7	2.1	2.5	1.5
Inland Waterway Transports	221.4	221.3	221.0	220.4	220.5	0.0	-0.2	-0.3	0.0
Overall	4223.0	4274.9	4344.9	4380.8	4402.9	1.2	1.6	0.8	0.5
Payload-distance									
Road Freight Transports	459.0	471.8	484.2	492.8	499.8	2.8	2.6	1.8	1.4
Rail Freight Transports	116.6	116.2	116.8	117.3	117.6	-0.4	0.6	0.4	0.3
of that: Comb. Transports	45.5	46.8	47.8	49.0	49.8	2.7	2.1	2.6	1.6
Inland Waterway Transports	55.3	54.3	54.5	54.6	54.9	-1.8	0.2	0.3	0.5
Overall	650.1	662.6	675.9	685.0	692.7	1.9	2.0	1.3	1.1

Table 1: Transport volume and performance per transport mode. *Source: SSP Consult (2017)*

The individual transport modes evolve differently: While the road freight transport sector will still grow at a rather high rate of 0.8% in transport volume and 1.6% in payload-distance until 2019, rail freight transports and inland waterway transports are expected to stay almost constant. Only in terms of payload-distance a small growth is expected. This is due to higher growth rates for combined transports, i.e. the movement of goods using two or more modes of transport, without handling the goods themselves in between, which is still a growing segment for transports by train and barge.

Figure 1 shows that more than 70% of the total payload-distance is done by trucks (for the sake of completeness we added transports per pipeline as a forth mode of land transportation here) and the modal split is expected to shift even further towards road transports in the following years. One reason for that is the lack of staff and equipment in the rail and inland water transportation segment so that during demand peaks train and barge cannot fulfill all the requested transports. However, transportation capacities on the road are short as well. While it is hard to find truck drivers and workers in logistics, the demand is steadily increasing. In addition to that more and more transport companies from Eastern Europe start competing with domestic companies. The transport volume of foreign trucks is expected to grow at a rate of almost 4% per year, while domestic freight companies will be able to increase their transport volume by only less than 1%. Nevertheless, all three segments are expected to grow overall, but the growth of transports by train and barge could be even bigger, if they were more flexible, especially during transport demand peaks.

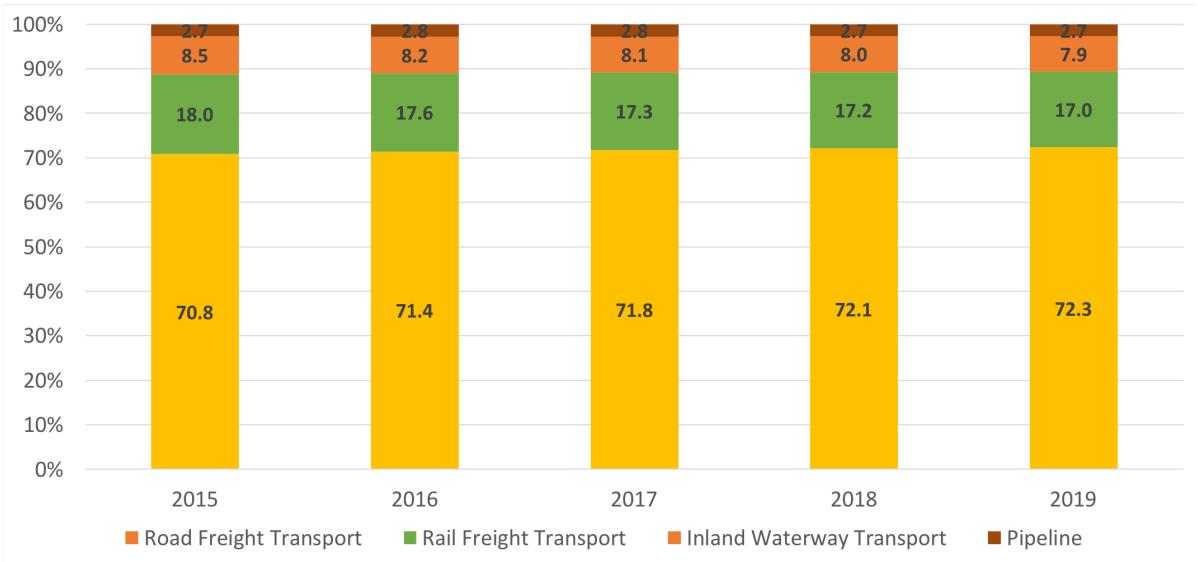

Figure 1: Modal split of land payload-distance in %. *Source: SSP Consult (2017)*

2.2 Volatility of the Transport Market

Another challenge is the increasing volatility of the transport market, i.e., an increasing fluctuation in transport volume. While the transport market was characterised by continuous growth of payload-distance (tkm) during the last decades, it has become significantly more volatile since the financial crisis in 2009. Therefore, forecasting is way more difficult nowadays and the risk of investments on the transport market has increased.

In 2013 the German "Bundesverband Materialwirtschaft, Einkauf und Logistik e.V. (BME)" conducted a survey about the market volatility in transport and logistics (see Wittenbrink and Gburek, 2013). 229 companies from the field of industry, commerce and logistics participated in the survey, from which about 75% were purchasers or shippers and 25% were from transport

and logistics companies.
According to this survey, 60% of the companies expect an increasing volatility in international transports. On one hand, this is because of the fact that more and more product parts are made in different countries. Therefore, the complexity of the underlying processes and consequently the uncertainty increase. On the other hand, the varying economic situations of different countries contribute to an increasing market volatility as well.
Exactly this increasing volatility makes it more difficult to plan transport capacities accordingly. The last few years have shown that phases of transport capacity surplus and shortage can alternate quite rapidly. For the transport companies this results in additional costs, either for covering peak periods or for having unused transport capacities.
But even with perfect foresight these fluctuations in transport volume can be reduced only to a certain degree. Therefore, alternative ways have to be found for dealing with this problem. For the vast majority of transport companies the way to go is accepting a shortage in transport capacity for short periods and not investing in an extension of the capacities immediately. This is closely related to the approach of adjusting the transport and storage capacities to the expected minimum of the transport demand and therefore optimizing the utilization of the own capacities. In the survey, 48% of the shippers and 67% of the transport companies stated to follow this approach. This also implies contracting certain transport orders out to subcontractors and renting trucks instead of buying them. This change in policy has been witnessed in the transport market for quite some time already and especially transports by trucks are carried out by subcontractors more and more often.
In addition, 72% of the shippers try to avoid entering any commitments regarding transport volumes to stay as flexible as possible. This is a well known phenomenon for transport companies and is particularly pronounced in the rail transportation segment. At the same time shippers expect transport companies to be able to deal with demand peaks, but only 49% of the transport companies are willing to accept this situation of very few long term agreements on one hand and contracting out orders to subcontractors on the other hand. One reason for that is that due to the lack of truck drivers it can be pretty difficult to find good subcontractors and transport companies would then have to establish long term relationships, which contradicts the idea of using subcontractors only for demand peaks.
Section 2.1 together with Section 2.2 show that freight forwarders have to deal with a difficult market situation today. While transport volume is stagnating or even shrinking, the volatility of transport volume is increasing. Under these circumstances it becomes more and more difficult to transport goods by train or barge cost-efficiently, but at the same time it is an important objective of world's politics to reduce the number of road transports, as the following section shows.

2.3 Green Logistics

An aspect that has become increasingly important over the last years is Green Logistics, meaning the process of recording and reducing the use of resources and emissions.

With the Paris Climate Agreement, which was signed on April 22nd in 2016, the international community agreed on tackling the problem of human made climate change. The goal of the agreement is to keep the increase of the average global temperature below 2 degrees Celsius compared to the pre-industrial era and to pursue efforts to even limit it to 1.5 degrees Celsius. The climate change is mainly induced by the emission of green house gases. The main cause of these emissions is globally as well as in Europe the burning of fossil fuels. Only by reducing this use to almost zero by the middle of this century the goal of the Paris agreement can be reached. Already today the average global temperature is about 1 degree above the pre-industrial level and the years 2014, 2015 and 2016 were the warmest years on record (Umweltbundesamt, 2017, p. 5).
Table 2 shows the change in CO2 emissions for Germany based on a prediction performed in 2010. We see that the overall CO2 output is expected to have increased by 7.8% in 2020 and by 9.5% in 2030 compared to 2005. Especially emissions caused by freight transportation are expected to increase significantly, which can be explained mainly by the increasing transportation distance. According to this prediction, at the same time the emissions from private transports will decrease. The comparatively big impact of commercial traffic on the CO2 emissions is, besides the increasing transport distance, due to the specific emissions of trucks, that are a multiple of the emissions of cars (Wittenbrink, 2014, p. 298).

	2005	2020	2030	2020 to 2005 in %	2030 to 2005 in %
Road Traffic	155.1	152.3	143.9	98.2	92.8
Private Transport	106.4	96.6	86.6	90.8	81.4
Buses	3.2	2.8	2.6	87.5	81.3
Commercial Traffic	45.5	52.9	54.7	116.3	120.2
Rail Traffic	8.5	9.2	9.6	108.2	112.9
Rail Passenger Transportation	5.7	5.6	5.5	98.2	96.5
Rail Freight Transportation	2.8	3.6	4.1	128.6	146.4
Inland Waterway Transports	2.0	2.3	2.6	115.0	130.0
Air Traffic	25.4	42.1	53.0	165.7	208.7
Overall	191.0	205.9	209.1	107.8	109.5

Table 2: Expected change in CO2 emissions for different modes of transport. *Source: (Wittenbrink, 2014, p. 298)*

Especially due to better engines and a better fuel quality, the emissions per payload distance decreased significantly for road transports since 1995. For instance, CO2 emissions decreased by 28% and SO2 emissions decreased by astonishing 99%. At the same time the overall CO2 emissions of road freight transports still increased by 11%. So we see that the decrease of emissions per ton kilometer caused by technological improvements is outweighed by the increase of traffic volume (Wittenbrink, 2014, p. 299).
According to Chapter 9.5 of Wittenbrink (2014), there are basically four approaches to reduce the emissions resulting from freight transportation:

- Giving incentives to reduce the transportation demand to preferably **avoid** traffic.
- Giving incentives to **shift** the (unavoidable) traffic to more environmentally friendly means

of transport like train or barge.

- Finding ways to **reduce** the emitted CO2 emissions when performing the transports.
- Finding ways to **compensate** the resulting emissions in other economic sectors with lower reduction costs.

We are going to talk only very briefly about reduction and compensation of CO2 emissions, as these two points are not directly related to vehicle routing.
CO2 compensation is offered for instance by non-profit organizations like myclimate.org. In order to compensate for unavoidable emissions, companies can spend money on climate protection projects to make their transports CO2-neutral. One example for a company that is already pursuing this approach is the Austrian Post AG, which has been delivering all letters and packages CO2-neutral since 2011 (see www.post.at/co2neutral).
Emission reduction on the other hand is a rather technological approach to reduce emissions. Since there is a close relation between fuel consumption and CO2 emissions, the key is to find ways to safe fuel. Installing a sensor to monitor the filling pressure of a truck's tires safes for instance about 4.05 tons of CO2 per year due to less fuel consumption. Alternatively, reducing the maximal speed will decrease the fuel consumption as well and therefore results in less CO2 emissions. The key point of all these measures is whether the costs of implementing them outweighs the reduced transportation costs due to less fuel consumption. The tire pressure sensor, for instance, costs about 1000 euros extra, but safes approximately 1900 euros in operating costs per year (Wittenbrink, 2014, p. 343).
Avoidance and shifting of traffic on the other hand are the two approaches we are more interested in: What measures give an incentive to reduce traffic? And if we cannot reduce it, what leads transportation companies to shift their transports from the road to the rail- or waterways? In the recent past the payload distance has almost always grown at a higher rate than the gross domestic product (GDP). The only exception was the year 2009 during the economic crises. Besides that, the transport intensity, i.e., the ratio between GDP and tkm, has always increased. For instance, between 1995 and 2007 the payload distance increased by 51% while the GDP grew by only 21%. During these years the transport intensity has increased on average by 1.9% per year (Wittenbrink, 2014, p. 323).
If we assume that the transport demand will not decrease significantly in the future, one important approach has to be to utilize the capacities of transport vehicles as efficiently as possible. For instance, a truck with a capacity of 25 tons has a fuel consumption of about 35 liters per 100 kilometers, while a van capable of transporting 1.5 tons needs about 12 liters of fuel per 100 kilometers. So the van's fuel consumption is almost 6 times higher than the one of the truck (the truck needs 1.4 liters per ton transported compared to the 8 liters per ton of the van). The goal has to be a high degree of consolidation of transports to be able to move them more fuel efficiently and therefore more environmentally friendly (Wittenbrink, 2014, p. 325).
To find such bundling potentials it is essential to have a look at the whole transport network, because a single transport destination with a high transportation demand does not necessarily

imply that larger trucks are more cost efficient. Especially on longer routes it can be very costly if there is no proper transport in the other direction. It also may be hard to tell what impact bundling on one route would have on all the other routes. Therefore, to be able to identify bundling potentials we cannot have a look at each vehicle individually, but instead we have to take all routes of all vehicles in the network into account, which comes down to solving the vehicle routing problem for the given transportation network.

If it is not possible to avoid traffic any further, another approach to reduce emissions is to shift transports from the road to more environmentally friendly transport modes like rail- or waterways. Despite the pretty impressive reduction of emissions of road freight transport during the last years, Table 3 and Figure 2 show that trucks still emit significantly more compared to trains or barges (Wittenbrink, 2014, p. 329).

In this context, CO2 equivalent (CO2e) is often used as a measure for green house gas emission. CO2 equivalent is a quantity describing the amount of carbon dioxide that would have the same effect on global warming over a specified timescale as the mix of actually emitted green house gases (Gohar and Shine, 2007).

Means of transport	Green house gas in CO2e	Nitric oxides NOx	Particulate matter
Truck	97.5	0.49	0.0079
Train	23.4	0.07	0.0012
Barge	33.4	0.55	0.0171
Aircraft	1539.6	3.46	0.0412

Table 3: Emissions of different transport modes in gram per ton kilometer (g/tkm) *Source: Wittenbrink, 2014, p. 329*

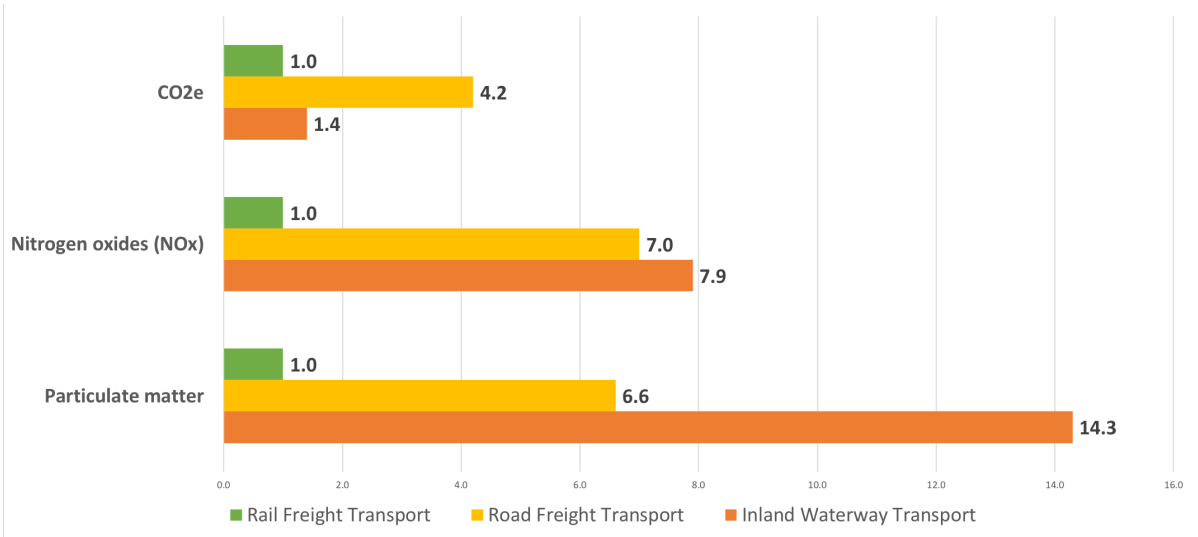

Figure 2: Ratio of specific emissions of different transport modes in gram per ton kilometer (g/tkm). Values normalized to 1 for rail freight transport. *Source: Wittenbrink, 2014, p. 330*

Like above, solving the vehicle routing problem is the only way to give an appropriate answer to the question whether a shift of the transport mode reduces the emitted gases and is cost

efficient at the same time. Trains and barges in general have way more capacity compared to a single truck, but at the same time have significantly higher fixed costs. So they will only be cost efficient, if the transportation demand is constantly on a high level. And again, a high demand on one direction of transport will not be enough either. Without a transport demand in the other direction the amount of unloaded mileage will become too big to operate cost efficiently. Another factor is that waterways are often much longer compared to the road connection between two cities. This implies that the transport costs per kilometer do not necessarily tell us the whole story and therefore we need to have a look at the whole network. On top of that, trucks are often needed for the "last mile" since the goods often have to be transported from the train station or the port to their final destination. According to Goodman (2005), up to 28 percent of all transportation costs occur on the last mile. Even though the report is from 2005 and the exact figure might be outdated by now, it still shows that this might be another important aspect that has to be considered.

3 Introduction to Vehicle Routing Problems

The family of vehicle routing problems can roughly be described the following way: Given a set of transport requests together with a fleet of transport vehicles, determine a route for each vehicle so that all transport requests are fulfilled and the costs for doing that are as little as possible. In particular, this means that one has to decide which vehicle fulfills which transport requests in what order, so that the resulting vehicle route is feasible and the overall costs are minimal.

The best studied type of VRP is the capacitated vehicle routing problem, or in short CVRP. In this problem, each vehicle has in addition only a certain transport capacity and therefore can only deliver a limited number of items. It first appeared in Dantzig and Ramser (1959) under the name "truck dispatching problem". Almost 60 years ago the authors tried to determine the optimal routing of a fleet of gasoline delivery trucks to supply a number of gas stations. Even though the CVRP has mainly only academic relevance we want to start with this very simple variant of the VRP to introduce the notation and some basic models. This will be helpful later on when we have a closer look on other variants of the VRP in Section 3.3. At first we start with the formal description of the problem. In the following, the used notation, the different model formulations and the classification of different variants of VRPs are based on the ones used in Irnich et al. (2014).

3.1 The Capacitated Vehicle Routing Problem

In the CVRP a certain number of goods have to be distributed from a single depot, denoted as point 0, to a set of n customers, which is denoted by $N = \{1, \dots, n\}$. Each customer $i \in N$ has a certain demand for goods, denoted by a scalar $q_i \geq 0$. One can think of q_i as the weight or the volume of the goods that have to be delivered to the customer. We assume that the vehicle fleet $K = \{1, \dots, |K|\}$ is homogeneous, i.e., there are $|K|$ identical vehicles available at the depot. They all are assumed to have the same transport capacity Q and cause the same costs when transporting a good to a certain destination. Each vehicle services a certain subset $S \subseteq N$ of customers by starting at the depot, moving once to each customer $i \in S$ and then returning back to the depot again. By traveling from point i to point j each vehicle incurs travel costs c_{ij}.

The whole transport network can be seen as a directed or an undirected graph. Therefore, let $V = \{0\} \cup N = \{0, 1, \dots, n\}$ be the set of vertices. Since there is no demand to the depot, let us define $q_0 := 0$. In the symmetric case, i.e., when the direction of movement has no effect on the resulting costs (i.e., $c_{ij} = c_{ji},\ \forall i, j \in V$), the graph is an undirected and complete graph with the edge set $E = \{(i, j) = (j, i) : i, j \in V, i \neq j\}$ and edge costs c_{ij} for all $(i, j) \in E$. Otherwise, if there exists at least one pair of vertices $i, j \in V$ for which the costs depend on the direction the vehicle is moving (i.e., $\exists\ i, j \in V : c_{ij} \neq c_{ji}$), the underlying graph is a complete digraph with the arc set $A = \{(i, j) : i, j \in V, i \neq j\}$ and arc costs c_{ij} for all $(i, j) \in A$. Note that both graphs have $\mathcal{O}(n^2)$ edges respectively arcs since $|E| = n(n + 1)/2$ and $|A| = n(n + 1)$. The

complete, weighted graph $G = (V, E, c_{ij}, q_i)$ or digraph $G = (V, A, c_{ij}, q_i)$, the fleet size $|K|$ and the transport capacity Q of the vehicles uniquely define an instance of the CVRP.
A vehicle route is a sequence of vertices $r = (i_0, i_1, \ldots, i_s, i_{s+1})$ with $i_0 = i_{s+1} = 0$. The set $S_r = \{i_1, \ldots, i_s\} \subseteq N$ represents the set of customers visited on tour r. The route's costs are equal to $c(r) = \sum_{p=0}^{s} c_{i_p, i_{p+1}}$. A vehicle route is feasible if no customer is visited more than once, i.e., $i_j \neq i_k$ for all $1 \leq j \leq k \leq s$, and the total capacity of the transported goods does not exceed the maximal capacity of the vehicle, i.e., $q(S_r) := \sum_{p=0}^{s} q_{i_p} \leq Q$. We then call S_r a feasible cluster. $|K|$ feasible routes, one for each vehicle $k \in K$, represent a solution of the CVRP. The solution is a feasible solution of the CVRP, if the corresponding clusters $S_1, S_2, \ldots, S_{|K|}$ form a partition of N. So finding a feasible solution of the CVRP consists of two tasks:

- partitioning the set of customers N into feasible clusters $S_1, S_2, \ldots, S_{|K|}$
- routing each vehicle $k \in K$ through $\{0\} \cup S_k$

Note that the seconds task involves solving the traveling salesman problem (TSP) over $\{0\} \cup S_k$. Both tasks are linked closely together since the vehicle routing is based on a given cluster and on the other hand the costs of a cluster depend on the vehicle routing.

3.2 Mathematical Programming Models

In this section, we want to have a look at different mathematical programming formulations of the CVRP, but first we start with defining some basic notation.
Let S be an arbitrary subset of vertices. For an undirected graph, let $E(S) = \{(i, j) \in E : i, j \in S\}$ be the set of all edges with both endpoints in S. Moreover, we define the cut set $\delta(S) = \{(i, j) \in E : i \in S, j \notin S\}$ as the set with exactly one endpoint in S. For a directed graph, let $\delta^- = \{(i, j) \in A : i \notin S, j \in S\}$ be the in-arcs of S and $\delta^+ = \{(i, j) \in A : i \in S, j \notin S\}$ be the out-arcs of S. For a singelton set $S = \{i\}$ let $\delta(i) := \delta(\{i\})$ (analogously for δ^- and δ^+). Finally, similarly to $E(S)$ we define $A(S) = \{(i, j) \in A : i, j \in S\}$ for a directed graph as the set of all arcs connecting two vertices in S.
For a customer set $S \subseteq N$ we define $r(S)$ as the minimum number of vehicle routes needed to serve S. Calculating the number $r(S)$ is equal to solving a bin-packing problem with items S of weight $q_i,\ i \in S$ and bins of the size Q. As this type of problem can be pretty hard to solve (see Section 5.1), the lower bound $\lceil q(S)/Q \rceil$ is often used instead of $r(S)$. Here $\lceil . \rceil$ denotes the ceiling function, i.e., $\lceil x \rceil = \min\{k \in \mathbb{Z} : k \geq x\}$.

3.2.1 Compact Formulations

Next we present two classical compact formulations of the CVRP. These models are mixed integer linear programming models (MILP) with a polynomial number of variables with respect to $|N|$ and $|K|$. At first we have a look at two vehicle-flow formulations with an exponential number of constraints and discuss modelling techniques to reduce the number of constraints

to a polynomial order. The advantage of compact formulations is that these models can be solved with techniques based on mathematical programming like Branch and Cut algorithms (see Chapter 4). However, this is only possible if the objective function and the constraints of a VRP can be expressed as summation over the visited vertices or visited edges or arcs. A counterexample where this is not possible is if the costs depend on the load of the vehicle.
The first important class of models has integer variables x_{ij} for each edge or arc (i, j) of the graph, indicating how often a vehicle moves from vertex i to vertex j. Since two indices are used in this formulations, they are known as two-index (vehicle-flow) formulations.
Laporte et al. (1986) introduced the following model for the directed CVRP:

$$\min_{x_{ij}:\ (i,j)\in A} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1a}$$

$$\text{subject to} \quad \sum_{j\in\delta^+(i)} x_{ij} = 1 \qquad \forall i \in N$$
$$\sum_{j\in\delta^-(j)} x_{ij} = 1 \qquad \forall j \in N \tag{1b}$$

$$\sum_{i\in\delta^+(0)} x_{0j} = |K| \tag{1c}$$

$$\sum_{(i,j)\in\delta^+(S)} x_{ij} \geq r(S) \qquad \forall S \subseteq N, S \neq \emptyset \tag{1d}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A \tag{1e}$$

In a very similar way, Laporte et al. (1985) published a model for the undirected CVRP a year earlier:

$$\min_{x_{ij}:\ (i,j)\in E} \quad \sum_{(i,j)\in E} c_{i,j}x_{ij} \tag{2a}$$

$$\text{subject to} \quad \sum_{j\in\delta(i)} x_{ij} = 2 \qquad \forall i \in N \tag{2b}$$

$$\sum_{i\in\delta(0)} x_{0j} = 2|K| \tag{2c}$$

$$\sum_{(i,j)\in\delta(S)} x_{ij} \geq 2r(S) \qquad \forall S \subseteq N, S \neq \emptyset \tag{2d}$$

$$x_{ij} \in \{0, 1, 2\} \qquad \forall (i, j) \in \delta(0)$$
$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in E\backslash\delta(0) \tag{2e}$$

The objectives (1a) and (2a) in the two models are to minimize the overall routing costs. Constraints (1b) and (2b) ensure that in a route each customer vertex is connected to exactly two other vertices, one where the vehicle is coming from and one where it is moving on to. In a similar way, constraints (1c) and (2c) state that the depot has exactly $|K|$ successor vertices, and in total $2|K|$ vertices are connected to it. This also implies that exactly $|K|$ routes are

constructed. If there are more vehicles available than actually needed, i.e., $|K| > r(N)$, the model solution does still contain $|K|$ routes. In that case, one could replace the equations (1c) and (2c) with "$\leq$"-inequalities. As increasing the number of maximal routes increases the space of feasible solutions, it is possible that the optimal routing costs decrease. Therefore, fleet size minimizing and routing cost minimizing are conflicting objectives. One way to cope with this opposing objectives is to add fixed costs of the vehicles to the model by altering the cost coefficients c_{0j}.

Constraints (1d) and (2d) have two purposes: On the one hand, they ensure that the capacity constraint is not violated and on the other hand, they serve as subtour elimination constraints (SECs). This can be seen as follows: First, if there is an infeasible route over a cluster $S \subseteq N$ with demand $q(S) > Q$, the minimum number of needed vehicles to serve this route would be greater than 1, i.e., $r(S) > 1$. So at least two routes from $S \backslash V$ are connected to S, which is a contradiction to S being the cluster of a route. Second, for any subtour (by subtour we mean a tour, that does not contain the depot) over a non-empty subset $S \subseteq N$ the sum $\sum_{(i,j)\in\delta^+(S)} x_{ij}$ is equal to 0, which is a contradiction to $r(S) \geq 1$. For the directed CVRP, constraints (1d) are equivalent to

$$\sum_{(i,j)\in\delta^-(S)} x_{ij} \leq r(S) \qquad \text{and} \qquad \sum_{(i,j)\in A(S)} x_{ij} \leq |S| - r(S).$$

For the undirected CVRP, they are equivalent to

$$\sum_{(i,j)\in E(S)} x_{ij} \leq |S| - r(S).$$

This follows straight from the fact that since S does not include the depot, a tour is a subtour if and only if the vehicle travels along $|S|$ edges or arcs within $A(S)$ respectively $E(S)$. This can easily be proven by summing up the constraints (1b) for $i \in S$ and $j \in S$. However, regardless of the formulation, the number of constraints increases exponentially with the number of customers N.

Basically there are two techniques to overcome the problem of the exponentially many constraints. One way is to simply leave out all or some of the SECs and solve this relaxation of the problem. Afterwards, the violated SECs are identified (using a so-called separation method, see Crowder and Padberg, 1980, p. 507f) and added to the problem again. This is done iteratively until no violated SEC is found. The second option is to replace the SECs with another set of constraints by adding additional variables. This approach was introduced by Miller, Tucker and Zemlin for the traveling salesman problem (see Miller et al., 1960) and is therefore known as the MTZ-formulation. Considering the directed CVRP, variables $u_1, \ldots, u_n$ are added denoting the accumulated demand that has already been fulfilled by the vehicle when it arrives at customer $i \in N$. Constraints (1d) are then replaced by MTZ specific SECs

$$u_i - u_j + Qx_{ij} \leq Q - q_j \qquad \forall (i,j) \in A(N)$$

and the capacity constraints

$$q_i \leq u_i \leq Q \qquad \forall i \in N.$$

Note that $x_{ij} = 1$ implies $u_j \geq u_i + q_j > u_i$. So for each tour $(i, j, \ldots, i)$ that does not contain the depot we get the contradiction $u_i > u_j > \ldots > u_i$. The big advantage of the MTZ-formulation is that it leads to only $\mathcal{O}(n^2)$ constraints. The drawback on the other hand is that finding strong lower bounds of the objective value turns out to be more difficult when using the MTZ-formulation (see Padberg and Sung, 1991). As we will see in Section 4, strong bounds are important for solving the MILPs.

Using the two-index formulation we model the underlying fleet only implicitly. The only thing we know is whether a specific edge (i, j) is used, but the route of which vehicle this edge is actually part of is not clear. Therefore it is not possible to model any vehicle-specific characteristics like different capacities or costs with this formulation, which is of course a big limitation. However, with this formulation there exist no symmetric solutions resulting from simply renumbering the vehicles, such that there world be a one-to-one relation between the feasible VRP solution and the feasible vector x.

To account for vehicle-specific characteristics, we now present a three-index (vehicle flow) formulation. Let $G(V, A)$ be a directed graph in which we split the depot 0 into two points o and d representing the starting point respectively the end point of a route. So the vertex and arc set is given by

$$V := N \cup \{o, d\} \qquad \text{and} \qquad A := (V \backslash \{d\}) \times (V \backslash \{o\}).$$

As the name suggests, in the three-index formulation the decision variables have three indices. Let x_{ijk} be a binary variable that indicates whether vehicle $k \in K$ moves along the arc $(i, j) \in A$. Furthermore, let y_{ik} be another binary variable that is equal to 1 if and only if vehicle k visits vertex $i \in V$. In addition, we add the variables u_i from the MTZ-formulation denoting (a upper bound on) the load in vehicle k before reaching customer i. Finally, let us define $q_o = q_d = 0$. The three-index formulation was originally proposed by Golden, Magnanti, and Nguyen (see Golden et al., 1977). The following model is not exactly the one they proposed in there paper as they also considered a maximum travel time for each vehicle and used a slight variation of the MTZ-formulation. For the sake of clarity, we omit these extra constraints and stay with our previously used MTZ subtour elimination constraints.

$$\min_{x_{ijk}:\ (i,j)\in A, k\in K} \quad \sum_{(i,j)\in A} \sum_{k\in K} c_{ij} x_{ijk} \tag{3a}$$

$$\text{subject to} \quad \sum_{k\in K} y_{ik} = 1 \qquad \forall i \in N \tag{3b}$$

$$\sum_{j\in\delta^+(i)} x_{ijk} - \sum_{j\in\delta^-(i)} x_{ijk} = \begin{cases} 1, & i = o \\ 0, & i \in N \end{cases} \qquad \forall i \in V\backslash\{d\}, k \in K \tag{3c}$$

$$\sum_{j \in \delta^+(i)} x_{ijk} = y_{ik} \qquad \forall i \in V \backslash \{d\}, k \in K \qquad (3d)$$

$$\sum_{i \in \delta^-(d)} x_{idk} = y_{dk} \qquad \forall k \in K \qquad (3e)$$

$$u_{ik} - u_{jk} + Q x_{ijk} \leq Q - q_j \qquad \forall (i,j) \in A, k \in K \qquad (3f)$$

$$q_i \leq u_{ik} \leq Q \qquad \forall i \in V, k \in K \qquad (3g)$$

$$x_{ijk} \in \{0,1\} \qquad \forall (i,j) \in A, k \in K \qquad (3h)$$

$$y_{ik} \in \{0,1\} \qquad \forall i \in V, k \in K \qquad (3i)$$

The objective (3a) is again to minimize the total routing costs, and the constraints (3b) state that each customer is visited exactly once. Constraints (3c) are the so-called path-flow constraints. Since the constraints imply $\sum_{j \in \delta^+(d)} x_{ijk} - \sum_{j \in \delta^-(d)} x_{ijk} = -1$, the route of a fixed vehicle $k \in K$, i.e., the arc set $\{(i,j) \in A : x_{ijk} = 1\}$, is an induced o-d-path on the graph G. Constraints (3d) and (3e) link the routing variables x_{ijk} with the binary variables y_{ik} so that y_{ik} is indeed 1 if and only if customer $i \in N$ is visited by vehicle $k \in K$. Finally, constraints (3f) and (3g) are vehicle-specific versions of the MTZ capacity constraints.
With this three-index formulation we are able to account for lots of vehicle specific characteristics with only a few changes in the model. We will have a look at different types of vehicle routing problems in Section 3.3. However, the big downside of this formulation is that especially for large fleets K the problem becomes hard to solve due to the high number of symmetric solutions. Since renumbering the vehicles already gives us a new solution, simple permutation of the indices of the vehicles results in $|K|!$ equivalent solutions.

3.2.2 Extensive Formulations

An extensive formulation of the CVRP was first introduced by Balinski and Quandt (1964). The basic idea is that a solution of the optimization problem is not defined by vehicle-flow variables x_{ijk} indicating whether a vehicle travels along a certain edge or arc, but rather by a set of entire routes. So the problem turns into an extended set partitioning or covering problem. Let Ω be the set of all feasible routes of the CVRP. Each route of Ω has the form $r = (i_0, i_1, \ldots, i_s, i_{s+1})$ with i_0 being the starting point o and i_{s+1} being the end point d. Then the total costs of a route are equal to $c_r = \sum_{j=0}^{s} c_{i_j i_{j+1}}$. In addition, let $a_{ir} \in \{0,1\}$ be a coefficient that is equal to 1 if and only if route r visits customer $i \in N$, i.e., $i \in \{i_1, \ldots, i_s\}$. The decision variable of the model is a binary variable, denoted by λ_r, that indicates whether route $r \in \Omega$ is part of the solution. The extensive formulation is then given by

$$\min_{\lambda_r:\ r \in \Omega} \quad \sum_{r \in \Omega} c_r \lambda_r \qquad (4a)$$

$$\text{subject to} \quad \sum_{r \in \Omega} a_{ir} \lambda_r = 1 \qquad \forall i \in N \qquad (4b)$$

$$\sum_{r \in \Omega} \lambda_r = |K| \qquad (4c)$$

$$\lambda_r \in \{0,1\} \qquad \forall r \in \Omega. \tag{4d}$$

Like before, the objective (4a) is to minimize the total routing costs. Constraints (4b) state that each customer is visited exactly once by one route and constraint (4c) ensures that all vehicles are used.
In the formulation above the model is a set partitioning problem. If the routing costs fulfill the triangular inequality (i.e. $c_{ij} + c_{jh} \geq c_{ih},\ \forall (i,j),(j,h) \in A$) we can replace the set partitioning constraints (4b) by set covering constraints (i.e., replace the"$= 1$" by a "≥ 1"). This is possible since removing one or several customers from a feasible route produces another feasible solution with less or equal routing costs. Moreover, if it is feasible to use fewer than $|K|$ vehicles, one can change constraint (4c) to an inequality, i.e., replace the "$=$" by a "$\leq$".
Compared to the compact formulation, the extensive formulation is way more flexible when it comes to defining whether a route is feasible or not. All this information is implicitly hidden in the definition of the set Ω, and so non-linear cost functions and complex intra-route constraints can be implemented in this model. In addition, this formulation does not suffer from symmetry problems. The big downside is obviously the potentially huge number of feasible routes $|\Omega|$. This makes it often impossible to explicitly set up the model and, even more importantly, find the optimal solution of (4a)-(4d) directly. Therefore, algorithms that work implicitly on huge sets like Ω are needed. Examples for such algorithms are column generation techniques at which we will have a closer in Section 4.3.

3.3 Classification of Vehicle Routing Problems

In this section we want to discuss various types of vehicle routing problems and their maybe small, but meaningful differences. The following classification is based on the one presented in Irnich et al. (2014). As we just want to give a broad overview of the most important VRP variants, we refer to the paper of Irnich et al. for a more detailed classification.
We are going to classify VRPs with respect to the following aspects:

- the road network structure (Section 3.3.1)
- the type of transportation request (Section 3.3.2)
- the constraints affecting single routes individually (Section 3.3.3)
- the fleet composition (Section 3.3.4)

3.3.1 Network Structure

There are basically two different ways to model a vehicle routing problem regarding the network structure. The CVRP for instance has transportation tasks that are related to points in space, as the task is to deliver a certain amount of goods to different customers. Usually we model these points as nodes of a graph and therefore VRPs like this are called node routing problems. The other type of problems are so-called arc routing problems (ARP, see Corberán and Laporte,

2015). These models are used if the task that needs to be fulfilled is performed on the arcs instead of on the vertices. Typical examples for that are the routing of vehicles for winter services like salt gritting or snow removal. Often services like garbage collection and mail delivery are seen as services on arcs as well since all the customers are located along a street segment. So in this formulation the street segment, where the households belong to, is serviced instead of the individual customers.
These two types of network structures can be split again into two different cases. The first one is the simpler one of symmetric travel costs. As we will see in Section 3.3.3, other characteristics like for instance travel time can be symmetric or asymmetric as well. If the whole problem is symmetric, it is possible to model the problem as an undirected graph. As soon as the direction of movement on a single edge is restricted or if any cost or any other parameter is asymmetric, the underlying graph of the network is asymmetric as well. But this does not mean that only because the transport network is symmetric, the model formulation has to be symmetric as well. As we have seen in Section 3.2, the MTZ-formulation requires the underlying graph to be a directed graph, since the order in which the customers are served is essential for the MTZ-formulation to work out.
The granularity of the underlying data or network is another thing that differentiates the VRP from the ARP. Usually in an ARP the arcs and edges represent single street segments. This is in contrast to the VRP where the arcs and edges are potentially a large number of street segments, since they represent the whole path between two points. Therefore the travel time and the corresponding costs first need to be computed by a Geographical Information System (GIS). Typically a weighted combination of distance and costs is used to calculate the shortest path between two points. However, the time of day may also have a big impact on the actual shortest path (with respect to travel costs and/or time).
The VRP is called stochastic, if some data is not known exactly but rather given by a random variable of a given probability distribution. If some data is not known at the beginning, but becomes available during the operation, we call the VRP to be dynamic. The CVRP as a very simple version of the VRP is obviously neither stochastic nor dynamic.

3.3.2 Transportation Request

The CVRP is the most basic variant of a VRP, where the task is to deliver a certain amount of goods from a central depot to a set of customers. In this section we want to classify VRPs regarding other types of transportation request.

Delivery and Collection: The opposite of delivering goods to customers would be to collect goods and transport them to a central depot. This type of transport can often be found either at the very beginning or the very end of a supply chain. An example for the beginning of the supply chain is the collection of raw milk that is then transported to a dairy factory (see Sankaran and Ubgade, 1994). The collection of garbage is an example for a pickup VRP that occurs at the very end of the supply chain (see Golden et al., 2002). Obviously, pure pickup and pure delivery

problems are equivalent as simply reversing the routes turns collection into delivery and vice versa.
So only if pickup and delivery tasks occur on the same route, a new type of problem is given. One unchanged assumption is that all goods have to be taken from or returned to a single depot. Therefore, in the case of distribution the VRP is often known as one-to-many problem, and on the other hand, in the case of collection as a many-to-one VRP. One simple variant of the VRP that contains collection and delivery is the VRP with Backhauls (VRPB, see Goetschalckx and Jacobs-Blecha, 1989). In this type of problem, customers are either so-called linehaul customers, to whom goods have to be delivered from a central depot, or backhaul customers, from whom items have to be transported back to the depot. The restriction thereby is that on each route all deliveries have to be made before the vehicles are allowed to collect anything from the backhaul customers. This can be motivated by the idea that for instance bulky material has to be transported and it is not possible to rearrange the loaded items inside the vehicle. Therefore all goods have to be delivered first so that the vehicle is empty when it reaches the first collection point. This problem can still be solved using the two-index formulation of Section 3.2 by simply removing the arcs leading from a backhaul customer to a linehaul customer (or alternatively setting the routing costs to a sufficiently large number M).
In the VRPB, each point is either a customer that requires delivery or collection, but not both. In the VRP with Simultaneous Pickup and Delivery (VRPSPD, see Min, 1989) each customer has two transportation request, a pickup and a delivery request, and both have to be performed by the same vehicle in one visit. A real-world example for that is the transport of newly arriving tourists from the airport to their hotels and bring back leaving tourists from their hotel to the airport. Interestingly, the capacity constraint that ensures that at no point in time more tourists are on the bus than the bus offers seats can be expressed easily: A set $S \subseteq N$ of customers can feasibly be serviced by a single vehicle, if neither the amount of the overall collected goods nor the amount of the overall delivered goods exceeds the vehicle's capacity. The customers just have to be served in the order that customers with a higher amount to be delivered to than collected from have to be served before the others.

Point-to-Point Transportation: Transportation problems with transport requests being point-to-point transports are called Pickup and Delivery Problems (PDP, not to be confused with the VRPSPD). To be more precise: A transport request consists of a pair of points, one from which a certain amount of goods has to be picked up and one where theses goods have to be delivered to. So in contrast to the VRP with Simultaneous Delivery and Pickup, customers are always either a delivery point or a collection point, but never both. In general, there exists no central depot in this type of problem. Instead, one could say that each delivery point has its own depot. We therefore refer to this type of problem as a many-to-many VRP (see Battarra et al., 2014).
Indeed, the vehicle routing problem we described in Section 1.1 is a PDP as the task is to transport different types of cars between factories and turnover points.

Non-split and Split Services: The Split Deliveries Vehicle Routing Problem (SDVRP) is a relaxed version of the standard VRP, in which the amount of goods that has to be transported to a demand point can be split between any number of vehicles. The problem was first formally introduced in the late 1980s (see Dror and Trudeau, 1989, and Dror and Trudeau, 1990). There are two reasons why it makes sense or maybe it is even necessary to split deliveries: First, if the demand exceeds the maximum capacity of a vehicle, it is obviously unavoidable to split the goods between more than one vehicle in order to be able to fulfill the transport request. The second reason arises when the maximal capacity of the vehicles is larger than the demand: Since the SDVRP is a relaxation of the standard VRP, the minimum costs to fulfill all the transport requests are equal or less compared to the problem without delivery splitting and therefore it might be possible to service all customers at less costs.
Dror and Trudeau gave the following example to illustrate the cost saving effects that are possible due to splitting deliveries: Assume three demand points with demands $q_1 = 3$, $q_2 = 4$, $q_3 = 3$. The costs between the depot and any demand point is 10, i.e., $c_{0,i} = 10$, for $i = 1, 2, 3$. The costs between two demand points are given by $c_{1,2} = c_{2,3} = 5$ and $c_{1,3} = 10$. The capacity of all vehicles is equal to 5. If no splits are allowed, the optimal solution has a total cost of 60 and requires 3 vehicles. For the SDVRP solution only two vehicles are required and the total costs reduce to 50 (see Figure 3).

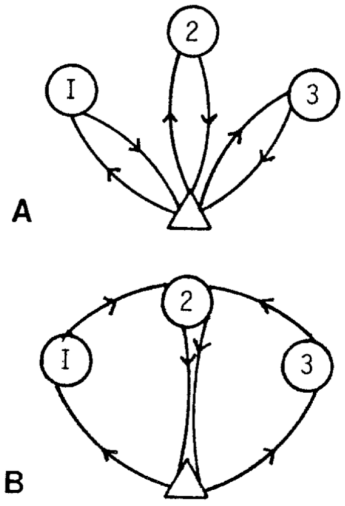

Figure 3: (A) The solution of the VRP and (B) the solution of the SDVRP. *Source: Dror and Trudeau, 1990*

Archetti and Speranza (2008) differentiate between the following variations of the SDVRP (for comparison we state the VRP as well):

- VRP: The problem of finding the optimal route if all demands are less than or equal to the vehicle capacity Q and each customer is visited exactly once.
- SDVRP: The problem of finding the optimal route if all demands are less than or equal to the vehicle capacity Q and each customer may be visited any number of times.

- VRP$^+$ (extended VRP): The problem of finding the optimal route if at least one demand is greater than the vehicle capacity Q and each customer is visited at most by the minimum possible number of times needed to fulfill the transport request, i.e., $t_i = \lceil q_i/Q \rceil$, where q_i is the demand of customer i and t_i is the number of visits.
- SDVRP$^+$ (extended SDVRP): The problem of finding the optimal route if at least one demand is greater than the vehicle capacity Q and each customer may be visited any number of times.

Usually when talking about the SDVRP an unlimited fleet is assumed. In that case there always exists a feasible and thereby an optimal solution as well. The variant with a limited fleet size is denoted by SDVRP-LF. Obviously, a feasible solution to the SDVRP with a fleet of $|K|$ vehicles exists if and only if $\sum_{i \in N} q_i \leq |K|Q$ holds.
Let $x_{ijk} \in \{0, 1\}$ again be a binary variable indicating whether vehicle $k \in K$ travels along the arc $(i, j) \in A$. In addition, let y_{ik} be the quantity of demand delivered by vehicle k to customer i. Then the SDVRP can be formulated as follows (Archetti et al., 2006):

$$\min_{x_{ijk}:\ (i,j)\in A, k\in K} \quad \sum_{(i,j)\in A} \sum_{k\in K} c_{ij} x_{ijk} \tag{5a}$$

$$\text{subject to} \quad \sum_{i\in N} \sum_{k\in K} x_{ijk} \geq 1 \qquad \forall j \in N \tag{5b}$$

$$\sum_{i=0}^{n} x_{ihk} - \sum_{j=0}^{n} x_{hjk} = 0 \qquad \forall h \in V, k \in K \tag{5c}$$

$$\sum_{i\in S} \sum_{j\in S} x_{ijk} \leq |S| - 1 \qquad \forall k \in K, S \subseteq N \tag{5d}$$

$$y_{ik} \leq d_i \sum_{j=0}^{n} x_{ijk} \qquad \forall i \in N, k \in K \tag{5e}$$

$$\sum_{k\in K} y_{ik} = d_i \qquad \forall i \in N \tag{5f}$$

$$\sum_{i\in N} y_{ik} \leq Q \qquad \forall k \in K \tag{5g}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall (i, j) \in A, k \in K \tag{5h}$$

$$y_{ik} \geq 0 \qquad \forall i \in V, k \in K. \tag{5i}$$

The objective (5a) is to minimize the total routing costs. Constraints (5b)-(5d) are classical vehicle routing constraints: Constraints (5b) state that each customer has to be visited at least once (contrary to exactly once like in the standard CVRP), equations (5c) impose that the number of vehicles traveling to a node has to be equal to the number of vehicles leaving the node and (5d) are subtour elimination constraints. These are similar to the ones we introduced in the two-index formulation, but since $r(S)$ being the minimum number of vehicle routes needed to serve a set of customers S makes no sense in the SDVRP context, they are replaced by this version of subtour elimination constraints. The next set of constraints (5e)-(5g) are about the

distribution of goods among the vehicles: Inequalities (5e) state that a customer can only be served by a vehicle, if it visits the corresponding node, equations (5f) ensure that the demand of all customers is served and constraints (5g) make sure that the vehicle capacity is not exceeded.

Combined Shipment and Multi-Modal Service: Combined transports is similar to the vehicle routing problem with split deliveries: In this variation of the VRP the transport of goods is split again, but this time the shipment itself is not split into smaller parts. Instead, the transport path is split at intermediate transfer points or consolidation centers into segments which are then serviced by different vehicles. We have already seen in Section 2.1 that combined shipment is still a growing segment, and when it comes to shifting transports to the train or the barge, multi-modal service is often unavoidable as there often is no train station or harbor near the customer. Combined shipment is also used to consolidate less-than-truckload loadings to full-truckload shipments.

Dynamic and Stochastic Routing: Like we mentioned in the introduction, the main task of the research project IPPO was to improve the planning security of freight forwarders. But forecasting production numbers of cars is obviously pretty difficult. One result of the research project was that even the car manufacturers themselves make prediction errors of up to $\pm$ 20%. Besides the customer demand, travel time and costs can be uncertain as well. Especially in urban areas travel time is affected by lots of different factors like accidents, traffic and weather conditions. Ignoring these travel time variations can lead to significantly longer travel times at the end what may cause problems when a customer has to be visited within a specific time window.
In general we say a problem is

- dynamic, if some or all relevant information is not know at the beginning and only becomes available over time;
- stochastic, if the system conditions are uncertain, but can be described by a given probability distribution.

So in the case of stochastic vehicle routing, some components of the problem like travel time or customer demand are only described by random variables. Vehicles traveling a pre-planned route might miss time windows or reach their transport capacity before even visiting all customers, which may cause extra costs or even make it impossible to fulfill all customer demands. Therefore, in that case the main focus is on analyzing the impact of the uncertainties on the routing costs or the reliability of the transport services.

3.3.3 Intra-Route Constraints

One of the most important aspects when it comes to differentiating between different types of vehicle routing problems is the determination of the feasibility of a route. In this section, we give an overview on the most important variants of constraints like maximal loading capacity,

time restrictions or a limited route length. In contrast to so-called inter-route constraints, intra-route constraints have in common that every route can be checked individually on its feasibility, independently of all the other routes.

Loading: Capacity constraints, like they occur in the CVRP, are one of the simplest types of constraints. They limit the consumption of a certain resource at every customer and therefore can simply be written as an overall bound on these resources for each transport vehicle:

$$\sum_{i \in N} q_i y_{ik} \leq Q, \qquad \forall k \in K. \tag{6}$$

Note that we use the three-index model again since with this formulation many modifications of the VRP can be described in a convenient way. For some problems there may be several resource constraints like maximum weight, maximum volume (for liquid goods) or maximum space (for instance a limited pallet storage area) such that each constraint individually restricts the feasibility of a route at the same time. In that case, for each restriction a constraint analogous to (6) with corresponding coefficients q_i and Q has to be added to the model.
The problem becomes way more complex when the capacity constraints can not be described by a scalar like the maximum weight, but instead shipments and cargo departments have to fulfill 2-dimensional or 3-dimensional restrictions. So the VRP has to be solved in combination with a multi-dimensional packing problem, which is already difficult to solve on its own.
Iori et al. (2007) consider a symmetric capcitated vehicle routing problem in which each customer has a demand consisting of a set of rectangular two-dimensional weighted items. Often goods are transported on top of rectangular bases like pallets, and due to their fragility or shape they may not be stacked on each other. In that case, the three-dimensional loading problem can be reduced to a two-dimensional loading problem of the rectangular item base on the vehicle loading area. In addition, they assume that each item has a fixed loading and unloading orientation, since common loading equipment like forklifts may not be able to pick up the items from any side. Furthermore, when a customer's demand consists of several items, they have to be delivered at once (i.e., split deliveries are not allowed) and each item to be unloaded must not be blocked by another item that has not been unloaded yet. This last restriction is due to the problem that rearranging items with forklifts can be very difficult, time consuming or might even be impossible. This variant of vehicle routing problem is known as CVRP with 2-dimensional Loading constraints (2L-CVRP).
Gendreau et al. (2006) introduce a tabu-search algorithm to solve a variant of the problem with three-dimensional loading constraints, denoted as 3L-CVRP. This time horizontal rotation of the items is allowed, while upside-down rotations are forbidden. In addition, they add some operational constraints that are often encountered in the real-world: If an item is fragile, no other item may be placed on top of it, and if boxes are stacked, the supporting surface needs to be large enough so that the stability of the loading is guaranteed.
Another interesting variation of the CVRP is the so-called VRP with Compartments (VRPC), introduced by Derigs et al. (2010). In industries with inhomogeneous goods, vehicles with differ-

ent compartments are often used in order to save transportation costs by allowing the transport of inhomogeneous goods together on the same vehicle. So besides an overall capacity constraint, each compartment has its own restrictions as well. In some settings, the separator of the compartments may be configurable so that one compartment's size can be up to the whole vehicle's capacity. In other settings, like for instance tanks in road tankers, the compartments may be fixed units. Regarding the item compatibility, two aspects have to be taken into account: First, only compatible items can be placed in the same compartment (item-item compatibility), e.g., due to certain hygiene regulations for the transport of medical products. Second, the compartment has to fulfill the requirements of the transported item (item-compartment compatibility), e.g., refrigerated cargo has to be transported in cooled compartments.

Route Length: Similar to the capacity constraints, the resource consumption on arcs or edges may be restricted. This again is a rather simple constraint since checking the feasibility of the solution just requires to some up the overall resource consumption for each vehicle and check whether it exceeds the vehicle's limit. The resulting problem is then denoted as distance-constrained CVRP (DCVRP, see Laporte et al., 1984). Let $d_{ij} > 0$ be the distance between two nodes i and j for all $(i, j) \in A$. In addition, let D be the maximal length of a route. Then the distance constraints are given as

$$\sum_{(i,j)\in A} d_{ij} x_{ijk} \leq D \qquad \forall k \in K.$$

Of course, not only spatial restrictions can be modeled with this formulation. The parameters d_{ij} can also denote the travel time to formulate constraints on the route duration. Similarly, the total routing costs can be limited. Another option would be to use indicator variables $d_{ij} \in \{0, 1\}$ to restrict the number of connections with a certain property.

Multiple Use of Vehicles: In many VRPs the standard assumption is that each vehicle performs at most or exactly one route over a planning horizon T. The Multi-Trip VRP (MTVRP, see Mingozzi et al., 2013) relaxes this restriction so that vehicles may perform several routes. In particular, a vehicle may perform a given set of routes with durations $T_1, T_2, \ldots, T_p$ if $T_1 + T_2 + \ldots + T_p \leq T$ holds. This is a relevant scenario for distribution companies using leased vehicles to service the customers. If the planning period is significantly larger compared to the usual transport duration, a single vehicle can perform several routes, and therefore one of the main concerns of a distribution company is to minimize the number of vehicles used. Conversely, if the maximal transport capacity Q is relatively small or any other restriction leads to a small number of customers serviced by a route, vehicles of a limited fleet size $|K|$ may have to be used multiple times to be even able to fulfill all transport requests. A rather simple approach is to solve the problem with unlimited fleet size and combine the resulting routes within the planning horizon, but this obviously yields suboptimal solutions to the MTVRP.
A possible scenario where this kind of problem occurs are carriers using electric vehicles for last-mile deliveries in order to reduce the emission of greenhouse gases. The limited battery

capacity of the vehicles forces them to regularly visit recharge stations during their tours. Then these stations can be seen as kind of depots to which the vehicles have to return to and the battery capacity is a restriction on their maximal tour length. This particular case of VRPs with needed recharging during trips has been studied by Schneider et al. (2014).

Time Windows and Scheduling Aspects: In many scenarios, constraints regarding the scheduling of transports are among the most important restrictions in a VRP, i.e., travel, service, and waiting times need to be considered as well as time windows in which a customer has to be serviced. This type of problem is generally know as VRP with time windows (VRPTW, see Cordeau et al., 2002). We again use the three-index formulation (3a)-(3i) to model this variant of a VRP. Let t_{ij} be the travel time from vertex i to vertex j for all $(i,j) \in A$. Furthermore, for each vertex $i \in V$ a service times s_i and a time-window $[a_i, b_i]$ are given. For the nodes o and d representing the depot, we define $s_o = s_d = 0$ and $[a_o, b_o] = [a_d, b_d] = [E, L]$, where E and L denote the earliest possible departure time from the depot and the latest possible arrival time at the depot, respectively. A feasible solution can obviously only exist if $a_o = E \leq \min_{i \in N} \ b_i - t_{oi}$ and $b_d = L \geq \max_{i \in N} \ a_i + s_i + t_{id}$. Note that an arc (i,j) can be removed from the model, if $a_i + s_i + t_{ij} > b_j$, as due to temporal restrictions it is impossible to service customer j in time after visiting customer i.
In addition to the flow variables x_{ijk}, indicating whether vehicle $k \in K$ travels along the arc $(i,j) \in A$, and y_{ik}, denoting whether customer $i \in N$ is visited by vehicle k, we introduce the time variables w_{ik}, specifying the start of service at customer i when serviced by vehicle k. Then (3a)-(3i) combined with the following additional constraints, are a formulation of the VRPTW:

$$x_{ijk}(w_{ik} + s_i + t_{ij} - w_{jk}) \leq 0 \qquad \forall (i,j) \in A, k \in K \tag{7a}$$

$$a_i y_{ik} \leq w_{ik} \leq b_i y_{ik} \qquad \forall i \in V, k \in K \tag{7b}$$

$$E \leq w_{ik} \leq L \qquad \forall i \in \{o,d\}, k \in K \tag{7c}$$

$$w_{ik} \in \{0,1\} \qquad \forall (i,j) \in A, k \in K. \tag{7d}$$

Note that (7b) forces w_{ik} to be equal to 0, if vehicle k does not visit customer i.
Since x_{ijk} are binary variables, the constraints (7a) can be linearized by means of MTZ-like constraints of the form

$$w_{ik} + s_i + t_{ij} - w_{jk} \leq (1 - x_{ijk}) M_{ij} \qquad \forall (i,j) \in A, k \in K, \tag{8}$$

where M_{ij} are a sufficiently large constants. M_{ij} can be replaced by $\max\{b_i + s_i + t_{ij} + a_j, 0\}$ and constraints (8) are only relevant for arcs $(i,j) \in A$, such that $M_{ij} > 0$. Otherwise, due to constraints (7b) the constraint is fulfilled for all values of x_{ijk}.
There are various ways to alter and generalize these time-window constraints: If there is not only one, but p possible intervals for the start of service at vertex i, we have a vehicle routing problem with multiple time windows. In that case (7b) has to be replaced by $w_{ik} \in \bigcup_{l=1}^{p} [a_i^l, b_i^l]$.

The travel time may also depend on the time of the day. In that case t_{ij} is not a constant, but a stepwise function $t_{ij}(w_{ik} + s_i)$ depending on the time the vehicle leaves the customer i. Chen et al. (2006) take the problem even further by considering a real-time time-dependent VRP with time windows (RT-TDVRPTW). In their problem, the travel time does not follow a predetermined function, but travel times may change at any time due to unexpected incidents. So this is an example of a dynamic VRP like we discussed in Section 3.3.2. The authors solved the problem by employing a rolling time horizon and solving a mixed integer programming submodel at every point in time when travel times change.
Time windows do not necessarily have to be hard constraints. Taillard et al. (1997) consider the VRP with soft time windows (VRPSTW) where each customer $i \in N$ still has a given time window $[a_i, b_i]$ in which they should be serviced, but they do not necessarily have to be serviced in that window. Nevertheless, the service at customer i still may not start before a_i. So if the vehicle arrives too early, it has to has to wait until a_i. On the other hand, if the customer is serviced too late, this is accounted for by an non-negative penalty function. A objective function is then given by

$$\min_{x_{ijk}:\ (i,j)\in A, k\in K} \quad \sum_{(i,j)\in A}\sum_{k\in K} c_{ijk}x_{ijk} + \sum_{i\in N}\sum_{k\in K} \alpha_i (w_{ik} - b_i)^+,$$

where α_i are lateness penalty coefficients and $p^+ = \max(p, 0)$. In other variants of the VRP, the overall waiting time or even the waiting time for each individual vehicle may be bounded or penalized as well. Note that by appropriately increasing the lateness penalty coefficients α_i, the problem becomes a vehicle routing problem with hard time windows again. Therefore, the VRPSTW can be seen as a more general formulation of the VRPTW.

3.3.4 Fleet Composition

Until now, we have only discussed vehicle routing problems with identical vehicles, all based in a single depot. In this section, we want to have a look at problems with fleets consisting of vehicles with different characteristics regarding capacity, costs, speed or customers they can serve. Furthermore, these vehicles may be stationed in different depots where they start and end their tour.

Multiple Depot VRP: At first, we want to consider a problem still consisting of a homogeneous fleet, but this time the vehicles may start and end their tour at different depots. This variant of a VRP is called Multi-Depot VRP (MDVRP, see Renaud et al., 1996).
In particular, the set of nodes is partitioned into two subsets $V_c = \{1, \dots, n\}$ and $V_d = \{n+1, \dots, n+p\}$, representing the set of customers and depots, respectively. With each customer $i \in V_c$ a non-negative demand q_i is associated, and $|K_j|$ identical vehicles with capacity Q are based at each depot $j \in V_d$. Depots may have only a limited capacity and the vehicle fleet may be limited or unlimited. To account for an unlimited fleet, $|K_j|$ can simply be set to be equal to n.

Another interesting variation of the problem arises when the vehicles may be replenished at intermediate depots along their route. This type of problem is known as Multi-Depot VRP with Inter-Depot Routes (MDVRPI). Crevier et al. (2007) introduced an algorithm for solving this type of problem, motivated by a real-life grocery distribution problem in Montana. The problem is closely related to the VRP with multiple use of vehicles we discussed in Section 3.3.3 with the difference that we now consider multiple depots.

Heterogeneous or Mixed Fleet VRP: In industry, a fleet hardly ever consists of only one type of vehicle. Since vehicles are usually operated for several years, new ones are acquired in the meantime and therefore the vehicles will usually have different levels of technology. Maintenance and operation costs will change over the years as well. In addition, fleet owners often wish to have a fleet consisting of diverse vehicle types to achieve more versatility. An extensive survey on the industrial aspects of fleet composition and routing is provided by Hoff et al. (2010). The authors distinguish between vehicle types regarding three main aspects:

- physical dimensions,
- compatibility constraints, and
- costs.

Physical dimensions like the length or the height of a vehicle roughly determine the capacity of a vehicle. Obviously, a train has a larger capacity than trucks, but there exist different types of trains as well: Smaller cars can be transported in double-deck railcars, which roughly doubles their capacity compared to single-deck railcars. On the other hand, bigger cars like SUVs do not fit on such railcars and therefore their capacity is effectively 0 in that case. But physical dimensions can constrain the usability of the vehicle as well: Double-deck railcars for instance are only allowed to travel on certain tracks, or large trucks may face restrictions due to limited space at ramps for loading or unloading.
Compatibility constraints define where a vehicle may travel and what type of products it may transport. Often special equipment is needed like for instance cooled compartments for food or drugs, or large items may not be able to be rear-loaded. In addition, certain certificates may be needed due to increasingly strong environmental restrictions in urban areas.
Costs are obviously an important factor when it comes to deciding which type of vehicle to use. While larger vehicles usually have lower transportation costs per unit due to a higher capacity utilization, they are less flexible to fluctuating transport volumes. When it comes to the age of the vehicles, older ones have in general higher maintenance and environmental costs, but on the other hand lower depreciation costs. Over a shorter time horizon, the main goal is to balance fixed costs of the fleet with potential extra costs arising when demand exceeds the fleet's capacity and external capacity needs to be bought. But extra capacity may have a positive value as well: Due to more flexibility, there may exist a more cost efficient routing solution.
Formally, this type of problem is known as Heterogeneous or Mixed Fleet VRP (HFVRP).

Considering a set of different vehicle types P, the fleet K is partitioned into $|P|$ subsets of homogeneous vehicles $K = K^1 \cup K^2 \cup \ldots \cup K^{|P|}$. All vehicles of type $p \in \{1, \ldots, |P|\}$ are characterized by a capacity $Q_k = Q^p$, travel times $t_{ijk} = t^p_{ij}$, variable transport costs $c_{ijk} = c^p_{ij}$, fixed costs $FC_k = FC^p$ and a subset of customers $N_k = N^p \subseteq N$ that are accessible by vehicles of type p.

Fixed costs reflect the costs of the driver as well as maintenance and operating costs of the vehicle, but they become only relevant if not all vehicles have to be used. Another factor regarding the importance of considering fixed costs is whether the fleet is owned by the decision maker. If the trucks are owned by a subcontractor, fixed costs will often simply be a part of the variable transport costs.

To account for vehicle specific attributes one just has to replace the general coefficients in (3a)-(3i) with the vehicle specific ones, e.g., the capacity Q with Q_k. Fixed costs can be considered by replacing c_{oj} with $c_{ojk} + FC_k$ for all $(o, j) \in \delta^+(o)$. Also inaccessible customers $j \in N \backslash N_k$ can be modeled by setting the costs c_{ijk} to a sufficiently large number M for all arcs $(i, j) \in \delta^-(j)$. In principle, the model represents a VRP with a limited fleet size since the fleet K consists of exactly $|K| = |K^1| + |K^2| + \ldots + |K^{|P|}|$ vehicles. However, to account for an unlimited fleet size, one just has to choose sufficiently large numbers of vehicles $|K^p|$.

4 Exact Solution Methods

In the following two chapters we want to give a broad overview on how to actually solve vehicle routing problems. All the various types of VRPs we introduced in Chapter 3 are formulated as Mixed Integer Linear Programs (MILP) and there are basically two approaches to find sufficiently good solutions for such problems. In this chapter we present exact solution methods and the most important concepts of finding the optimal solution of an optimization problem. In theory, an exact algorithm finds the optimal solution within a finite number of steps, but often the time it actually takes for doing that is way too long to be applicable in many real-world scenarios. If this is the case, heuristic solution methods may be the better way to solve such problems. This type of optimization algorithms will be discussed in Chapter 5.
To start things off, let us consider an Integer Program of the form

$$z = \min\{c(x) : x \in X \subseteq \mathbb{Z}^n\} \tag{IP}$$

with an objective function $c(.)$ and decision variables x. A rather "naive" way to derive the optimal solution x^* would be finding a lower bound $\underline{z} \leq z$ and an upper bound $\overline{z} \geq z$ such that $\underline{z} = z = \overline{z}$. Practically, this means finding an algorithm that generates a decreasing sequence

$$\overline{z}_1 \geq \overline{z}_2 \geq \ldots \geq \overline{z}_s \geq z$$

of upper bounds and an increasing sequence

$$\underline{z}_1 \leq \underline{z}_2 \leq \ldots \leq \underline{z}_t \leq z$$

of lower bounds until a stopping criterion of the form

$$\overline{z}_s - \underline{z}_t \leq \epsilon$$

is fulfilled, where ϵ is a sufficiently small value. This simple idea is an important aspect of branch and bound algorithms, which we are going to discuss in the following section. For that purpose, the definitions and propositions are largely based on the ones presented in Wolsey (1998).

4.1 Branch and Bound Algorithms

The basic idea of branch and bound algorithms is to break up the problem into a series of subproblems, which can be solved more easily, and then put the information back together to solve the original problem. Thus, branch and bound algorithms follow a divide and conquer approach to solve the problem.

Proposition 4.1. *Let $X = X_1 \cup \ldots \cup X_K$ be a decomposition of X into smaller subsets and $z^k = \min\{c(x) : x \in X_k\}$ for $k = 1, \ldots, K$. Then $z = \min_k z^k$.*

Even though this proposition seems pretty trivial, it is the key of branch and bound methods. The recursive decomposition of X can be represented by an enumeration tree. Figure 4 shows what such a enumeration tree might look like for $X \subseteq \{0,1\}^3$.

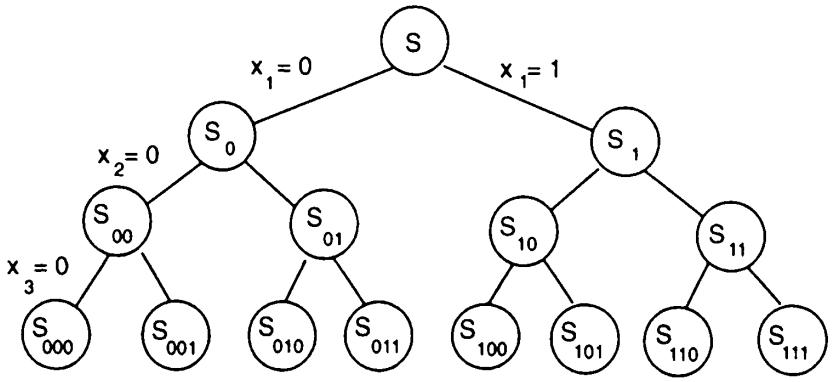

Figure 4: Binary enumeration tree. *Source: Wolsey, 1998, p. 92*

In the TSP context, one could branch on the arcs leaving for instance node 1. So X is then divided into subsets $X_{12}, \ldots, X_{1n}$, where X_{ij} is the set of all tours containing the arc $(i,j) \in A$. A complete enumeration, as it is shown in Figure 4, is impossible for most problems, as the number of different cases one has to consider explodes even for a small number of nodes. The idea is to find lower and upper bounds for each node in the enumeration tree and thus be able to tell beforehand whether the optimal solution can be in a certain branch. If we know that this is not the case, we can prune the tree at the node and ignore all solutions down that branch. This is known as implicit enumeration, as the enumeration tree is not unfolded completely.

Proposition 4.2. *Let $X = X_1 \cup \ldots \cup X_K$ be a decomposition of X into smaller subsets, $z^k = \min\{c(x) : x \in X_k\}$ for $k = 1, \ldots, K$, $\underline{z}^k$ be a lower bound on z^k and $\overline{z}^k$ be an upper bound on z^k. Then $\underline{z} = \min_k \underline{z}^k$ is a lower bound on z and $\overline{z} = \min_k \overline{z}^k$ is an upper bound on z.*

Based on Proposition 4.2 we can deduce three rules when the enumeration tree can be pruned at the k-th node:

- **Pruning by optimality:** If $\underline{z}^k = \overline{z}^k$, the optimal value z^k of the subproblem on X_k has been found and therefore X_k does not have to be divided any further into smaller subproblems.
- **Pruning by bound:** If $\underline{z}^k > \overline{z}$, the optimal solution in X_k is worse than an upper bound already found. Therefore, the optimal solution cannot be in that branch and it can be pruned.
- **Pruning by infeasibility:** If $X_k = \emptyset$, there exists no feasible solution down that node and the branch can obviously be pruned.

What we have not discussed so far is how to obtain the upper and lower bounds to be able to prune the tree. In the context of a minimization problem, upper bounds are often referred to as

primal bounds, whereas lower bounds are called dual bounds (why this is the case will become clear pretty soon).

Primal Bounds: Every feasible solution $x \in X$ provides an upper bound $\overline{z} = c(x) \geq z$ of the problem. Unfortunately, this is essentially the only way to find upper bounds. For some problems it might be easy to find feasible solutions and the actual task is to find "good" solutions. For other problems, especially if the set of feasible solutions is restricted by a set of complex constraints, finding a feasible solution might be almost as difficult as finding the optimal solution. Feasible solutions are usually found by using heuristics like a greedy heuristic or local search (or a combination of both). A greedy heuristic starts with an empty set and adds step by step the immediate "best" item. For the TSP that would mean the algorithm starts at a random point and in each step it adds the point closest to it that has not been visited yet. This easily generates a feasible solution, but it is quite likely that this solution is far off from being optimal. Afterwards, local search can be used to improve this first initial solution, called the incumbent solution. Local search compares the incumbent solution with other solutions in a neighborhood close to it. If a better one is found, the incumbent is replaced by that solution and the procedure is repeated. If the solution cannot be improved any further, it is locally optimal with respect to the neighborhood and the algorithm terminates. We will have a closer look at local search algorithms in Section 5.3.

Dual Bounds: Finding lower bounds of z is a different problem. The most important approach to find such bounds is via relaxation. Basically the idea is to replace the "difficult" original problem by an "easier" optimization problems whose optimal value is not larger than the one of the original problem. Obviously, there are two possibilities to relax a problem:

- Enlarge the set of feasible solutions so that the optimization is performed on a larger set.
- Replace the objective function by a function that has the same or smaller value for all feasible solutions.

Definition 4.3. *A problem RP* $z^R = \min\{f(x) : x \in T \subseteq \mathbb{R}^n\}$ *is a relaxation of the problem IP* $z = \min\{c(x) : x \in X \subseteq \mathbb{R}^n\}$ *if and only if*

(i) $X \subseteq T$

(ii) $c(x) \geq f(x),\ \forall x \in X.$

Proposition 4.4. *If RP is a relaxation of IP, then* $z^R \leq z$.

Proof. Let x^* be the optimal solution of IP, i.e., $x^* \in X$ and $z = c(x^*) \geq f(x^*)$. Since $x^* \in T$, $f(x^*)$ is an upper bound of z^R and thus $z \geq f(x*) \geq z^R$. □

Another important aspect of relaxations is that they can be used to show that a solution is optimal or a problem is infeasible.

Proposition 4.5.

(i) If a relaxation RP is infeasible, the original problem IP is infeasible.

(ii) Let x^* be an optimal solution of RP. If $x^* \in X$ and $f(x^*) = c(x^*)$, then x^* is an optimal solution of IP.

Proof. Ad *(i)*: As the RP is infeasible, $T = \emptyset$ and so $X \subseteq T = \emptyset$.
Ad *(ii)*: As $x^* \in X$, we obtain $z \leq c(x^*) = f(x^*) = z^R$. According to Proposition 4.4 we know that $z^R \leq z$ and thus $c(x^*) = z = z^R$. □

Now the interesting question is how to construct useful relaxations. One of the most often used relaxations is the linear programming relaxation. It provides an easy way to compute dual bounds within a short time. The only downside is that it can only be applied to linear integer programs.

Definition 4.6. *For the linear integer program $z = \min\{c^T x : x \in P \cap \mathbb{Z}^n\}$ with formulation $P = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$, the linear programming relaxation is the linear program $z^{LP} = \min\{c^T x : x \in P\}$.*

This is clearly a relaxation of the original problem as $P \cap \mathbb{Z}^n \subseteq P$ and the objective function is the same for both programs.
Another possible way of relaxing a linear program is the so-called Lagrangian relaxation. The idea is to drop all constraints $Ax \leq b$ and solve the resulting relaxation $z' = \min\{c^T x : x \in \mathbb{Z}^n\}$. In Section 3.2, we mentioned that one way of dealing with the exponentially many constraints of the VRP is to drop the subtour elimination constraints at first and add them iteratively until a feasible and therefore optimal solution is found. This already is an example of a Lagrangian relaxation. The idea can be extended by not just dropping the constraints, but adding them into the objective function with Lagrange multipliers.

Definition 4.7. *For the linear integer program $z = \min\{c^T x : x \in P \cap \mathbb{Z}^n\}$ with formulation $P = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$ and $u \geq 0$, the Lagrange relaxation is the linear program $z(u) = \min\{c^T x - u^T(b - Ax) : x \in \mathbb{Z}^n\}$.*

Another possibility to obtain dual bounds is to utilize the dual problem (this is where the dual bounds got their name from). The key property of the dual problem is that every feasible solution provides a lower bound on the original problem.

Definition 4.8. *The two problems*

$$z = \min\{c(x) : x \in X\} \tag{IP}$$
$$w = \max\{\omega(u) : u \in U\} \tag{D}$$

form a (weak)-dual pair, if $c(x) \geq \omega(u)$ for all $x \in X$ and all $u \in U$. If $z = w$, they form a strong-dual pair.

The advantage of the dual problem compared to a relaxation is that the dual problem does not have to be solved to optimality to obtain a lower bound. Instead, every feasible solution already generates a dual bound. An example for a weak-dual pair is the following:

Proposition 4.9. *The integer program* $z = \min\{c^T x : Ax \leq b, x \in \mathbb{Z}^n_+\}$ *and the linear program* $w^{LP} = \max\{u^T b : u^T A \geq c^T, u \in \mathbb{R}^m_+\}$ *form a weak-dual pair.*

Analogously to Proposition 4.5, we can use to the dual problem to proof the infeasibility of the problem or the optimality of a solution.

Proposition 4.10. *Suppose that IP and D form a weak-dual pair.*

(i) If D is unbounded, IP is infeasible.

(ii) If $x^* \in X$ *and* $u^* \in U$ *satisfy* $c(x^*) = w(u^*)$*, then* x^* *is optimal for IP and* u^* *is optimal for D.*

Finally, we want to tackle the question, which nodes in an enumeration tree should be divided. Thereby, two contradictory arguments can be brought up:

- The tree can only be pruned significantly, if a feasible solution giving a hopefully good upper bound has been found. Descending as quickly as possible in the enumeration tree is the key to find feasible solutions and therefore one should follow a Depth-First Search strategy. Another advantage of this strategy is that resolving the LP-relaxation can easily be done when just a constraint is added.

- Another idea is to minimize the number of nodes that are evaluated in the tree and therefore always choose the active node with the best bound (i.e., choose the node s where $\underline{z}^s = min_k\ \underline{z}^k$). By doing so, one will never divide a node whose lower bound $\underline{z}^k$ is larger than the optimal value z. So this observation suggests the use of a Best-Node First strategy.

In practice, it is often the best idea to follow an approach that is the combination of both. For instance, one may start with a depth-first strategy until at least one feasible solution has been found and from there on mix best-node and depth-first to on one hand try to prove optimality and on the other hand find better upper bounds by finding better feasible solutions.

4.2 Cutting Plane Algorithms

In the next section we are going to discuss alternative formulations of integer programs and what distinguishes a "good" from a "bad" formulation. Let us consider a mixed integer linear program of the form

$$z = \min\{c(x) : x \in X \subseteq (\mathbb{Z}^n \times \mathbb{R}^p)\}. \tag{IP}$$

Then the formulation of the problem can formally be defined as follows:

Definition 4.11. *A subset of $\mathbb{R}^n$ described by a finite set of linear constraints $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ is a polyhedron.*

Definition 4.12. *A polyhedron $P \subseteq \mathbb{R}^{n+p}$ is a formulation of a set $X \subseteq \mathbb{Z}^n \times \mathbb{R}^p$ if and only if $X = P \cap (\mathbb{Z}^n \times \mathbb{R}^p)$.*

In theory, there are infinitely many formulations of a given problem. So which one of these is the best and what characterizes a good formulation? The answer is closely related to the linear programming relaxation: The smaller the set P is, the better are the dual bounds gained from the relaxation. Formally, this is described in the following proposition.

Proposition 4.13. *Suppose P_1 and P_2 are two formulations of the mixed integer linear program $z = \min\{c^T x : x \in X \subseteq (\mathbb{Z}^n \times \mathbb{R}^p)\}$ such that $P_1 \subset P_2$. If $z_i^{LP} = \min\{c^T x : x \in P_i\}$ for $i = 1, 2$ are the values of the associated LP relaxations, then $z_1^{LP} \geq z_2^{LP}$.*

Based on that observation, we use the following definition to compare two alternative formulations:

Definition 4.14. *Given a set $X \subseteq \mathbb{R}^n$ and two formulations P_1 and P_2 for X, P_1 is a better formulation than P_2, if $P_1 \subset P_2$.*

So, is there a best formulation that is a subset of all other formulations? In Figure 5 we see three different formulations of the set

$$X = \{(1,1), (2,1), (3,1), (1,2), (2,2), (3,2), (2,3)\}.$$

The formulation P_3 is better than the other two as it is a subset both of P_1 and of P_2. Furthermore, it is ideal in the sense that when solving the LP relaxation, the optimal solution will always be an extreme point and here all extreme points are integer. Thus, the LP relaxation already gives us the optimal solution.
Formally P_3 represents the convex hull of X, what leads us to the following formal description of this idea:

Definition 4.15. *Given a set $X \subseteq \mathbb{R}^n$, the convex hull of X, denoted* $\mathrm{Conv}(X)$*, is defined as*

$$\mathrm{Conv}(X) = \left\{ \sum_{i=1}^{|X|} \lambda_i x_i : \sum_{i=1}^{|X|} \lambda_i = 1, \lambda_i \geq 0 \ \forall i = 1, \ldots, |X| \right\}.$$

Proposition 4.16. $\mathrm{Conv}(X)$ *is a polyhedron.*

Proposition 4.17. *All extreme points of* $\mathrm{Conv}(X)$ *lie in X.*

Indeed, $\mathrm{Conv}(X)$ is the best formulation of X as there does not exist a polyhedron containing X and being at the same time a subset of $\mathrm{Conv}(X)$. So theoretically, once we have found a characterization of the convex hull, all we have to do is to solve the LP relaxation. The problem is that in most cases there is no simple characterization of $\mathrm{Conv}(X)$ and an enormous number

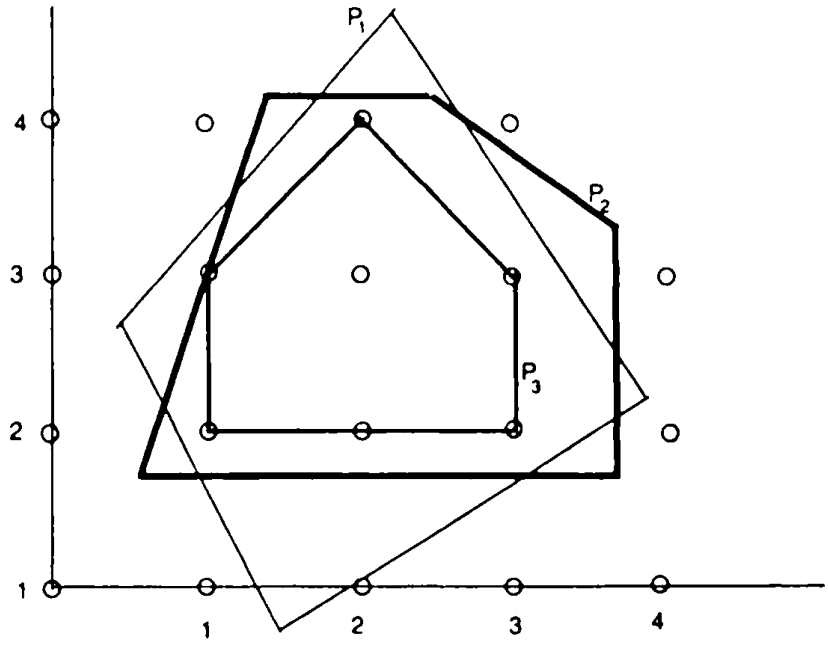

Figure 5: Different formulations of an IP. *Source: Wolsey, 1998, p. 15*

of inequalities would be needed to describe it. Therefore, we are looking for an effective way to approximate Conv(X) for a given instance.
This leads us to cutting plane algorithms: The basic idea is to solve the LP relaxation of the problem. and if the solution is not an integer solution, a new inequality is added to the formulation that "cuts off" this solution. The procedure is repeated until an integer solution is obtained.
When adding inequalities to the formulation, it is important that no feasible solution is excluded. If this is not the case, we call the inequality a valid inequality.

Definition 4.18. *An inequality $\pi^T x \leq \pi_0$ is a valid inequality for a set $X \subseteq \mathbb{R}^n$, if $\pi^T x \leq \pi_0$ holds for all $x \in X$.*

Although Dantzig et al. (1954) had already developed an algorithm for the TSP to cut off non-integer solutions of the LP relaxation, Gomory (1958) was the first to introduce a general approach for an arbitrary linear integer program, the so-called Gomory Cuts.

Gomory Cuts

Let us consider a linear integer program in standard form, i.e., $z = \min\{c^T x : x \in X = P \cap \mathbb{Z}^n_+\}$ with $P = \{x : Ax = b\}$. Note that each linear IP with fractional coefficients can be written that way by scaling all coefficients to integers and adding slack variables afterwards to rewrite the inequalities as equations.
Let x^* be the optimal solution of the LP relaxation obtained by using the simplex algorithm. The set $B \subseteq \{1, \ldots, n\}$ denotes the indices of the basis variables with $x^*_B = A_B^{-1} b - A_B^{-1} A_N x_N$

and $x_N^* = 0$, where $N = \{1, \ldots, n\} \backslash B$. If x^* is an integer solution, we have already found the optimal solution of the IP and the algorithm terminates. Otherwise, there exists an index $i \in B$ with x_i^* being fractional. For convenience, we define $\bar{a}_{ij} = (A_B^{-1} A_N)_{ij}$ and $\bar{b}_i = (A_B^{-1} b)_i$. Then we get an equation of the form

$$x_i + \sum_{j \in N} \bar{a}_{ij} x_j = \bar{b}_i, \tag{9}$$

with x_i being the basic variable and x_j for $j \in N$ being nonbasic variables. In the next step, we rewrite the equation so that all integer parts are on the left-hand side and all fractional parts are on the right-hand side:

$$x_i + \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j - \lfloor \bar{b}_i \rfloor = \bar{b}_i - \lfloor \bar{b}_i \rfloor - \sum_{j \in N} \left(\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor \right) x_j. \tag{10}$$

As all feasible $x \in X$ have to be integers, the left-hand side has to be an integer as well. In addition, the right-hand side has to be smaller than 1 as $x \geq 0$ and for any $\alpha \in \mathbb{R}$ it holds that $0 \leq \alpha - \lfloor \alpha \rfloor < 1$. Combining those two observations, we get

$$\bar{b}_i - \lfloor \bar{b}_i \rfloor - \sum_{j \in N} \left(\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor \right) x_j \leq 0. \tag{11}$$

Moreover, (11) is violated by the current solution x^* as $x_N = 0$ and $\bar{b}_i = x_i^* \notin \mathbb{Z}$:

$$\bar{b}_i - \lfloor \bar{b}_i \rfloor - \sum_{j \in N} \left(\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor \right) x_j = \bar{b}_i - \lfloor \bar{b}_i \rfloor > 0.$$

Finally after subtracting (9) from (11) we obtain

$$x_i + \sum_{j \in N} \lfloor \bar{a}_{ij} \rfloor x_j \leq \lfloor \bar{b}_i \rfloor, \tag{12}$$

which is, as we have seen, a valid inequality that cuts off the noninteger solution when added to the formulation. Note that again a slack variable is needed to rewrite the inequality as an equation, but since all coefficients in (12) are already integers, the slack variable is an integer as well. This procedure is then repeated until an integer solution is obtained. The derivation of the cuts can be found in Marchand et al. (2002). More than 50 years ago, Gomory (1963) has already proven that with a particular type of cuts an integer solution is obtained after only a finite number of steps.

The downside of this approach is, however, that it cannot be applied to Mixed Integer Linear Problems (MILP), i.e., problems of the form $\min\{c^T x : Ax = b, x \in (\mathbb{Z}_+^p \times \mathbb{R}_+^{n-p})\}$. The problem arises in equation (10) as the left-hand side does not have to be an integer any more. However, Gomory (1960) presented an extension of his first approach to MILPs, introducing the "mixed-integer cuts", known today as Gomory mixed-integer cuts (GMI cuts).

By using the same notation as before, again we get an equation of the form (9) after applying the

simplex method. As x_i is an integer, this expression is equivalent to $\sum_{j \in N} \bar{a}_{ij} x_j = \bar{b}_i - \lfloor \bar{b}_i \rfloor + k$ for some $k \in \mathbb{Z}$. In addition, we now define $N^+ = \{j \in N : \bar{a}_{ij} x_j \geq 0\}$ and $N^- = N \backslash N^+$ to distinguish the two cases: $\sum_{j \in N} \bar{a}_{ij} x_j \geq 0$ and $\sum_{j \in N} \bar{a}_{ij} x_j < 0$.
In the first case, as the left-hand side is nonnegative, k has to be nonnegative too, and therefore,

$$\bar{b}_i - \lfloor \bar{b}_i \rfloor \leq \sum_{j \in N^+} \bar{a}_{ij} x_j + \sum_{j \in N^-} \bar{a}_{ij} x_j \leq \sum_{j \in N^+} \bar{a}_{ij} x_j.$$

In the second case, k has to be negative and therefore we get $\sum_{j \in N^-} \bar{a}_{ij} x_j \leq \bar{b}_i - \lfloor \bar{b}_i \rfloor - 1$, which is equivalent to

$$\bar{b}_i - \lfloor \bar{b}_i \rfloor \leq - \frac{\bar{b}_i - \lfloor \bar{b}_i \rfloor}{1 - \bar{b}_i - \lfloor \bar{b}_i \rfloor} \sum_{j \in N^-} \bar{a}_{ij} x_j.$$

So whatever case occurs, it is always true that

$$\bar{b}_i - \lfloor \bar{b}_i \rfloor \leq \sum_{j \in N^+} \bar{a}_{ij} x_j - \frac{\bar{b}_i - \lfloor \bar{b}_i \rfloor}{1 - \bar{b}_i - \lfloor \bar{b}_i \rfloor} \sum_{j \in N^-} \bar{a}_{ij} x_j, \tag{13}$$

as the right-hand side is the sum of two nonnegative numbers, of which at least one is $\geq \bar{b}_i - \lfloor \bar{b}_i \rfloor$. Like before, as the nonbasic variables x_N^* are all equal to 0, this inequality is violated by the current optimal solution x^* and thus cuts it off.
Gomory (1960) showed that an algorithm that iteratively adds inequalities of the form (13) solves the MILP $z = \min\{c^T x : x \in X = P \cap (\mathbb{Z}_+^p \times \mathbb{R}_+^{n-p})\}$ with $P = \{x : Ax = b\}$ within a finite number of steps, provided that $c^T x \in \mathbb{Z}$ for all $x \in X$.
Even though Gomory's results were a great theoretical breakthrough as he was able to reduce an integer linear program to a sequence of linear programs, it turned out pretty soon that his approach was practically unusable. One problem of cutting plane algorithms is that usually a large number of iterations is needed until an integer solution is found: "...they do not work well in practice. They fail because an extremely large number of these cuts frequently are required for convergence" (Nemhauser and Wolsey, 1989, p. 486). Furthermore, an even bigger problem is the poor computational performance: "The main difficulty has come, not from the number of iterations, but from numerical errors in computer arithmetic." (Parker and Rardin, 1988, p. 290).

Branch and Cut Algorithms

Therefore, only little attention was payed to Gomory cuts until the mid 1990s. This changed when Balas et al. (1996) introduced a branch and cut algorithm that uses Gomory mixed-integer cuts at each iteration of a branch and bound algorithm. They were able to solve 86% of the test instances compared to only 55% of the pure branch and bound algorithm within the given time and space limitations. One problem of branch and cut algorithms has always been that the cuts are only valid at a particular node of the enumeration tree as some variable values are

fixed, but they managed to generate cuts that are valid for the whole MILP by what they call "lifting" the cut. Indeed, this hugely improved the algorithm: In a computational experiment they compared their algorithm to one where the cutting planes were used only locally and as a consequence the code did not terminate in almost half of the instances. In addition, they did not use all cuts at every iteration, what enabled them to keep the linear program relatively small. And finally, they of course benefited from the faster and numerically more stable LP solvers that they had at hand compared to Gomory 30 years before.
Besides Gomory cuts, other types of cuts like "Mixed-integer-rounding cuts" or "Lift-and-project cuts" are used in branch and cut algorithms as well. For a detailed description of those cuts we refer to Marchand et al. (2002). The number of cuts added to the problem is always a trade-off, because adding lots of cuts can significantly tighten the formulation, but on the other hand reoptimization may be much slower than before. Even though branch and cut looks very similar to branch and bound, it is quite a change of philosophy. Rather than aiming for a fast reoptimization, branch and cut algorithms try to find tight dual bounds at each node. Therefore, the number of visited nodes in the enumeration tree is significantly smaller compared to pure branch and bound algorithms.

4.3 Column Generation Algorithms

At last, we present the basic idea of column generation algorithms, which are needed to solve VRP models in extensive formulation as we have seen them in Section 3.2.2.
Considering the IP in extensive form (4a)-(4d), the obvious problem is the huge number of possible routes $|\Omega|$. Since enumerating all the routes is (besides really small models) impossible, a so-called column generation technique is applied. The basic idea is as follows: Only a portion of routes is enumerated and the LP-relaxation is solved, considering only this partial route set. The solution of the relaxation is then used to find new routes that should be included in the formulation to further reduce the optimal function value. This is called the column generation step. To determine such a new route that should be added to the formulation, the dual variables are used to construct a simpler optimization problem that then has to be solved. Afterwards the LP relaxation of the expanded problem is solved again. This process is repeated until no additional route can be found to further reduce the transportation cost.
Let us now have a look at this procedure in more detail. First, we have to generate a subset of all possible routes $\Omega' \subseteq \Omega$ with which we start the algorithm. Finding a good initial set of routes can greatly improve the performance of the algorithm. The LP relaxation with respect to this subset of routes is then given as

$$\min_{\lambda_r:\ r\in\Omega'} \quad \sum_{r\in\Omega'} c_r\lambda_r \tag{14a}$$

$$\text{subject to} \quad \sum_{r\in\Omega'} a_{ir}\lambda_r = 1 \qquad \forall i \in N \tag{14b}$$

$$\sum_{r\in\Omega'} \lambda_r = |K| \tag{14c}$$

$$\lambda_r \in \{0,1\} \qquad \forall r \in \Omega'. \tag{14d}$$

Here again, c_r are the routing costs of route $r \in \Omega$, a_{ir} indicates whether route r visits customer $i \in N$ and λ_r is the decision variable denoting whether or not route r is part of the solution.
Now let $\bar{\lambda}$ be the optimal solution of the relaxation. Furthermore, let $\bar{\pi} = \{\bar{\pi}_1, \ldots, \bar{\pi}_n\}$ be the corresponding optimal dual variables associated with constraints (14b) and $\bar{\theta}$ be the dual variable associated with constraint (14c). The question is whether or not $\bar{\lambda}$ already is the optimal value of the LP relaxation of the original problem with respect to the whole set of routes Ω. This is equivalent to whether $(\bar{\pi}, \bar{\theta})$ is the optimal solution to the dual problem of the LP relaxation. Therefore, let us have a look at the dual problem:

$$\max_{\theta, \pi_i:\ i \in N} \quad \sum_{i \in N} \pi_i - |K|\theta \tag{15a}$$

$$\text{subject to} \quad \sum_{i \in N} a_{ir}\pi_i - \theta \leq c_r \qquad \forall r \in \Omega. \tag{15b}$$

Clearly, $(\bar{\pi}, \bar{\theta})$ is the optimal solution of the dual problem only if all constraints (15b) are fulfilled. In that case $\bar{\lambda}$ is the optimal solution of the primal problem. So how can we check whether $(\bar{\pi}, \bar{\theta})$ fulfills all constraints? Note that we only have to find a single route r such that

$$-\bar{\theta} > c_r - \sum_{i \in N} a_{ir}\bar{\pi}_i \tag{16}$$

and the solution is not feasible as then constraint (15b) is violated for that route. So if we minimize $c_r - \sum_{i \in N} a_{ir}\bar{\pi}_i$ and find a route r such that the quantity is less than $-\bar{\theta}$, we have found a violated constraint. Then the solution $(\bar{\pi}, \bar{\theta})$ is not optimal for the problem, but we can use this information to add the corresponding column to the formulation of the primal problem and resolve it. This process is then repeated until no violated constraint is found. In that case $(\bar{\pi}, \bar{\theta})$ is the optimal solution of the dual problem and $\bar{\lambda}$ is therefore a lower bound of the minimal total routing costs.
So specifically, the column generation problem is to find a route $r \in \Omega$ that satisfies (16). Let S_r be the set of customers appearing in route r and define $\bar{c}_r$ to be the reduced cost of column r, i.e., $\bar{c}_r = c_r + \bar{\theta} - \sum_{i \in S_r} a_{ir}\bar{\pi}_i$ for each $r \in \Omega$. For a CVRP, the column generation problem is then formally given as

$$\min_{r \in \Omega} \left\{ \bar{c}_r : \sum_{i \in S_r} q_i \leq Q \right\}.$$

What might sound not too difficult at first sight turns out to be not easy at all: The problem is again $\mathcal{NP}$-hard (see Section 5.1), as even for a given set S_r determining c_r requires solving a TSP with respect to the vertices $S_r \cup \{0\}$.
In summary, the column generation algorithm for solving the LP relaxation of a VRP in extensive form can be described as follows:

Step 1: Generate an initial set of routes Ω'.

Step 2: Solve the LP relaxation to get optimal solutions $\bar{\lambda}$ and $(\bar{\pi}, \bar{\theta})$, respectively.

Step 3: Solve the column generation problem or equivalently find a route $r \in \Omega$ with $\bar{c}_r < 0$.

Step 4: For every $r \in \Omega$ satisfying $\bar{c}_r < 0$, add the route r to Ω' and go to step 2.

Step 5: If there are no routes with $\bar{c}_r < 0$, stop.

The result of this procedure is a vector $\bar{y}$ which is the optimal solution of the LP-relaxation and therefore a lower bound of the original problem. For a detailed description of different approaches to solve the column generation problem we refer to Bramel and Simchi-Levi (2002).

5 Heuristic Optimization Methods

The big advantage of exact solution methods is that they deliver the optimal solution and guarantee its optimality. Unfortunately, it often takes way too long to solve a given problem with an exact algorithm. This is where heuristic solution methods may be the better option as they provide high-quality solutions within a reasonable amount of time. However, with these methods there is no guarantee of finding the optimal solution.
Before discussing heuristic optimization methods, we want to have a quick look at why it is so difficult to solve vehicle routing problems with an exact algorithm. Therefore, we start with a short excursus on the complexity of problems.

5.1 The $\mathcal{P}$ versus $\mathcal{NP}$ Problem

In the last decades, the computational power has dramatically increased while the cost of computing has drastically decreased. At the same time, algorithms have improved as well and still there are certain problems that cannot be solved within a reasonable amount of time. In complexity theory, these problems are called $\mathcal{NP}$-hard, but before we can talk about different classes of complexity, we need some formal definitions. All the following definitions can be found in Chapter 6 in Wolsey (1998).
First of all, complexity theory does not address exactly optimization problems like we have discussed so far. Instead, decision problems having a YES or NO answer are considered, but as we will see in a moment, the results can easily be carried over. So instead of an optimization problem

$$\min\{c^T x : x \in F\},$$

where F is the set of all feasible solutions, we consider the decision problem

$$\text{Is there an } x \in F \text{ with value } c^T x \geq k?$$

for any real number k.
Next we want to state a formal definition of the running time of an algorithm. Since the length of the input is not only given by the number of variables, constraints or nodes, but by the size of the numbers occurring in the data as well, we first have to define what the length of the input actually is.

Definition 5.1. *For a problem instance X, the length of the input $L = L(X)$ is the length of the binary representation of the instance.*

Definition 5.2. *Given a problem P, an algorithm A and an instance X, let $f_A(X)$ be the number of elementary calculations required to run the algorithm A on the instance. $f_A^*(l) = \sup_X\{f_A(X) : L(X) = l\}$ is the running time of algorithm A. An algorithm A is polynomial for a problem P, if $f_A^*(l) = \mathcal{O}(l^p)$ for some positive integer p.*

Now we are able to define the class of $\mathcal{NP}$-problems.

Definition 5.3. *$\mathcal{NP}$ is the class of decision problems with the property that for any instance for which the answer is YES, there is a polynomial proof that the answer is indeed YES.*

Note that any optimization problem with the associated decision problem lying in $\mathcal{NP}$, can be solved by answering the decision problem a polynomial number of times. This can be done by using bisection on the objective value. A subset of $\mathcal{NP}$ is the class of the "easy" problems $\mathcal{P}$:

Definition 5.4. *$\mathcal{P}$ is the class of decision problems in $\mathcal{NP}$ for which there exists a polynomial algorithm.*

Now that we have defined what we consider as easy problems, we can move on to the "most difficult" problems, namely the $\mathcal{NP}$-complete problems. As a consequence, this also leads us to the $\mathcal{P}$ versus $\mathcal{NP}$ problem as we will see in a moment.
To define $\mathcal{NP}$-complete problems, we first have to define the property of a problem being polynomially reducible to another problem:

Definition 5.5. *If $P, Q \in \mathcal{NP}$ and if an instance of P can be converted in polynomial time to an instance of Q, then P is polynomially reducible to Q.*

Polynomially reducible means that there exists an algorithm f with polynomial running time that transforms instances p of the problem P to instances $q = f(p)$ of the problem Q, so that the answer of the decision problem p is YES if and only if the answer of $f(p)$ is YES. So if we have an algorithm for Q, it can be applied on the problem P with an overhead that is only polynomial in the size of the instance.

Definition 5.6. *$\mathcal{NPC}$, the class of $\mathcal{NP}$-complete problems, is the subset of problems $P \in \mathcal{NP}$ such that for all $Q \in \mathcal{NP}$, Q is polynomially reducible to P.*

The remarkable thing about the set of $\mathcal{NP}$-complete problems is that it is not only non-empty, but instead there are quite a few decision problems belonging to this class. Karp (1972) showed that a large number of classic problems like the set covering problem, the problem of finding an (un-)directed Hamiltonian circuit or the knapsack problem are $\mathcal{NP}$-complete.

Proposition 5.7. *Suppose problems $P, Q \in \mathcal{NP}$.*

(i) If $Q \in \mathcal{P}$ and P is polynomially reducible to Q, then $P \in \mathcal{P}$.

(ii) If P is $\mathcal{NP}$-complete and polynomially reducible to Q, then $Q \in \mathcal{NPC}$

Proof. (i) is trivially true. To proof (ii) consider a problem $R \in \mathcal{NP}$. Since P is $\mathcal{NP}$-complete, R is polynomially reducible to P. By assumption, P is reducible to Q and thus R is polynomially reducible to Q as well. As R was an arbitrary $\mathcal{NP}$-problem, Q is $\mathcal{NP}$-complete. □

This proposition has the following important corollary.

Corollary 5.8. *If $\mathcal{P} \cap \mathcal{NPC} \neq \emptyset$, then $\mathcal{P} = \mathcal{NP}$.*

Proof. Suppose $Q \in \mathcal{P} \cap \mathcal{NPC}$ and let $R \in \mathcal{NP}$. Since $Q \in \mathcal{NPC}$, R is polynomially reducible to Q. As $Q \in \mathcal{P}$ and R is polynomially reducible to Q, *(i)* implies that $R \in \mathcal{P}$. Therefore, $\mathcal{NP} \subseteq \mathcal{P}$ and thus $\mathcal{P} = \mathcal{NP}$. □

The question whether $\mathcal{P} = \mathcal{NP}$ was first brought up by Cook (1971). In 2000, the $\mathcal{P}$ versus $\mathcal{NP}$ problem was named one of the seven Millennium Prize Problems, for which the Clay Mathematics Institute (CMI) offered a 1,000,000$ prize for the first correct solution to any of the problems. But as of today, nobody has been able to find a proof or a counter example for the $\mathcal{P}$ versus $\mathcal{NP}$ problem (Out of the seven millenium problems so far only the Poincaré conjecture has been proven). However, due to the large number of $\mathcal{NPC}$ problems for which no polynomial algorithm has been found yet, it is likely that there does not exists an algorithm with polynomial running time for any $\mathcal{NP}$-complete problem.

Definition 5.9. *An optimization problem for which the decision problem lies in $\mathcal{NPC}$ is called $\mathcal{NP}$-hard.*

One might ask now, why all this is important for us trying to solve a specific vehicle routing problem. The quite obvious answer is: The VRP is unfortunately $\mathcal{NP}$-hard. As the VRP is a generalization of the Traveling Salesman Problem (TSP), it is sufficient to show that this spezial case is $\mathcal{NP}$-hard. The TSP is the problem of finding the shortest tour to visit a given set of customers. To see that the TSP is $\mathcal{NP}$-hard, we first have to reformulate the problem. The optimization problem is associated with the following decision problem:

Is there a feasible tour x with length less or equal to k?

The problem lies obviously in $\mathcal{NP}$ since one just has to sum up the length of all tour segments to be able to proof whether or not its length is less or equal to a scalar k. As we have mentioned before, Karp (1972) showed that the Hamiltonian circuit problem (HCP) is $\mathcal{NP}$-complete. By using Proposition 5.7, it is therefore sufficient to show that the Hamiltonian circuit problem can be polynomially reduced to the TSP. The HCP asks the following question:

Given a graph $G = (V, E)$, is there a cycle which includes each node exactly once?

Note that the TSP is traditionally defined on a complete graph. The HCP can be reduced to the TSP in the following way: Set the length of all edges in E equal to 1 and the remaining ones equal to 2. As a cycle containing $|V| = n$ nodes has exactly n edges connecting these nodes, the HCP can be reduced to the TSP by setting $k = n$. Then the answer to the question whether there exists a tour of at most length k is YES if and only if there exists a Hamiltonian circuit on the Graph G. So the TSP and thus the VRP are $\mathcal{NP}$-hard.
In the following sections we are going to show different approaches to solve such difficult optimization problems by using heuristic optimization methods. They are thereby based on the Chapters 1-3 of Talbi (2009).

5.2 Common Concepts of Metaheuristics

Heuristic methods represent a family of approximate methods that are often used if optimization problems become intractable for exact algorithms. In general, heuristics can either be problem specific algorithms or an abstract sequence of steps that can be applied on any optimization problem. Such general frameworks are then called metaheuristics. Nevertheless, the single steps of a metaheuristic have to be implemented problem specifically as well. Metaheuristics themselves can be divided into two groups: Single solution based metaheuristics (S-metaheuristics) and population based metaheuristics (P-metaheuristics). S-metaheuristics start with a single solution which is then improved iteratively. Popular optimization algorithms like simulated annealing and tabu-search belong to this class of metaheuristics. P-metaheuristics on the other hand operate on a population of solutions, and at each iteration a whole new population is generated. This new population of solutions is then integrated into the current population using some selection procedures. Popular examples for this type of heuristics are evolutionary algorithms and swarm intelligence inspired algorithms like ant colony optimization. We are going to discuss single solution based metaheuristics in Section 5.3 and population based metaheuristics in Section 5.4, respectively.
Basically, there are two conflicting aspects when it comes to designing a metaheuristic: Intensification versus diversification (see Figure 6). On the one hand, once a good solution is found, one may hope to find even better solutions with similar characteristics and so it might be expedient to further explore this particular region of the search space. On the other hand, unexplored areas in the search space should be visited as well so that not only a reduced number of regions is explored.

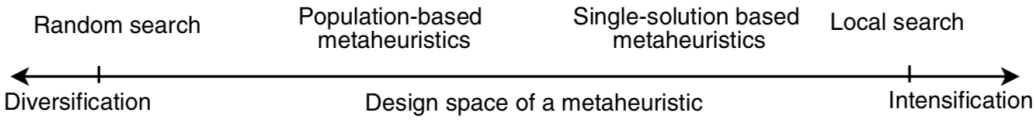

Figure 6: Two conflicting criteria in designing a metaheuristic: Intensification versus diversification *Source: Talbi, 2009, p. 24*

The extreme algorithm when it comes to diversification is a random search algorithm, where at each iteration a random solution is picked, independently of the solutions that were previously found. The most intensification focused algorithms are local search algorithms: At each iteration the algorithm picks the neighboring solution that improves the current one the most. By doing that the algorithm may get stuck in a small area of the search space, but with the advantage that this region is then explored really well. Single solution based metaheuristics are more exploitation oriented and therefore perform better when it comes to intensifying the search in local regions. In comparison, population based algorithms are more exploration oriented and thus allow a better diversification throughout the whole search space.
Even though we have just seen that there are lots of different metaheuristics, there are still some common design questions related to all iterative metaheuristics.

Representation

One of the fundamental design questions in developing a metaheuristic is how to encode a solution. The representation has to be suitable and relevant for the given optimization problem, but at the same time solutions should be easy to evaluate, and the way the search operator functions has also be taken into consideration. All this together is essential for the efficiency and effectiveness of the representation of any metaheuristic.
Although the same problem may have lots of different representations, it always has to have the following characteristics:

- **Completeness:** All solutions associated with the problem have to be represented.
- **Connexity:** There has to be a search path between any two solutions of the search space. In particular, it has to be possible to attain the global optimum from any other solution.
- **Efficiency:** The representation has to be easy to manipulate by the search operators.

For instance, for the traveling salesman problem with n cities, the order in which the cities are visited could be encoded as a permutation of size n. Then each permutation represents a unique solution and the whole solution space is represented by the set of permutations. A possible search operator would be to exchange two cities in the sequence. This search operator is known as city swap.
In addition to the representation, a mapping function is needed to transform the encoding to a particular solution of the problem. There are three possible cases when it comes to mapping the solution space with the encoding space: The traditional case is where a solution is represented by a single encoding and each encoding corresponds to a single solution (**one-to-one mapping**). So there is is no redundancy and no reduction of the original search space. It is also possible that one solution is represented by more than one encoding (**one-to-many mapping**). This will lead to a larger search space, but may have an impact on the effectiveness of the metaheuristic. Finally, the third case is that several solutions are represented by the same encoding (**many-to-one mapping**). So the representation does not include all the details of a solution, which reduces the size of the search space. This is also referred to as indirect encoding.
When using an indirect representation, a decoder is needed to derive a complete solution given by the encoding. Depending on how much information is missing, it can be computationally quite intensive to derive the solution. The advantage of indirect encoding is that the decoder can handle some or even all of the constraints so that the search operator does not have to guarantee the validity of the solution.

Objective Function

Another important aspect in designing a metaheuristic is defining the objective function. It describes the quality or the fitness of a solution by assigning a real value to it so that it is possible to order solutions in the search space according to their objective value. If a decoder is necessary, it has to be applied beforehand to generate a solution that the objective function

can then evaluate. The objective function plays an important role as it guides the search towards "good" solutions in the search space. A poorly defined objective function can lead to unacceptable solutions, independently of the metaheuristic that is used.
For some optimization problems the objective function is simply the original objective function of the problem, but often it has to be transformed to improve the convergence of the metaheuristic. For instance, this can be the case, if the objective function only takes a small number of values (the extreme case would be a binary function), and therefore the differentiation between two solutions may be difficult.
For some metaheuristics it may be sufficient to determine whether a solution is better than another solution. An example for that are population based metaheuristics in which the selection strategy only needs the relative fitness of two solutions. In that case the absolute quality of the solution has no importance.
Often the evaluation of the objective value is a quite time-consuming task. One way to improve the performance of the heuristic is to replace the objective function with a so-called proxy function that is easier to evaluate. This approach can also be helpful if an analytical objective function is not available. In that case one can replace the objective function by an approximate function that is derived by using a sample of solutions generated by physical experiments or simulations.

Constraint Handling

Next we want to address different strategies for dealing with constraints. Indeed, many optimization problems are constrained, and handling those constraints is not trivial at all.

Reject strategies: These strategies represent the simple approach of keeping only feasible solutions during the search and discarding infeasible solutions. This approach is only conceivable if the portion of infeasible solutions is relatively small, but even then it may be interesting to explore these areas as well. As optimal solutions lie in general on the boundary between feasible and infeasible regions, infeasible solutions may contain useful information to guide the search towards globally optimal solutions.

Penalizing strategies: These strategies extend the objective function by a penalty function to penalize infeasible solutions. In some cases it may be sufficient to simply penalize the number of violated constraints. However, this approach becomes useless, if the number of constraints is small or the constraints are tight. Another strategy is to penalize the amount of infeasibility. This takes into account how close a solution is to a feasible region and so the penalty can be seen as the costs of repairing the solution. Hereby the initialization of the penalty coefficients is important: If the coefficients are too small, the algorithm might converge towards an infeasible solution. On the other hand, if the coefficients are too high, the heuristic might get stuck in a suboptimal region. A possible way to overcome this problem is changing the coefficients over time. For instance, the weight of a violated constraint could increase over time so that at the

beginning of the search even highly infeasible solutions are kept, but the longer the search process continues, the stronger will be the penalties guiding the search towards a feasible solution. But even with dynamic penalty coefficients initialization is still crucial: If the coefficients increase too slowly, a longer search is needed to find feasible solutions. If they increase too fast, the algorithm may again get stuck in a suboptimal region. Adaptive penalty functions even go one step further and adapt the penalty coefficients according to information gained from previously found solutions. A possible strategy would be to decrease the penalty during the search if many feasible solutions are generated and conversely increase the penalty coefficients if many infeasible solutions are generated. Here again we have a trade-off between diversification and intensification of the search process.

Repairing strategies: As the name already suggests, repairing strategies transform infeasible solutions into feasible ones. These strategies are for instance used if some constraints are not considered by the chosen search operator and it therefore potentially generates infeasible solutions. For applying this strategy, problem specific repairing heuristics are needed, and thus the success of such strategies depends largely on the availability of such efficient heuristics.

Decoding strategies: When using indirect encodings, this type of constraint handling strategies can be applied. As we have seen before, in that case a decoder function is used to map each representation with a feasible solution. Hereby it is important that each representation corresponds to a feasible solution and each feasible solution has a representation corresponding to it. In addition, the distance between two solutions in the representation space has to be positively correlated with the distance between the corresponding feasible solutions. Otherwise it may be difficult to guide the search towards good solutions. Of course, this approach is only useful if the computational complexity of the decoder is sufficiently small.

Preserving Strategies: The last type of constraint handling strategies we are going to discuss are preserving strategies. Hereby a specific representation and a search operator are defined so that only feasible solutions will be generated during the search process. This of course requires the representation and operator to be designed specifically for the problem and therefore cannot be generalized to handle any constraints of an arbitrary optimization problem. Moreover, for some problems it may be even difficult to find a feasible solution to be able to start the search process.

5.3 Single Solution Based Metaheuristics

Single solution based metaheuristics (S-metaheuristics) perform an iterative procedure that moves from a current solution to another solution in the search space. Iteratively, a generation and replacement procedure is applied to first generate a set of new candidate solutions and then select one of these solutions to replace the current one. This iterative process continues until a certain stopping criterion is fulfilled. If the generation and replacement procedure is based only

on the current solution, we say the metaheuristic is memoryless. Otherwise, some information of the previous solutions is used for the generation of the candidates and/or the selection of the new solution. In general, there are two concepts that all S-metaheuristics have in common: The definition of the neighborhood structure and the determination of the initial solution.

Neighborhood

The definition of a neighborhood is an important aspect in designing an S-metaheuristic, as the structure of the neighborhood is crucial for the performance of the algorithm. A heuristic with a poorly defined neighborhood will be inefficient or maybe even fail to solve the problem. Formally, a neighborhood is defined via a neighborhood function.

Definition 5.10. *A neighborhood function N defined on a search space S is a mapping $N : S \to 2^S$ that assigns to each solution $s \in S$ a set of solutions $N(s) \subseteq S$.*

A solution s' is said to be a neighbor of s, if $s' \in N(s)$. A neighbor of s is generated by applying a move operator that slightly changes the solution s. The main property a neighborhood has to fulfill is locality. By locality we mean the effect on the solution when applying the move operator on the representation. Small changes in the representation have to correspond to small changes in the solution. In that case we say the neighborhood has a strong locality. Weak locality is characterized by a large effect on the solution, even for small changes made to the representation. In the extreme case of weak locality there is almost no connection between two neighboring solutions and so the search converges towards a random search.
As an example let us again consider the TSP: We have already introduced the city-swap operator that swaps two cities in the order in which the cities are visited. Besides that, another popular move operator is the so-called k-opt operator. It removes k edges of the tour and replaces them with k other edges. So in that case s' is a neighboring solution of s if the corresponding tours differ by exactly k edges. As the variation to the solution is small (provided that k is not insanely large), the neighborhood has strong locality. Indeed, the k-opt operator turns out to be a very efficient operator for the TSP.
Now that we have introduced the concept of a neighborhood, we can discuss what it means for a solution to be locally optimal.

Definition 5.11. *A solution $s \in S$ is a local optimum with respect to a neighboring function N and an objective function $f : S \to \mathbb{R}$, if $f(s) \leq f(s')$ for all $s' \in N(s)$.*

Note that for the same optimization problem, a solution may be a local optimum for a neighborhood N_1, but not regarding a different neighborhood N_2. In designing an S-metaheuristic it is often a compromise between the size and quality of a neighborhood and the computational complexity to explore it. Large neighborhoods may improve the quality of the obtained solutions as there is a larger number of solutions considered at each iteration. However, it also increases the computational time to generate and especially evaluate the larger neighborhood. The effects of a larger neighborhood are illustrated in Figure 7.

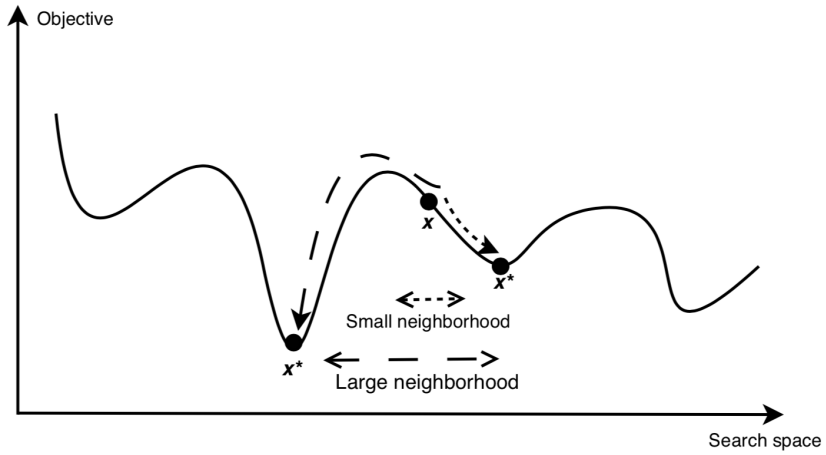

Figure 7: Larger neighborhoods may improve the quality of a search at the expense of a higher computational time. *Source: Talbi, 2009, p. 94*

Initial Solution

The second common concept of all S-metaheuristics is the determination of an initial solution. The two main strategies are a random and a greedy approach. Which one of the two leads to better results always depends on the problem at hand as well as the used heuristic. In general, there again is a trade-off between the quality of the solution and the computational time of the algorithm. For instance, a worse starting solution will have a smaller impact on the overall performance of the heuristic if the neighborhood is large.
A random initial solution may be generated quickly, but the price for that may be that it takes the heuristic pretty long to converge towards a local optimum afterwards. For some optimization problems it may be even difficult to randomly generate a feasible solution due to the high complexity of the constraints. Greedy algorithms lead in general to better initial solutions and so the number of iterations needed by the metaheuristic may be smaller. Nevertheless, a better starting solution does not necessarily lead to better local optima. For some specific problems it also may also be an option to define the initial solution beforehand.
Next we want to discuss a few of the most popular S-metaheuristics, starting with one of the simplest methods, namely local search (LS).

Local Search

The basic idea of a local search algorithm is the following: Starting with an initial solution, at each iteration a neighboring solution is chosen that improves the objective function. This process is continued until no improving solution can be found, meaning that the current solution is a local optimum. Different variations of LS can be distinguished according to the way neighboring solutions are generated (deterministic/stochastic) and the strategies used to select a new solution. Common selection strategies are the following:

- **Best improvement:** In this strategy all neighboring solutions are evaluated and afterwards the best solution, i.e., the solution that improves the objective function the most, is

chosen. As the exploration of the neighborhood is exhaustive, this strategy may be very time-consuming, especially for large neighborhoods.

- **First improvement:** This strategy consists of selecting the first solution found that improves the current solution. So the strategy involves only a partial evaluation of the neighborhood. Nevertheless in the worst case, i.e., if the current solution already is a local optimum, still the complete neighborhood is evaluated.

- **Random selection:** In this strategy, a solution is selected randomly among all neighboring solutions that improve the current one.

It has been observed that in practice on many applications the first improvement strategy leads to the same quality of solutions as the best improvement strategy while having a smaller computational time. In addition, premature convergence to a local optimum is less an issue in the first improvement strategy (Talbi, 2009, p. 124).
In general, local search is easy to design and implement and gives very quickly fairly good solutions. The big downside is, however, that the algorithm converges towards local optima. Moreover, the algorithm can be very sensitive to the initial solution for some problems: Depending on the starting region of the algorithm, the quality of the final solutions can differ drastically. Another problem is that there is no way to estimate the relative error from the global optimum, and the number of steps needed until the algorithm converges is not known beforehand. Local search works well if the number of local optima in the search space is not too large and their quality is more or less similar.
As the main disadvantage of LS is the convergence towards local optima, various alternative algorithms have been proposed to escape from those local optima. Figure 8 shows three different approaches that can be used to avoid local optima.

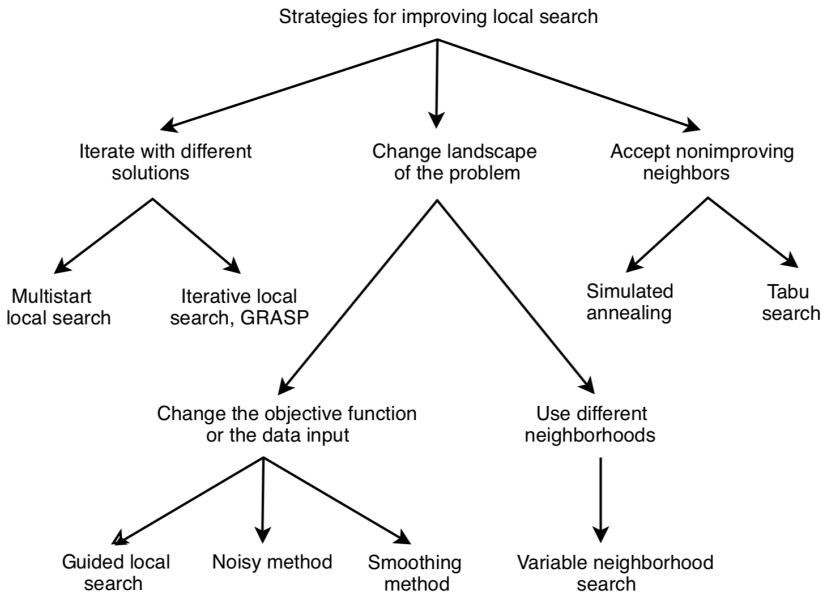

Figure 8: Strategies for escaping from local optima in LS. *Source: Talbi, 2009, p. 125*

Two of the most popular S-metaheuristics are simulated annealing and tabu-search. They both accept moves that momentarily degrade the current solution and thereby allow the algorithm to move out the basin of attraction of a local optimum. At the end of this section we want to have a closer look on those two variants of LS.

Simulated Annealing

Simulated annealing (SA) is inspired from the annealing technique used in metallurgy. When a crystalline solid is heated and then allowed to cool very slowly, it achieves the most regular possible crystal lattice configuration and thus is free of crystal defects. This corresponds also to the fact that it reaches its minimum lattice energy state. Simulated annealing algorithms now carry this type of thermodynamic behavior over to the search for global optima.
At each iteration of the algorithm, a neighboring solution is randomly generated and compared with the current solution. Improving solutions are always accepted, while a fraction of non-improving solutions are accepted as well, hoping to thereby escape a local optimum. The probability of accepting a non-improving solution s' depends on the amount of degradation $\Delta E = f(s') - f(s)$ and the temperature parameter T, which is decreasing over time. In general, this probability follows the Boltzmann distribution:

$$P(\Delta E, T) = \exp\left(-\frac{\Delta E}{T}\right).$$

The key feature of SA is that it provides a way to escape local optima by performing moves that temporarily worsen the objective function value. Those moves are also called hill-climbing moves. As the temperature parameter decreases to zero, the probability of such a hill-climbing move decreases as well and so degrading solutions are accepted less often. The algorithm stops, if the probability of accepting a non-improving solution is negligibly small, i.e., a sufficiently small final temperature T_F is reached.
So in addition to the common design concepts of S-metaheuristics like defining a neighborhood and generating an initial solution, SA has the following two algorithm specific design issues:

- The probability function enabling non-improving solutions to be selected
- The cooling schedule defining the temperature at each step of the algorithm

For a more detailed description of these aspects we refer to Chapter 2.4 in Talbi (2009).

Tabu-Search

Tabu-search (TS) is basically an extension of a local search algorithm using a best improvement strategy. The big difference is that TS allows hill-climbing moves if all neighboring solutions are non-improving. In contrast to simulated annealing, where a random neighbor is selected, TS usually explores the whole neighborhood deterministically. As in LS, if an improving neighbor is found, it replaces the current solution. If the algorithm reaches a local optimum, though, it does not stop, but instead selects the next best neighboring solution. This approach is

likely to contain cycles, as at the next iteration the algorithm might try to move back to the previous locally optimal solution. Therefore, TS memorizes the recent search trajectory and discards previously visited solutions. Note that in contrast to SA, tabu-search is not memoryless anymore. This memory of solutions or moves is called *tabu list*.
Usually not the visited solutions, but instead the applied moves are stored in the tabu list. As the list is updated at each iteration, storing all moves may become space and time consuming. Indeed, at each iteration the algorithm has to check whether a move belongs to the tabu list of already applied moves. Therefore, the tabu list usually contains only a constant number of moves.
Sometimes the tabu list may be too restrictive. By storing only the applied moves, we loose some information about the search trajectory and so a solution may be tabu even if it has not been visited before. Therefore, a so-called aspiration criterion is introduced: If a move is "good", i.e., it fulfills some predefined criteria, it is accepted even if the move is actually tabu. The admissible solutions are therefore those that are not on the tabu list or that fulfill the aspiration criteria.
So in addition to the common design concepts of S-metaheuristics, tabu-search has the following specific design issues:

- A tabu list to prevent the search trajectory from revisiting previously visited solutions.
- An aspiration criterion to be able to select good moves even if they are tabu. A common aspiration criterion is that a tabu move may be accepted, if it generates a solution that is better than any previously found solution.

The tabu list is also referred to as the short-term memory. Often, two additional mechanics are introduced to further improve the search process:

- **Medium-term memory (intensification):** The medium-term memory stores the best solutions found during the search process. The idea is that those solutions may have certain attributes that all good solutions have in common. So the search should be guided towards solutions containing these attributes.
- **Long-term memory (diversification):** The long-term memory stores information about all visited solutions to detect unvisited areas in the search space and consequently guide the search towards those areas. So some attributes of elite solutions may be discouraged to further diversify the search process.

A detailed discussion of the fundamental concepts of tabu-search can be found in Gendreau and Potvin (2010).

5.4 Population-Based Metaheuristics

Most population-based metaheuristics (P-metaheuristics) start with an initial population of solutions, although there also exist algorithms starting from a partial or empty set of solutions.

At each iteration, a new population of solutions is created (generation phase) and afterwards a selection from the current and the new population is carried out (replacement phase). Again, these two phases are called memoryless, if only the information of the current population is used to generate and select the solutions. This procedure is continued iteratively until a certain stopping criterion is fulfilled.
Most of the P-metaheuristics are inspired by natural phenomena. Popular examples are evolutionary algorithms, ant colony optimization or particle swarm optimization. These algorithms differ in the way they generate and select the new population and how they use the search memory during the search process:

- **Search memory:** The search memory represents all the information extracted and used during the search. Depending on the P-metaheurstics, the memory can be limited to the current population of solutions (e.g., in evolutionary algorithms) or updated and extended at each iteration (e.g., in ant colony algorithms).
- **Generation:** At each iteration, a new population of solutions is generated. In general, depending on the P-metaheuristics one of the following two generation strategies is used:
  - Evolution based: New solutions are generated based on attributes belonging to the current population. A variation operator thereby acts directly on the representation of the solutions.
  - Blackbox based: Here all solutions participate in constructing a shared memory, which is then used to generate new solutions. In this class of P-metaheuristics the recombination of solutions is indirect through the shared memory.
- **Selection:** The last step at each iteration is the selection of new solutions from the union of the current and the newly generated population. The traditional strategy is to simply select the generated population as the new population. However, some algorithms select the best solutions of the two sets, and in blackboard based P-metaheuristics there is no explicit selection as the new population updates the shared memory, which consequently affects the generation of the new populations.

As for S-metaheuristics, there are certain search concepts that all P-metaheuristics have in common. These are the determination of the initial population and the stopping criteria.

Initial solution

In comparison to S-metaheuristics, population based metaheuristics are usually more exploration oriented due to the large diversity of initial populations. The generation of the initial population is crucial for the effectiveness and efficiency of a P-metaheuristic because if the initial population is not well diversified, the algorithm may stop without considering solutions in certain areas of the search space. This could happen for instance if the initial population is generated using a greedy heuristic or an S-metaheuristic. This results in an initial population consisting of high quality solutions, but they may lack of diversity. Some P-metaheuristics such

as scatter search explicitly take some diversification criteria into account when generating the initial population of solutions (see Chapter 3.5 in Talbi, 2009).

Stopping criteria

The stopping criteria are usually based on some statistics on the current population or the evolution of the population over the last iterations. Often the criteria are related to the diversification of the population: If the population becomes too uniform according to a certain measure, the algorithm stops as this is a sign of stagnation of the population, and therefore a continuation of the search is useless. Similar to S-metaheuristics, there are two stopping procedures:

- **Static procedure:** In a static procedure, the end of the search may already be known at the beginning of the search process. Static stopping criteria can for instance be a fixed number of iterations or a maximum number of objective function evaluations.
- **Adaptive procedure:** In an adaptive procedure, the end of the search is determined during the search process and therefore is a priori unknown. A possible adaptive stopping criterion can be a fixed number of iterations without any improvement of the current best solution.

At the end of this section, we have a look at two examples of popular P-metaheuristics.

Evolutionary Algorithms

In the 1980s, the theory of creation and evolution of a new species, first brought up by Charles Darwin in his famous book *On the Origin of Species* (see Darwin, 1859), inspired computer scientists to develop evolutionary algorithms. These algorithms are stochastic P-metaheursitics that simulate the process of the evolution of a species. The initial population is usually generated randomly, whereby each individual of the population is an encoded version of a potential solution. The objective function assigns a fitness value to each solution, indicating its suitability to the problem. At each iteration, individuals are selected to form parents whereby individuals with better fitness are more likely to be selected. The selected individuals then generate new solutions ("offsprings") using variation operators like crossover or mutation. In the last step, by applying a replacement procedure, the individuals of the next generation are selected from the parents and the offsprings. Each iteration represents a generation, and the procedure is repeated until a stopping criterion is fulfilled.
For a detailed description of the specific search components of selection, reproduction, and replacement of the solutions in evolutionary algorithms we refer to Chapter 3.3 in Talbi (2009).

Swarm Intelligence

Swarm intelligence algorithms imitate the collective social behavior of species like ants, bees, fish or birds. The main characteristics of such algorithms are agents, cooperating by an indirect

communication medium and moving around in the decision space. As one of the most successful swarm intelligence inspired algorithms, we will have a closer look at ant colony optimization.

Ant Colony Optimization: As the name already suggests, ant colony optimization (ACO) mimics the cooperative behavior of real ants to solve optimization problems. These algorithms can be seen as multiagent systems, where each agent represents a single ant. Even though a single ant is a rather simple organism, a colony uses collective behavior to solve difficult tasks such as finding shortest paths to the food sources. Ants cannot see very well, but they manage to find a food source by following a chemical trail (pheromone) that is left on the ground by the other ants. The more pheromone is on a particular path, the more likely it is for ants to select that path. Furthermore, this chemical substance evaporates over time and the quantity left on the ground depends on the amount of food found by the ant.
In Figure 9 we see an illustration of an ant colony having their way blocked by an obstacle. At the beginning, it is equally likely for ants to choose the right or the left path. Over time, as the travel time of the right path is shorter, more pheromone is left on that path and therefore more ants select the right one. This effect is further increased by the evaporation of the pheromone on the left path. ACO mimics this principle of finding the shortest path between two points by using this simple communication mechanism.

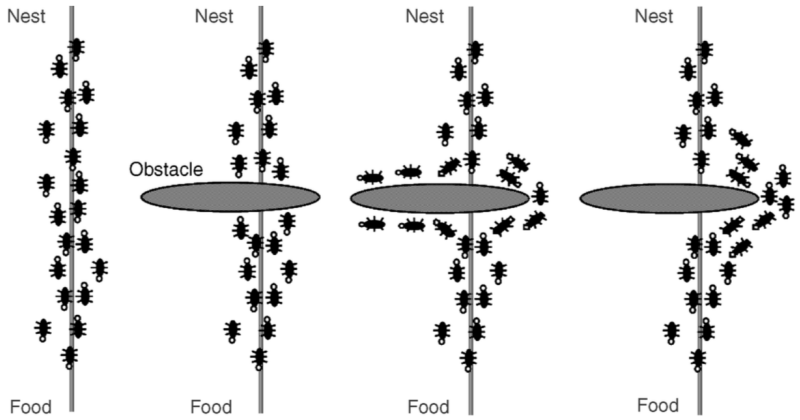

Figure 9: ACO is inspired by ant colonies searching for an optimal path between a food source and their nest. *Source: Talbi, 2009, p. 241*

In addition to the common search components of metaheuristics like the representation of solutions and the objective function, ACO has the following main design issues:

- **Pheromone information:** The pheromone model is the central component of an ACO algorithm. A vector of model parameters τ called pheromone trail parameters has to be defined, whereby the single pheromone values $\tau_i \in \tau$ should reflect relevant information when it comes to constructing new solutions for a given problem.
- **Solution construction:** The main task is to find a local heuristic that constructs a new solution guided by the the pheromone information. Therefore, an ACO algorithm is easier

to implement for problems where an efficient greedy heuristic already exists that then only needs to be adapted to the pheromone model.

- **Pheromone update:** The update of the pheromone information consists of two steps. In the evaporation phase, all pheromone trails are decreased to encourage the diversification of solutions and to avoid premature convergence toward a suboptimal solution. In the second phase, the reinforcement phase, the pheromone information is updated according to the generated solutions. Defining the reinforcement learning strategy is a further main design issue of ACO.

A more detailed description of ant colony optimization algorithms can be found in Dorigo et al. (2006). Another popular metaheuristic inspired by swarm intelligence is particle swarm optimization (PSO), which mimics the behaviours of natural organisms like a flock of birds or a school of fish, when searching for a place with enough food. For more information on that we refer to Chapter 3.6 in Talbi (2009).

6 Model for the Car Distribution in Central Europe

Now that we have discussed different types of vehicle routing problems and how to solve them using exact and heuristic algorithms, it is time that we get back to the actual topic of this thesis: Solving the VRP of the car distribution in Central Europe. We want to start with a detailed description of the problem and introduce a mixed integer linear model describing the problem. After that we present an exact algorithm as well as a heuristic method to solve the problem. Finally we evaluate and compare these two solution approaches by applying them to real-world transportation data.

6.1 Model Description

Let (V, A) be a directed graph, whereby the set of nodes V represents the different locations and the arc set A denotes the different routes connecting those locations. The locations can either be car factories or turnover points, whereby from each factory $i \in V$ a certain amount of cars q_{ij} has to be transported to a particular turnover point j. We are going to refer to arcs with a positive transport demand as a transport relation. In general, it is also possible that from a single factory multiple turnover points have to be supplied, or conversely multiple factories service the same turnover point. As we focus on the car distribution in Central Europe, there are three modes of transport available: Cars can be carried on the road by trucks, by trains via the rail network or by barges on the inland waterways. The total vehicle fleet K consists of a fixed number of vehicles of each vehicle type. For each vehicle $k \in K$ and each arc $(i, j) \in A$ leading from node i to node j, we know the travel cost c_{ijk} and whether the vehicle is available on that arc. This is denoted by the parameter $a_{ijk} \in \{0, 1\}$, which is equal to 1 if and only if vehicle k can go from location i to location j. This restriction is given by the fact that either some types of cars can only be transported by a certain transport mode or that there is no connection between the two places for that type of transport vehicle. The type of cars also affects the number of cars a vehicle can transport in a single run. As the type of car produced in a certain factory does not change, we also know the capacity b_{ijk} of vehicle k on arc (i, j). Finally, we also have the distance between two locations, which in general also depends on the type of transport vehicle. Water ways are for instance usually significantly longer than the corresponding road connection. The distance is denoted by d_{ijk} for each vehicle k and each arc (i, j). As a last restriction, each vehicle k has a maximal distance D_k it can travel within a month, which is the planning horizon we are going to consider.

According to the classification in Section 3.3, the given problem is a Pickup and Delivery Problem (PDP), as the transport requests are point-to-point requests. Furthermore, the number of cars that have to be transported on a single transport relation are in general significantly higher than the transport capacity of a single vehicle. So the transported cars have to be split between several vehicles, which corresponds to a VRP with Split Deliveries (SD). Finally, there are different types of transport vehicles, and therefore our fleet is heterogeneous (HF). All together, the given problem is a Heterogeneous Fleet Pickup and Delivery Problem with Split

Deliveries (HFPDPSD).
As the given problem is a split delivery problem, we are going to use the SDVRP model of Archetti et al. (2006) that we have already presented in Section 3.3.2. In contrast to the problem considered by Archetti et al. (2006), the given VRP has no central depot, and the transport vehicles are allowed to travel multiple times along the same arc. Therefore, we have to modify the model slightly:
Let x_{ijk} be the number of times vehicle $k \in K$ travels along arc $(i, j) \in A$. Then the objective function is given as

$$\min_{x_{ijk}} \sum_{i \in V} \sum_{j \in V} \sum_{k \in K} x_{ijk} c_{ijk}.$$

First, we have to ensure that all tours of the vehicles are meaningful. One criterion is that the number of times a vehicle travels to a node has to be equal to the number of times it leaves that node. This corresponds to the following constraints:

$$\sum_{l \in V} x_{ilk} = \sum_{l \in V} x_{lik} \qquad \forall i \in V, k \in K.$$

At the same time, tours must not contain any subtours. This is where things becomes tricky, because as there is no central depot and our vehicles may move along the same arc multiple times, we can not use the subtour elimination constraints of the model introduced by Archetti et al. (2006). At first we define a binary variable z_{ijk} that is 1 if and only if vehicle k moves at least once from point i to point j. That means,

$$z_{ijk} = \begin{cases} 1 & \text{if } x_{ijk} > 0 \\ 0 & \text{otherwise.} \end{cases}$$

To ensure that, let M be a sufficiently large number and let the following two inequalities hold:

$$\begin{aligned} x_{ijk} &\leq M z_{ijk} \qquad \forall (i, j) \in A, k \in K \\ z_{ijk} &\leq x_{ijk} \qquad \forall (i, j) \in A, k \in K. \end{aligned}$$

Next let S be the set of all closed tours. Note that a tour is a set of routes, i.e., $S \subseteq 2^A$. Analogously to the notation used in Section 3.2, let $\delta^+(s) = \{(i, j) \in A : \exists (h, i) \in s\}$ be the set of all arcs connected to an end point of an arc in $s \in S$. Furthermore, for $s \in S$ and $k \in K$ we define w_{sk} as

$$w_{sk} = \begin{cases} 1 & \text{if vehicle } k \text{ runs tour } s \\ 0 & \text{otherwise.} \end{cases}$$

A vehicle runs a tour s if and only if it moves along all arcs contained in s and no arc that leads away from that tour. So to ensure that w_{sk} is indeed 1 if and only if vehicle k runs tour s, the

following constraints have to be fulfilled

$$\sum_{(i,j)\in s} z_{ijk} - \sum_{(i,j)\in\delta^+(s)} z_{ijk} - w_{sk} \leq |s| - 1 \qquad \forall s \in S, \tag{17}$$

with $|s|$ being the number of arcs contained in s. Finally, we are able to define a subtour elimination constraint that fits our problem: Every vehicle is only allowed to run a single tour, which corresponds to

$$\sum_{s\in S} w_{sk} \leq 1 \qquad \forall k \in K.$$

Now that we have ensured that all vehicles run meaningful tours, we can concentrate on the other restrictions we stated before. Let us start with a set of constraints to make sure that all demands are fulfilled. If b_{ijk} is the transport capacity of vehicle k on arc (i,j) and q_{ij} is the number of cars that have to be transported along that arc, than it has to hold that

$$\sum_{k\in K} x_{ijk} b_{ijk} \geq q_{ij} \qquad \forall (i,j) \in A.$$

Furthermore, if D_k is the maximal travel distance of vehicle k and d_{ijk} is the distance of the connection between location i and j for vehicle k, then it has to hold that

$$\sum_{(i,j)\in A} x_{ijk} d_{ijk} \leq D_k \qquad \forall k \in K.$$

Finally, we have to make sure that vehicles only travel along arcs that they actually are allowed to. With a_{ijk} being equal to 1 if and only if vehicle k can move along the arc (i,j), the corresponding constraints are

$$a_{ijk} \leq z_{ijk} \qquad \forall (i,j) \in A, k \in K.$$

In the course of the research project IPPO this model has already been published as a part of Brunnthaller et al. (2018).
In the following two Section we are going to present the basic idea of the algorithms used to solve the model above. We start with an exact algorithm before introducing a tabu-search heuristic. Both algorithms were implemented in Matlab. The corresponding codes of the algorithms can be found in the appendices.

6.2 Core Elements of the Exact Algorithm

When solving the problem with an exact algorithm, an obvious problem is the exponential number of variables w_{sk}, as there is one for each possible tour a vehicle can run. We are going to use the approach presented in Section 3.2.1 to at first leave out all variables w_{sk} as well as the corresponding inequalities in (17). The optimization problem is then solved iteratively,

and at each iteration, if the optimal solution contains a vehicle running a certain subtour, the corresponding w_{sk} variables are identified and added to the problem together with their inequalities. In particular, this means that at each iteration all tours run by a vehicle have to be identified, and if more than one tour is found, a w_{sk} variable for each tour has to be added to the problem. The identification of the tours of a vehicle is done by the function **findSubtours**, which takes all the arcs along which the vehicles move as an input and then iterates from one visited node to the next one until all cycles are found. Cycles having a node in common are combined to a single tour afterwards, and the nodes and arcs of all tours are then returned to the main function. The implementation of the function findSubtours can be found in Appendix A.2. The second difficult task is to solve the MILP at each iteration. This is done by the MILP solver **intlinprog** from the Matlab Optimization Toolbox (see MathWorks, 2017). The solver applies different strategies to find the optimal solution, and whenever it cannot solve the problem with a certain strategy, it moves on to the next one. In total, intlinprog executes six strategies:

1. **Linear Program Preprocessing:** The algorithm tries to detect redundant variables and linear constraints and eliminates them. Furthermore, it strengthens the bounds on the variables and checks the feasibility of the model. While this initial step may take some time, it usually lowers the overall computation time and therefore potentially makes larger problems solvable.

2. **Linear Programming:** Intlinprog solves the LP relaxation to get an initial lower bound of the problem. Besides this initial relaxed problem, all lower bounds in the later branch and bound algorithm are generated by applying this relaxation technique.

3. **Mixed-Integer Program Preprocessing:** Similar to the first step, intlinprog tries to tighten the problem even further. Now the algorithm also takes the integrality restriction of the variables into account to determine whether:

   - The problem is infeasible.
   - Some bounds can be tightened.
   - Some inequalities are redundant and so can be ignored or removed.
   - Some inequalities can be strengthened.
   - Some integer variables can be fixed.

4. **Cut Generation:** At the next step, various types of integer cuts like Mixed-integer rounding cuts or Gomory cuts are generated and added to the problem formulation. These additional constraints attempt to improve the LP relaxation so that the solutions are closer to integer points.

5. **Heuristics for Finding Feasible Solutions:** The algorithm runs a number of different heuristics to find a good feasible solution and therefore a good upper bound of the problem. Currently, intlinprog uses these advanced heuristic techniques only at the root node, not during the branch and bound iterations.

6. **Branch and Bound:** Finally, a branch and bound algorithm is applied to split the problem into a sequence of subproblems in order to converge to a solution of the overall MILP. As mentioned before, intlinprog uses the LP relaxations to generate lower bounds at each node. At which node the tree should be split is determined by a predefined rule, which is passed on to the solver. Possible splitting rules are, for instance, to choose the variable with its fractional part closest to 0.5 or with the maximal corresponding absolute value in the objective function. The branch and bound algorithm continues until one of the following stopping criteria is met:

   - The algorithm exceeds a predefined maximal computation time.
   - The difference between the lower and upper bounds on the objective function is less than a predefined tolerance.
   - The number of explored nodes exceeds the predefined maximal number.
   - The number of integer feasible points exceeds the predefined maximal number.

6.3 A Tabu-Search Algorithm for the HFPDPSD

Next we want to present a tabu-search algorithm for the HFPDPSD as an alternative to the exact algorithm. The basic idea of the algorithm is based on the tabu-search algorithm of Archetti et al. (2006) for solving a SDVRP. As for the model formulation, we take the key concepts of the algorithm and adjust them to our routing problem.

Representation: The solution is encoded by a list containing an object for each available vehicle. Each of these vehicle objects consists of the tour run by the vehicle as well as the current travel distance and the travel costs of the vehicle. A tour is decoded as a list of cycles consisting of the order in which certain relations are serviced by the vehicle together with the number of units transported on each relation. The number of times the cycle is run is then simply the number of units transported divided by the capacity of the vehicle on that relation rounded up to the next integer.
At first glance, it may seem a bit odd as we kind of model a tour as a set of cycles that do not necessarily have to be connected. This is necessary, as the number a cycle is run is linked to the cycle itself and therefore a vehicle could otherwise only service all relations contained in the tour equally often. Especially in combination with the maximal travel distance restriction, it is not unlikely that a vehicle services some relations more often than the other ones. When calculating the travel distance and the travel costs we of course connect the different cycles so that the result is again a valid tour.
This is, by the way, the only part of our representation that is encoded indirectly, as we do neither store the order in which the cycles are run nor the nodes at which the cycles are connected. The problem with using this encoding is that the resulting lack of information is quite huge, as the corresponding decoding function would have to solve a TSP consisting of all relations serviced by the vehicle. Therefore we do not calculate the travel costs and distance

exactly, but instead use a greedy heuristic as a proxy function. The idea behind this is that if the greedy connection is suboptimal, the algorithm should construct a new cycle that connects the other cycles in an optimal way.

Constraint handling: When it comes to constraint handling, we use a combination of a penalization and a preserving strategy. Most of the constraints are handled by the move operator we are going to introduce in a moment. The only constraint the operator does not take care of is the maximal travel distance restriction. This is due to the fact that this restriction makes it potentially quite difficult to find a feasible starting solution. The amount of violation is multiplied by a penalty coefficient and added to the objective function, which is for this problem simply the total transport cost. The search starts with a penalty coefficient of zero that increases whenever the search gets stuck in a region of infeasible solutions for a certain number of iterations. In doing so, we allow the search trajectory to pass regions of the search space consisting of infeasible solution, hoping that it will move over time to a region of good feasible solutions.

Neighborhood: The move operator that defines the neighborhood structure of the search space is defined via two functions: First, the function **order** (see Appendix B.2) generates a list of savings, if a single run of a given relation is removed from a cycle belonging to a tour of a vehicle. This list is then ordered by the transport cost reduction per unit removed from the cycle. So the first entry of the list is the cycle with the highest cost reduction, if the relation is partly or completely removed from it. After that, the function **bestNeighbor** (see Appendix B.3) looks for the cycle to which adding the given relation results in the highest overall cost reduction. To improve the performance of the algorithm, not all possible combinations are considered: The algorithm starts with the first entry of the savings list and moves the optimal number of units transported from one cycle to the other. Once this optimal number is found, this number is fixed and it continues with the second entry of the list searching from the optimal number of units to be moved from one cycle to the other. This procedure is executed for all relations so that at the end we get the optimal number of units transported at a certain relation to be moved from one or many cycles to a single other cycle.
If any number of units belonging to a certain relation was removed from a cycle, it is tabu to add any number of units belonging to that relation again to that cycle for a given number of iterations. Analogously, it is forbidden to remove any units of a certain relation from a cycle for a given number of iterations, if units of that relation had been added to that cycle recently. So as described in Section 5.3, we do not consider the particular transfer of units from one cycle to another to be tabu, but rather adding or removing units of a certain relation to or from a certain cycle.
The implementation of the main function of the tabu-search algorithm is called **TabuVRP** and can be found in the Appendix B.1. It generates the initial solution using a greedy heuristic, checks the stopping criteria after each tabu-search iteration and potentially adapts the penalty coefficient according to the criteria described above.

6.4 Test Scenarios

Next we want to compare the performance of the two algorithms introduced above. As a test scenario we will take the monthly transport demand of 10 real-world transport relations throughout Central Europe for the years 2014 and 2015. This also was the data used in the IPPO research project to test the developed algorithm for predicting future transport demands and optimizing the fleet composition based on that information. Figure 10 shows the number of cars that need to be transported on the given relations. We see that the demands fluctuate a lot: While there is usually a peak around March as the business year of many companies ends around that time, there is typically also a low during the summer marking the company holidays of the car manufacturers. Thus we see that only because it may be profitable to use a train as a transport vehicle for a certain month, it is not given that it is true for all months.

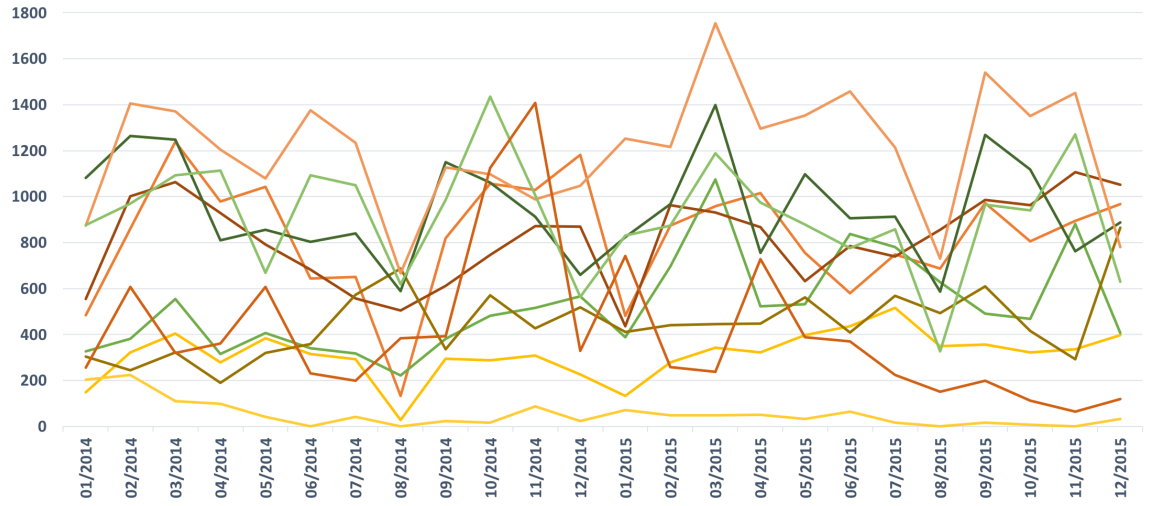

Figure 10: Transport demand of the 10 test relations over time

The transport distances were calculated by using the online geographic informations systems Google Maps and ecotransit.org. All the other parameters needed to evaluate the algorithms are based on the doctoral thesis of Pascher (2017), who developed a detailed model to calculate the number of cars a transport vehicle can carry as well as the resulting transportation costs depending on the type and number of cars transported by the vehicle.

Depending on the size of the cars, a single truck can transport between 6 and 11 cars. In our test scenarios, only mid sized cars have to be transported and so the transport capacity of a single truck lies between 7 and 9 cars. A block train is usually allowed to have a length of up to 600 meters, which is equivalent to 18 railcars. A single-deck railcar can carry between 5 and 8 cars. A double-deck railcar can carry about twice as many, but is not available for all transport relations or car types. On the relations of our test scenarios, trains have a capacity between 113 and 216 cars. For barges it is not that easy to generally state a fixed transport capacity, as it largely depends on the type of barge that is available or can be used on a certain relation. In our test scenarios, the capacity varies between 236 and 294 cars.

As we consider transport requests on a monthly basis, we have to adjust the transport capacity

for the large vehicles: If for instance only 200 cars are produced at a certain location per month, it is obviously unrealistic that a single train can pickup all the cars at once. Therefore, we reduce the transport capacity according to the number of cars produced per day multiplied by the number of days it takes the vehicle to perform a round trip from the source to the sink and back again. Within the research project IPPO we were told by the traffic managers of Hödlmayr International AG that they use the same approach for their strategical planning.
The calculation of the transportation costs takes various fixed and variable cost factors into account. Fixed costs accrue independently of the use of the vehicles. In order to be able to account for these costs, they are added to the single transports proportionally to the duration of the transport. Examples for fixed costs are the fixed depreciation of the vehicle, taxes, insurance or personnel costs. Variable costs depend on the distance traveled by the vehicle. So for trucks these are, for instance, the variable part of the depreciation, maintenance costs, tires or toll fees. The transport costs for trains and barges are calculated similarly. In contrast to the costs of trucks it is also possible to lease or rent these vehicles, which is also considered in the cost calculation. Furthermore, there are some additional cost factors like infrastructure costs for the rail system or costs for the change of traction of a train. A detailed description of the cost calculation can be found in Chapter 5.7 of Pascher (2017).

6.5 Computational Results

Now that we have described the in overall 24 test scenarios, we want to apply the two algorithms to them and compare their performance. Previous tests have already shown that - as expected - the performance of the exact algorithm becomes worse the more vehicles and locations are considered in the problem. So by setting the number of vehicles to a large number, we insure that the problem has a feasible solution, but the exact algorithm would not be able to find a solution. Therefore, we calculated the travel distance of the optimal solution using only trucks with only one eighth of the transport demand. This can easily be done with the exact algorithm. The resulting travel distance was multiplied by eight again and then divided by the maximal travel distance of a truck giving an estimate of the number of trucks needed to satisfy all transport requests. Finally we increased the number of trucks by 10% so that potentially the optimal solution is not the only feasible one.
At first we want to start off with calculating the optimal routes when using only trucks and for only half of the test size, i.e., we divided the transport demand and the number of trucks available by two. Each algorithm was given 300 seconds to find the optimal routing of the vehicles. The results can be found in Table 4.
The first column contains the period of the test scenarios, followed by two columns with the optimal transportation costs and the run time of the exact algorithm. The fourth column “optimal solution found” indicates whether the exact algorithm converged within the given time limit. If it was able to initially solve the problem, but the solution contains at least one subtour (keep in mind that we add the subtour elimination constraints only iteratively), we still state the resulting transportation costs. In that case, the costs are only a lower bound as it is the

test period	optimal costs exact algorithm	run time exact algorithm	optimal solution found	optimal costs tabu-search	run time tabu-search	optimality gap
01/2014	566480 €	54.752 sec	Yes	600590 €	302.96 sec	6.02%
02/2014	840650 €	301.25 sec	No	—	310.53 sec	—
03/2014	864700 €	300.40 sec	No	908060 €	305.71 sec	$\leq 5.01\%$
04/2014	753950 €	301.03 sec	No	786560 €	304.43 sec	$\leq 4.33\%$
05/2014	738120 €	302.56 sec	No	—	300.60 sec	—
06/2014	654840 €	300.24 sec	No	674880 €	313.35 sec	$\leq 3.06\%$
07/2014	635730 €	33.829 sec	Yes	693400 €	314.28 sec	9.07%
08/2014	382930 €	46.786 sec	Yes	388030 €	306.49 sec	1.33%
09/2014	698820 €	298.74 sec	Yes	—	304.33 sec	—
10/2014	977170 €	300.03 sec	No	—	305.22 sec	—
11/2014	956260 €	300.05 sec	No	—	316.47 sec	—
12/2014	683800 €	300.57 sec	No	699550 €	305.50 sec	$\leq 2.3\%$
01/2015	643410 €	13.616 sec	Yes	691940 €	310.18 sec	7.54%
02/2015	727700 €	16.939 sec	Yes	751260 €	301.28 sec	3.24%
03/2015	909540 €	289.94 sec	Yes	—	309.55 sec	—
04/2015	818620 €	300.05 sec	No	—	302.31 sec	—
05/2015	702100 €	50.654 sec	Yes	736410 €	309.84 sec	4.89%
06/2015	705640 €	300.32 sec	No	737720 €	304.91 sec	$\leq 4.55\%$
07/2015	680700 €	61.339 sec	Yes	702510 €	308.13 sec	3.20%
08/2015	504240 €	6.5204 sec	Yes	511380 €	300.91 sec	1.42%
09/2015	780040 €	300.05 sec	No	—	300.15 sec	—
10/2015	699280 €	161.86 sec	Yes	—	310.81 sec	—
11/2015	784870 €	12.199 sec	Yes	814360 €	303.58 sec	3.76%
12/2015	636540 €	302.36 sec	No	650300 €	306.02 sec	$\leq 2.16\%$

Table 4: Half of the transportation demand with only trucks and a maximal computation time of 300 seconds.

solution of a relaxation of the problem. The next two columns contain the optimal costs and the run time of the tabu-search algorithm. As the algorithm checks the elapsed time only after each tabu iteration, it usually runs a few seconds longer than the maximal computation time, but since the improvement of a single move is only marginally, this does not have a big effect on the results. The last column of the table states the optimality gap, i.e., how far the tabu-search algorithm is away from the optimal solution. If the exact algorithm only found a lower bound of the optimal solution, we only know an upper bound of the difference to the optimal solution. We see that the exact algorithm was able to solve 12 out of the 24 test instances within the given 300 seconds. For 8 of those instances it took less than a minute to find the optimal solution, for the instance of August 2015 even only astonishing 6.5 seconds. Note that it is quite reasonable that the algorithm performs better for the summer months, as the transport demand is significantly smaller for those periods due to the company holidays of the manufacturers, also corresponding to a smaller number of trucks considered in the problem.

The tabu-search algorithm was able to find a feasible solution for 15 of the 24 instances with optimality gaps ranging from 1.33% to 9.07%. Remarkably, the solutions of only 4 instances have costs that are more than 5% greater than the optimal routing costs.

Next, we take the original transportation demand, but again consider only trucks as transport vehicles. As the size of the instance doubles, we also increase the maximal computation time from 300 to 600 seconds. The test results can be found in Table 5.

test period	optimal costs exact algorithm	run time exact algorithm	optimal solution found	optimal costs tabu-search	run time tabu-search	optimality gap
01/2014	1128400 €	600.17 sec	No	—	601.74 sec	—
02/2014	1667400 €	600.05 sec	No	—	608.84 sec	—
03/2014	1721700 €	600.86 sec	No	—	631.72 sec	—
04/2014	1497900 €	600.09 sec	No	1583700 €	605.36 sec	$\leq 5.73\%$
05/2014	1480900 €	605.57 sec	Yes	1550900 €	629.30 sec	4.73%
06/2014	1297100 €	600.06 sec	No	1376800 €	625.67 sec	$\leq 6.14\%$
07/2014	1272300 €	601.15 sec	No	1362500 €	615.25 sec	$\leq 7.09\%$
08/2014	764460 €	600.07 sec	No	783760 €	603.92 sec	$\leq 2.52\%$
09/2014	1392500 €	601.57 sec	No	—	628.86 sec	—
10/2014	1945100 €	600.07 sec	No	—	621.54 sec	—
11/2014	1914500 €	600.08 sec	No	2033100 €	608.81 sec	$\leq 6.19\%$
12/2014	1341100 €	600.99 sec	No	1377100 €	600.25 sec	$\leq 2.68\%$
01/2015	1272600 €	600.10 sec	No	—	618.12 sec	—
02/2015	1455500 €	600.06 sec	No	1536200 €	611.60 sec	$\leq 5.54\%$
03/2015	1806300 €	601.36 sec	No	—	626.93 sec	—
04/2015	1613400 €	600.05 sec	No	1726100 €	631.37 sec	$\leq 6.99\%$
05/2015	1404200 €	600.11 sec	No	—	624.42 sec	—
06/2015	1407700 €	600.05 sec	No	1512300 €	606.24 sec	$\leq 7.43\%$
07/2015	1361000 €	600.11 sec	No	—	616.85 sec	—
08/2015	1000300 €	600.10 sec	No	1017500 €	609.40 sec	$\leq 1.72\%$
09/2015	1553300 €	600.06 sec	No	—	622.01 sec	—
10/2015	1384800 €	600.11 sec	No	—	632.17 sec	—
11/2015	1574500 €	600.06 sec	No	1650500 €	607.52 sec	$\leq 4.83\%$
12/2015	1259700 €	254.99 sec	Yes	1290000 €	601.78 sec	2.41%

Table 5: Full transportation demand with only trucks and a maximal computation time of 600 seconds.

The exact algorithm was only able to find the optimal solution for two test scenarios, namely for May 2014 and December 2015. We see that the algorithm reaches its limits as the problem size increases. In contrast to that, the tabu-search heuristic was still able to find feasible solutions for 13 out of the 24 test scenarios, and the optimality gaps are again within an acceptable range. For the two scenarios for which we know the optimal solution, the heuristic is only 4.73% respectively 2.41% away from the optimal solution. Both algorithms might be able to find solutions for more instances if they were given some additional computation time, but nevertheless it is obvious that the exact algorithm is clearly outperformed by the heuristic at these larger problem instances.

As a last test we add two trains and a barge to the previous scenarios to test the performance of the algorithms with a heterogeneous fleet. For all but two transport relations, it is possible to transport cars by train or by barge. This restriction is not given as there would be no connection for that mean of transport, but rather that the transport requests occur pretty spontaneously on these relations and therefore it is not possible to service them by the less flexible transport vehicles train and barge. The results are shown in Table 6.

Interestingly, neither the solutions of the exact algorithm nor the tabu-search heuristic contain a barge for any test instance. On the other hand, this result is not really surprising as Pascher (2017) has already shown that a barge is only very seldom cost efficient when it comes to transporting cars. This is due to the fact that cars in comparison to bulk cargo or containers

test period	optimal costs exact algorithm	run time exact algorithm	optimal solution found	optimal costs tabu-search	run time tabu-search	optimality gap
01/2014	1086100 €	352.50 sec	Yes	1169400 €	602.54 sec	7.67%
02/2014	—	600.22 sec	No	1486900 €	606.88 sec	—
03/2014	1411800 €	600.20 sec	No	1457200 €	624.62 sec	$\leq$ 3.22%
04/2014	1344100 €	600.13 sec	No	1396500 €	612.73 sec	$\leq$ 3.90%
05/2014	—	600.12 sec	No	1404700 €	607.90 sec	—
06/2014	—	600.46 sec	No	1267300 €	618.85 sec	—
07/2014	1221700 €	600.16 sec	No	1274800 €	618.80 sec	$\leq$ 4.35%
08/2014	763040 €	600.08 sec	No	829010 €	585.79 sec	$\leq$ 8.65%
09/2014	—	600.28 sec	No	1297200 €	608.90 sec	—
10/2014	1675200 €	600.18 sec	No	1733900 €	603.60 sec	$\leq$ 3.50%
11/2014	—	600.24 sec	No	—	622.58 sec	—
12/2014	1204300 €	51.63 sec	Yes	1253400 €	615.52 sec	4.08%
01/2015	1274200 €	600.24 sec	No	1361700 €	612.58 sec	$\leq$ 6.87%
02/2015	—	600.2 sec	No	1276500 €	604.49 sec	—
03/2015	—	600.27 sec	No	1594300 €	611.51 sec	—
04/2015	—	600.27 sec	No	1549500 €	601.60 sec	—
05/2015	1310000 €	155.68 sec	Yes	1345100 €	603.58 sec	2.68%
06/2015	1361300 €	600.17 sec	No	1406900 €	613.50 sec	$\leq$ 3.35%
07/2015	—	600.14 sec	No	1376000 €	612.62 sec	—
08/2015	967960 €	25.75 sec	Yes	991380 €	619.04 sec	2.42%
09/2015	1335400 €	600.18 sec	No	1383200 €	614.38 sec	$\leq$ 3.58%
10/2015	1248900 €	600.14 sec	No	1270400 €	617.16 sec	$\leq$ 1.72%
11/2015	1333700 €	601.23 sec	No	1415500 €	600.51 sec	$\leq$ 6.13%
12/2015	1134800 €	600.13 sec	No	1144000 €	608.70 sec	$\leq$ 0.81%

Table 6: Full transportation demand with a heterogeneous fleet and a maximal computation time of 600 seconds.

have only very little weight in relation to their volume and are therefore not suitable for the transport on waterways.

The performance of the tabu-search algorithm improved significantly as it was easier for the heuristic to find feasible solutions due to the additional transport capacities. The results of the exact algorithm on the other hand are still not satisfying: Although the number of instances that the exact algorithm was able to solve increased slightly from two to four, it could not find any solution for nine instances, even completely without any subtour elimination constraints. The tabu-search heuristic was able to find a feasible solution for 23 out of the 24 test instances with a median optimality gap of only 3.58%. It is also worth noting that for the instance of August 2014 it is the first and only time that the tabu-search algorithm converged, for all the other instances it reached the time limit beforehand.

7 Conclusion

In this thesis, we first described the current situation of the freight transport market. It is characterized by an increasing volatility of the transport requests due to an increasing globalization of the manufacturing processes as well as different economic situations in the countries being part of the production chain. Consequently, trucks are the most used transport vehicles due to their higher flexibility compared to trains or barges. Conflicting to that, we also showed that especially trains are significantly more environmentally friendly in comparison to trucks. To evaluate the impact of using alternative transport modes, we discussed various types of vehicle routing problems and how these can be modeled mathematically as mixed integer linear programs. After that we presented different ways to solve the resulting problems with exact algorithms and how to find reasonably good solutions by applying heuristic optimization methods. Based on that, we derived a MILP model describing the VRP of the car distribution in Central Europe for which we then developed and implemented an exact optimization algorithm as well as a tabu-search heuristic. Finally, the algorithms were applied to 24 real-world test instances to evaluate and compare their performance.

Both algorithms were able to solve the test scenarios to a certain extent, whereby with an increasing size of the problem instances, the tabu-search heuristic clearly outperformed the exact algorithm. The heuristic was able to deliver feasible solutions with an optimality gap of less than 5% for a large number of instances. Especially if the vehicle fleet is large enough so that finding a feasible solution is not too difficult, the heuristic algorithm really stands out.

The big advantage of the exact algorithm is that it proves the optimality of the solution or at least gives an estimation of how close the current solution is to the optimal one. This is something the tabu-search algorithm alone is not able to do. Another downside of the tabu-search algorithm is the rather slow move operator, as it evaluates the impact of all relations added to any tour of a vehicle. This is one of the reasons why the algorithm only very rarely comes to the point of converging to a local optimum so that it has to perform a tabu-search typical up-hill move.

Even though the tabu-search heuristic performed quite well, it may be worth investigating whether the move operator could be sped up in order to be able to optimize even bigger instances. It could also be interesting to implement medium-term and long-term memories to further intensify and especially diversify the search process. Furthermore, the model does not take into account the transport from the train station or the port to the final destination, as this was not relevant for the given example case of the project IPPO. This might be an interesting aspect to add to the model as the transport costs occurring on the “last mile” can be quite substantial.

Appendices

The appendices contain the core pieces of the Matlab implementation of the exact and heuristic algorithms presented in Chapter 6.

A Matlab-Code of the Exact Algorithm

A.1 Function SolveVRP

The function *SolveVRP* is the main function of the exact optimization algorithm. It takes all relevant parameters describing the problem instance as input parameters and returns the minimal transportation costs as well as the exact routing of all vehicles.

```
1  function [costopt,x_opt,z_opt,opt_tours] = ...
       SolveVRP(cost,avail,capacity,distance_limit,demand,distance,Tmax)
2
3  %cost numArc x numVeh matrix; cost(i,j) = cost of vehicle j traveling along arc i
4  %avail numArc x numVeh matrix; avail(i,j) = 1 if vehicle j may travel along arc i
5  %capacity numArc x numVeh matrix; capacity(i,j) = number of units vehicle j can ...
       transport on arc i
6  %distance_limit numVeh x 1 vector; distance_limit(i) = maximal travel distance ...
       of vehicle i
7  %demand numArc x 1 vector; demand(i) = number of units that have to be ...
       transported on arc i
8  %distance numArc x numVeh vector; distance(i,j) = distance of arc i for vehicle j
9  %Tmax 1x1 double; maximal computation time
10
11 c = clock; % starting time
12
13 % Define some constants and parameters
14 % big M: maximal number a vehicle could travel along an arc
15 M = ceil(max(demand)/min(min(capacity(capacity>0))));
16 TolInt = 1e-5;
17 RelTolGap = 1e-2;
18 opts = optimoptions('intlinprog','Display','off','TolInteger',TolInt, ...
       'RelativeGapTolerance',RelTolGap,'CutGeneration','intermediate', ...
       'HeuristicsMaxNodes',200,'Heuristics','rss','IntegerPreprocess','advanced');
19 numNodes = 0.5*(1+sqrt(4*size(cost,1)+1));
20 numVeh = size(cost,2);
21 numArc = size(cost,1);
22 numVar = 2 * numArc * numVeh;
23 idx_z = numArc*numVeh; %column index, after which the first z variable is
24
25 % Eliminate irrelevant arcs to improve computational performance
26 isSink = zeros(numNodes,1);
27 isSource = zeros(numNodes,1);
28
29 for n=1:numNodes
```

```
30      % get the indices of the arcs starting and ending at the node
31      arcs_in = getIdx(1:numNodes,n,1,numNodes);
32      arcs_out = getIdx(n,1:numNodes,1,numNodes);
33
34      % set isSink to 1 if there is a positive demand on an arc to the node
35      % set isSource to 1 if there is a positive demand on an arc from the node
36      if any(demand(arcs_in)>0) && any(demand(arcs_out)>0)
37          disp(['Error: Node ',n,' is source and sink.']);
38          return;
39      elseif any(demand(arcs_in)>0)
40          isSink(n) = 1;
41      elseif any(demand(arcs_out)>0)
42          isSource(n) = 1;
43      end
44  end
45
46  % vehicles always travel from a source to the corresponding sink and from
47  % a sink to a source -> set avaialability of all other arcs to 0
48  for n=1:numNodes
49      if isSink(n) == 1
50          arcs = getIdx(n,find(isSink),1,numNodes);
51          avail(arcs,:) = 0;
52      end
53
54      if isSource(n) == 1
55          arcs = getIdx(n,1:numNodes,1,numNodes);
56          avail(arcs(demand(arcs) == 0),:) = 0;
57      end
58
59      if isSink(n)+isSource(n) == 0
60          arcs = [getIdx(n,1:numNodes,1,numNodes);getIdx(1:numNodes,n,1,numNodes)];
61          avail(arcs,:) = 0;
62      end
63  end
64
65  % transform parameter matrices to column vectors
66  cost = reshape(cost,numel(cost),1);
67  avail = reshape(avail,numel(avail),1);
68  distance = reshape(distance,numel(distance),1);
69
70  cost = [cost;zeros(size(cost,1),1)]; %z variables have cost 0
71
72  % flow constraint; number of trips to the node = number of trips to the
73  % node for all nodes and vehicles
74  Aeq = spalloc(numNodes*numVeh,numVar,numArc*numVeh); % allocate sparse matrix
75  for i = 1:numNodes
76      for j = 1:numNodes
77          for k = 1:numVeh
78              if i~=j
79                  % set all arcs leading to node i to 1
```

```
                Aeq(i+(k-1)*numNodes,getIdx(j,i,k,numNodes)) = 1;
                % set all arcs ledaing from node i to -1
                Aeq(i+(k-1)*numNodes,getIdx(i,j,k,numNodes)) = -1;
            end
        end
    end
end

beq = zeros(numNodes*numVeh,1);


% capacity constraint; all demands have to be fulfilled
A = [];

for k=1:numVeh
    A = [A,sparse(diag(-capacity(:,k)))];
end

A = [A,zeros(size(A))]; % add matrix of zeros for the z variables
b = -demand;

%distance constraint; vehicles must not exceed their maximum travel distance
for k=1:numVeh
    A = [A;sparse(ones(numArc,1),getIdx(1,2,k,numNodes):getIdx(numNodes, ...
        numNodes-1,k,numNodes),distance(getIdx(1,2,k,numNodes):getIdx(numNodes, ...
        numNodes-1,k,numNodes)),1,numVar)];
end
b = [b;distance_limit];

% define z variables
rowidx_tmp = size(A,1);

A = [A;spalloc(2*numArc*numVeh,numVar,numVeh*numArc*4)];
b = [b;zeros(2*numArc*numVeh,1)];

for i=1:numNodes
    for j=1:numNodes
        if i~=j
            for k=1:numVeh
                A(rowidx_tmp + getIdx(i,j,k,numNodes), getIdx(i,j,k,numNodes)) ...
                    = 1;
                A(rowidx_tmp + getIdx(i,j,k,numNodes), idx_z + ...
                    getIdx(i,j,k,numNodes)) = -M;
                A(rowidx_tmp + numArc*numVeh + getIdx(i,j,k,numNodes), ...
                    getIdx(i,j,k,numNodes)) = -1;
                A(rowidx_tmp + numArc*numVeh + getIdx(i,j,k,numNodes), idx_z + ...
                    getIdx(i,j,k,numNodes)) = 1;
            end
        end
    end
```

```
124 end
125
126 % subtour elemination constraints (SEC)
127 idx_cycle = size(A,1); % rowindex where the SEC start
128 A = [A;zeros(numVeh,size(A,2))];
129 b = [b;ones(numVeh,1)];
130
131 % define some additional parameters
132 intcon = 1:numVar; % index of integer variables
133 lb = zeros(numVar,1); % lower bounds
134 ub = [Inf.*ones(numVar/2,1);avail]; % upper bounds
135
136 multiple_tours = true;
137
138 while ((multiple_tours == true) && etime(clock,c) < Tmax)
139
140     % set maximal computation time
141     opts = optimoptions(opts,'MaxTime',max([0,Tmax-etime(clock,c)]));
142
143     % solve MILP using intlinprog
144     [x_opt,costopt,exitflag] = intlinprog(cost,intcon,A,b,Aeq,beq,lb,ub,opts);
145
146     if exitflag==-2
147         disp('no integer solution found.');
148         return
149
150     elseif exitflag==0
151         disp('intlinprog stopped prematurely. No integer feasible point found.');
152         return
153
154     end
155
156     % eliminate subtours
157     multiple_tours = false;
158
159     z_opt = round(x_opt((idx_z+1):numVar));
160
161     for k=1:numVeh
162
163         % findSubtours returns a cell containing all tours travelled by
164         % vehicle k; if more than 1 tour is found then vehicle k has a subtour
165         [sub_nodes,sub_arcs] = findSubtours(z_opt,k,numNodes);
166         num_tours = length(sub_nodes);
167
168         % add SEC, if vehicle has more than 1 tour
169         if num_tours>1
170             multiple_tours = true;
171
172             % allocate space for additional variables
173             extra_var = num_tours;
```

```
174             var_old = size(cost,1);
175             cost = [cost; zeros(extra_var,1)];
176             intcon = [intcon,var_old+(1:extra_var)];
177             A = [A,zeros(size(A,1),extra_var)];
178             Aeq = [Aeq,zeros(size(Aeq,1),extra_var)];
179             lb = [lb;zeros(extra_var,1)];
180             ub = [ub;ones(extra_var,1)];
181
182             A(idx_cycle+k,(end-extra_var+1):end) = ones(extra_var,1);
183
184             % define and add new SEC
185             for j=1:num_tours
186                 arcs_out = [];
187                 for sub_tour=sub_nodes{j}
188                     arcs_out = getIdx(sub_tour,1:numNodes,k,numNodes);
189                 end
190
191                 new_constr = ...
                        sparse(ones(size(sub_arcs{j},1)+size(arcs_out,1)+1,1), ...
                        [idx_z+sub_arcs{j};idx_z+arcs_out;var_old+j], ...
                        [ones(size(sub_arcs{j},1),1);-ones(size(arcs_out));-1], ...
                        1,var_old+extra_var);
192
193                 if ~any(ismember(A(idx_cycle+numVeh+1:end,1:numVar), ...
                        new_constr(1:numVar),'rows'))
194                     A = [A;new_constr];
195                     b = [b;size(sub_arcs{j},1)-1];
196                 end
197             end
198         end
199
200     end
201
202 end
203
204 z_opt = round(x_opt((idx_z+1):numVar));
205 x_opt = x_opt(1:idx_z);
206
207 for k=1:numVeh
208     opt_tours{k} = findSubtours(z_opt,k,numNodes);
209 end
210
211 end
```

A.2 Function findSubtours

As described in Section 6.2, in order to deal with the exponentially number of subtour elimination constraints, they are added only iteratively to the problem. To find out which one has to be added to the model, the function *findSubtours* returns for a given solution all tours run by

a defined vehicle, i.e., if the function returns more than one tour, the vehicle runs a subtour.

```
function [cycle_nodes,cycle_arcs] = findSubtours(z_opt,veh,numNodes)

numArc = numNodes*(numNodes-1);

% get arcs that are visited by the vehicle
visited_arcs = find(z_opt(((veh-1)*numArc+1):(veh*numArc)));
unvisited_arcs = zeros(numArc,1);
unvisited_arcs(visited_arcs) = 1;

% allocate a cell for all cycles and an index vector for all the cycles that
% are not closed yet
cycles = {};
open_cycles = [];

% repeat until all visited arcs have been assigned to cycles
while sum(unvisited_arcs)>0

    % start with the first arc that has not been considered yet
    starting_arc = find(unvisited_arcs,1);
    unvisited_arcs(starting_arc) = 0;

    % create a new cycle
    cycles{end+1} = getNodes(starting_arc,numNodes)';
    open_cycles = [open_cycles;1];

    % repeat until all new cycles are closed
    while sum(open_cycles) > 0
        for i=find(open_cycles)'

            % find all visited arcs leaving from the last node of the cycle
            curr_cycle = cycles{i};
            end_node = curr_cycle(end);
            arcs_out = ...
                intersect(visited_arcs,getIdx(end_node,1:numNodes,1,numNodes));

            % for each of these arcs, a new cycle is created; the original
            % one is deleted at the end of the loop
            for j=1:size(arcs_out,1)

                % get the node that should be added to the cycle
                node_to_add = getNodes(arcs_out(j),numNodes,2);

                % if the cycle does not become closed by adding the new
                % node, create a new cycle
                if any(curr_cycle==node_to_add)
                    cycles{end+1} = [cycles{i};node_to_add];
                    open_cycles = [open_cycles;1];
                else
```

```
48                      % else check whether the cycle has already been found; if not,
49                      % store the new cycle at the end of the cell and set the open
50                      % cycle index to 0
51                      new_cycle = curr_cycle(find(curr_cycle==node_to_add,1):end);
52                      if ~any(cellfun(@(x)isequal(sort(x),sort(new_cycle)), ...
                            cycles(1:(i-1))))
53                          cycles{end+1} = curr_cycle( ...
                                find(curr_cycle==node_to_add,1):end);
54                          open_cycles = [open_cycles;0];
55                      end
56                  end
57              end
58
59              % delete the original cycle
60              cycles{i} = {};
61              open_cycles(i) = 0;
62
63              % set the index of the considered arcs to 0
64              unvisited_arcs(arcs_out) = 0;
65          end
66      end
67  end
68
69  % delete all empty entries of the cell
70  cycle_nodes = cycles(~cellfun('isempty',cycles));
71  cycle_arcs = {};
72
73  % derive the arcs corresponding to the nodes of the cycles
74  for i=1:size(cycle_nodes,2)
75      cycle_arcs{i} = [];
76      curr_cycle = cycle_nodes{i};
77      for j=1:(size(curr_cycle,1)-1)
78          cycle_arcs{i} = ...
                [cycle_arcs{i};getIdx(curr_cycle(j),curr_cycle(j+1),veh,numNodes)];
79      end
80      cycle_arcs{i} = ...
            [cycle_arcs{i};getIdx(curr_cycle(j+1),curr_cycle(1),veh,numNodes)];
81  end
82
83  % as a tour may consist of several cycles, all cycles with a common node have
84  % to be merged to a single tour
85  for n=1:numNodes
86
87      % determine all cycles that contain node n
88      idxs=[];
89      for i=1:size(cycle_nodes,2)
90          if any(cycle_nodes{i} == n)
91              idxs=[idxs,i];
92          end
93      end
```

```
94
95      % if more than one cycle contains node n, merge the cycles
96      if size(idxs,2) > 1
97          cycle_nodes{idxs(1)} = unique(cat(1, cycle_nodes{idxs}));
98          cycle_nodes(idxs(2:end)) = [];
99          cycle_arcs{idxs(1)} = unique(cat(1, cycle_arcs{idxs}));
100         cycle_arcs(idxs(2:end)) = [];
101     end
102 end
103
104 end
```

B Matlab-Code of the Tabu-Search Algorithm

B.1 Function TabuVRP

Analogously to *SolveVRP*, the function *TabuVRP* is the main function of the tabu-search algorithm. It again takes all parameters describing the problem instance as an input and returns the best solution found together with its total routing costs.

```
1  function [costopt,solution_overall] = ...
       TabuVRP(cost,avail,capacity,distance_limit,demand,distance,Tmax)
2
3  %cost numArc x numVeh matrix; cost(i,j) = cost of vehicle j traveling along arc i
4  %avail numArc x numVeh matrix; avail(i,j) = 1 if vehicle j may travel along arc i
5  %capacity numArc x numVeh matrix; capacity(i,j) = number of units vehicle j can ...
       transport on arc i
6  %distance_limit numVeh x 1 vector; distance_limit(i) = maximal travel distance ...
       of vehicle i
7  %demand numArc x 1 vector; demand(i) = number of units that have to be ...
       transported on arc i
8  %distance numArc x numVeh vector; distance(i,j) = distance of arc i for vehicle j
9  %Tmax 1x1 double; maximal computation time
10
11 % generate an initial solution
12 solution = initialSolution(cost,distance,capacity,avail,distance_limit,demand);
13
14 solution_overall = solution;
15
16 %get number of transport relations
17 numRelations = getNumRelations(solution);
18
19 % initialize parameters for the stopping criteria
20 fbest_overall = getOverallCosts(solution);
21 c = clock; % starting time
22 a = 0; % number of moves without improvement
23 a_limit = 6; % stop after a_limit moves without improvement
24 penaltyFactor = 1; % penalty coefficient
25
```

```
26 % repeat until the time limit is reached or if a feasible solution was
27 % found and did not improve for a_limit moves
28 while etime(clock,c) <= Tmax && (a <= a_limit || ...
       checkDistanceMaxViolated(solution) == true)
29
30     % set best found objective value to infinite
31     fbest_inc = Inf;
32
33     % iterate through all relations and find the one that when added to a
34     % different tour improves the current solution the most
35     for i=1:numRelations
36
37         % find the best solution, if relation i is added to a tour
38         [sol_tmp] = bestNeighbor(i,solution);
39
40         % update the incumbent solution (within the loop), if its ojective
41         % value is better than the best found so far
42         if getOverallCosts(sol_tmp) < fbest_inc
43             sol_inc = sol_tmp;
44             fbest_inc = getOverallCosts(sol_tmp)*longTermPenalty;
45         end
46     end
47
48     % update current solution
49     solution = sol_inc;
50
51     % update the overall best solution, if its objective value is less than
52     % the one of the current best solution or if the maximal travel
53     % distance was violated by the previous best solution, but the current
54     % one does not violate it anymore. If a new best solution is found, set
55     % the number of moves without imrpovement to 0. If not, increase a by 1
56     if checkDistanceMaxViolated(solution) == false
57         if (fbest_overall-fbest_inc)>1e-8 || ...
               checkDistanceMaxViolated(solution_overall) == true
58             fbest_overall = getOverallCosts(solution);
59             solution_overall = solution;
60             a = 0;
61         else
62             a = a+1;
63         end
64     elseif (fbest_overall-fbest_inc)>1e-8 && ...
           checkDistanceMaxViolated(solution_overall) == true
65         fbest_overall = getOverallCosts(solution);
66         solution_overall = solution;
67         a = 0;
68     else
69         a = a+1;
70     end
71
72     % after a_limit moves without improvement, increase the penalty
```

```
73      % coefficient by 1
74      if a > a_limit
75          a = 0;
76          penaltyFactor = penaltyFactor + 1;
77          solution = setPenaltyFactor(solution,penaltyFactor);
78      end
79
80  end
81
82  % calculate the total routing costs
83  costopt = getOverallCosts(solution_overall);
84
85  end
```

B.2 Function order

The function *order* calculates for a given relation the cost reduction if a vehicle services the relation one time less. The savings of all vehicles are then returned as a list ordered by the cost reduction per less transported unit.

```
1   function [ savings ] = order(relation, solution)
2
3   numVehicles = getNumVehicles(solution);
4   savings = [];
5
6   % for each vehicle, calculate the cost reduction if the number the relation
7   % is run by the vehicle is reduced by 1
8   for k=1:numVehicles
9
10      % get the number of tours run by the vehicle
11      numTours = getVehicleNumTours(solution,k);
12
13      % calculate the potential cost reduction for each tour run by the vehicle
14      for j=1:numTours
15          tourRelations = getTourRelations(solution,k,j);
16          relationIdx = find(tourRelations==relation,1);
17
18          % consider only tours which indeed contain the relation and from
19          % which it is not tabu to remove the relation from
20          if ~isempty(relationIdx) && isTabuRemove(solution,k,j,relation) == false
21
22              % get total number of units transported and the capacity of the
23              % vehicle for that relation
24              tourLoadings = getTourLoadings(solution,k,j);
25              unitsTransportet = tourLoadings(relationIdx,2);
26              vehicleCapacity = tourLoadings(relationIdx,3);
27
28              % reduce the number the relation is run by the vehicle by 1
29              % this corresponds to reducing the number of unit transported
```

```
30              % by the vehicle's capacity
31              sol_tmp = ...
                    removeUnitsTransported(solution,k,j,relation,vehicleCapacity,false);
32
33              % calculate the cost reduction per less transported unit and
34              % add a new line to the savings matrix
35              savings(end+1,:) = [k,j,unitsTransportet,vehicleCapacity,0];
36              savings(end,5) = max((getVehicleCurrentCosts(solution,k) - ...
                    getVehicleCurrentCosts(sol_tmp,k)) / vehicleCapacity,0);
37
38          end
39
40
41      end
42  end
43
44  % sort the savings in decreasing order by the cost reduction per less
45  % transported unit
46  savings = sortrows(savings,-5);
47
48  end
```

B.3 Function bestNeighbor

The function *bestNeighbor* moves for a given relation to the best neighboring solution by removing the relation from one or more vehicle tours and adding them to a single other vehicle tour. The function is called by the main function for each relation in order to determine the overall best neighboring solution.

```
1  function [solution_new] = bestNeighbor(relation, solution)
2  % get the potential cost reduction for different tours, if the relation is
3  % removed from it
4  savings = order(relation, solution);
5
6  % set the objective value of the best solution found so far to infinite
7  fbest = Inf;
8
9  % get the number of vehicles
10 numVehicles = getNumVehicles(solution);
11
12 % find the tour, to which the relation that should be added "fits best"
13 % iterate over all vehicles
14 for k=1:numVehicles
15
16     % get the number of tours run by the vehicle
17     numTours = getVehicleNumTours(solution,k);
18
19     % iterate over all tours of the vehicle plus a new (=empty) tour
20     for j=1:numTours+1
```

```
21
22          % check whether adding the relation to the tour is tabu
23          if isTabuAdd(solution,k,j,relation) == false
24
25              % set objective value of incumbent solution to infinite and
26              % initialize the incumbent solution with the originial one
27              fbest_tmp = Inf;
28              sol_inc = solution;
29
30              % iterate over all rows of the savings matrix, i.e. all tours
31              % from which the relation could be removed from
32              for i=1:size(savings,1)
33
34                  % check that the relation is not added to the exact same
35                  % tour it would be removed from
36                  if savings(i,1)~=k || savings(i,2)~=j
37
38                      % calculate the maximal number of runs the relation may
39                      % be removed from the tour
40                      numRunsMax = ceil(savings(i,3)/savings(i,4));
41
42                      % set the basic solution, to which the runs are added
43                      % to, to the incumbent solution
44                      sol_basic = sol_inc;
45
46                      for n=1:numRunsMax
47
48                          %calculate the number of unit that are transfered
49                          units_transfered = ceil(savings(i,3) - ...
                                (numRunsMax-n)*savings(i,4));
50
51                          % transfer the units from one tour to the other one
52                          sol_tmp = addUnitsTransported(sol_basic,k,j,relation, ...
                                units_transfered,false);
53                          sol_tmp = removeUnitsTransported(sol_tmp,savings(i,1), ...
                                savings(i,2),relation,units_transfered,false);
54
55                          % updateSolution recalculates and updates the costs
56                          % and travel distance of the solution
57                          sol_tmp = updateSolution(sol_tmp,[k,savings(i,1)],false);
58
59                          % update the incumbent solution, if the newly
60                          % generated one improves it
61                          if getOverallCosts(sol_tmp) <= fbest_tmp
62                              fbest_tmp = getOverallCosts(sol_tmp);
63                              sol_inc = sol_tmp;
64                          end
65                      end
66                  end
67              end
```

```
68
69              % due to the way relations are added and removed, it is
70              % possible that a vehicle has two tours containing the same
71              % relations. If that is the case, removeDoubleTours merges these
72              % tours
73              sol_inc = removeDoubleTours(sol_inc);
74
75              % if the incumbent solution is better than the previous best
76              % solution found, update it
77              if fbest_tmp < fbest
78                  fbest = fbest_tmp;
79                  solution_new = sol_inc;
80              end
81          end
82      end
83  end
84
85  end
```

References

Archetti, C. and Speranza, M. G. (2008). The split delivery vehicle routing problem: A survey. In Golden, B., Raghavan, S., and Wasil, E., editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces*, pages 103–122. Springer, Boston, MA.

Archetti, C., Speranza, M. G., and Hertz, A. (2006). A tabu search algorithm for the split delivery vehicle routing problem. *Transportation Science*, 40(1):64–73.

Balas, E., Ceria, S., Cornuéjols, G., and Natraj, N. (1996). Gomory cuts revisited. *Operations Research Letters*, 19(1):1–9.

Balinski, M. L. and Quandt, R. E. (1964). On an integer program for a delivery problem. *Operations Research*, 12(2):300–304.

Battarra, M., Cordeau, J.-F., and Iori, M. (2014). Pickup-and-delivery problems for goods transportation. In Vigo, D. and Toth, P., editors, *Vehicle Routing*, MOS-SIAM Series on Optimization, chapter 6, pages 161–191. SIAM, Philadelphia, PA.

Bramel, J. and Simchi-Levi, D. (2002). Set-covering-based algorithms for the capacitated VRP. In Toth, P. and Vigo, D., editors, *The Vehicle Routing Problem*, chapter 4, pages 85–108. SIAM, Philadelphia, PA.

Brunnthaller, G. and Stein, S. (2017). Entwicklung eines prognoseunterstützten Modells zur kontinuierlichen Vorplanung des Transportkapazitätsbedarfs zur Gestaltung nachhaltiger Transportketten. In Dörner, K. F., Prandstetter, M., Starkl, F. P., and Wakolbinger, T., editors, *Jahrbuch der Logistikforschung*. Trauner Verlag, Linz.

Brunnthaller, G., Stein, S., Schett, G., and Sihn, W. (2018). Development of a multi-step approach for continuous planning and forecasting of required transport capacity for the design of sustainable transport chains. In *Proceedings of the 7th Transport Research Arena*.

Chen, H.-K., Hsueh, C.-F., and Chang, M.-S. (2006). The real-time time-dependent vehicle routing problem. *Transportation Research Part E: Logistics and Transportation Review*, 42(5):383–408.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing - STOC '71*, Ohio, USA. ACM Press.

Corberán, Á. and Laporte, G. (2015). *Arc Routing: Problems, Methods, and Applications*. MOS-SIAM Series on Optimization. SIAM, Philadelphia, PA.

Cordeau, J.-F., Desaulniers, G., Desrosiers, J., Solomon, M. M., and Soumis, F. (2002). VRP with time windows. In Toth, P. and Vigo, D., editors, *The Vehicle Routing Problem*, chapter 7, pages 157–193. SIAM, Philadelphia, PA.

Crevier, B., Cordeau, J.-F., and Laporte, G. (2007). The multi-depot vehicle routing problem with inter-depot routes. *European Journal of Operational Research*, 176(2):756–773.

Crowder, H. and Padberg, M. W. (1980). Solving large-scale symmetric travelling salesman problems to optimality. *Management Science*, 26(5):495–509.

Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410.

Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, 6(1):80–91.

Darwin, C. (1859). *On the origins of species by means of natural selection.* John Murray, London, UK.

Derigs, U., Gottlieb, J., Kalkoff, J., Piesche, M., Rothlauf, F., and Vogel, U. (2010). Vehicle routing with compartments: applications, modelling and heuristics. *OR Spectrum*, 33(4):885–914.

Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39.

Dror, M. and Trudeau, P. (1989). Savings by split delivery routing. *Transportation Science*, 23(2):141–145.

Dror, M. and Trudeau, P. (1990). Split delivery routing. *Naval Research Logistics (NRL)*, 37(3):383–402.

Gendreau, M., Iori, M., Laporte, G., and Martello, S. (2006). A tabu search algorithm for a routing and container loading problem. *Transportation Science*, 40(3):342–350.

Gendreau, M. and Potvin, J.-Y. (2010). Tabu search. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, chapter 2, pages 41–59. Springer, Boston, MA.

Goetschalckx, M. and Jacobs-Blecha, C. (1989). The vehicle routing problem with backhauls. *European Journal of Operational Research*, 42(1):39–51.

Gohar, L. and Shine, K. (2007). Equivalent CO2 and its use in understanding the climate effects of increased greenhouse gas concentrations. *Weather*, 62(11):307–311.

Golden, B. L., Assad, A. A., and Wasil, E. A. (2002). Routing vehicles in the real world: Applications in the solid waste, beverage, food, dairy, and newspaper industries. In Toth, P. and Vigo, D., editors, *The Vehicle Routing Problem*, chapter 10, pages 245–286. SIAM, Philadelphia, PA.

Golden, B. L., Magnanti, T. L., and Nguyen, H. Q. (1977). Implementing vehicle routing algorithms. *Networks*, 7(2):113–148.

Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278.

Gomory, R. E. (1960). An algorithm for the mixed integer problem. Technical report, RAND Corporation, Santa Monica, CA.

Gomory, R. E. (1963). An algorithm for integer solutions to linear programs. *Recent advances in mathematical programming*, 64:260–302.

Goodman, R. (2005). Whatever you call it, just don't think of last-mile logistics, last. *Global Logistics & Supply Chain Strategies*, 9(12):46–51.

Hoff, A., Andersson, H., Christiansen, M., Hasle, G., and Løkketangen, A. (2010). Industrial aspects and literature survey: Fleet composition and routing. *Computers & Operations Research*, 37(12):2041–2061.

Iori, M., Salazar-González, J.-J., and Vigo, D. (2007). An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transportation Science*, 41(2):253–264.

Irnich, S., Toth, P., and Vigo, D. (2014). The family of vehicle routing problems. In Vigo, D. and Toth, P., editors, *Vehicle Routing*, chapter 1, pages 1–33. SIAM, Philadelphia, PA.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R., Thatcher, J., and Bohlinger, J., editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer, Boston, MA.

Laporte, G., Desrochers, M., and Nobert, Y. (1984). Two exact algorithms for the distance-constrained vehicle routing problem. *Networks*, 14(1):161–172.

Laporte, G., Mercure, H., and Nobert, Y. (1986). An exact algorithm for the asymmetrical capacitated vehicle routing problem. *Networks*, 16(1):33–46.

Laporte, G., Nobert, Y., and Desrochers, M. (1985). Optimal routing under capacity and distance restrictions. *Operations Research*, 33(5):1050–1073.

Marchand, H., Martin, A., Weismantel, R., and Wolsey, L. (2002). Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1-3):397–446.

MathWorks (2017). Matlab optimization toolbox. `https://mathworks.com/help/optim/ug/mixed-integer-linear-programming-algorithms.html`. Natick, MA.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the Association for Computing Machinery*, 7(4):326–329.

Min, H. (1989). The multiple vehicle routing problem with simultaneous delivery and pick-up points. *Transportation Research Part A: General*, 23(5):377–386.

Mingozzi, A., Roberti, R., and Toth, P. (2013). An exact algorithm for the multitrip vehicle routing problem. *INFORMS Journal on Computing*, 25(2):193–207.

Nemhauser, G. L. and Wolsey, L. A. (1989). Integer programming. In Nemhauser, G., Kan, A. R., and Todd, M., editors, *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science*, chapter 6, pages 447–527. Elsevier, Amsterdam.

Padberg, M. and Sung, T.-Y. (1991). An analytical comparison of different formulations of the travelling salesman problem. *Mathematical Programming*, 52(1-3):315–357.

Parker, R. G. and Rardin, R. L. (1988). *Discrete Optimization*. Computer Science and Scientific Computing. Academic Press, New York, NY.

Pascher, H. (2017). *Entwicklung einer Methode zur Quantifizierung von Kosten und Umweltauswirkungen in der automobilen Distributionslogistik*. Dissertation, Technische Universität Wien.

Renaud, J., Laporte, G., and Boctor, F. F. (1996). A tabu search heuristic for the multi-depot vehicle routing problem. *Computers & Operations Research*, 23(3):229–235.

Sankaran, J. K. and Ubgade, R. R. (1994). Routing tankers for dairy milk pickup. *Interfaces*, 24(5):59–66.

Schneider, M., Stenger, A., and Goeke, D. (2014). The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science*, 48(4):500–520.

SSP Consult, Beratende Ingenieure GmbH (2017). Gleitende Mittelfristprognose für den Güter-und Personenverkehr - Kurzfristprognose Sommer 2017. `https://www.bag.bund.de/SharedDocs/Downloads/DE/Verkehrsprognose/Verkehrsprognose_Sommer_2017.pdf`.

Taillard, É., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J.-Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186.

Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*. Wiley, Hoboken, NJ.

Umweltbundesamt (2017). Klimaschutzbericht 2017. `http://www.umweltbundesamt.at/fileadmin/site/publikationen/REP0622.pdf`.

Wittenbrink, P. (2014). *Transportmanagement*. Springer Fachmedien, Wiesbaden.

Wittenbrink, P. and Gburek, G. (2013). Marktvolatilität in Transport und Logistik, Ergebnisse einer Befragung der Dualen Hochschule Baden-Württemberg mit dem Bundesverband Materialwirtschaft, Einkauf und Logistik (BME). Frankfurt a. M.

Wolsey, L. A. (1998). *Integer Programming*. Wiley-Interscience, Hoboken, NJ.