FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

A Heuristic Framework for Dynamic Vehicle Routing with Site-Dependent Constraints

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Matthias Horn
Matrikelnummer 1027929

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao. Univ. Prof. Dipl. -Ing. Dr. techn. Günther Raidl
Mitwirkung: Dr. Andreas Chwatal

Wien, 20. Februar 2017

Matthias Horn

Günther Raidl

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

A Heuristic Framework for Dynamic Vehicle Routing with Site-Dependent Constraints

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Matthias Horn
Registration Number 1027929

to the Faculty of Informatics

at the TU Wien

Advisor: Ao. Univ. Prof. Dipl. -Ing. Dr. techn. Günther Raidl
Assistance: Dr. Andreas Chwatal

Vienna, 20th February, 2017

Matthias Horn	Günther Raidl

Erklärung zur Verfassung der Arbeit

Matthias Horn
Wiesfeldstraße 28 3130 Herzogenburg

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. Februar 2017

Matthias Horn

Danksagung

Als erstes möchte ich mich bei Prof. Raidl für die Betreuung dieser Arbeit bedanken. Er hat sich sofort damit einverstanden erklärt und gab mir während der gesamten Zeit seine volle Unterstützung. Des Weiteren half er mir, durch seine wertvollen Hinweise und Kommentare, mich in dem der Arbeit zugrunde liegendem Fachgebiet einzuarbeiten.

Mein besonderer Dank gilt auch Andreas Chwatal für die Mitbetreuung der vorliegenden Arbeit. Er ist der Geschäftsführer unserer Firma und der beste Chef den man sich nur wünschen kann. Während der Entstehung dieser Arbeit hat er sich sehr viel Zeit für mich genommen um meine Fragen zu beantworten. Viele wichtige Ideen und Anregungen sind während dieser Diskussionen mit ihm entstanden.

Des Weiteren möchte ich mich bei Mario Ruthmair und Markus Leitner von der Universität Wien für ihre Anregungen hinsichtlich der mathematischen Programmierung bedanken. Ferner gebührt meine Dankbarkeit Thibaut Vidal von der Pontifical Catholic University of Rio de Janeiro für die Bereitstellung einiger seiner Problem-Instanzen und zugehörigen Lösungen zum Vergleich mit dieser Arbeit.

Überdies möchte ich mich besonders bei meinem langjährigen und guten Freund Simeon Kuran bedanken. Ich durfte mit Simeon während der letzten Jahre sowohl in unserer Firma als auch im Studium eng zusammenarbeiten. Er hat mir während dem letzten Semester den Rücken freigehalten, sodass ich zügig diese Arbeit beenden konnte.

Eine ganz besondere Dankbarkeit empfinde ich gegenüber meiner Familie. Allen voran gegenüber meinen Eltern Anna und Gerhard Horn für die Unterstützung während meiner Studienzeit. Ohne sie wäre mein Studium und insbesondere diese Arbeit nicht möglich gewesen.

Acknowledgements

I would first like to express my sincere gratitude to my advisor Prof. Raidl. He immediately agreed to supervise this thesis and during the whole time gave me his full support. He gave me important hints and comments and helped me to explore this research field!

Furthermore I want to thank Andreas Chwatal for co-supervising this thesis. He is the CEO of our company and one of the best director an employee could wish for. He gave me many important inputs and many ideas emerged in discussions with him. Moreover he took a lot of time for me if a had questions regarding my research or writing.

I also want to thank Mario Ruthmair and Markus Leitner from University of Vienna for their inputs in particular to the mathematical programming methods. I also want to thank Thibaut Vidal from Pontifical Catholic University of Rio de Janeiro for providing some of his problem instances and solutions for a comparison in this work.

I also want to thank Simeon Kuran with whom I worked closely together during the last years in my studies as well as our company. He supported me in our studies during the last semester such that I could quickly finished my master thesis.

Finally I want to express my very profound gratitude to my parents Anna and Gerhard Horn for their support throughout my years of study. Without them the accomplishment of this thesis would not have been possible!

Kurzfassung

Im Rahmen dieser Arbeit wurde ein auf variable Nachbarschaftssuche (VNS) basierender Algorithmus entwickelt, um eine Variante des *Site-Dependent Vehicle Routing Problem with Time Windows (SDVRPTW)* zu lösen. Die Besonderheit dieser Variante ist, dass zum einen nicht alle Kunden von allen Fahrzeugen beliefert werden dürfen und zum anderen, dass jeder Kunde nur innerhalb eines vorgegebenen Zeitfensters beliefert werden darf.

Motivation dieser Arbeit war weitere für die Praxis relevante Eigenschaften zu berücksichtigen, wie zum Beispiel große Test-Instanzen mit einen Planungshorizont über mehrere Wochen. Der Planungshorizont wird dynamisch um jeweils einen Tag verschoben, sodass neue Kundenaufträge hinzugefügt werden und alte Aufträge aus dem Planungshorizont herausfallen. Ein anderes Szenario wäre, dass sich ein Zeitfenster von einem Kunden ändert und dieser neu eingeplant werden muss oder, dass ein Kunde nur von einem bestimmten Fahrzeug beliefert werden kann.

In dieser Arbeit wurden sowohl exakte als auch heuristische Methoden für das beschriebene Problem entworfen. Die heuristische Methode basiert auf der VNS Metaheuristik, welche systematisch und teilweise randomisiert nach besseren Lösungen sucht. Hierbei werden mit der Cyclic Exchange Nachbarschaft, welche verschiedene Kunden von verschiedenen Routen zirkular austauscht, auch Large Neighborhood Search Methoden eingesetzt.

Eine Besonderheit ist, dass der Algorithmus ohne einen expliziten Konstruktionsalgorithmus auskommt, da die benutzten Nachbarschaften in der Lage sind, eine Lösung zu erzeugen bzw. zu erweitern. Die VNS braucht daher keine Initial-Lösung, sondern kann mit einer leeren Lösung oder einer Teillösung starten.

Die Ergebnisse des entwickelten Algorithmus wurden mit den publizierten Ergebnissen von der Literatur verglichen. Es lässt sich zeigen, dass die Ergebnisse robust sind sowie eine hohe Lösungsgüte besitzen.

Zusätzlich zur heuristischen Methode, wurde eine auf Techniken der mathematischen Programmierung basierende exakte Methode entwickelt. Das zugrundeliegende Modell basiert auf einer Miller-Tucker-Zemlin Formulierung, die durch die dynamische Separierung von weiteren Ungleichungen gestärkt wird. Diese exakte Methode ist in der Lage, kleine Problem-Instanzen des SDVRPTW zu lösen.

Trotz des Schwerpunktes auf der operativen Anwendbarkeit in der Praxis im dynamischen Kontext erreichen die Algorithmen nahe state-of-the-art Ergebnisse auch auf akademischen Benchmark-Instanzen für die statische Problemvariante.

Abstract

In this work a Variable Neighborhood Search (VNS) algorithm is developed to tackle an extension of the Site-Dependent Vehicle Routing Problem with Time Windows (SDVRPTW). That is, not all vehicles are allowed to visit all customers and each customer could only be serviced within a specific time window.

The motivation was to design algorithms which are able to solve large real-world instances which have a planning horizon over several weeks. The planning horizon is shifted day by day, such that customers continuously leave and enter the planning horizon. The algorithms must handle cases where for instance customers are additionally added to the original problem. Another scenario may be that a customer's time window changed and the customer must be rescheduled or there may be situations where a customer should only be serviced by a particular vehicle.

Within this thesis, both, heuristic and exact methods are developed for the considered problem. The heuristic method is based on VNS, which searches neighboring solutions of a current incumbent solution in a systematic, but also randomized way. By using the Cyclic Exchange Neighborhood, which exchanges several customers from several different routes in a cyclic manner, large neighborhoods search techniques are also applied to the problem.

One of the main characteristics of the presented algorithm is that it does not require an explicit construction algorithm, since the neighborhoods are able to construct and enhance solutions. Hence, the VNS does not need a feasible initial solution but can start from an empty or old partial solution i.e. can do a *warm* start.

The results of the proposed algorithm are compared to recent results published in the literature for some benchmark instances and it is shown that the algorithms have good overall performance regarding robustness, solution quality and time.

In addition to the heuristic VNS approach, a mathematical programming formulation based on Miller-Tucker-Zemlin inequalities is presented. Moreover, the approach is extended to a branch-and-cut algorithm by separating set inequalities to eliminate invalid subtours. These exact methods are able to solve small instances of the SDVRPTW.

Despite the strong focus on applicability for real world operational use in context of the dynamic problem variant, the algorithms also closely achieve state-of-the-art results on academic benchmark instances for the static problem variant.

Contents

CHAPTER 1

Introduction

This chapter explains in Section 1.1 the motivation for this work. Furthermore an informal problem description is presented in Section 1.2 followed by Section 1.3 which explains the aim of this work. Section 1.4 describes the used methodological approaches to tackle the described problem in Section 1.2. The last Section 1.5 gives insights in the structure of this work.

1.1 Motivation

Logistics plays an important role in industry and economy. A lot of companies have to deal with the transportation and distribution of goods. This often directly involves production decisions and warehouse management. Hence, there are complex decisions to make, which have a huge influence to delivery costs and therefore total logistic costs.

A huge reduction of costs could for instance be achieved by attempts to reduce the total number of kilometers driven by the fleet of vehicles. It is worth mentioning, that this has a significant impact to the environment, as CO2 emissions could be reduced by such attempts. Another interesting and frequently arising goal is to determine the optimal size of the fleet of vehicles to perform all required deliveries of goods to customers or subsidiaries..

The importance of this field is also reflected in the scientific research in the last decades. Many publications address the Vehicle Routing Problem (VRP) and its variations and have presented a magnitude of solution techniques.

However, there is still some room for improvement, when it comes to applying those techniques to complex real-world problems. Despite all academic research efforts there are still some obstacles for a seamless and convenient application of these results to planning scenarios in practice.

1.2 Problem Description

A multitude of routing problems which deal with cost-optimal distribution of goods have been studied in the literature. Many different operations research techniques of heuristic and exact nature have been studied so far. There is, however, room for algorithmic improvement, in particular regarding several problem aspects arising in the retail furniture sector.

In addition to typical constraints like time-windows and maximal tour duration, this variant contains restrictions regarding required qualifications of assembly crews. For instance, some deliveries require electricians, where others might require a plumber or a carpenter. An example of this circumstance is shown in Figure 1.1. The goal of the considered optimization problem is to determine a high-quality, low-cost routing solution, which is also robust regarding new or changed customer orders.

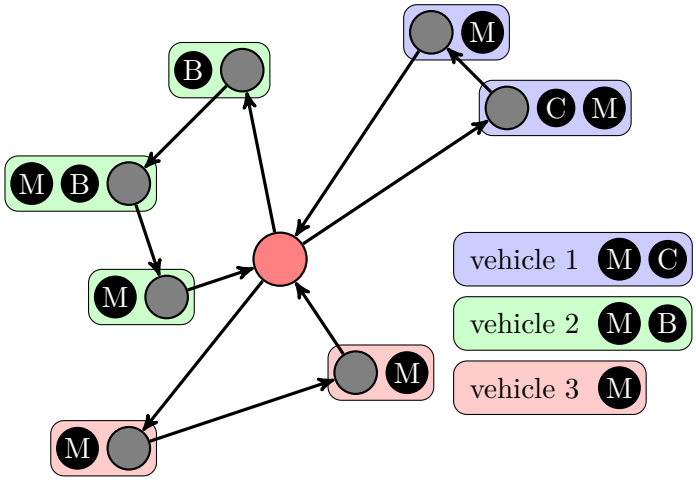


Figure 1.1: Site Dependent Vehicle Routing Problem

Furthermore there are often dynamic aspects which arise in real-world planning scenarios. In such cases, planning is usually done for time horizons of several weeks. New customer orders continuously enter this planning horizon. Delivery times are proposed to customers based on the planning solutions of the system. Hence, the algorithms need to be able to handle situations with fixed appointments and assembly crews. This fixed appointments are modeled with time windows, such that each customer with a fixed appointment could only be serviced within a tight time window. Where in practice the term *tight* could range from several minutes to several hours. The considered problem is a special variant of the Site-Dependent Vehicle Routing Problem (SDVRP), however, with several extensions. The SDVRP itself is an extension to the well known *vehicle routing problem* (VRP) with additional constraints defined for each site such that only a subset of the defined vehicles is allowed to visit them.

The projected work was realized in a cooperation of the *Algorithm and Complexity Group*

of TU Wien and *Destion – IT Consulting & Software Solutions GmbH*, an Austrian operations research software company, where I am a 10% shareholder.

1.2.1 Problem Complexity

Since the described problem which covers some dynamic aspects of real-world planning scenarios, is a special case of the SDVRP, the problem is $\mathcal{NP}$-hard [CL01]. Hence, it is very unlikely that there is an algorithm which could solve moderate to large size problem instances to optimality in polynomial time. Therefore heuristic approaches must be applied in order to solve such large instances efficiently.

1.3 Aim of this Work

The aim of this work is to develop a sophisticated algorithmic framework, which is able to produce high quality solutions. In addition, the algorithms must handle several requirements which arise in the field of furniture industry. In particular they must produce planning solutions for planning horizons of several weeks, and enable to adopt these solutions to new circumstances quickly.

Moreover the solution technique must deal with partial solutions where not all customers are scheduled yet, or some customers are already fixed regarding a specific delivery time and/or vehicle.

1.4 Methodological Approach

In order to be able to apply optimization methods to the considered problem, good runtime performance and the ability to cope with large input data sets is essential in practice. Hence, a Variable Neighborhood Search (VNS) algorithm is developed in this thesis.

For the VNS several large neighborhoods are considered which are not only applied to improve an existing solution, but also to construct solutions and add currently unscheduled customers to it. Besides classical standard neighborhoods, we apply a cyclic exchange neighborhood and a destroy and repair neighborhood. The cyclic exchange neighborhood considers exchanges of customers not only between two routes, but rather between multiple routes in a cyclic manner. The destroy and repair neighborhood destroys a whole route and tries to reschedule the customers of the destroyed route.

Furthermore exact solution approaches based on existing mathematical programming techniques are studied. In particular, a Branch & Cut (B&C) algorithm where several valid inequalities are separated as cutting planes are presented in this work. The mathematical programming formulation is based on Miller-Tucker-Zemlin. The formulation is further strengthened by separating inequalities to cut away invalid tours. As invalid tour is defined a tour with either time window violations or maximum tour duration violations or a tour which does not start and end at the depot.

1.5 Structure of this Work

Chapter 2 gives a brief overview over the VRP and its variants. Moreover the used methodological approaches in the literature for both, exact and heuristic methods to solve the Site-Dependent Vehicle Routing Problem with Time Windows (SDVRPTW) are discussed. The next Chapter 3 gives an introduction to the used methodological approaches of this work.

Chapter 4 gives a formal description of the problem and defines the set of feasible solutions. After this formal problem description, Chapter 5 presents a B&C algorithm to tackle the problem. The used cutting planes will be explained in detail as well as the mathematical programming formulation.

The VNS algorithm is presented in Chapter 6 together with all used neighborhoods and algorithmic details.

Chapter 7 compares the results obtained from the VNS algorithm and from the B&C algorithm with results obtained from the literature for several benchmark instances. Furthermore algorithmic parameters are determined and some algorithmic components are analyzed.

Finally Chapter 8 gives a summery over this work with some outlook of further investigations.

CHAPTER 2

Related Work

The VRP was intensely discussed in the literature in the last decades. Both heuristic and exact methods have been applied to the VRP. This Chapter first gives in Section 2.1 an overview of publications regarding the classical VRP and its variants. Then Sections 2.2 and 2.3 present publications which tackle the SDVRPTW by applying heuristic and exact methods, respectively.

2.1 Overview

The VRP became popular in the literature since Dantzig and Ramser [DR59] first introduced it as *"The Truck Dispatching Problem"* in 1959. Dantzig and Ramser solved the problem by generalizing the Traveling Salesman Problem (TSP) such that a central bulk terminal must be revisited after the gasoline truck has visited several service stations. The revisitation is necessary if the demand for oil of the service stations exceeds the capacity of the gasoline truck.

Clarke and Wright [CW64] generalized the problem formulation such that more than one vehicle is used to deliver goods. The authors developed one of the first construction heuristics for the VRP known as the *savings method*.

Over the years several variants of the original VRP arose, which are more related to practice. For instance popular extensions are to consider *service times* for each customer or a *maximum tour duration* for each route. Hence, the vehicle has to wait a specific amount of time before leaving the customer or the total tour duration must not exceed the maximum tour duration. This extensions are often used in combination with *time window constraints*. The Vehicle Routing Problem with Time Windows (VRPTW), first introduced by Solomon [Sol83], considers time windows for each customer, such that the arrival time of the vehicle must be within the customer's time window. An overview

of solution methods for the VRPTW is for instance provided by Bräsys and Gendreau [BG05a], [BG05b].

Another extension of the VRP is the consideration of a heterogeneous vehicle fleet, known as the Heterogeneous VRP (HVRP). Instead of assuming that every vehicle has the same properties, vehicles can differ for instance with respect to the maximum load capacity or the maximum tour duration.

One extension which is very important in practical logistics systems is the integration of multiple depots. This variant is known as the Multiple Depots Vehicle Routing Problem (MDVRP) and uses multiple depots instead of only one depot. A survey of the MDVRP is presented in Montoya-Torres, Franco, Isaza, Jiménez and Herazo-Padilla [MTFI$^+$15].

Another VRP variant which is closely related to the discussed problem in this work is the Dynamic VRP (DVRP). This variant deals with dynamic and stochastic approaches such that not all information is provided in advance. Hence, there are dynamic events which indicate for instance a new customer request or a change of service times, travel times or demands as well. Since these dynamic events arrive when the fleet is already executing the current tours, the replanning has to be in *real time*. An overview of variants of the DVRP can be obtained from Pillac, Gendreau, Guéret and Medaglia [PGGM13] as well as from Ritzinger, Puchinger and Hartl [RPH16].

Over the last years more and more papers studied multiconstraint VRPs. Hence, the VRP is extended by several constraints, for instance with time windows, multiple depots or site-dependency constraints. Those multiconstrained VRPs form a new class called Rich VRP (RVRP). A comprehensive taxonomy of the RVRP is presented by Lahyani, Khemakhem and Semet [LKS15].

A classification of variants of the VRP is presented in Eksioglu, Vural and Reisman [EVR09] as well as in Braekers, Ramaekers and Nieuwenhuyse [BRVN16].

2.2 Heuristic Methods

Although the VRPs have been studied extensively in the literature, to the author's best knowledge very few publications adressing the SDVRP do exist. In the taxonomy of Eksioglu, Vural and Reisman [EVR09] there are only two papers dealing with the SDVRP.

Cordeau and Laporte [CL01] show that the SDVRP is a special case of the Periodic VRP (PVRP) and solve the problem with a Tabu Search (TS) heuristic for the PVRP [CGL97]. One disadvantage of this approach is that SDVRP instances are limited to a planning horizon of one day, since the vehicle types of the SDVRP are mapped to single days of the PVRP. Brandao and Mercer [BM97] consider a multi-trip vehicle routing and scheduling problem with many constraints including that only certain vehicles can visit certain customers. They also tackle the problem with a tabu search heuristic.

Another tabu search heuristic is presented in Alonso, Alvarez and Beasley [AAB08] to tackle the Site-Dependent Multi-Trip Periodic Vehicle Routing Problem (SDMTPVRP). The problem extends the classical VRP in the manner that vehicles are allowed to have multiple routes on the same day as long as the maximum operation time is not exceeded. The tabu search heuristic is able to outperform the tabu search heuristic from [CL01].

Pisinger and Ropke [PR07] present a unified heuristic to solve five variants of the VRP including the SDVRP. This is done by transforming the variants of the VRP instances into a pickup and delivery problem instance and applying an adaptive large neighborhood search heuristic to them. Amorim and Parragh [APSAL14] also use an adaptive large neighborhood search framework to solve a heterogeneous fleet site dependent vehicle routing problem with multiple time windows in collaboration with a Portuguese food distribution company. The company has to face this problem in real live on a daily basis, where customers have multiple interdependent time windows.

Belhaiza [Bel11] comes up with a hybrid VNS combined with a TS algorithm. Although the algorithm improves 20 out of 24 best known solutions of the year 2010 for the test instances created by [CL01], the algorithm is not very computation time efficient. Another VNS algorithm, where the shaking phase of the VNS is replaced by a TS is presented by Sicilia, Quemada, Beatriz and Escuín [SQRE16]. The algorithm solves the RVRP, considering constraints which are important regarding freight distribution in large urban areas. Considered constraints are among others time window constraints and site-dependency constraints. The algorithm is tested on data extracted from real-world scenarios which arise by a large transport company in Spain.

Zare-Reisabadi and Mirmohammadi [ZRM15] present two meta-heuristic algorithms for solving the SDVRP with soft time windows: an Ant Colony Optimization (ACO) with Local Search (LS), and a TS. These algorithms were compared to each other by adopting the data sets from [Sol87]. The ACO finds better solutions in most cases, however, if the size of the test instances increases, the runtime of the ACO exceeds the runtime of the TS significantly. Therefore the ACO may not be practical for very large instances.

Vidal, Crainic, Gendreau and Prins [VCGP13] introduce the *Genetic Search with Advanced Diversity Control* algorithm which can efficiently solve a wide range of large-scale VRPs. The algorithm is a combination of the metaheuristics Genetic Algorithm (GA) and LS with diversity management mechanisms. All classical benchmarks for the MDVRP, PVRP and SDVRP are outperformed by the algorithm.

2.3 Exact Methods

Whereas, there are many publications with heuristic methods to tackle the VRPTW or the SDVRPTW, there are only a few which use exact methods.

An exact method consisting of a dual ascent procedure in combination with a column-and-cut generation algorithm, is presented in [BBMR10] where the SDVRP is treated as a special case of the HVRP. The HVRP is mathematically modeled so that the routing

cost of an edge in the routing graph does not only depend on the start and end location but also on the vehicle using this edge. Hence, by setting the routing cost to infinity of the corresponding edges one can treat each SDVRP instance as an HVRP instance. However, for larger instances the runtime of exact methods is not acceptable.

A special case of the SDVRP is the Skill Vehicle Routing Problem in [CGS11], where an ordering among the vehicle types exists, such that a customer which requires a certain vehicle type can be delivered by vehicles with at least that type or vehicles with higher type according the ordering of vehicle types. In [CGS11] there are various Integer Linear Programming models developed and tested on randomly generated instances.

Regarding the considered problem variant, a key limitation of the above mentioned work is that it does not address the dynamic aspects of the SDVRPTW. It is important that new customers can be added to an existing solution rather than building up a solution from the scratch. Therefore the new algorithm must be able to start from an existing solution and try to schedule new customers, which are not yet in the solution.

CHAPTER 3

Heuristic Optimization Methods

This chapter first gives in Section 3.1 a common definition of combinatorial optimization problems. Furthermore a brief overview will be given of the main solution approaches of such problems. Particular different metaheuristics will be explained. Sections 3.2-3.4 describe in detail some metaheuristics which will be used in this work in later chapters.

3.1 Combinatorial Optimization Problems

In practice many problems arise where not only a feasible solution of a specific problem is of interest, but rather the best solution among all possible solutions. Those problems where the best solutions have to be found are called Optimization Problems (OP). OPs can be divided into two categories: those where solutions can be expressed through *real-valued* variables and those where solutions can be expressed through *discrete* variables [PS82] and [BR03]. The latter ones are called Combinatorial Optimization Problems (COP).

Definition 3.1.1.
A COP $P = (S, f)$ consists of

- *a set of variables $X = \{x_1, \ldots, x_n\}$ together*
- *with corresponding variable domains $D_1, \ldots, D_n$*
- *and a set of constraints C among variables, defined on subsets of $D = D_1 \times D_2 \times \cdots \times D_n$.*

Depending on the problem, the objective function $f : D \to \mathbb{R}$ has to be minimized or maximized. The set of feasible solution donated as solution space is defined as

$$S = \{s = \{(x_1, v_1), \ldots, (x_n, v_n)\} \,|\, v_i \in D_i, s \text{ satisfies } C\} \tag{3.1}$$

[BR03].

The goal of a COP is to find a solution $s^* \in S$ such that

$$f(s^*) \leq f(s) \quad \forall s \in S \tag{3.2}$$

holds. The solution s^* denotes the global optimal solution of P. Since every maximization problem can by considered as a minimization problem by setting $f(.)$ to $-f(.)$, we will only consider minimization problems.

Since the size of the solution space S for many COPs grows exponentially with the input, finding a globally optimal solution $s^* \in S$ by performing an exhausted search in S is therefore hardly possible. Often there is no algorithm known which is able to explore the whole search space efficiently.

There are two approaches to solve a COP: *Exact solution approaches* and *Heuristic approaches*. The former finds the global optimal solution s^* by exploring the solution space S such that areas of S are omitted where it is guaranteed that there could not be the global optimal solution s^* within that areas. Often such approaches are based on the *divide and conquer* principle. Hence, the solution space is recursively partitioned into subspaces. If it is guaranteed that s^* can not be found in a specific subspace, that subspace is ruled out and not further partitioned into smaller subspaces. The efficiency of those algorithms strongly depends on how often subspaces could be ruled out during the search. A typical example of such an algorithm is the Branch & Bound (B&B) algorithm first introduced in [LD60] by Land and Doig. The B&B algorithm estimates for each subspace an upper and a lower bound. If the lower bound exceeds the upper bound, the corresponding subspace can be omitted. As already mentioned, because of the exponentially growing of the solution space S with the input, even exact solution approaches which use sophisticated methods like B&B are only practicable for small instances.

Heuristic approaches on the other hand, use the fact, that in practice it is often sufficient to find a solution which is near the global optimum solution s^*. In order to find such near-optimal solutions it may not be required to explore the whole solution space S, but rather to explore only a subset of S. Those subset can be efficiently found and explored by *metaheuristics*.

Metaheuristics are high level strategies to explore the solution space efficiently, by using different methods [BR03]. The suffix *meta* means that those search strategies are problem independent, whereas the word *heuristic* indicates that the search procedure may not find the global optimal solution s^*. The goal is to explore subsets of the solution space efficiently to find near-optimal solutions. Metaheuristics are usually non-deterministic.

There are many ways to classify metaheuristics. Single solution based methods focus on improving a single solution. Those methods are often called *trajectory methods*, because they follow a single trajectory through the search space. Examples of metaheuristics which

use such methods are LS, VNS, Simulated Annealing (SA) and TS. Most single solution based metaheuristics are extension or modifications of LS. They are trying to escape a local optima in different ways. Since this work is based on VNS, the metaheuristics LS and VNS will be explained in more detail in the following subsections.

On the contrary, metaheuristics like ACO, GA or Particle Swarm Optimization (PSO) use population based methods. Hence, they improve multiple solutions simultaneously. Population based metaheuristics are often inspired by nature. The ACO for instance is inspired by ants seeking a source of food. If an ant finds a source of food, the ant will lay down a pheromone trail from the source of food back to their colony such that other ants could follow the pheromone trail. The GA is inspired by the process of natural selection. Hence, only the best solutions of the current solution population are selected in order to generate new solutions by recombination and mutation. Furthermore the PSO is inspired by the movements of a bird flock or a fish swarm.

The reason for the diversity of metaheuristics is explained by the *no free lunch theorem* by Wolpert and Macready [WM97]. One simple interpretation of the theorem is that *"a general-purpose universal optimization strategy is theoretically impossible, and the only way one strategy can outperform another is if it is specialized to the specific problem under consideration"* [HP02]. Therefore a metaheuristic must be applied and adapted to a specific COP by making use of a prior problem specific knowledge. Only then it makes sense to compare the metaheuristic with other metaheuristics which are also adapted to the specific COP.

Chapter 2 already summarized metaheuristics which are applied to the SDVRP. The next subsections will explain the ideas behind the LS and VNS methods. Chapter 6 will then apply the VNS method to the SDVRP.

3.2 Local Search

Since an exhausted search through the solution space S is not practicable due to the *combinatorial explosion* of S with respect to the input size, one approach is to scan only a small subspace of S. This gives rise to the definition of a neighborhood $\mathcal{N}(.)$ [BR03].

Definition 3.2.1.
Let $P = (S, f)$ be a COP, a neighborhood is a mapping

$$\mathcal{N} : S \to 2^S \tag{3.3}$$

that maps a feasible solution $s \in S$ to a set of neighbor *solutions $\mathcal{N}(s) \subseteq S$.*

Often the neighborhood $\mathcal{N}(s)$ of solution s is defined by the set of solutions which can be obtained by applying a single operation on s.

The definition of a neighborhood allows us to define a locally optimal solution.

Definition 3.2.2.
Let $P = (S, f)$ be a COP and $\mathcal{N}$ be a neighborhood. A locally optimal solution $\hat{s} \in S$ with respect to $\mathcal{N}$ satisfies

$$f(\hat{s}) \leq s \quad \forall s \in \mathcal{N}(\hat{s}). \tag{3.4}$$

A locally optimal solution $\hat{s}$ is called strict locally optimal if

$$f(\hat{s}) < s \quad \forall s \in N(\hat{s}). \tag{3.5}$$

holds.

Algorithm 3.1 finds such a locally optimal solution $\hat{s}$ with respect to a neighborhood $\mathcal{N}$. Starting from a feasible solution s_{init}, Algorithm 3.1 selects a neighbor solution s' from s. If the objective value of the selected solution s' is smaller than the objective value of solution s, a better solution is found and stored into s. The procedure stops if some stopping criteria are satisfied. For instance if there is no new best solution found in the current neighborhood $\mathcal{N}(s)$ or the search time has expired. Another stopping criteria could be that the number of steps exceeds a predefined threshold. The procedure is called LS procedure.

Algorithm 3.1: Local Search

Input: An initial solution s_{init}, a neighborhood $\mathcal{N}$
Output: A probably improved solution s

```
1 s ← s_init;
2 repeat
3 |   select s' ∈ N(s);
4 |   if f(s') < f(s) then
5 |   |   s ← s';
6 |   end
7 until stopping criteria satisfied;
8 return s;
```

One critical aspect of the LS procedure is the *select* operator in line 3 of Algorithm 3.1. The operator defines how the current neighborhood $\mathcal{N}(s)$ is scanned and how the solution s' is selected. The selection of a neighbor could have a great influence to the overall performance of the LS procedure. Usually there are three possibilities to define the select operator.

Best Improvement: In the first place the whole neighborhood $\mathcal{N}(s)$ is searched through and the best solution is returned.

First Improvement: Alternatively, one can search through neighborhood $\mathcal{N}(s)$ and return the first solution s' which is better than solution s. The order in which the neighborhood is explored may play an important role.

Random Neighbor: Another option is to randomly select a neighbor s' of the neighborhood $\mathcal{N}(s)$.

One important influence of the decision which select operator to use could be the size of the considered neighborhood. If the size is rather small then *Best Improvement* could be used to select a neighbor solution. Otherwise, *First Improvement* could be a better choice. If the neighborhood is extremely large then only *Random Neighbor* may be practicable.

3.3 Variable Neighborhood Search

One disadvantage of the LS procedure is that it explores only the area around one local optimum with respect to a specific neighborhood, although a COP could have many local optima. Furthermore the objective value of the global optimum could extremely differ from the average objective value of local optima. Hence, it is worthwhile trying to explore different local optima to hopefully get better solutions near the global optimum.

One way to escape local optima is the VNS method, first introduced in [MH97] by Mladenović and Hansen. VNS is based on the idea to not just only search through only one neighborhood, but rather to use more than one neighborhood. Those neighborhood structures are alternated in a systematic way. VNS relies on the following principles [HMBP10]:

- A local optimum with respect to one neighborhood structure is not necessarily one for another.
- A global optimum is a local optimum with respect to all possible neighborhood structures.
- For many problems local optima with respect to one or several neighborhoods are relatively close to each other.

The alternation of the neighborhoods could be in a deterministic and / or stochastic way. Algorithm 3.2 shows the Variable Neighborhood Descent (VND) meta-heuristic which alternates the neighborhood structure $\mathcal{N}_k$, $(k = 1, \ldots, k_{\max})$ in a deterministic way. The neighborhoods $\mathcal{N}_k$ are usually ordered with respect to their increasing sizes or after their increasing effort to search through them. Typically best improvement is applied to find the best solution s in the current neighborhood $\mathcal{N}_k(\hat{s})$ of the current incumbent solution $\hat{s}$. If the local optimum is found with respect to the current neighborhood $\mathcal{N}_k(\hat{s})$, the next neighborhood $\mathcal{N}_{k+1}(\hat{s})$ in the neighborhood structure will be considered. Otherwise the new solution s becomes the new incumbent solution $\hat{s}$ and the neighborhood structure is reset to the first neighborhood $\mathcal{N}_1$. If the local optimum of the last neighborhood $\mathcal{N}_{k_{\max}}$ in the neighborhood structure is found, the returned solution $\hat{s}$ is locally optimal with respect to all neighborhoods $\mathcal{N}_k(\hat{s})$ with $k = 1, \ldots, k_{\max}$.

Algorithm 3.2: VND

Input: An initial solution s_{init}, a neighborhood structure $\mathcal{N}_k$
Output: A probably improved solution $\hat{s}$

```
1  ŝ ← s_init;
2  k ← 1;
3  repeat
4  |   s ← arg min_{s'∈N_k(ŝ)} f(s');
5  |   if f(s) < f(ŝ) then
6  |   |   ŝ ← s;
7  |   |   k ← 1;
8  |   else
9  |   |   k ← k + 1;
10 |   end
11 until k = k_max;
12 return ŝ;
```

Algorithm 3.3 shows the Basic VNS (BVNS) method, which combines both deterministic and stochastic changes of neighborhoods. The algorithm can be split into two parts. Line 5 shows the shaking phase which represents the stochastic part. A solution s' is randomly selected from the current neighborhood $\mathcal{N}_k(s)$ of the current incumbent solution s. The deterministic part is listed in line 6 and refers to the descent phase of Algorithm 3.3. Whereas in the descent phase a locally optimum solution $\hat{s}$ with respect to neighborhood $\mathcal{N}_{\text{ls}}(s')$ is found, in the shaking phase BVNS tries to get out of the currently explored valley.

There exist many variants of the VNS. If the descent phase in Algorithm 3.3 is neglected, the resulting algorithm is called Reduced VNS (RVNS). Another VNS variant is the General VNS (GVNS), which is obtained by using a VND search instead of a local search procedure in the descent phase.

3.4 Skewed Variable Neighborhood Search

The Skewed VNS (SVNS) is an extension of the VNS to explore valleys which are far away from the current best solution [HJMP00]. If the local optimum is found within a large region, then it may be necessary to move far away from the best solution to find another better local optimum. Solutions in such a distant region could differ significantly from the best solution found so far. The SVNS allows such far jumps from the incumbent solution, without degenerating to the multistart heuristic [HMBP10]. This is avoided by defining a distance metric

$$\rho(s_1, s_2) : S \times S \mapsto R_{\geq 0} \tag{3.6}$$

between two solutions s_1 and s_2 and let the SVNS only jump into a far away region if the distance between the incumbent solution and the new solution is wide enough.

Algorithm 3.3: Basic VNS

Input: A initial solution s_{init}, a neighborhood structure $\mathcal{N}_k$, a neighborhood $\mathcal{N}_{\text{ls}}$
Output: A probably improved solution s

```
1  s ← s_init;
2  repeat
3  |   k ← 1;
4  |   repeat
5  |   |   randomly select s' ∈ N_k(s);                    // shaking phase
6  |   |   ŝ ← LocalSearch(s', N_ls);                      // descent phase
7  |   |   if f(ŝ) < f(s) then
8  |   |   |   s ← ŝ;
9  |   |   |   k ← 1;
10 |   |   else
11 |   |   |   k ← k + 1;
12 |   |   end
13 |   until k = k_max;
14 until stopping criteria satisfied;
15 return s;
```

Algorithm 3.4 shows in detail the SVNS method. Variable s stores the current incumbent solution, whereas variable s_{best} stores the best solution found so far. As with the BVNS Algorithm 3.3, the SVNS Algorithm 3.4 has a shaking phase and a descent phase. Variable s' is the randomly selected solution of the current neighborhood $\mathcal{N}_k(s)$ from the incumbent solution s. The local optimum solution with respect to neighborhood $\mathcal{N}_{\text{ls}}$ is stored into variable $\hat{s}$ after the descent phase. The core of Algorithm 3.4 is line 8. If the condition

$$f(\hat{s}) - \alpha_{\text{svns}}\,\rho(s,\,\hat{s}) < f(s) \tag{3.7}$$

holds, variable $\hat{s}$ becomes the new incumbent solution. If the distance between solution s and $\hat{s}$ is wide enough, then $\hat{s}$ is accepted as the new incumbent solution even if $\hat{s}$ has a worse objective value than s. Parameter α_{svns} controls how distant a solution $\hat{s}$ must be in order to perform the jump operation.

In the next chapter the variant of the vehicle routing problem with site dependencies will be formally described. Then in Chapter 6 the VNS as well as the SVNS will be applied to the problem. In Chapter 7 the results will be presented and compared with the results from the literature.

Algorithm 3.4: Skewed VNS

Input: A initial solution s_{init}, a neighborhood structure $\mathcal{N}_k$, a neighborhood $\mathcal{N}_{\text{ls}}$
Output: A probably improved solution s

$s \leftarrow s_{\text{init}}$;
$s_{\text{best}} \leftarrow s_{\text{init}}$;
repeat
  $k \leftarrow 1$;
  repeat
    randomly select $s' \in \mathcal{N}_k(s)$; `// shaking phase`
    $\hat{s} \leftarrow$ `LocalSearch(`s', $\mathcal{N}_{ls}$`)`; `// descent phase`
    if $f(\hat{s}) - \alpha_{\text{svns}}\,\rho(s, \hat{s}) < f(s)$ **then**
      $s \leftarrow \hat{s}$;
      $k \leftarrow 1$;
    else
      $k \leftarrow k + 1$;
    end
  until $k = k_{\max}$;
  if $f(s) < f(s_{\text{best}})$ **then**
    $s_{\text{best}} \leftarrow s$;
  end
  $s \leftarrow s_{\text{best}}$;
until *stopping criteria satisfied*;
return *s;*

CHAPTER 4

Formal Description

In this chapter we formulate first the Static Site-Dependent Vehicle Routing Problem with Time Windows (SSDVRPTW) $P = (S, f)$ as a mathematical model in Section 4.1-4.4. The model consists of input parameters which describe a specific problem instance of the SSDVRPTW together with decision variables and constraints which describes the set of feasible solution S of the SSDVRPTW. Furthermore the objective function f will be defined.

Based on the model for the SSDVRPTW the model $\breve{P} = (\breve{S}, \breve{f})$ for the Dynamic Site-Dependent Vehicle Routing Problem with Time Windows (DSDVRPTW) will be defined in Section 4.5.

4.1 Parameters

The formulation of the SSDVRPTW is based on the formulation of the SDVRP with soft time windows from [ZRM15]. The SSDVRPTW is defined on a graph $G = (N, A)$ with a set of nodes $N = \{0, 1, \ldots, n_c, n_c + 1\}$ and a set of arcs $A = \{(i, j) \,|\, i, j \in N\}$. The vertices correspond to n_c customers except vertex 0 and $n_c + 1$ which both represent the depot. Hence, each tour has to start at the depot with vertex 0 and end at the depot with vertex $n_c + 1$ to indicate the start and end of a tour. Let $N' = \{1, \ldots, n_c\} \subset N$ further denote the set of all customers.

- For each arc $(i, j) \in E$ there is a distance $c_{i,j}^{\text{dist}} \in \mathbb{R}_{\geq 0}$ and a travel time $c_{i,j}^{\text{time}} \in \mathbb{R}_{\geq 0}$ defined. The distances and travel times are arranged in the distance matrix $\mathbf{C}^{\text{dist}} = (c_{i,j}^{\text{dist}}), \forall i, j \in N$ and in the time matrix $\mathbf{C}^{\text{time}} = (c_{i,j}^{\text{time}}), \forall i, j \in N$, respectively. We assume that all entries of this matrices are greater or equal than zero and all diagonal elements are zero. Furthermore the matrices are not necessarily symmetric and it can be assumed that the triangle inequality does hold.

- The planning horizon $D = \{1 \dots, n_d\}$ is a non-empty finite total ordered set with n_d days.
- Each customer $i \in N'$ has a demand $d_i \geq 0$ of goods. The time it takes to deliver this demand is denoted by the service time $s_i \geq 0$. For simplicity the variables s_0 and d_0 for depot 0 are also defined and set to zero.

  Furthermore for each customer i a set of allowed delivery days $D_i \subseteq D \setminus \emptyset$ together with a time window $W_i^{\text{cus}} = [e_i^{\text{cus}}, l_i^{\text{cus}}]$ is defined so that the delivery of goods to customer i can only take place on a day $d \in D_i$ between the earliest service time e_i^{cus} and the latest service time l_i^{cus}.
- Each vehicle $v \in V$ has a maximum load capacity $C_v^{\max}$.

  Moreover each vehicle v is associated with an availability time window $W_v^{\text{veh}} = [s_v^{\text{veh}}, e_v^{\text{veh}}]$ such that the vehicle can only start from the depot and arrive at the depot between the availability start time s_v^{veh} and the availability end time e_v^{veh}.
- Let $Q = \{1, \dots, n_q\}$ be the index set of n_q qualifications. Each customer $i \in N'$ is associated with a set of required qualifications $Q_i^{\text{req}} \subseteq Q$, and each vehicle $v \in V$ is associated with a set of provided qualifications $Q_v^{\text{pro}} \subseteq Q$.

  A vehicle $v \in V$ can only deliver goods to a customer $i \in N'$ on day $d \in D$ if and only if the required set of qualifications of customer i is a subset of the provided set of qualifications of vehicle v and day d is in the set of allowed delivery days D_i. Hence, let

$$A_i = \{(d, v) \in D \times V \mid d \in D_i \wedge Q_i^{\text{req}} \subseteq Q_v^{\text{pro}}\}, \quad \forall i \in N' \tag{4.1}$$

  be the set of day and vehicle combinations which are allowed to visit customer i.

4.2 Variables

The decision variables are:

- $X_{ij}^{dv} \in \{0, 1\} \quad \forall d \in D, \forall v \in V, \forall (i, j) \in A$,
  indicates if vehicle $v \in V$ travels by arc $(i, j) \in A$ on day $d \in D$.
- $Y_0^{dv} \in \mathbb{R}_{\geq 0} \quad \forall d \in D, \forall v \in V$
  indicates the start time from the depot of vehicle $v \in V$ on day $d \in D$.

Furthermore we make use of the following auxiliary variables:

- $R^{dv} \quad \forall d \in D, \forall v \in V$
  indicates a path in graph G which is followed by vehicle $v \in V$ on day $d \in D$. A path is a finite sequence of vertices $(\tau_1, \tau_2, \dots, \tau_k)$, so that $(\tau_i, \tau_{i+1}) \in A$ for $i = 1, \dots, k$ and there are no repeated vertices in the sequence. Furthermore let $|R^{dv}| = k$ be the length of the sequence.

- $Y_i^{dv} \in \mathbb{R}_{\geq 0} \quad \forall d \in D, \forall v \in V, \forall i \in N \setminus \{0\}$
  indicates the arrival time of vehicle $v \in V$ to customer $i \in N'$ on day $d \in D$. The variables $Y_{n_c+1}^{dv}$ denote the arrival time at the depot of vehicle v on day d.

- $W_i^{dv} \in \mathbb{R}_{\geq 0} \quad \forall d \in D, \forall v \in V, \forall i \in N$
  indicates the waiting time of vehicle $v \in V$ while arriving at customer $i \in N'$ on day $d \in D$ in order to hold the earliest service time e_i^{cus} of the customer's time window $[e_i^{\text{cus}}, l_i^{\text{cus}}]$. The variables W_0^{dv} and $W_{n_c+1}^{dv}$ are defined for simplicity and set to zero.

- $L_{dvi}^{\text{cus}} \in \mathbb{R}_{\geq 0} \quad \forall d \in D, \forall v \in V, \forall i \in N$
  indicates the lateness of vehicle $v \in V$ while arriving at customer $i \in N'$ on day $d \in D$ by exceeding the latest service time l_i^{cus} of the customer's time window $[e_i^{\text{cus}}, l_i^{\text{cus}}]$. The variables L_{dv0}^{cus} and $L_{dv(n_c+1)}^{\text{cus}}$ are defined for simplicity and set to zero.

- $L_{dv}^{\text{veh}} \in \mathbb{R}_{\geq 0} \quad \forall d \in D, \forall v \in V$
  indicates the overtime of vehicle $v \in V$ on day $d \in D$ by exceeding the availability end time e_v^{veh} of the vehicle's availability time window $[s_v^{\text{veh}}, e_v^{\text{veh}}]$.

- $O^{dv} \in \mathbb{R}_{\geq 0} \quad \forall d \in D, \forall v \in V$
  indicates the overtime related to the maximum tour duration $D^{\max}$ of vehicle $v \in V$ on day $d \in D$.

which can be computed from X_{ij}^{dv} and Y_0^{dv}. These variables are used in Section 4.3 to describe the constraints which must be fulfilled by a feasible solution.

Figure 4.1 shows an example of a feasible route of vehicle $v \in V$ on day $d \in D$. The vehicle starts from the depot at s_v^{veh} and arrives too early at customer $i \in N'$ at Y_i^{dv} to hold the customer's time window. Therefore the vehicle waits to hold the time window which is described through the waiting time W_i^{dv}. If the earliest service time e_i^{cus} is reached, the vehicle starts to unload the goods (service time s_i). After that, the vehicle leaves customer i and arrives at customer $j \in N'$ at Y_j^{dv}. This time the vehicle arrives too late to meet the customer's time window. This indicates the lateness variable L_j^{cus}. After unloading the goods at customer j the vehicle drives back and reaches the depot at $Y_{n_c+1}^{dv}$. The variable O^{dv} indicates that the total tour duration $Y_{n_c+1}^{dv} - Y_0^{dv}$ exceeds the maximums tour duration $D^{\max}$ and variable L_v^{veh} indicates that the vehicle arrives too late to hold the vehicle's time window.

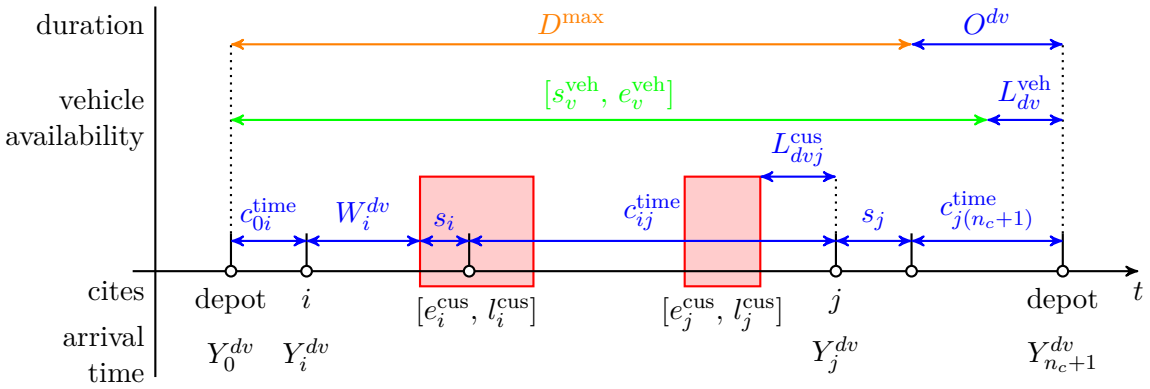


Figure 4.1: Example on a feasible route. The vehicle $v \in V$ arrives at customer $i \in N'$ too early. Hence, a waiting time W_i^{dv} is added to hold the customer's time window. Furthermore the vehicle reaches customer $j \in N'$ too late, which indicates the variable L_{dvj}^{cus}.

4.3 Solution Space

Let S be the solution space of the SSDVRPTW. A feasible solution

$$s = \left\{ \left(R^{dv}, Y_0^{dv} \right) \mid d \in D,\, v \in V \right\} \in S \tag{4.2}$$

of the SSDVRPTW is a set of routes R^{dv} together with the corresponding start times Y_0^{dv} for each vehicle $v \in V$ and for each day $d \in D$.

A solution is feasible if and only if

- for each arc $(i,j) \in A$, vehicle $v \in V$ and day $d \in D$ holds that

$$X_{ij}^{dv} = 1 \Leftrightarrow (i,j) \in R_{ij}^{dv} \quad \forall d \in D,\, \forall v \in V,\, \forall (i,j) \in A \tag{4.3}$$

  if arc (i,j) is used by vehicle v on d then arc (i,j) is also in route R^{dv} driven by vehicle v starting on day d, and vice versa.

- each route $R^{dv} = \left(\tau_1, \tau_2, \ldots, \tau_{|R^{dv}|} \right)$ in solution s starts and ends at the depot. Hence, it holds that $\tau_1 = 0$ and $\tau_{|R^{dv}|} = n_c + 1$. Since it is possible that a vehicle is not scheduled on all days in the planning horizon D, there are *empty routes* which contains only the start depot 0 and the end depot $n_c + 1$. Therefore

$$\left| R^{dv} \right| \geq 2 \quad \forall R^{dv} \in s \tag{4.4}$$

  must hold for reach route R^{dv} in solution s. Furthermore a route must not visit the start depot 0 or the end depot $n_n + 1$ more than once. Hence, the condition

$$(i,0) \notin R^{dv} \wedge (n_c + 1, i) \notin R^{dv} \quad \forall R^{dv} \in s,\, \forall i \in N' \tag{4.5}$$

  must hold.

- each customer $i \in N'$ is served by a vehicle $v \in V$ so that the customer's required set of qualifications Q_i^{reg} are a subset of the vehicle's provided set of qualifications Q_v^{pro} and the start day is in the set of allowed delivery days D_i. Therefore the condition
$$i \in R^{dv} \implies (d, v) \in A_i \quad \forall R^{dv} \in s, \forall i \in N' \tag{4.6}$$
must hold. Furthermore each customer $i \in N'$ can only be served one time. Hence, customer i can only be included into one Route R^{dv} of solution s.

- the arrival times Y_j^{dv} for each customer $j \in N \setminus \{0\}$ can be recursively computed through
$$Y_j^{dv} = \begin{cases} Y_i^{dv} + s_i + c_{ij}^{\text{time}} + W_i^{dv} & \text{if } (i,j) \in R^{dv} \\ 0 & \text{otherwise} \end{cases} \quad \forall d \in D, \forall v \in V \tag{4.7}$$
together with the waiting times
$$W_j^{dv} = \begin{cases} \max\left(0,\, e_j^{\text{cus}} - Y_j^{dv}\right) & \text{if } j \in R^{dv} \\ 0 & \text{otherwise} \end{cases} \quad \forall d \in D, \forall v \in V. \tag{4.8}$$
Note that these conditions make sure that the earliest service time e_j^{cus} of customer j is not violated.

- the lateness
$$L_{dvi}^{\text{cus}} = \begin{cases} \max\left(0,\, Y_i^{dv} - l_i^{\text{cus}}\right) & \text{if } i \in R^{dv} \\ 0 & \text{otherwise} \end{cases} \quad \forall d \in D, \forall v \in V \tag{4.9}$$
of vehicle $v \in V$ arriving at customer $i \in N'$ can be computed from the arrival time Y_i^{dv} and the latest service time l_i^{cus}.

- the equation
$$L_{dv}^{\text{veh}} = \begin{cases} \max\left(0,\, Y_{n_c+1}^{dv} - e_v^{\text{veh}}\right) & \text{if } (0, n_c+1) \notin R^{dv} \\ 0 & \text{otherwise} \end{cases} \quad \forall d \in D, \forall v \in V \tag{4.10}$$
of the overtime L_{dv}^{veh} holds as well as the equation
$$O^{dv} = \max\left(0,\, Y_{n_c+1}^{dv} - Y_0^{dv} - D^{\max}\right) \quad \forall d \in D, \forall v \in V \tag{4.11}$$
of the overtime O^{dv} related to the maximum tour duration $D^{\max}$ holds for each vehicle $v \in V$ and for each day $d \in D$.

- each vehicle $v \in V$ does not violate the vehicle's availability start time s_v^{veh}. Therefore the equation
$$s_v^{\text{veh}} \leq Y_0^{dv} \quad \forall d \in D, \forall v \in V \tag{4.12}$$
must hold for each vehicle $v \in V$ on each day $d \in D$.

- the amount of goods which are delivered by route $R^{dv} \in s$ does not exceed the maximum load capacity $C_v^{\max}$ of vehicle $v \in V$. Thus the equation

$$\sum_{i \in R^{dv}} d_i \leq C_v^{\max} \quad \forall R^{dv} \in s \tag{4.13}$$

must hold for each route $R^{dv} \in s$.

4.4 Objective Function

Before the overall objective function f for feasible solutions $s \in S$ of the SSDVRPTW is discussed, the route objective function to evaluate a single route R^{dv} will be considered. Function

$$f_{\mathrm{R}}\left(R^{dv}, Y_0^{dv}\right) = \sum_{(i,j) \in R^{dv}} \left(\alpha\, c_{i,j}^{\mathrm{dist}} + (1-\alpha)\, c_{i,j}^{\mathrm{time}}\right) \tag{4.14a}$$

$$+ \sum_{i \in R^{dv}} \left(\beta\, W_i^{dv} + \gamma\, L_{dvi}^{\mathrm{cus}}\right) \tag{4.14b}$$

$$+ \delta\, L_{dv}^{\mathrm{veh}} + \varepsilon\, O^{dv} \tag{4.14c}$$

consists of four terms and maps a route $\left(R^{dv}, Y_0^{dv}\right) \in s$ together with the corresponding start time Y_0^{dv} to a single real objective value. Term (4.14a) computes the total travel costs consisting of the total driving time and the number of meters driven. The parameter $\alpha \in [0,1]$ describes in what extent the costs should be taken into account. If $\alpha = 1$, only the distance costs will be minimized. However, if $\alpha = 0$, only the driving time costs will be minimized. The second term (4.14b) penalizes the total waiting time by parameter β as well as the total customer time window violation by parameter γ. The third and forth term (4.14c) calculate the vehicle time window violation and the overtime of route R^{dv} which are weighted by the parameters δ and ϵ, respectively.

The overall objective function $f : S \mapsto \mathbb{R}$ maps a feasible solution s to a single real valued objective value. Thus, the function

$$f(s) = \sum_{\left(R^{dv}, Y_0^{dv}\right) \in s} f_{\mathrm{R}}\left(R^{dv}, Y_0^{dv}\right) \tag{4.15a}$$

$$+ \eta \sum_{j \in N'} \left(1 - \mathrm{isServed}\left(j\right)\right) \tag{4.15b}$$

$$+ \theta \sum_{R^{dv} \in s} \left(1 - \mathrm{isEmpty}\left(R^{dv}\right)\right) \tag{4.15c}$$

is the additive sum (4.15a) of all routing objective values together with two additional terms. Term (4.15b) penalizes each customer which is not visited by any vehicle on any day. The function $\mathrm{isServed} : N' \mapsto \{0,1\}$,

$$\mathrm{isServed}\left(i\right) = \begin{cases} 1 & \text{if } \exists R^{dv} \in s \text{ s.t. } i \in R^{dv} \\ 0 & \text{otherwise} \end{cases} \tag{4.16}$$

maps each customer $i \in N'$ to one if i is included into any route of solution s and zero otherwise. Term (4.15c) penalizes each used vehicle $v \in V$ for each day $d \in D$ by parameter θ. Vehicle v is used on day d, if the corresponding route R^{dv} is not *empty*. The function isEmpty : $s \mapsto \{0, 1\}$ maps each route $R^{dv} \in s$ to one if

$$\text{isEmpty}\left(R^{dv}\right) = \begin{cases} 1 & \text{if } \exists\,(0, n_c + 1) \in R^{dv} \\ 0 & \text{otherwise} \end{cases} \tag{4.17}$$

the arc $(0, n_c + 1)$ is included into route R^{dv} and zero otherwise.

Table 4.1 listed all used penalty factors in objective functions (4.15) and (4.14). The setting minimizes the total travel time rather than the total distance.

α	Distance and time factor
β	Penalty factor for the waiting time
γ	Penalty factor for customer time window violations
δ	Penalty factor for vehicle time window violations
ε	Penalty factor for exceeding the maximum tour duration
η	Penalty factor for the number of unvisited customers
θ	Penalty factor for each time a vehicle is used

Table 4.1: Penalty Factors used in Objective Functions f (4.15) and f_{R} (4.14).

4.5 Dynamic Problem

The DSDVRPTW $\breve{P} = (\breve{S}, \breve{f})$ is modeled as a series of SSDVRPTWs. Hence, in order to solve an instance of the DSDVRPTW a sequence of instances $P_k(S_k, f)$ of the SSDVRPTW with $k \geq 0$ has to be solved such that a solution $s_k \in S_k$ of an instance $P_k(S_k, f)$ is the initial solution of the next instance $P_{k+1}(S_{k+1}, f)$. Before the next instance $P_{k+1}(S_{k+1}, f)$ can be solved, there are usually some modifications regarding the problem parameters. For instance the time window of a customer could be changed or a customer should only be delivered by a certain vehicle. Moreover the current planning horizon could be shifted such that new unscheduled customers enter the planning horizon and other customers leave the planning horizon. Note, that this definition of the DSDVRPTW differs from the definition of the DVRP from the literature. Since the DSDVRPTW is modeled as a series of SSDVRPTWs, there does not occur any *real time* planning. The time between the solving of two consecutive instances of the SSDVRPTW is not restricted.

Figure 4.2 visualizes the shift of the planning horizon. In this example the planning horizon consists of six days and is shifted day by day ($\breve{D}_1$-$\breve{D}_4$). Each customer $i \in N' = \{1, 2, 3, 4\}$ has a different set of allowed delivery days D_i. If $D_i \cap \breve{D}_k \neq \emptyset$, then customer i is included into the current static problem instance $P_k(S_k, f)$.

Usually the first static problem instance starts with customers whose time window is set to [0, ∞], meaning that the solver should plan optimal routes regarding logistic costs.

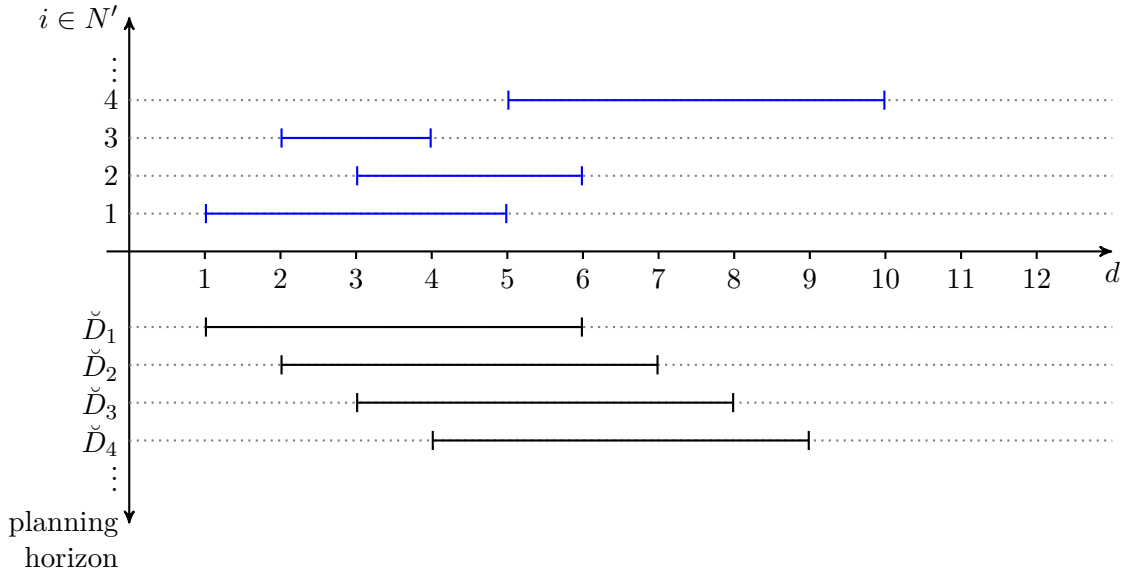


Figure 4.2: Example about the day by day shift of the planning horizon of an instance of the SDVRPTW.

Before the second static problem is solved, the customers are fixed such that a tight time window is set according to the solution from the first static problem instance.

CHAPTER 5

Mixed Integer Programming Approaches

In this chapter a Mixed Integer Program (MIP) will be described to solve instances of the SSDVRPTW in order to analyze the solution quality of the VNS algorithm presented in chapter 6.

Section 5.1 gives a brief introduction into the field of integer linear programming. Then in Section 5.2 and 5.3 a compact mathematical programming formulation will be presented for the SSDVRPTW. Section 5.4 describes a B&C algorithm together with used cutting-planes.

5.1 Introduction

For a deep insight into linear integer programming we refer to [NW88a] and [Lee04a].

A Linear Program (LP)

$$\min \mathbf{c}^{\mathrm{T}}\mathbf{x} \tag{5.1a}$$
$$\text{s.t. } \mathbf{A}\mathbf{x} \leq \mathbf{b} \tag{5.1b}$$
$$\mathbf{x} \in \mathbb{R}^n \tag{5.1c}$$

consists of a vector $\mathbf{x} \in \mathbb{R}^n$ of n variables together with vectors $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{c} \in \mathbb{R}^n$ of coefficients and a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ of coefficients. The task is to determine vector $\mathbf{x}$ such that the objective function (5.1a) is minimized taking into consideration m linear inequalities (5.1b). Those linear inequalities define geometrically a feasible region which is a convex polyhedron. Since the objective function is also linear, therefore also convex it can be shown that every local minimum is a global minimum. Assuming that a

solution exists, the solution must be on the border of the polyhedron formed by the linear inequalities (5.1b).

LPs in the form of (5.1) are first independently studied by Leonid Kantorovich and George B. Dantzig in 1939 and 1946, respectively [Dan02]. Dantzig invented the *simplex* algorithm which can in most cases efficiently solve LPs.

Many practical problems can be formulated as a LP. However, to formulate COPs, a more general formulation is needed since COPs are dealing with discrete variables and constraint (5.1c) allows only real-valued variables. Therefore a MIP

$$\min \mathbf{c}_1^{\mathrm{T}}\mathbf{x}_1 + \mathbf{c}_2^{\mathrm{T}}\mathbf{x}_2 \tag{5.2a}$$

$$\text{s.t. } \mathbf{A}_1\mathbf{x}_1 + \mathbf{A}_2\mathbf{x}_2 \leq \mathbf{d} \tag{5.2b}$$

$$\mathbf{x}_1 \in \mathbb{Z}^{n_1} \tag{5.2c}$$

$$\mathbf{x}_2 \in \mathbb{R}^{n_2} \tag{5.2d}$$

consists in addition to a vector $\mathbf{x}_2 \in \mathbb{R}^{n_2}$ of n_2 real-valued variables, of a vector $\mathbf{x}_1 \in \mathbb{Z}^{n_1}$ of n_1 *discrete* variables. Furthermore the MIP contains the vectors $\mathbf{c}_1 \in \mathbb{R}^{n_1}$, $\mathbf{c}_2 \in \mathbb{R}^{n_2}$ and $\mathbf{d} \in \mathbb{R}^m$ of coefficients together with the matrices $\mathbf{A}_1 \in \mathbb{R}^{m \times n_1}$ and $\mathbf{A}_2 \in \mathbb{R}^{m \times n_2}$ of coefficients. Since the variables of vector $\mathbf{x}_1$ are discrete, the feasible region is restricted to discrete points within the polyhedron formed by the m inequalities (5.2b). This makes the MIP $\mathcal{NP}$-hard.

If the real-valued variables in vector $\mathbf{x}_2$ are omitted by setting $\mathbf{A}_2 = \mathbf{0}^{m \times n_2}$, and $\mathbf{c}_2 = \mathbf{0}^{n_2}$, the LP is called Integer Program (IP). Furthermore if the discrete variables are restricted to $\{0, 1\}$ the program is called Binary Integer Program (BIP).

One way to exactly solve an instance of a MIP is the combination of a B&B approach together with *relaxations* and *duality*.

5.1.1 Relaxation

One of the key techniques in integer programming are relaxations, where some constraints (5.2b)-(5.2d) are omitted to obtain simpler problems. Those simpler problems can hopefully be solved efficiently to get lower bounds which can be used in the B&B algorithm to cut away branches of the B&B search tree.

The obtained solutions of the simpler problem need not be feasible regarding the original problem. Therefore one relaxation is to neglect the discrete value constraint (5.2c) and change the domain of vector $\mathbf{x}_1$ from $\mathbb{Z}^{n_1}$ to $\mathbb{R}^{n_1}$. The resulting problem is a LP which can be efficiently solved with the *simplex* algorithm to obtain a lower bound of the original problem.

5.1.2 Duality

The dual problem to a LP (5.1) is defined by

$$\min \mathbf{b}^{\mathrm{T}}\mathbf{y} \tag{5.3a}$$

$$\text{s.t. } \mathbf{A}^{\mathrm{T}}\mathbf{y} \leq \mathbf{c} \tag{5.3b}$$

$$\mathbf{y} \in \mathbb{R}^m \tag{5.3c}$$

with the corresponding dual vector $\mathbf{y} \in \mathbb{R}^m$ consisting of m variables. While the relaxation of a MIP to a LP provides a lower bound for the original MIP, the dual problem of the relaxation provides an upper bound for the MIP. This upper bound can be used by the B&B algorithm to cut away branches of the B&B search tree.

5.1.3 Branch and Bound

The B&B algorithm already mentioned in chapter 3 is an exact technique to solve a MIP based on the *divide and conquer* principle. This is done by solving the LP relaxation of the MIP. The solution of the LP relaxation will contain some variable x_i with fractional variable values $\widetilde{x}_i$. One way of branching is to create two new subproblems by adding to subproblem one the inequality $x_i \leq \lfloor \widetilde{x}_i \rfloor$ and to subproblem two the inequality $x_i \geq \lceil \widetilde{x}_i \rceil$. For each new subproblem the corresponding LP relaxation as well as the dual LP is solved. Then the B&B algorithm checks if the branch could be pruned due to the obtained lower and upper bounds, otherwise the subproblem will be split again into two new subproblems.

5.2 Objective Function and Basic Constraints

In order to solve even small instances of the DSDVRPTW exactly, the solution space S must be reduced. Therefore we change some soft constraints to hard constraints by setting the penalty factors γ, δ and ε from Table 4.1 to infinity. Hence, a solution of the following MIP formulation is feasible if

$$L_{dvi}^{\text{cus}} = 0 \quad \forall d \in D, \forall v \in V, \forall i \in N', \tag{5.4}$$

$$L_{dv}^{\text{veh}} = 0 \quad \forall d \in D, \forall v \in V \text{ and} \tag{5.5}$$

$$O^{dv} = 0 \quad \forall d \in D, \forall v \in V \tag{5.6}$$

holds, or, in other words if there are not any time window violations, vehicle time window violations or maximum tour duration violations. Furthermore by setting η to infinity it will be claimed that all customers are served by one vehicle. Moreover the objective function should not consider waiting times and open route costs. Therefore β and θ are set to zero.

To reduce the complexity further, it is assumed that the planning horizon $D = \{1\}$ consists of only one day. Therefore, the day index d will be neglected in this chapter.

This restrictions lead to the following MIP formulation

$$\min \sum_{v \in V} \sum_{i,j \in N} \left(\alpha \, c_{i,j}^{\text{dist}} + (1 - \alpha) \, c_{i,j}^{\text{time}} \right) X_{ij}^{v} \tag{5.7}$$

subject to

$$\sum_{v \in A_j} \sum_{i \in N \setminus \{n_c+1\}} X_{ij}^v = 1 \qquad \forall j \in N' \tag{5.8}$$

$$\sum_{i \in N \setminus \{n_c+1\}} X_{ij}^v - \sum_{i \in N \setminus \{0\}} X_{ji}^v = 0 \qquad \forall j \in N', \forall v \in A_j \tag{5.9}$$

$$\sum_{v \notin A_j} \sum_{i \in N} X_{ij}^v + X_{ji}^v = 0 \qquad \forall j \in N' \tag{5.10}$$

$$\sum_{j \in N'} X_{0j}^v = Z^v \qquad \forall v \in V \tag{5.11}$$

$$\sum_{j \in N'} d_j \sum_{i \in N} X_{ij}^v - Z^v C_v^{\max} \leq 0 \qquad \forall v \in V \tag{5.12}$$

$$\sum_{v \in V} \sum_{i \in N \setminus \{n_c+1\}} \sum_{j \in N \setminus \{0\}} X_{ij}^v = \sum_{v \in V} Z^v + |N'| \tag{5.13}$$

$$\sum_{v \in V} X_{ij}^v = 0 \qquad \forall i \in N', \forall j \in N', \quad e_i^{\text{cus}} + s_i + c_{ij}^{\text{time}} > l_j^{\text{cus}} \tag{5.14}$$

$$\sum_{(i,j) \in P} X_{ij}^v \leq |P| - 1 \qquad \forall \text{ invalid paths } P \subseteq A \text{ of } v \in V \tag{5.15}$$

$$\sum_{v \in V} X_{ii}^v + X_{i0}^v + X_{(n_c+1)i}^v = 0 \qquad \forall i \in N \tag{5.16}$$

$$\sum_{i \in N} X_{ij}^{v_1} - \sum_{k=1}^{k<j} \sum_{i \in N} X_{ik}^{v_2} \leq 0 \qquad \forall j \in N', \forall v_1, v_2 \in V, Q_{v_1}^{\text{pro}} = Q_{v_2}^{\text{pro}}, v_2 < v_1 \tag{5.17}$$

$$X_{ij}^v \in \{0,1\} \qquad \forall v \in V, \forall i,j \in N \tag{5.18}$$

$$Z^v \in \{0,1\} \qquad \forall v \in V \tag{5.19}$$

with the objective function (5.7) consisting of the total driving time and the number of meters driven. The parameter $\alpha \in [0,1]$ describes in what extent the costs should be taken into account. If $\alpha = 1$, only the distance costs will be minimized. On the other hand, if $\alpha = 0$, only the driving time costs will be minimized.

Constraint (5.8) restricts the number of visits to a customer to at most one visit over all vehicles. Furthermore constraint (5.9) ensures that each vehicle which visits a customer, leaves that customer again while (5.10) forbids vehicles to visit customers if their qualifications do not match. Equation (5.11) puts variable Z^v on par with the number of routes of vehicle $v \in V$. Hence, variable Z^v is set to one if vehicle $v \in V$ is used and set to zero otherwise. The maximum load capacity $C_v^{\max}$ of each vehicle $v \in V$ is hold through constraint (5.12). Constraint (5.14) forbids arcs that are a priori invalid due to a time window violation.

Since there are vehicles which provide the same qualification, there are many solutions which are in fact identical, except two vehicles are exchanged. This symmetry is prohibited by the symmetry breaking constraint (5.17). The idea is to define an ordering between customers as well as between vehicles. A vehicle $v \in V$ can only visit a customer $j \in N'$ if there is at least one vehicle $v' \in V$ such that $v' < v$ and v' provides the same qualifications as vehicle v and visits a customer $k \in N'$ such that $k < j$ holds.

Note, that constraint (5.15) produce exponentially many inequalities to prohibit subtours and time window violations. Since this is only for small instances practicable, in the following sections, there are different approaches presented to avoid adding exponentially many inequalities.

5.3 Miller-Tucker-Zemlin Constraints

The Miller-Tucker-Zemlin based formulation [MTZ60]

$$Y_i^v + s_i + c_{ij}^{\text{time}} - Y_j^v - \left(1 - X_{ij}^v\right) M_{ij} \leq 0 \qquad \begin{array}{l} \forall i \in N \setminus \{n_c + 1\}, \\ \forall j \in N \setminus \{0\},\ i \neq j, \\ \forall v \in A_i \cap A_j, \\ e_i^{\text{cus}} + s_i + c_{ij}^{\text{time}} \leq l_j^{\text{cus}} \end{array} \tag{5.20}$$

$$s_v^{\text{veh}} \leq Y_0^v \leq e_v^{\text{veh}} \qquad \forall v \in V \tag{5.21}$$

$$s_v^{\text{veh}} \leq Y_{n_c+1}^v \leq e_v^{\text{veh}} \qquad \forall v \in V \tag{5.22}$$

$$Y_0^v - Y_{n_c+1}^v \leq 0 \qquad \forall v \in V \tag{5.23}$$

$$Y_{n_c+1}^v - Y_0^v - Z^v D^{\max} \leq 0 \qquad \forall v \in V \tag{5.24}$$

$$e_i^{\text{cos}} - Y_i^v \leq 0 \qquad \forall v \in V, \forall i \in N' \tag{5.25}$$

$$Y_i^v - l_i^{\text{cus}} \sum_{j \in N} X_{ij}^v - e_i^{\text{cus}} \left(1 - \sum_{j \in N} X_{ij}^v\right) \leq 0 \qquad \forall v \in V, \forall i \in N' \tag{5.26}$$

$$Y_i^v \in \mathbb{R} \qquad \forall v \in V, \forall i \in N \tag{5.27}$$

uses the variables Y_i^v to define an ordering between customer $i \in N \setminus \{n_c + 1\}$ and customer $j \in N \setminus \{0\}$ to avoid subtours. Furthermore variables Y_i^v represent the start of service time of vehicle $v \in V$ by customer $i \in N'$, whereas Y_0^v and $Y_{n_c+1}^v$ represent the start time from the depot and the arrival time at the depot, respectively. Hence, if arc X_{ij}^v is used, the start of service time Y_j^v of customer $j \in N \setminus \{0\}$ must be greater or equal to the start of service time Y_i^v of customer $i \in N \setminus \{n_c + 1\}$ plus the service time s_i of customer i and the travel time c_{ij}^{time} from customer i to customer j. If arc X_{ij}^v is not used, the constant

$$M_{ij} = \max\left(l_i^{\text{cus}} + s_i + c_{ij}^{\text{time}} - e_j^{\text{cus}}, 0\right) \tag{5.28}$$

ensures that constraint (5.20) is always fulfilled.

Constraints (5.21)-(5.22) make sure that the start time from the depot Y_0^v and the arrival time at the depot $Y_{n_c+1}^v$ are within the corresponding time window of vehicle $v \in V$. Furthermore constraint (5.23) takes care that each vehicle $v \in V$ starts from the depot before it arrives at the depot. The tour duration $Y_{n_c+1}^v - Y_0^v$ is limited by the maximum tour duration $D^{\max}$ through constraint (5.24). Time window violations are prohibited by constraints (5.25)-(5.26).

5.4 Cutting Plane Separation

Another approach to prohibit subtours and time window violations is the usage of cutting planes. Cutting planes are basically linear inequalities which are not initially added to the MIP, but dynamically separated. Those linear inequalities are also called *cuts*. At each node at the B&B-tree those cuts are applied to the current LP to cut away invalid IP solutions or to strengthen the LP-relaxation. The overall procedure is called Branch & Cut (B&C).

The tasks of the cutting planes developed for the program (5.7)-(5.19) are to eliminate subtours and time window violation as well as to eliminate maximum tour duration violation. Furthermore one cutting plane tries to strengthen the LP-relaxation.

5.4.1 Subtour Elimination

We consider here two subtour elimination approaches [NW88b], [Lee04b]. The cycle elimination cuts are based on the idea that if there is a path from the end point to the start point of an arc, then there is an invalid subtour. This subtour is then prohibited by adding an additional constraint to the MIP. The directed connection cuts separate cycle elimination cuts based on the max-flow min-cut theorem.

Cycle Elimination Cuts

The constraints

$$\sum_{(i,j)\in C} X_{ij}^v \leq |C| - 1 \quad \forall v \in V, \forall \text{ cycles } C \in G \tag{5.29}$$

are sufficient to enforce that an IP-solution of program (5.7)-(5.19) does not contain any cycles. Unfortunately there are exponentially many inequalities (5.29), such that adding them initially to the problem is only possible for very small instances. Therefore the constraints (5.29) are added dynamically during the B&B procedure.

At each B&B-node Algorithm 5.1 is applied to the current LP solution to find invalid cycles and cut them away. This is done by associating to each arc $(i, j) \in A$ of the routing graph $G = (N, A)$ a weight

$$w_{ij} := 1 - \widetilde{X}_{ij}^v \quad \forall (i,j) \in A \tag{5.30}$$

where $\widetilde{X}_{ij}^v$ denotes the current fractional value of variable X_{ij}^v per vehicle $v \in V$. Next, Algorithm 5.1 tries to find for each arc $(i, j) \in A$ the shortest path p from j to i. If such

a path exists and the sum of the fractional values is greater than the length of the path minus one, Algorithm 5.1 has found an invalid cycle and adds a corresponding constraint

$$X_{ij}^v + \sum_{(i',j') \in p} X_{i'j'}^v \leq |C| - 1 \tag{5.31}$$

with $C = p \cup \{(i,j)\}$ to forbid the cycle.

Algorithm 5.1: Add Cycle Elimination Constraints

Input: A LP-solution s_{LP}, a routing graph $G = (N, A)$

```
foreach v ∈ V do
    foreach (i, j) ∈ A do
        w_ij ← 1 − X̃^v_ij;
    end
    foreach (i, j) ∈ E do
        if X̃^v_ij ≤ 0.5 then continue;
        Find shortest path p from j to i;
        if X̃^v_ij + Σ_{(i',j')∈p} X̃^v_{i'j'} > |p| − 1 then
            Add constraint X^v_ij + Σ_{(i',j')∈p} X^v_{i'j'} ≤ |p|;
        end
    end
end
```

Directed Connection Cuts

Algorithm 5.2 solves for each vehicle $v \in V$ and for each customer $j \in N'$ the max-flow min-cut problem. This is done by creating a flow network such that each arc $(i', j') \in A$ of the routing graph $G = (N, A)$ is associated with a capacity

$$c_{i'j'} = \widetilde{X}_{i'j'}^v \quad \forall (i', j') \in A \tag{5.32}$$

where $\widetilde{X}_{i'j'}^v$ denotes the current fractional value of variable $X_{i'j'}^v$. Furthermore the depot is set as sink and customer j is set as target. Next Algorithm 5.2 calculates the minimum cut set C with capacity $c(C)$. If the capacity $c(C)$ is smaller than the sum of the flow entering customer j, then an invalid subtour is detected and can be prohibited by adding constraint

$$\sum_{(i'j') \in \delta^+(C)} X_{i'j'}^v >= \sum_{i \in N \setminus \{0\}} X_{ij}^v \tag{5.33}$$

to the MIP [NW88b], [Lee04b].

Algorithm 5.2: Add Directed Connection Cuts

Input: A LP-solution s_{LP}, a routing graph $G = (N, A)$

foreach $(i, j) \in A$ **do**
  $w_{ij} \leftarrow \widetilde{X}^v_{ij}$;
end
foreach $v \in V$ **do**
  foreach $j \in N'$ **do**
    $f \leftarrow \sum_{i \in N \setminus \{0\}} \widetilde{X}^v_{ij}$;
    if $f = 0$ **then continue**;
    Get minimum cut set C with the depot as source and customer j as target;
    if $c(C) \geq f$ **then continue**;
    Add constraint $\sum_{(i'j') \in \delta^+(C)} X^v_{i'j'} >= \sum_{i \in N \setminus \{0\}} X^v_{ij}$;
  end
end

5.4.2 Time Window Cuts

In order to detect and eliminate time window violations, Algorithm 5.3 is applied to each LP-solution at each node of the B&B-tree. Algorithm 5.3 applies a Depth-First Search (DFS) for each vehicle $v \in V$ and each node $i \in N$ in the routing graph $G = (N, A)$.

During the DFS, at each expanded node k the start of service time t is calculated with respect to the current root node i. If t is between the customers time window $[e^{\text{cus}}_k, l^{\text{cus}}_k]$, then node k will be further expanded. Hence k is pushed on the stack s. Otherwise, if t exceeds the latest possible service time l^{cus}_k, then a time window violation is detected. The path p from the start node i to the current expanded node k is reconstructed from the predecessor array *pred*. Since this path leads to a time window violation, the path is prohibited by adding the constraint

$$\sum_{(i',j') \in p} X^v_{i'j'} \leq |p| - 1. \tag{5.34}$$

5.4.3 Maximum Tour Duration Cuts

In order to detect routes which are violating the maximum tour duration constraint, algorithm 5.4 is applied at each LP-solution.

As in Section 5.4.1 each arc $(i, j) \in A$ from the routing graph $G = (N, A)$ is associated with weight w_{ij} (5.30). Next, algorithm 5.4 calculates the shortest path r from the start depot to the end depot. If the duration of route r exceeds the maximum tour duration

Algorithm 5.3: Add Time Window Violation Constraints

Input: A LP-solution s_{LP}, a routing graph $G = (N, A)$
Data: Predecessor array $pred[v]$, Stack s

foreach $v \in V$ **do**
  foreach $i \in N$ **do**
    `Init(`s`);`
    `Push(`s`,` (i, e_i^{cus})`);`
    $pred[i] \leftarrow i$;
    while ¬`IsEmpty(`s`)` **do**
      $(j, t) \leftarrow$`Pop(`s`);`
      $t \leftarrow t + s_j$;
      foreach $k \in \delta^+(j)$ **do**
        if $\widetilde{X}^v_{jk} = 0.0$ **then continue**;
        if k *is already discovered* **then continue**;
        if $e_k^{\text{cus}} \leq t + c_{jk}^{\text{time}} \leq l_k^{\text{cus}}$ **then**
          `Push(`s`,` $(k, t + c_{jk}^{\text{time}})$`);`
          $pred[k] \leftarrow j$;
        else if $t + c_{jk}^{\text{time}} > l_k^{\text{cus}}$ **then**
          Extract path p from i to k from $pred$;
          Add constraint $\sum_{(i',j') \in p} X^v_{i'j'} \leq |p| - 1$;
      end
    end
  end
end

$D^{\max}$, then r is prohibited by adding the constraint

$$\sum_{(i',j') \in r} X^v_{i'j'} \leq |r| - 2 \tag{5.35}$$

to the MIP.

5.4.4 Packing Constraints Strengthening

This section describes the strengthening of packing constraint (5.12). A set $I \subset N'$ is called a *cover* if the condition

$$\sum_{i \in C} d_i > C_v^{\max} \tag{5.36}$$

Algorithm 5.4: Add Maximum Tour Duration Constraints

Input: A LP-solution s_{LP}, a routing graph $G = (N, A)$

foreach $v \in V$ **do**
  foreach $(i, j) \in A$ **do**
    $w_{ij} \leftarrow 1 - \widetilde{X}^v_{ij}$;
  end
  Find shortest path $r = (\tau_0 = 0, \tau_1, \tau_2, \ldots, \tau_{|r|} = n_c + 1)$ from start depot 0 to end depot $n_c + 1$;
  Calculate start time from the depot Y^v_0;
  Calculate arrival time at the depot $Y^v_{n_c+1}$;
  if $Y^v_{\tau_{n_c+1}} - Y^v_0 > D^{\max}$ **then**
    Add constraint $\sum_{(i',j')\in r} X^v_{i'j'} \leq |r| - 2$;
  end
end

for some vehicle $v \in V$ holds. Since constraint (5.12) will be violated if all customers of I are visited by vehicle v, the inequality

$$\sum_{i \in I} \sum_{j \in N \setminus \{0\}} X^v_{ij} \leq |C| - 1 \tag{5.37}$$

will strengthen the LP-relaxation. Inequalities (5.37) are called Cover Inequalitys (CI) and were simultaneously and independently introduced by [Wol75], [HJP75] and [Bal75]. A cover I is called *minimal* if the cardinality of I is minimal such that (5.36) still holds. Minimal covers yield to stronger CIs than with non minimal covers.

Heuristic 5.5 computes such minimal covers for each LP-solution s_{LP}. First, all customers $i \in N'$ are sorted in ascending order according to $d_i \sum_{j \in N \setminus \{0\}} \widetilde{X}^v_{ij}$. Next, the sorted list is iterated, such that all customers are added to cover I until condition (5.36) is satisfied. If the corresponding sum of the fractional variable values exceeds $|I| - 1$, the CI (5.37) is added to strengthening the LP-relaxation.

Algorithm 5.5: Add Minimal Cover Inequalities

Input: A LP-solution s_{LP}, a routing graph $G = (N, A)$

foreach $v \in V$ **do**
  $\bar{N} \leftarrow$ sorted list $i \in N'$ in ascending order according to $d_i \sum_{j \in N \setminus \{0\}} \widetilde{X}_{ij}^v$;
  $I \leftarrow \emptyset$;
  foreach $i \in \bar{N}$ **do**
    $I \leftarrow I \cup \{i\}$;
    if $\sum_{j \in I} d_j > C_v^{\max}$ **then break**;
  end
  if $\sum_{i \in I} \sum_{j \in N \setminus \{0\}} \widetilde{X}_{ij}^v > |I| - 1$ **then**
    Add constraint $\sum_{i \in I} \sum_{j \in N \setminus \{0\}} X_{ij}^v \leq |I| - 1$;
  end
end

CHAPTER 6

Variable Neighborhood Search

This chapter describes the VNS algorithm to tackle the SSDVRPTW. Section 6.1 presents local search neighborhoods which are used within the descent phase of the VNS. The shaking neighborhoods are described in Section 6.2. Overall there are seven different shaking neighborhoods developed for the VNS. The ordering of execution of the shaking neighborhoods is described in Section 6.3.

Algorithm 6.1 gives an overview of the VNS algorithm. The shaking neighborhood structure $\mathcal{N}_k$ will be described in Section 6.3 together with the shaking neighborhoods in Section 6.2. At the beginning of the loop, a neighbor solution s' will be randomly selected from the shaking neighborhood $\mathcal{N}_k(s_{best})$ of the current incumbent solution s_{best}. In the next step, the selected solution s' will be improved by local search procedures to obtain solution s''. The neighborhoods used in the local search procedures are described in Section 6.1. If the objective value of s'' is less than the objective value of s_{best}, a new incumbent solution is found. One more local search procedure will be applied to s'' with respect to neighborhood $\mathcal{N}_{2\text{opt}*}$. The obtained solution s''' will be assigned to s_{best} and the shaking will be reset to $k = 1$. Because of the large size of neigborhood $\mathcal{N}_{2\text{opt}*}$ the local search will be applied only if a new best solution is found rather than applying the local search in every step of the VNS.

6.1 Local Search Neighborhoods

For the local search procedures we use the well known 2-opt and 3-opt neighborhoods which will be described in more detail in Section 6.1.1 and 6.1.2. Both neighborhoods improve only one route of solution s' in Algorithm 6.1. Therefore the local search procedure is applied to every changed route during the shaking phase. Additionally every new best solution is improved by a local search procedure using the inter-route exchange neighborhood 2-opt* introduced by Potvin and Rousseau [JYP95].

Algorithm 6.1: VNS

Input: An initial solution s
Output: A probably improved solution s'

```
1  s_best ← s;
2  k ← 1;
3  for stopping criteria satisfied do
4  |  Select s' ∈ N_k(s_best);
5  |  s'' ← LocalSearch(s', N_3opt);
6  |  s'' ← LocalSearch(s'', N_2opt);
7  |  if f(s'') < f(s_best) then
8  |  |  s''' ← LocalSearch(s'', N_2opt*);
9  |  |  s_best = s''';
10 |  |  k ← 1;
11 |  else
12 |  |  k ← k + 1;
13 |  end
14 |  if k > k_max then
15 |  |  k ← 1;
16 |  end
17 end
18 return s_best;
```

Each local search procedure uses first improvement, hence, they do not scan the whole neighborhood but rather stop the search when the first improved solution is found.

6.1.1 2-opt Neighborhood

A move in the 2-opt neighborhood is performed by removing two edges from a route and connecting the remaining two paths by adding two new edges such that a new route arises. Figure 6.1 shows an example of a possible neighbor solution $s' \in \mathcal{N}_{2\text{opt}}(s)$ from solution s. The two red arcs in solution s are replaced by two green arcs.

$s \in S$ $\quad$ $s' \in \mathcal{N}_{2\text{opt}}(s)$

Figure 6.1: 2-opt Neighborhood Example

6.1.2 3-opt Neighborhood

A move in the 3-opt neighborhood is performed by removing three edges from a route and connecting the remaining three paths by adding three new edges such that a new route arises. On the contrary to the 2-opt neighborhood, where there is only one way to get a new route from the remaining segments by adding new edges, the 3-opt neighborhood has more than one option to get a new route from the remaining segments. The local search strategy is to choose randomly one option at the start of the search and use this option during the whole search. Figure 6.2 shows one possible neighbor solution s' from the 3-opt neighborhood $\mathcal{N}_{3\text{opt}}(s)$ of solution s.

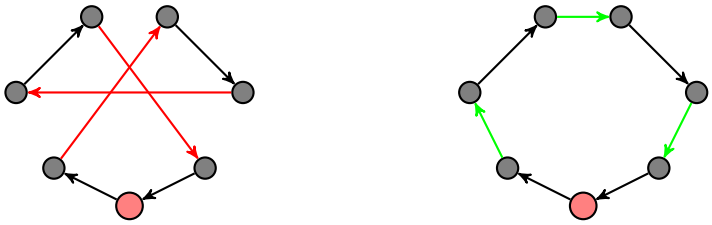


Figure 6.2: 3-opt Neighborhood Example

6.1.3 2-opt* Neighborhood

The 2-opt* Neighborhood $\mathcal{N}_{2\text{opt}*}(s)$ combines two routes by cutting each route into two segments of customers. The resulting second segment of one route is exchanged with the second segment of the other route. Figure 6.3 shows an example of a possible 2-opt* move. The two routes are cut by the red edges and the resulting segments with the green pigmented vertices are exchanged. In our application for each pair of routes starting on the same day all possible cuts and exchanges are tried.

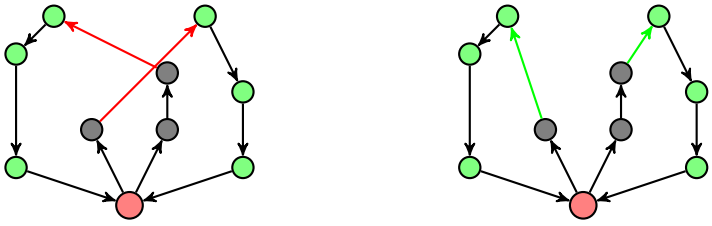


Figure 6.3: 2-opt* Neighborhood Example

6.2 Shaking Neighborhoods

The shaking neighborhood structure used in Algorithm 6.1 and 6.10 consists of classical neighborhoods as well as very large neighborhoods. All used shaking neighborhoods will be described in detail in the following subsections.

6.2.1 Move Segment Neighborhood

The neighborhood $\mathcal{N}_{\text{move}}(s, k)$ includes all solutions which can be obtained by moving a segment of customers from one route $R^{dv} \in s$ to another route $R^{dv_2} \in s$ such that both routes start on the same day d. Parameter k describes the maximum length of the moved segment of customers. Figure 6.4 shows an example of a valid move operation with $k = 1$. Algorithm 6.2 describes the random selection of a neighbor solution s' from

Figure 6.4: Move Segment Neighborhood Example

the neighborhood $\mathcal{N}_{\text{move}}(s, k)$. After the selection of two different routes R^{dv_1} and R^{dv_2} such that both routes starts on the same day d, a segment $\delta \in R^{dv_1}$ is moved from route R^{dv_1} to route R^{dv_2}. The segment δ is inserted in a greedy way into R^{dv_2}.

Algorithm 6.2: Move Segment Neighborhood $\mathcal{N}_{\text{move}}(s, k)$

Input: A solution s, a parameter k
Output: A solution s'

Create a copy s' from solution s;
Choose randomly two different routes $R^{dv_1}, R^{dv_2} \in s'$ so that both routes are driven on the same day d;
repeat
  Choose randomly a segment $\delta = (\tau_i, \ldots, \tau_j)$ from route $R_1^{dv_1}$;
until $j - i \leq k$;
if v_2 *is not allowed to supply customers in* δ **then return** s ;
Erase δ from route $R_1^{dv_1}$;
Greedy insert δ into route $R_2^{dv_2}$;
return s';

6.2.2 Exchange Segment Neighborhood

The neighborhood $\mathcal{N}_{\text{exchange}}(s, k)$ contains all solutions which can be derived from solution s by selecting two routes R^{dv_1}, R^{dv_2} and exchange two segments $\delta_1 \in R^{dv_1}$ and $\delta_2 \in R^{dv_2}$. Parameter k limits the length of the segments δ_1 and δ_2. Figure 6.5 shows a possible neighbor solution $s' \in \mathcal{N}_{\text{exchange}}(s, 1)$ from solution s with $k = 1$. Hence, only two single customers from two different routes are exchanged. Algorithm 6.3 selects randomly

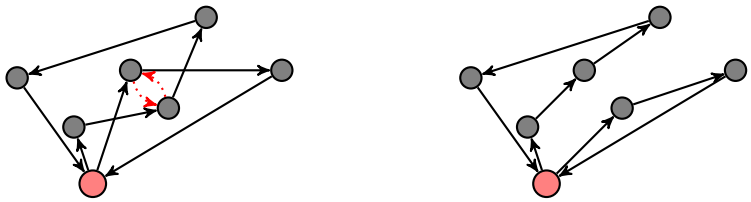


Figure 6.5: Exchange Segment Neighborhood Example

a neighbor solution s' from the Exchange Segment Neighborhood $\mathcal{N}_{\text{exchange}}(s, k)$ of solution s.

Algorithm 6.3: Exchange Segment Neighborhood $\mathcal{N}_{\text{exchange}}(s, k)$

Input: A solution s, parameter k
Output: A solution s'

Create a copy s' from solution s;
Choose randomly two different routes $R^{dv_1}, R^{dv_2} \in s'$ so that both routes are driven on the same day d;
repeat
  Choose randomly a segment $\delta_1 = (\tau_i, \ldots, \tau_j)$ from route R^{dv_1};
  Choose randomly a segment $\delta_2 = (\tau_{i'}, \ldots, \tau_{j'})$ from route R^{dv_2};
until $j - i \leq k$ ***and*** $j' - i' \leq k$;
if v_2 *is not allowed to supply customers in* δ_1 **then return** s ;
if v_1 *is not allowed to supply customers in* δ_2 **then return** s ;
Erase δ_1 from route R^{dv_1} and greedy insert δ_2;
Erase δ_2 from route R^{dv_2} and greedy insert δ_1;
return s';

6.2.3 Use Edge Neighborhood

The idea behind the Use Edge Neighborhood $\mathcal{N}_{\text{edge}}(s)$ is to use an arc which is not already used in solution s. Figure 6.6 shows an example of a possible neighbor solution $s' \in \mathcal{N}_{\text{edge}}(s)$. The red arc is not currently used in solution s and included in solution s'

such that the adjacent customers are both moved from their original routes to a different third route. Algorithm 6.4 describes how a random solution $s' \in \mathcal{N}_{\text{edge}}(s)$ is selected

Figure 6.6: Use Edge Neighborhood Example

from the Use Edge Neighborhood. After the selection of two customers n_1 and n_2 from two different non-empty routes R^{dv_1} and R^{dv_2}, the newly created segment $\delta = (n_1, n_2)$ is greedy inserted into one route of the set $R^d \setminus \{R^{dv_1}, R^{dv_2}\}$ under the condition that the qualification constraint is fulfilled for segment δ. The set R^d contains all routes which start on the same day $d \in D$.

Algorithm 6.4: Use Edge Neighborhood $\mathcal{N}_{\text{edge}}(s)$

Input: A solution s
Output: A solution s'

1 Create a copy s' from solution S;
2 Choose randomly two different routes $R^{dv_1}, R^{dv_2} \in s'$ so that both routes are driven on the same day d;
3 Choose randomly a customer n_1 from R^{dv_1} and a customer n_2 from R^{dv_2};
4 Create new segment $\delta = (n_1, n_2)$;
5 Remove customer n_1 from R^{dv_1} and n_2 from R^{dv_2};
6 Greedy insert δ into one route of $R^d \setminus \{R^{dv_1}, R^{dv_2}\}$;
7 **return** s';

6.2.4 Include Unserved Customers Neighborhood

Since we do not use a construction heuristic to create an initial solution for the VNS Algorithm 6.1, it is up to the Include Unserved Customers Neighborhood $\mathcal{N}_{\text{include}}(s)$ to fill up the solution with customers which are not already visited by a vehicle. Hence, the neighborhood $\mathcal{N}_{\text{include}}(s)$ is defined as the set of solutions which serves one more customer than solution s. If all customers are visited by a vehicle, the size of the neighborhood decreases to zero. Figure 6.7 shows an example of a possible neighbor solution $s' \in \mathcal{N}_{\text{include}}(s)$. In this example two customers are not scheduled in the current incumbent solution s. A possible neighbor solution s' has included one of the two unscheduled customers. Algorithm 6.5 selects randomly a neighbor solution

Figure 6.7: Include Unserved Customer Neighborhood Example

$s' \in \mathcal{N}_{\text{include}}(s)$. First the algorithm chooses randomly a not yet visited customer $n \in N'$. Customer n is greedy inserted into route R^{dv} such that the objective value is the smallest among all other routes and vehicle v is allowed to visit customer n.

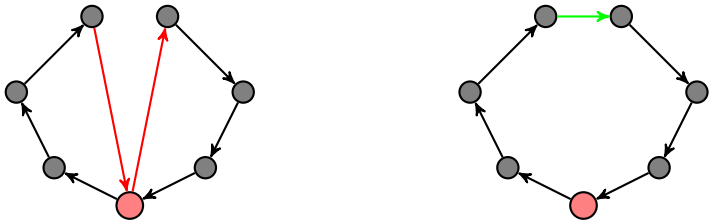
Algorithm 6.5: Include Unserved Customers Neighborhood $\mathcal{N}_{\text{include}}(s)$

Input: A solution s
Output: A solution s'
Create a copy s' from solution s;
Choose randomly an unserved customer $n \in N'$;
if *There are no unserved customer* **then**
  return *s;*
end
$R^{dv} \leftarrow$ empty;
$o \leftarrow \infty$; `// objective value`
foreach $R^{d'v'} \in s'$ **do**
  if v' *is not allowed to supply* n **then continue**;
  Greedy insert n into $R^{d'v'}$;
  if $f(R^{d'v'}) < o$ **then**
    $o \leftarrow f(R^{d'v'})$;
    $R^{dv} \leftarrow R^{dv'}$
  end
end
if R^{dv} *is not empty* **then**
  Replace R^{dv} in solution s;
end
return *s;*

6.2.5 Merge Routes Neighborhood

The Merge Routes Neighborhood $\mathcal{N}_{\text{merge}}(s)$ tries to reduce the number of used vehicles by merging two routes together. The neighborhood $\mathcal{N}_{\text{merge}}(s)$ contains all solution s'

which are obtained by appending all customers from one route $R^{dv_1} \in s$ to another route $R^{dv_2} \in s$ such that route R^{dv_1} gets empty. Figure 6.8 shows an example of such a neighbor solution $s' \in \mathcal{N}_{\text{merge}}(s)$ as well as Algorithm 6.6 describes how a neighbor solution s' is randomly selected.



$s \in S$ $\quad$ $s' \in \mathcal{N}_{\text{merge}}(s)$

Figure 6.8: Merge Routes Neighborhood Example

Algorithm 6.6: Merge Routes Neighborhood $\mathcal{N}_{\text{merge}}(s)$

Input: A solution s
Output: A solution s'

Create a copy s' from solution s;
Choose randomly two different routes $R^{dv_1}, R^{dv_2} \in s'$ such that both routes starting on the same day $d \in D$;
Append customers from route R^{dv_1} to route R^{dv_2};
if *vehicle* v_2 *is not allowed to supply customers in route* R^{dv_2} **then**
    return *s;*
else
    $R^{dv_1} \leftarrow$ empty;
    return *s′;*
end

6.2.6 Cyclic Exchange Neighborhood

Instead of exchanging only two different customers from two different routes, the cyclic exchange neighborhood $\mathcal{N}_{\text{cyclic}}(s, k, \delta_{\max})$ exchanges up to k different segments of customers from k different routes in a cyclic manner. The number of customers in one segment does not exceed the maximum segment length $\delta_{\max}$. Figure 6.9 shows an example of a possible neighbor solution s' of solution s.

The cyclic exchange neighborhood introduced by Thompson and Orlin [TO89] can be applied to any COP where a set $B = \{b_1, \ldots, b_n\}$ of n elements must be divided into m partitions $B_1, \ldots, B_m$, such that the objective function of each partition is independent. Simple neighborhoods would be a two-exchange neighborhood which does exchange

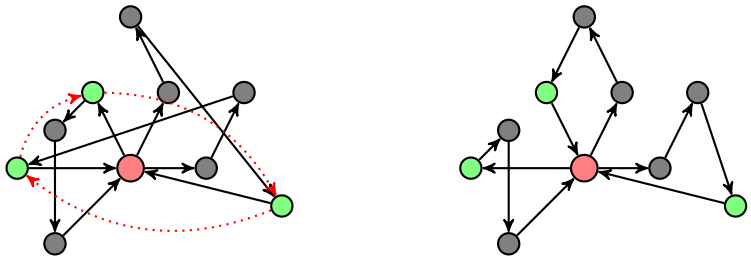


Figure 6.9: Cyclic Exchange Neighborhood Example

two elements of two partitions or a move neighborhood which moves an element from one partition to another partition. The cyclic exchange neighborhood can be seen as a generalization of the two-exchange neighborhood by exchanging up to k elements from k different partitions in a cyclic manner. The size of the cyclic exchange neighborhood $|\mathcal{N}_{\text{cyclic}}(s, k, \delta_{\max})| = \Omega(n^k)$ can be exponential growing if k is not fixed, whereas the two-exchange neighborhood has a size of $\Omega(n^2)$. Thompson and Orlin [TO89] and Thompson and Psaraftis [TP93] show how to find efficiently an improved neighbor solution by searching the cyclic exchange neighborhood using network flow techniques. In particular, a cyclic exchange move leads to an improved solution if a negative cost subset-disjoint cycle in an improvement graph is found. The following definition describes such an improvement graph for the SSDVRPTW.

Definition 6.2.1.
Let $P = (S, f)$ be the SSDVRPTW, $s \in S$ a solution, $R \subseteq s$ a subset of routes and δ_{max} the maximum length of segments. The improvement graph

$$G_{imp}\left(R, \delta_{max}\right) = \left(V_{imp}, A_{imp}\right) \tag{6.1}$$

is a weighted directed graph constructed from routes in R with vertex set

$$V_{imp} = V_{\delta} \cup V_{\emptyset} \cup \{\kappa\} \tag{6.2}$$

and arc set A_{imp}. The vertex set V_{imp} consists of the set of segments

$$V_{\delta} = \left\{ \left(R^{dv}, \delta\right), \left(R^{dv}, \delta'\right) \,\middle|\, R^{dv} \in R,\, \delta \subseteq R^{dv} \setminus \{0, n_c + 1\},\, |\delta| \leq |\delta_{max}| \right\} \tag{6.3}$$

together with the set of pseudo vertices

$$V_{\emptyset} = \left\{ \left(R^{dv}, \emptyset\right) \,\middle|\, R^{dv} \in R \right\} \tag{6.4}$$

and the source vertex κ. There is an arc $(v^1_{imp}, v^2_{imp}) \in A_{imp}$ between two vertices $v^1_{imp} \in V_{imp}$ and $v^2_{imp} \in V_{imp}$ if and only if one of the conditions

- $v^1_{imp} = (R^{d_1 v_1},\, \delta_1) \in V_\delta, v^2_{imp} = (R^{d_2 v_2}, \delta_2) \in V_\delta \cup V_\emptyset$ *s.t.*

$$d_1 \neq d_2 \,\wedge\, v_1 \neq v_2 \,\wedge\, \forall i \in \delta_1 : (d_2, v_2) \in A_i \,\wedge \sum_{i \in R^{d_2 v_2} \setminus \delta_2 \cup \delta_1} d_i \leq C^{max}_{v_2} \tag{6.5}$$

- $v^1_{imp} \in V_\emptyset,\, v^2_{imp} = \kappa$
- $v^1_{imp} = \kappa,\, v^2_{imp} \in V_\delta$

is fulfilled. Furthermore let $c : A_{imp} \to \mathbb{R}$ *be the cost function of an arc with*

$$c(v^1_{imp},\, v^2_{imp}) = \begin{cases} f_R\left(R^{d_2 v_2} \setminus \delta_2 \cup \delta_1\right) - f_R\left(R^{d_2 v_2}\right) & \text{if } v^1_{imp} \in V_\delta,\, v^2_{imp} \in V_\delta \cup V_\emptyset, \\ f_R\left(R^{d_2 v_2} \setminus \delta_2\right) - f_R\left(R^{d_2 v_2}\right) & \text{if } v^1_{imp} = \kappa,\, v^2_{imp} \in V_\delta, \\ 0 & \text{if } v^1_{imp} \in V_\emptyset,\, v^2_{imp} = \kappa. \end{cases} \tag{6.6}$$

For each segment of customers $\delta = (\tau_i, \ldots, \tau_j) \subseteq R^{dv}$ with $R^{dv} \in R$ such that $|\delta| \leq |\delta_{\max}|$ there is a vertex $v_{\text{imp}} = (R^{dv},\, \delta)$ and a vertex $v'_{\text{imp}} = (R^{dv},\, \delta')$ in graph $G_{\text{imp}}(R, \delta_{\max})$ where $\delta' = (\tau_j, \ldots, \tau_i)$ denotes the *reverse* segment of customers. An arc $(v^1_{\text{imp}},\, v^2_{\text{imp}}) \in A_{\text{imp}}$ in $G_{\text{imp}}(R, \delta_{\max})$ means that the segment of customers δ_2 in route $R^{d_2 v_2}$ is replaced by the segment of customers δ_1 from route $R^{d_1 v_1}$ by inserting δ_2 into route $R^{d_1 v_1}$ in a greedy manner (expressed through $R^{d_2 v_2} \setminus \delta_2 \cup \delta_1$). The arc exists if and only if vehicle $v_2 \in V$ is allowed to visit all customers in segment δ_1 on day $d_2 \in D$ and the maximum load capacity of vehicle v_2 is not exceeded. In order to not only perform cyclic exchanges of segments of customers but also to perform *moves* of segments of customers, there is for each route $R^{dv} \in R$ an additional pseudo vertex $(R^{dv}, \emptyset)$ and one additional source vertex κ. An arc from a vertex associated with a segment of customers $\delta \subseteq R^{d_1 v_1}$ to a pseudo vertex of a route $R^{d_2 v_2}$ means that the segment is greedy inserted into route $R^{d_2 v_2}$. The only outgoing arc from the pseudo vertex is to the source vertex κ. An arc from the source vertex κ to a vertex associated with a segment of customers $\delta \subseteq R^{dv}$ means that segment δ is removed from route R^{dv}. Costs $c(v^1_{\text{imp}},\, v^2_{\text{imp}})$ describe the *benefit* of replacing the segment of customers δ_2 by the segment of customers δ_1.

Thompson and Orlin [TO89] prove that if the objective function $f_{\text{R}}(R^{dv})$ of a route $R^{dv} \in R$ is independent of other routes in R, then there exists a valid cyclic exchange move leading to an improved solution if and only if there is a negative cost subset-disjoint cycle in improvement graph $G_{\text{imp}}(R)$ such that the negative costs of the cycle describe the benefit of the move.

A subset disjoint cycle means that all vertices of this cycle belong to different routes (subsets). The Problem of finding such a cycle is called Subset-Disjoint Cycle Problem (SDCP) and is $\mathcal{NP}$-complete [TO89]. Unless $\mathcal{NP} = \mathcal{P}$, it is not possible that there is an algorithm which can efficiently solve the SDCP. We thus use a heuristic based on a label

Algorithm 6.7: Detect Minimum Cost Subset-Disjoint Cycle

Input: An improvement graph $G_{\text{imp}} = (V_{\text{imp}}, A_{\text{imp}})$
Output: A possible empty cycle C_{best}
Data: Distance array $dist[v]$, Predecessor array $pred[v]$, Queue q, Path p

```
1  C_best ← empty;
2  while s ∈ V_imp do
3      dist[s] ← 0;
4      pred[s] ← s;
5      dist[v] ← ∞   ∀v ∈ V_imp \ {s};
6      Push(q, s);
7      while ¬IsEmpty(q) do
8          i ←Pop(q);
9          if ¬IsSubsetDisjoint(p[i]) then continue;
10         foreach (i, j) ∈ A_imp do
11             if dist[j] > dist[i] + c(i, j) then
12                 if j ∈ p[i] then
13                     C ← subpath from j to p[i];
14                     if c(C) < c(C_best) then
15                         C_best ← C;
16                     end
17                 else
18                     if j.R^dv is not in already in p[i] then
19                         dist[j] ← dist[i] + c(i, j);
20                         pred[j] ← i;
21                         Pop(q,j);
22                     end
23                 end
24             end
25         end
26     end
27 end
28 return C_best;
```

correcting algorithm for finding shortest paths to find subset-disjoint cycles similar to [AMO93] and [Sha02].

Algorithm 6.7 gets as input an improvement graph G_{imp} and returns a negative cost cycle C_{best} which could be empty if there was no cycle found during the search. Since the algorithm is based on a label correcting algorithm, it starts from a start vertex $s \in V_{\text{imp}}$ and tries to find the shortest distances to all other vertices, considering that the vertices in the shortest paths must be subset-disjoint, i. e. that each vertex in a path belongs to a different route R^{dv}. The algorithm maintains three data structures. The distance array $dist[v]$ stores the shortest currently known distance from vertex v to the start vertex s and the predecessor array $pred[v]$ stores the predecessor vertex of vertex v. With $pred[v]$ it is possible to reconstruct the shortest path with distance $dist[v]$ for each vertex v. The third data structure is a queue, storing all vertices which must be explored during the search. In the inner while loop of Algorithm 6.7 the next vertex i to be explored will be taken from the queue. If this vertex belongs to a path $p[i]$ which is not subset disjoint, the algorithm continues with the next vertex from the queue. Otherwise each adjacent vertex j will be considered. If the currently known distance $dist[j]$ is greater than the distance we get by taking the path from s via i to j and vertex j appears already in path $p[i]$, then obviously we have found a new negative cost cycle. Otherwise, if j does not appear in $p[i]$, we have found a new shortest path from s to j. Furthermore if the associated route R^{dv} of vertex j does not already appear in path $p[i]$, we have not only found a new shortest path but also a new subset disjoint path from s to j. In this case, we update the data structures accordingly and continue with the next vertex in queue q. The outer loop starts the algorithm from each vertex in graph G_{imp} to increase the possibility of finding a negative subset-disjoint cost cycle [AMO93]. Note, that Algorithm 6.7 does not guarantee to find any negative subset-disjoint cost cycle even if there are some cycles in graph G_{imp}.

Algorithm 6.8: Cyclic Exchange Neighborhood $\mathcal{N}_{\text{cyclic}}(s, k, \delta_{\max})$

Input: A solution s, parameter k, maximum segment length $\delta_{\max}$
Output: A solution s'

1 Create a copy s' from solution s;
2 Choose randomly a day $d \in D$;
3 Choose randomly k non-empty routes $R \subseteq R^d$ from all routes starting on day d;
4 $G_{\text{imp}}(R, \delta_{\max}) \leftarrow$ `CreateImprovementGraph(`R`)`;
5 $cycle \leftarrow$ `DetectMinimumCostSubsetDisjointCycle(`$G_{\text{imp}}(R, \delta_{\max})$`)`;
6 **if** `IsEmpty(`$cycle$`)` **then return** s ;
7 Apply $cycle$ on s';
8 **return** s';

Algorithm 6.8 describes how an improved solution is obtained from the cyclic exchange neighborhood $\mathcal{N}_{\text{cyclic}}(s, k, \delta_{\max})$ of solution s. The algorithm selects randomly k routes R such that each of the selected route starts on the same day. The goal is to find a

cyclic exchange move which can be applied on routes R such that the new obtained solution is improved. Therefore an improvement graph $G_{\text{imp}}(R, \delta_{\max})$ is created from the selected routes R to search for a negative subset-disjoint cycle. If such a cycle is found, the corresponding cyclic exchange move is applied and the new obtained solution s' is returned.

6.2.7 Rearrange Tours Neighborhood

To escape a local optimum, the Rearrange Tours Neighborhood $\mathcal{N}_{\text{rearrange}}(s, k)$ tries to explore a valley far away from the incumbent solution. This is done in Algorithm 6.9 by selecting k randomly routes and removing all customers from them. The removed customers are greedy inserted into the solution again by a random order. After each time a customer is inserted into a route R^{dv}, two local search procedures are applied. The local search procedures try to exchange two customers between route R^{dv} and another route $R^{dv'}$ or move a customer from route R^{dv} to another route $R^{dv'}$, respectively. The local search procedures use best improvement to select an improved solution.

Algorithm 6.9: Rearrange Tours Neighborhood $\mathcal{N}_{\text{rearrange}}(s, k)$

Input: A solution s, parameter k
Output: A solution s'

Create a copy s' from solution s;
Choose randomly a day $d \in D$;
Choose randomly k routes $R \subseteq R^d$ from all routes starting on day d;
Remove all customers from routes R and store them into set N_R;
while ¬`IsEmpty`(N_R) **do**
    Choose randomly a removed customer $n \in N_R$;
    Greedy insert n into a route $R^{dv} \in R^d$ such that $f(s')$ has the lowest increase;
    Apply exchange customers local search procedure among R^{dv} and R^d;
    Apply move customer local search procedure among R^{dv} and R^d;
    Remove n from N_R;
end
return s';

6.3 Shaking Neighborhood Structure

Table 6.1 listens the used neighborhood structure $\mathcal{N}_k$ with over $k = 48$ different neighborhoods. The constant $\kappa_{\max}$ and $\delta_{\max}$ describe the number of rearrange tour neighborhoods and the maximum length of considered segments in the cyclic exchange neighborhoods $\mathcal{N}_{\text{cyclic}}(s, 3-5, \delta_{\max})$, respectively. The values for $\kappa_{\max}$ and $\delta_{\max}$ are both determined in Section 7.3. One important neighborhood is $\mathcal{N}_{\text{include}}(s)$, which includes unserved customers. The task of the neighborhoods $\mathcal{N}_{\text{move}}(s, 1-15)$, $\mathcal{N}_{\text{exchange}}(s, 1-15)$,

$\mathcal{N}_{\text{edge}}(s)$ and $\mathcal{N}_{\text{merge}}(s)$ is to quickly make place for new customers such that eventually $\mathcal{N}_{\text{include}}(s)$ could include them. The very large neighborhoods $\mathcal{N}_{\text{cyclic}}(s, 3-5, \delta_{\max})$ and $\mathcal{N}_{\text{rearrange}}(s, 1-\kappa_{\max})$ should finally find new improved solutions in valleys far away from the incumbent solution. The last neighborhood $\mathcal{N}_{\text{cyclic}}(s, 20, 1)$ try to find cyclic exchange moves among up to 20 randomly selected routes. Since the exploration of this neighborhood could take same time, the maximum segment length is restricted to one.

$\mathcal{N}_{1-20}(s)$	$\mathcal{N}_{\text{move}}(s, 1-15)$
$\mathcal{N}_{21-40}(s)$	$\mathcal{N}_{\text{exchange}}(s, 1-15)$
$\mathcal{N}_{41}(s)$	$\mathcal{N}_{\text{edge}}(s)$
$\mathcal{N}_{42}(s)$	$\mathcal{N}_{\text{merge}}(s)$
$\mathcal{N}_{43}(s)$	$\mathcal{N}_{\text{include}}(s)$
$\mathcal{N}_{44-46}(s)$	$\mathcal{N}_{\text{cyclic}}(s, 3-5, \delta_{\max})$
$\mathcal{N}_{47-47+\kappa_{\max}}(s)$	$\mathcal{N}_{\text{rearrange}}(s, 1-\kappa_{\max})$
$\mathcal{N}_{48+\kappa_{\max}}(s)$	$\mathcal{N}_{\text{cyclic}}(s, 20, 1)$

Table 6.1: Shaking Neighborhood Structure $\mathcal{N}_k$

6.4 Skewed VNS

A skewed-VNS is implemented to escape local optima and explore valleys which are far away from the incumbent solution. This is done by accepting a possible worse solution if the found solution is far away from the current incumbent solution. This VNS variant is called skewed-VNS described in Algorithm 6.10. The variable s_{ref} describes the current incumbent solution whereas the variable s_{best} stores the best solution found so far. Similar to the VNS Algorithm 6.1 a solution s' is randomly selected from the shaking neighborhood structure $\mathcal{N}_k(s_{\text{ref}})$ at the beginning of the loop. After applying local search procedures a new incumbent solution is accepted if

$$f(s'') - \alpha_{\text{svns}}\, \rho(s'',\, s_{\text{ref}}) < f(s_{\text{ref}}) \tag{6.7}$$

holds. The distance metric $\rho : S \times S \mapsto \mathbb{R}_{\geq 0}$ describes the distance between solution s'' and s_{ref}. Parameter α controls to which extent the distance between the two solutions goes into the decision of accepting a worse solution. The number of different arcs in solution s'' and s_{ref} is used as distance metric ρ. Only if a new incumbent solution is accepted, one more local search procedure will be applied to s'' with respect to neighborhood $\mathcal{N}_{2\text{opt}*}$. If the resulting solution s_{ref} is better than the best solution found so far, s_{best} will be updated accordingly. Every time a new incumbent solution is accepted the shaking neighborhoods are reset. Otherwise the next shaking neighborhood in the structure will be selected.

Algorithm 6.10: Skewed-VNS

Input: An initial solution s
Output: A probably improved solution s'

$s_{\text{best}} \leftarrow s$;
$s_{\text{ref}} \leftarrow s$;
$k \leftarrow 1$;
for *stopping criteria satisfied* **do**
  Select $s' \in \mathcal{N}_k(s_{\text{ref}})$;
  $s'' \leftarrow$ `LocalSearch(`s', $\mathcal{N}_{3opt}$`)`;
  $s'' \leftarrow$ `LocalSearch(`s'', $\mathcal{N}_{2opt}$`)`;
  if $f(s'') - \alpha_{\text{svns}}\,\rho(s'', s_{\text{ref}}) < f(s_{\text{ref}})$ **then**
    $s_{ref} \leftarrow$ `LocalSearch(`s'', $\mathcal{N}_{2opt*}$`)`;
    if $f(s_{\text{ref}}) < f(s_{\text{best}})$ **then**
      $s_{\text{best}} \leftarrow$ `LocalSearch(`s'', $\mathcal{N}_{2opt*}$`)`;
    end
    $k \leftarrow 1$;
  else
    $k \leftarrow k + 1$;
  end
  if $k > k_{\max}$ **then**
    $k \leftarrow 1$;
  end
end
return s_{best};

6.5 Computation of Start Times

There are many ways to determine the start time Y_0^{dv} (see Section 4.2) from the depot of a given route $R^{dv} = (\tau_0, \tau_1, \ldots, \tau_{|R^{dv}|})$ of vehicle $v \in V$ at day $d \in D$. The cheapest method regarding computation time would be to always set Y_0^{dv} to the earliest possible availability start time s_v^{veh} of vehicle v, thus

$$Y_0^{dv} = s_v^{\text{veh}}. \tag{6.8}$$

This method minimizes time window violations, since the start from the depot is always as earliest as possible. However this approach is not optimal regarding the tour duration $Y_{n_c+1}^{dv} - Y_0^{dv}$, since the tour duration could exceed the maximum tour duration $D^{\max}$ although this could maybe be avoided by starting later from the depot. Hence, this method may cause maximum tour duration violations, although they could be avoided.

A simple improvement to reduce route duration is to set

$$Y_0^{dv} = \max(s_v^{\text{veh}}, e_{\tau_1}^{\text{cus}} - c_{\tau_0\tau_1}^{\text{time}}), \tag{6.9}$$

where τ_1 denotes the first visited customer. However this approach has the same disadvantages as the first described method.

There are cases where the start time Y_0^{dv} could be further delayed in order to minimize the tour duration and without violating any time windows. The concept of delaying the departure time from any node without violating any time window is called *forward time slack* and was first introduced by Savelsbergh [Sav92]. The forward time slack for the depot F_0 can be recursively defined as

$$F_0 = \min_{0 \leq i \leq |R^{dv}|} \left(\sum_{0 < k \leq i} W_{\tau_k} + \max \left(l_{\tau_j}^{\text{cus}} - Y_{\tau_j}^{\text{cus}} - W_{\tau_j}, 0 \right) \right). \tag{6.10}$$

By setting the start time to

$$Y_0^{dv} = s_v^{\text{veh}} + F_0 \tag{6.11}$$

the route duration will be minimized without violating any time window if possible. Although the computation of the slack time is more expensive regarding running time, Cordeau, Laporte and Mercier [CLM04] could show that the obtained results for the SDVRPTW instances from [CL01] are better when considering the time slack as without it. Thus, in the successive experiments in Chapter 7 we always compute the starting time of a route by considering the slack time.

CHAPTER 7

Computational Results

In this chapter I present the results of the computational experiments performed with the algorithms described in the previous chapters. This chapter starts with a description of the computational environment in section 7.1. In section 7.2 the benchmark instances are described, and their transformation to the considered model is presented in 7.2.1.

7.1 Used Libraries and Framework

All tests are performed on a computing cluster consisting of 14 machines with two Intel Xeon E5540 processors running on 2.53 GHz with 24 GB RAM. Furthermore two machines with two Intel Xeon E5649 processors running on 2.53 GHz with 60 GB RAM and one machine with two Xeon E5649 processors running on 2.53 GHz with 72 GB RAM have been used for the tests.

The program was written in C++ and compiled with GCC version 4.8.4 and the following additional software libraries and frameworks are used:

- Boost library (version 1.59.0): collection of many smaller C++ libraries;
- IBM ILOG CPLEX (version 12.6): A software package which can solve very large MIPs.

7.2 Test Instances

The implementations of the VNS and SVNS algorithms are tested on several test instances from the literature. There are all in all 36 test instances created by Cordeau and Laporte in [CL01] and [CGL97] for the SDVRP and the SDVRPTW. The characteristics of the test instances are shown in Table 7.1 and 7.2. In both tables the column n indicates the

number of customers which have to be served. The column t and m corresponds to the number of different vehicle types and the number of vehicles per vehicle type, respectively. The number of customers ranges from 48 to 1008, whereas the size of the fleet ranges from 4 to 108 with up to 6 different vehicle types. Furthermore the developed algorithms will be compared with 20 test instances from Vidal, Crainic, Gendreau and Prins [VCGP13] listed in Table 7.2. These instances contain 360 to 880 nodes corresponding to customers and up to 120 vehicles with up to 6 different vehicle types.

Instance	n	t	m	$f(s^*)$
1	48	4	1	1385.47
2	96	4	2	2322.08
3	144	4	3	2626.97
4	192	4	4	3575.79
5	240	4	5	4479.34
6	288	4	6	4583.46
7	72	6	1	1990.68
8	144	6	2	3103.40
9	216	6	3	3732.06
10	288	6	4	4782.17
11	1008	4	21	13479.90
12	720	6	10	9938.57

Table 7.1: Test Instances for the SDVRP from Cordeau and Laporte [CL01]

Moreover, there are 8 new reduced test instances for the SDVRPTW created from the test instances of [CL01] to test the B&C algorithm. These instances are shown in Table 7.3 and contain 27 to 57 customers with a fleet size up to 8 vehicles and up to 6 different vehicle types.

7.2.1 Transformation of the Benchmark Instances

Since the model of the SSDVRPTW differs from the models of the SDVRPTW used in [CL01] and [VCGP13], a mapping from an instance of the SDVRPTW to the SSDVRPTW must be provided.

The SDVRPTW from the literature is defined on a directed graph $\bar{G} = (\bar{V}, \bar{A})$ with vertex set $\bar{V} = \{\bar{v}_0, \ldots, \bar{v}_n\}$ and arc set $\bar{A} = \{(\bar{v}_i, \bar{v}_j) \,|\, \bar{v}_i \in \bar{V}, \bar{v}_j \in \bar{V}, \bar{v}_i \neq \bar{v}_j\}$. Vertex $\bar{v}_0$ represents the depot where m vehicles of t different types are located. Each vehicle type $l \in \{1, \ldots, t\}$ has a maximum capacity $\bar{Q}_l$. Each vertex $\bar{v}_i \in \bar{V}$ is associated with a time window $[\bar{e}_i, \bar{l}_i]$, a nonnegative service duration $\bar{s}_i$ and a nonnegative demand $\bar{d}_i$. Furthermore, each customer $\bar{v}_i \in \bar{V} \setminus \{\bar{v}_0\}$ has a set $\bar{R}_i \subseteq \{1, \ldots, t\}$ of allowed vehicle types. Moreover each arc $(\bar{v}_i, \bar{v}_j) \in \bar{A}$ is associated with a travel cost $\bar{c}_{ij}$.

Cordeau and Laporte					Vidal, Crainic, Gendreau and Prins				
Instance	n	t	m	BKS	Instance	n	t	m	BKS
1a	48	4	2	1658.84	11a	360	4	11	9924.11
2a	96	4	3	3003.21	12a	480	4	14	12251.66
3a	144	4	4	3450.46	13a	600	4	17	14491.25
4a	192	4	5	4746.27	14a	720	4	21	16547.89
5a	240	4	6	6521.95	15a	840	4	25	19090.19
6a	288	4	7	6066.22	16a	960	4	29	21413.65
7a	72	6	2	2179.06	17a	360	6	8	10547.07
8a	144	6	3	4029.71	18a	520	6	12	13963.49
9a	216	6	4	5085.57	19a	700	6	16	18855.51
10a	288	6	5	6218.15	20a	880	6	20	22513.44
11a	1008	4	27	16535.51	-	-	-	-	-
12a	720	6	14	12639.84	-	-	-	-	-
1b	48	4	2	1429.35	11b	360	4	9	7962.22
2b	96	4	3	2537.07	12b	480	4	11	9508.68
3b	144	4	4	2850.20	13b	600	4	14	11562.67
4b	192	4	5	3828.65	14b	720	4	17	13623.28
5b	240	4	6	4842.61	15b	840	4	20	15437.52
6b	288	4	7	5039.36	16b	960	4	23	17834.61
7b	72	6	2	1872.98	17b	360	6	6	8562.99
8b	144	6	3	3190.48	18b	520	6	9	11477.72
9b	216	6	4	4081.26	19b	700	6	12	14894.65
10b	288	6	5	5298.60	20b	880	6	15	18566.66
11b	1008	4	27	14263.65	-	-	-	-	-
12b	720	6	14	10382.74	-	-	-	-	-

Table 7.2: Test Instances for the SDVRPTW [CL01] and [VCGP13]

An instance of the SSDVRPTW can be created by setting

$$G := \bar{G} \tag{7.1}$$

$$n_d := 1 \tag{7.2}$$

$$n_c := n \tag{7.3}$$

$$[e_i^{\text{cus}}, l_i^{\text{cus}}] := [\bar{e}_i, \bar{l}_i] \qquad \forall \bar{v}_i \in \bar{V} \setminus \{\bar{v}_0\} \tag{7.4}$$

$$s_i := \bar{s}_i \qquad \forall \bar{v}_i \in \bar{V} \tag{7.5}$$

$$d_i := \bar{d}_i \qquad \forall \bar{v}_i \in \bar{V} \tag{7.6}$$

$$\mathbf{C}^{\text{time}} := (\bar{c}_{ij}) \qquad \forall \bar{v}_i \in \bar{V}, \forall \bar{v}_j \in \bar{V} \tag{7.7}$$

$$\mathbf{C}^{\text{dist}} := \mathbf{0}. \tag{7.8}$$

Inst	n	t	m
mip 1a	27	4	1
mip 2a	57	4	2
mip 3a	56	4	2
mip 4a	54	4	2
mip 5a	57	4	2
mip 6a	57	4	2
mip 7a	48	6	1
mip 8a	44	6	1

Table 7.3: B&C Results

Vehicles: The set of vehicles is defined by

$$V := \{(l,i) \mid l \in \{1,\dots,t\}, i \in \{1,\dots,m\}\}. \tag{7.9}$$

In contrary to the depot of the SSDVRPTW, the depot $\bar{v}_0$ is associated with a time window. This time window is mapped to the vehicle time windows

$$[s_v^{\text{veh}}, e_v^{\text{veh}}] := [\bar{e}_0, \bar{l}_0] \quad \forall v \in V. \tag{7.10}$$

Furthermore the maximum load capacity

$$C_v^{\max} := \bar{Q}_l \quad \forall v = (l,i) \in V \tag{7.11}$$

is set to the maximum load capacity of the corresponding vehicle type l.

Qualifications: It remains to map the set of allowed vehicle types $\bar{R}_i \, \forall v_i \in \bar{V} \setminus \{v_0\}$ to the sets of provided and required qualifications $Q_i^{\text{pro}} \, \forall i \in N'$ and $Q_v^{\text{req}} \, \forall v \in V$, respectively. This is done by interpreting the set of allowed vehicle types $\bar{R}_i \, \forall v_i \in \bar{V} \setminus \{v_0\}$ as a t-bit vector $q_i = (b_1 b_2 \dots b_t)$ such that

$$b_j = \begin{cases} 1 & \text{if } j \in \bar{R}_i \\ 0 & \text{otherwise} \end{cases} \qquad \forall j \in \{1,\dots,t\}. \tag{7.12}$$

The set of qualifications

$$Q := \left\{ q_i = (b_1 b_2 \dots b_t) | \bar{v}_i \in \bar{V} \setminus \{v_0\} \right\} \tag{7.13}$$

contains all t-bit vectors. Furthermore, for each customer $\bar{v}_i \in \bar{V} \setminus \{v_0\}$ with qualification $q_i = (b_1 b_2 \dots b_t)$ the sets

$$Q_i^{\text{req}} := \{q_i\} \tag{7.14}$$

$$Q_v^{\text{pro}} := Q_v^{\text{pro}} \cup \{q_i\} \quad \forall v = (l,j) \in V : b_l = 1 \tag{7.15}$$

are filled.

Objective Function: The objective function from the literature covers only the total travel time of a solution s. Therefore the penalty factors from Table 4.1 must be set accordingly and are shown in Table 7.4. Since in [CL01] and [VCGP13] only travel times

α	0	Distance and time factor
β	0	Penalty factor for the waiting time
γ	200	Penalty factor for customer time window violations
δ	100	Penalty factor for vehicle time window violations
ε	200	Penalty factor for exceeding the maximum tour duration
η	10^4	Penalty factor for the number of unvisited customers
θ	0	Penalty factor for each time a vehicle is used

Table 7.4: Penalty Factors

are used as distance measure between two customers, parameter α is set to zero. For the same reasion parameter β and θ are set to zero, ignoring waiting times and open route costs. To guarantee that scheduling a customer has a very high priority, the penalty factor η is set very high in contrast to the other penalty factors $\eta \gg \gamma, \delta, \varepsilon$. The values for the penalties of time window violations, vehicle time window violations and maximum tour duration violations were determined in extensive tests.

7.3 Determination of Strategic Optimization Parameters

There are some strategic optimization parameters which could be used for fine tuning the overall performance of the algorithm. One important parameter which has a great influence on the overall performance is the distance factor α_{SVNS} of the SVNS algorithm.

Another parameter is the maximum number of rearrange tours $\kappa_{\max}$ of the neighborhood $\mathcal{N}_{\text{rearrange}}(s, \kappa_{\max})$. The parameter controls how many tours are destroyed during the execution of the neighborhood and has a great influence on the execution time of the neighborhood.

The third parameter to fine tune the VNS algorithm is the maximum length $\delta_{\max}$ of segments which are considered by a cyclic exchange move. This parameter has also a great influence of the execution time of the neighborhood $\mathcal{N}_{\text{cyclic}}(s, k, \delta_{\max})$.

Hence, the focus lies on the parameter set $(\alpha_{\text{SVNS}}, \kappa_{\max}, \delta_{\max})$, and they were tested in the ranges

$$\alpha_{\text{SVNS}} \in \{0.3, 0.4, 0.5, 0.6\} \tag{7.16}$$

$$\kappa_{\max} \in \{1, 2, 3\} \tag{7.17}$$

$$\delta_{\max} \in \{1, 2, 3, 4\}. \tag{7.18}$$

This leads to $4 \times 3 \times 4 = 48$ different combinations. To save running time, the different parameter sets are not tested on all instances from [CL01]. Table 7.5 shows the used test

instances for the parameter calibration. Furthermore for each combination of parameters, there are 10 runs performed.

Inst	n	t	m
2a	96	4	3
3a	144	4	4
6a	288	4	7
9a	216	6	4
2b	96	4	3
3b	144	4	4
6b	288	4	7
9b	216	6	4

Table 7.5: Test Instances for Parameter Calibration

The best results could be obtained by a distance factor of $\alpha_{\text{svns}} = 0.5$, by destroying only $\kappa_{\max} = 1$ route in neighborhood $\mathcal{N}_{\text{rearrange}}(s, k_{\max})$ and by considering a maximum segment length of $\delta_{\max} = 3$ in neighborhood $\mathcal{N}_{\text{cyclic}}(s, k)$. We therefore use these values in all successive experiments.

7.4 Analysis of the Contribution of Individual Algorithmic Components

In order to analyze the contribution of individual algorithmic components the shaking neighborhood structure is changed such that only the currently analyzed neighborhoods together with the Included Unserved Customers Neighborhood is included into the structure.

The tests are performed on all test instances of Cordeau and Laporte [CL01] and are executed with the VNS algorithm as well as with the SVNS algorithm. For each test instance there were 15 runs performed. The obtained results are compared with the results obtained by Vidal, Crainic, Gendreau and Prins [VCGP13].

In particular the neighborhoods $\mathcal{N}_{\text{cyclic}}(s, k, \delta_{\max})$ and $\mathcal{N}_{\text{rearrange}}(s, k)$ were analyzed.

7.4.1 Analysis of the minimal Neighborhood Structure

In this section, the VNS algorithm as well as the SVNS algorithm are executed with the shaking neighborhood structure listed in Table 7.6. Hence, only standard neighborhoods such as Move Segment Neighborhood $\mathcal{N}_{\text{move}}(s, 1-15)$ or Exchange Segment Neighborhood $\mathcal{N}_{\text{exchange}}(s, 1-15)$ together with the Include Unserved Customers Neighborhood $\mathcal{N}_{\text{include}}(s)$ are used. Table 7.18 shows the results for both algorithms. The SVNS algorithm produce 3 and the VNS algorithm produce 11 infeasible solutions, where some

$\mathcal{N}_{1-20}(s)$	$\mathcal{N}_{\text{move}}(s, 1-15)$
$\mathcal{N}_{21-40}(s)$	$\mathcal{N}_{\text{exchange}}(s, 1-15)$
$\mathcal{N}_{41}(s)$	$\mathcal{N}_{\text{edge}}(s)$
$\mathcal{N}_{42}(s)$	$\mathcal{N}_{\text{merge}}(s)$
$\mathcal{N}_{43}(s)$	$\mathcal{N}_{\text{include}}(s)$

Table 7.6: Minimal Shaking Neighborhood Structure $\mathcal{N}_k$

customers are not scheduled. The average gap between the obtained results and the obtained results from [VCGP13] regarding the average objective values are 6.50% for the SVNS algorithm and 9.99% for the VNS algorithm. The average gap regarding the best solution found during the 15 performed runs obtained by the SVNS and the VNS algorithm are 3.56% and 6.61%, respectively. Hence, the SVNS algorithm outperforms the VNS algorithm.

7.4.2 Analysis of the Cyclic Exchange Neighborhood

Table 7.7 shows the shaking neighborhood structure used to analyze the Cyclic Exchange Neighborhood $\mathcal{N}_{\text{cyclic}}(s, k, \delta_{\max})$. The VNS as well as the SVNS algorithm are applied on the test instances from [CL01]. The obtained results are showed in Table 7.18. The

$\mathcal{N}_1(s)$	$\mathcal{N}_{\text{include}}(s)$
$\mathcal{N}_{2-4}(s)$	$\mathcal{N}_{\text{cyclic}}(s, 2-3, \delta_{\max})$
$\mathcal{N}_5(s)$	$\mathcal{N}_{\text{cyclic}}(s, 20, 1)$

Table 7.7: Cyclic Exchange Neighborhood Structure $\mathcal{N}_k$

average gap regarding the average objective values is 5.83% and 7.93% obtained by the SVNS algorithm and by the VNS algorithm, respectively. Whereas the average gap regarding the best objective value is 2.98% and 4.78% obtained by the SVNS algorithm and by the VNS algorithm, respectively. The SVNS algorithm produces 17 infeasible solutions where some customers are not scheduled, whereas the VNS algorithm could not schedule all customers 26 times.

7.4.3 Analysis of the Rearrange Tour Neighborhood

Table 7.8 shows the shaking neighborhood structure used to analyze the Rearrange Tours Neighborhood $\mathcal{N}_{\text{rearrange}}(s, k, \delta_{\max})$. The VNS as well as the SVNS algorithm are applied on the test instances from [CL01]. The SVNS as well as the VNS algorithm could always schedule all customers with the used neighborhood structure in Table 7.8. The average gap is 3.41% and 1.65% regarding the average objective values and the best objective values obtained by the SVNS algorithm, respectively.

$\mathcal{N}_1(s)$	$\mathcal{N}_{\text{include}}(s)$
$\mathcal{N}_{2-\kappa_{\max}}(s)$	$\mathcal{N}_{\text{rearrange}}(s, \kappa_{\max})$

Table 7.8: Rearrange Tours Neighborhood Structure $\mathcal{N}_k$

7.4.4 Analysis Summery

Table 7.18 shows the results of the analysis of individual algorithmic components. There are four different shaking neighborhood structures (Table 7.6-7.8) tested with both the VNS and the SVNS algorithm. Two sample t-tests were conducted to compare the different test cases. The results of the t-tests are shown in Table 7.10. It can be shown that the SVNS algorithm outperforms the VNS algorithm in most test scenarios. The best results could be obtained with the shaking neighborhood structure listed in Table 6.1. Since it is very unlikely to find randomly a move or exchange maneuver between two routes such that the incumbent solution could be improved when site dependency and time window constraints have to be considered, the neighborhood structure with only the standard neighborhoods in Table 7.6 produced the worst results.

	VNS			SVNS		
Neighborhood Structure	avg	$\sigma(avg)$	best	avg	$\sigma(avg)$	best
Minimal	9.99%	2.98	6.61%	6.50%	2.09	3.56%
Cyclic Exchange	7.93%	2.23	4.78%	5.83%	1.86	2.98%
Rearrange Tours	4.20%	1.59	2.08%	3.41%	1.61	1.65%
All	4.30%	1.73	2.15%	3.11%	1.32	1.32%

Table 7.9: Results of the Analysis of Algorithmic Components. The values indicate the averaged gap (7.19) over all instances from [CL01] compared with the results obtained from [VCGP13].

Better results could be obtained by using the neighborhood structure with the Cyclic Exchange Neighborhood $\mathcal{N}_{\text{cyclic}}(s, k, \delta_{\max})$ shown in Table 7.7. A major drawback of this neighborhood are the huge computational costs to create the improvement graph $G_{\text{imp}}(R, \delta_{\max})$, where R is a set of routes such that $|R| = k$ and $\delta_{\max}$ is the maximum length of considered segments of customers. A graph with $k > 5$ routes and a maximum segment length of $\delta_{\max} > 3$ is already too expensive to create. Nevertheless the neighborhood has a good performance for small k and small $\delta_{\max}$ in contrary to the Move or Exchange Neighborhoods.

The neighborhood structure with the Rearrange Tours Neighborhood shown in Table 7.8 could clearly outperform the Cyclic Exchange Neighborhood Note that the number of destroyed tours k is a critical parameter regarding the running time of the neighborhood. Good results could only be obtained for small values of k.

	SVNS			
VNS	Minimal	Cyclic Exchange	Rearrange Tours	All
Minimal	**(4.29/2.03)**	(1.07/2.03)		
Cyclic Exchange	**(2.48/2.03)**	**(2.24/2.03)**	**(4.40/2.03)**	
Rearrange Tours		**(6.09/2.03)**	(1.57/2.02)	(0.62/2.03)
All			(-0.19/2.02)	**(2.43/2.03)**

Table 7.10: Results of the t-Test: (t Stat / t Critical). If t Stat > t Critical, then there is a significant difference between the two corresponding test cases. Cells in the upper triangular matrix corresponds to results of t-tests where the compared neighborhood structures are executed with the SVNS algorithm, whereas cells in the lower triangular matrix corresponds to results of t-tests where the compared neighborhood structures are executed with the VNS algorithm. Diagonal cells indicate the comparison between the VNS and SVNS algorithm.

7.5 Computational Experiments with the VNS Framework

Table 7.11 and 7.12 show the results of the SVNS algorithm executed on the test instances from [CL01]. The parameter α_{SVNS} of the SVNS was set to 0.5. The shaking neighborhood structure from Table 6.1 in Section 6.3 was used during the execution of the SVNS algorithm.

The algorithm was executed 15 times for each of the 24 instances. Column n in Table 7.11 and 7.12 correspond to the number of customers, whereas column t and column m corresponds to the number of different vehicle types and the number of vehicles per type, respectively.

The columns $\overline{f}$ and $\overline{T}$ in Table 7.11 show the average objective values and average running times in seconds, respectively obtained from [VCGP13]. The *Hybrid Genetic Search with Advanced Diversity Control* [VCGP13] algorithm was executed 10 times for each instance. The next three columns listen the result of the SVNS algorithm. Column $\overline{f}$ reports the average objective values of 15 test runs, whereas column $\sigma(f)$ shows the corresponding standard deviation. The average running time in seconds is shown in column $\overline{T}$. The last column reports the gap

$$\text{gap} = \frac{\overline{f}_{\text{Vidal}} - \overline{f}_{\text{SVNS}}}{\overline{f}_{\text{Vidal}}} \tag{7.19}$$

between the average objective values. The last row shows the average gap.

Table 7.12 compares the objective values of the best solution obtained during 15 test runs in column $\overline{f_{\text{best}}}$ with the objective value of the best solution from [VCGP13] in column $\overline{f_{\text{best}}}$. The column $T(\overline{f_{\text{best}}})$ lists the running time of the SVNS algorithm. The gap of the two objective values is listened in column *gap*.

				Vidal		Skewed-VNS			
Inst	n	t	m	$\overline{f}$	$\overline{T}$ [s]	$\overline{f}$	$\sigma(f)$	$\overline{T}$ [s]	gap
1a	48	4	2	1655.42	13.80	1661.04	7.42	15.00	0.34%
2a	96	4	3	2904.13	42.00	2961.58	39.60	66.00	1.98%
3a	144	4	4	3320.44	96.00	3436.24	61.19	162.00	3.49%
4a	192	4	5	4437.19	351.00	4548.63	57.50	608.00	2.51%
5a	240	4	6	5681.48	698.40	5876.16	101.62	1256.00	3.43%
6a	288	4	7	5666.20	760.80	5872.29	49.57	1281.00	3.64%
7a	72	6	2	2166.88	25.20	2233.08	53.87	32.00	3.06%
8a	144	6	3	3880.58	141.00	3980.92	31.54	206.00	2.59%
9a	216	6	4	4797.72	336.00	4931.65	58.19	602.00	2.79%
10a	288	6	5	5876.38	694.80	6086.26	76.63	1221.00	3.57%
11a	1008	4	27	15198.10	-	-	-	-	-
12a	720	6	14	11475.15	-	-	-	-	-
1b	48	4	2	1429.35	13.20	1442.76	9.61	15.00	0.94%
2b	96	4	3	2479.56	59.40	2555.37	34.69	91.00	3.06%
3b	144	4	4	2779.09	136.80	2856.85	22.66	231.00	2.80%
4b	192	4	5	3660.66	394.20	3821.25	55.36	654.00	4.39%
5b	240	4	6	4625.79	483.60	4780.21	46.67	787.00	3.34%
6b	288	4	7	4755.59	917.40	4979.94	60.76	1380.00	4.72%
7b	72	6	2	1837.94	30.60	1884.33	27.27	44.00	2.52%
8b	144	6	3	3149.49	129.00	3212.05	32.55	207.00	1.99%
9b	216	6	4	3894.67	534.00	4110.36	79.29	854.00	5.54%
10b	288	6	5	4962.62	721.80	5244.45	61.24	1146.00	5.68%
11b	1008	4	27	13226.60	-	-	-	-	-
12b	720	6	14	9857.89	-	-	-	-	-
									3.12%

Table 7.11: SDVRPTW - Skewed VNS Results: Average Objective Function Values

Note that the SVNS algorithm has been implemented within a complex algorithmic framework, designed to solve the DSDVRPTW. Hence, the framework supports many complex operations like fixing service times or vehicles for some customer. These operations are not necessarily needed to solve the SSDVRPTW. However, it can be seen that the implemented SVNS algorithm is competitive with results obtained by the literature in some cases. Table 7.12 shows that for some instances the same best objective value could be obtained as in [VCGP13]. All other best solutions obtained by the SVNS algorithm are close to the obtained solutions from [VCGP13] such that the average gap is 1.32%.

				Vidal	Skewed-VNS		
Inst	n	t	m	$\overline{f_{\text{best}}}$	$\overline{f_{\text{best}}}$	$T(\overline{f_{\text{best}}})$ [s]	gap
1a	48	4	2	1655.42	1655.42	4.00	0.00%
2a	96	4	3	2904.13	2904.13	41.00	0.00%
3a	144	4	4	3304.13	3349.35	192.00	1.37%
4a	192	4	5	4427.25	4474.91	554.00	1.08%
5a	240	4	6	5647.76	5682.71	1329.00	0.62%
6a	288	4	7	5637.48	5792.68	1340.00	2.75%
7a	72	6	2	2166.88	2166.88	43.00	0.00%
8a	144	6	3	3873.40	3934.35	247.00	1.57%
9a	216	6	4	4777.61	4826.75	648.00	1.03%
10a	288	6	5	5858.82	5910.87	1355.00	0.89%
11a	1008	4	27	15080.68	-	-	-
12a	720	6	14	11402.01	-	-	-
1b	48	4	2	1429.35	1429.35	2.00	0.00%
2b	96	4	3	2479.56	2494.71	108.00	0.61%
3b	144	4	4	2775.61	2817.07	246.00	1.49%
4b	192	4	5	3649.72	3754.16	632.00	2.86%
5b	240	4	6	4611.16	4706.03	874.00	2.06%
6b	288	4	7	4729.96	4869.02	1495.00	2.94%
7b	72	6	2	1837.94	1837.94	48.00	0.00%
8b	144	6	3	3152.69	3163.12	231.00	0.33%
9b	216	6	4	3883.94	3983.38	695.00	2.56%
10b	288	6	5	4932.40	5137.58	1263.00	4.16%
11b	1008	4	27	13067.52	-	-	-
12b	720	6	14	9777.44	-	-	-
							1.32%

Table 7.12: SDVRPTW - Skewed VNS Results: Best Objective Function Values

The average gap in Table 7.11 between the average objective values obtained by the SVNS algorithm and the average objective values obtained from [VCGP13] is 3.12%. Although the obtained average objective values are only in some cases close to the average objective values from [VCGP13], it can be seen that the obtained standard deviation values are rather small. Hence, the results of the SVNS algorithm are robust and reliable and therefore well suited for practical applications.

Furthermore the SVNS algorithm is applied to the SDVRP instances from [CL01]. The obtained results are compared with the results obtained from Pisinger and Ropke [PR07] and listed in Table 7.13 and 7.14. For each instance 15 runs were performed. The average gap regarding the average objective value and best objective value is 2.19% and

				Pisinger		Skewed-VNS			
Inst	n	t	m	$\overline{f}$	$\overline{T}$ [s]	$\overline{f}$	$\sigma(f)$	$\overline{T}$ [s]	gap
1	48	4	1	1393.85	19	1381.24	0.70	61	-0.90%
2	96	4	2	2330.60	63	2339.85	27.17	210	0.40%
3	144	4	3	2607.66	140	2663.12	44.64	388	2.13%
4	192	4	4	3489.51	191	3597.11	49.06	510	3.08%
5	240	4	5	4431.16	251	4549.61	30.08	791	2.67%
6	288	4	6	4465.18	314	4623.06	50.40	772	3.54%
7	72	6	1	1916.50	39	1968.76	39.43	124	2.73%
8	144	6	2	3007.99	135	3049.51	33.01	413	1.38%
9	216	6	3	3567.15	226	3686.82	35.67	680	3.35%
10	288	6	4	4673.67	322	4839.76	46.22	729	3.55%
									2.19%

Table 7.13: SDVRP - Skewed VNS Results: Average Objective Function Values

				Pisinger	Skewed-VNS		
Inst	n	t	m	$\overline{f_{\text{best}}}$	$\overline{f_{\text{best}}}$	$T(\overline{f_{\text{best}}})$ [s]	gap
1	48	4	1	1380.77	1380.77	10	0.00%
2	96	4	2	2311.54	2316.51	112	0.22%
3	144	4	3	2602.13	2606.35	398	0.16%
4	192	4	4	3474.01	3519.08	583	1.30%
5	240	4	5	4416.38	4508.91	840	2.10%
6	288	4	6	4444.52	4532.65	930	1.98%
7	72	6	1	1889.82	1889.82	53	0.00%
8	144	6	2	2977.50	3004.52	436	0.91%
9	216	6	3	3536.20	3629.15	630	2.63%
10	288	6	4	4648.76	4734.87	825	1.85%
							1.11%

Table 7.14: SDVRP - Skewed VNS Results: Best Objective Function Values

1.11%, respectively. Hence, the average gaps are lower than the average gaps obtained by solving the SDVRPTW problem. However, there are 19 out of total 150 solutions, where not all customers are included. One reason for this behavior may be that the number of overall vehicles is smaller for each SDVRP instance as for the SDVRPTW instances. Therefore it is harder for the Include Unserved Customers neighborhood $\mathcal{N}_{\text{include}}(s)$ to find a free vehicle v without violating for instance the maximum load capacity $C_v^{\max}$ constraint.

				Vidal		Skewed-VNS			
Inst	n	t	m	$\overline{f}$	$\overline{T}$ [s]	$\overline{f}$	$\sigma(f)$	$\overline{T}$ [s]	gap
11a	360	4	11	9958.05	834.60	10351.34	55.01	1620.00	3.95%
12a	480	4	14	8011.50	953.40	8238.96	8.05	1664.00	2.84%
13a	600	4	17	12371.95	1826.40	12849.86	175.02	3382.00	3.86%
14a	720	4	21	9566.13	2007.00	9978.21	3.70	3587.00	4.31%
15a	840	4	25	14562.53	2695.80	15118.54	78.82	6994.00	3.82%
16a	960	4	29	11609.39	4488.60	12199.40	130.82	7785.00	5.08%
17a	360	6	8	16620.70	4207.20	17103.68	89.83	8242.00	2.91%
18a	520	6	12	13693.79	9427.20	14450.40	55.83	16691.00	5.53%
19a	700	6	16	19283.90	6665.40	19843.28	128.92	13194.00	2.90%
20a	880	6	20	15589.83	11472.60	16141.80	0.00	18705.00	3.54%
11b	360	4	9	21803.57	10575.00	22260.72	204.65	20995.00	2.10%
12b	480	4	11	17920.79	15141.00	18678.56	2.55	29437.00	4.23%
13b	600	4	14	10581.02	732.00	11245.36	143.65	1445.00	6.28%
14b	720	4	17	8629.90	792.00	9018.76	34.12	1395.00	4.51%
15b	840	4	20	14009.53	1293.60	14415.82	54.83	2428.00	2.90%
16b	960	4	23	11525.05	3216.60	11992.72	140.25	5266.00	4.06%
17b	360	6	6	18998.48	5717.40	19407.30	18.79	11174.00	2.15%
18b	520	6	9	14918.54	5529.00	15591.28	70.74	10462.00	4.51%
19b	700	6	12	22655.72	9033.60	23163.76	28.24	17442.00	2.24%
20b	880	6	15	18666.10	13726.80	19394.72	137.60	23918.00	3.90%
									3.78%

Table 7.15: SDVRPTW - Skewed VNS Results: Large Scale Instances - Average Objective Function Values

Besides the instances from [CL01], the SVNS algorithm is applied to overall 20 large-scale instances from [VCGP13] as well. The number of customers ranges from 360 to 880 with a fleet size up to 120 vehicles with up to 6 different vehicle types. Table 7.15 and 7.16 reports the obtained results from the SVNS algorithm. For each instance only 5 runs instead of 15 runs are performed to reduce running time. The obtained average objective values are reported in column $\overline{f}$ together with the standard derivation in column $\sigma(f)$ and the average running time in seconds in column $\overline{T}$ in Table 7.15. The best objective value $\overline{f_{\text{best}}}$ together with the corresponding running time $T(\overline{f_{\text{best}}})$ in seconds is listed in Table 7.16. In both tables the columns n, t and m corresponds to the number of customer, number of vehicle types and number of vehicles per vehicle type, respectively. The average gap (7.19) regarding the average objective value is 3.78%, whereas the average gap regarding the best objective values is 3.97%.

				Vidal	Skewed-VNS		
Inst	n	t	m	$\overline{f_{\text{best}}}$	$\overline{f_{\text{best}}}$	$T(\overline{f_{\text{best}}})$ [s]	gap
11a	360	4	11	9924.11	10295.60	1653.00	3.74%
12a	480	4	14	7962.22	8235.36	1549.00	3.43%
13a	600	4	17	12251.66	12633.20	3376.00	3.11%
14a	720	4	21	9508.68	9975.35	3655.00	4.91%
15a	840	4	25	14491.25	15039.00	7169.00	3.78%
16a	960	4	29	11562.67	12118.70	8950.00	4.81%
17a	360	6	8	16547.89	17037.80	7757.00	2.96%
18a	520	6	12	13623.28	14386.90	15505.00	5.61%
19a	700	6	16	19090.19	19750.70	13002.00	3.46%
20a	880	6	20	15437.52	16141.80	17480.00	4.56%
11b	360	4	9	21413.65	22162.40	21029.00	3.50%
12b	480	4	11	17834.61	18674.30	29721.00	4.71%
13b	600	4	14	10547.07	11090.60	1447.00	5.15%
14b	720	4	17	8562.99	8995.72	1213.00	5.05%
15b	840	4	20	13963.49	14385.50	2418.00	3.02%
16b	960	4	23	11477.72	11930.00	4558.00	3.94%
17b	360	6	6	18855.51	19389.80	10987.00	2.83%
18b	520	6	9	14894.65	15509.50	11044.00	4.13%
19b	700	6	12	22513.44	23126.30	17986.00	2.72%
20b	880	6	15	18566.66	19286.90	25091.00	3.88%
							3.97%

Table 7.16: SDVRPTW - Skewed VNS Results: Large Scale Instances - Best Objective Function Values

7.6 Experiments with the Branch-and-Cut Algorithm

Primary goal of the development of the exact methods was to analyze the solution quality of the VNS algorithm, i.e. to find optimal solutions for small problem instances. The compact model (Miller-Tucker-Zemlin-Formulation) itself was able to find feasible solutions, in particular when symmetry-breaking constraints (5.17) have been used. In some cases it was even possible to prove the optimality, i.e. to close the gap between primal solution and LP-relaxation. However, running times and memory consumption was not really satisfactory. For instance it took approximately 10 hours to obtain optimal solutions for the smallest instances from [CL01].

Significant improvements regarding the run-time behaviour could be obtained by separating valid inequalities by means of cutting-plane separation. Various combinations of these cuts have been implemented and tested.

In order to analyze the B&C algorithm, eight new test instances are created from the first eight instances pr01 - pr02 from [CL01]. This was done, by removing some customers and reducing the number of vehicles per type. Table 7.17 shows the characteristics of the new instances mip 1a - mip 8a as well as the results obtained by the B&C algorithm and the SVNS algorithm. Furthermore smallest instance from [CL01] is listened. Column n reports the number of customers for each instance, whereas columns t and m listen the number of different vehicle types and the number of vehicles per type, respectively. The obtained primal bound of the B&C algorithm is reported in column f as well as the running time in seconds of the B&C algorithm in column T. The number of B&B-Nodes is shown in column *Nodes*. The next five columns *twc* (time window cuts), *cec* (cycle elimination cuts), *dcc* (directed connection cuts), *tdc* (tour duration cuts) and *pcc* (packing constraints cuts) listen the number of separated inequalities. The column *gap* reports the gap between the primal bound and the dual bound. Hence, if the column reports a value of zero, then the B&C algorithm could prove the optimality of the obtained solution. The obtained results from the SVNS algorithm are reported in column $\overline{f}$ and $\overline{T}$. For each instance there are 15 runs performed. The obtained average objective value is listened in column $\overline{f}$, whereas the average running time in seconds is listened in column $\overline{T}$.

7.6.1 Time Window Cutting-Planes

Depending on the characteristics of the test instances the time window cutting planes turned out to be more or less useful. In case the time-windows are very narrow, i.e. the solution is already constrained very much, they were not really useful. In this situation the LP-relaxation of the MTZ-model already provides a relatively good description of the integral polyhedron. However, in case of wide time windows many of these cuts have been separated and helped to close the LP-gap faster. In this case the LP-relaxation of the compact model does not describe the polyhedron as good as in the case of narrow time windows. However, the cutting planes are not even very strong from a theoretical point of view. In most cases they were only able to cut off small parts of the polyhedron, or, in other terms the valid inequalities were only violated by relatively small values.

7.6.2 Subtour-Elimination Cuts

In this work two different cycle-elimination cuts have been separated. Both, the *cycle-elimination-cuts* 5.4.1 as well as the *directed-connection-cuts* 12 would ensure feasible routes (without subtours) on integral solutions, and thus render the MTZ constraints redundant. These MTZ-constraints give, however, a good polyhedral description when time-windows do exist. On the other hand, subtour elimination constraints themselves do not provide a full description of the facets of the integer polyhedron. Thus, they cannot be used as an alternative to the MTZ-formulation. They can, however, be used to strengthen the formulation.

The computational experience showed, that cycle-elimination cuts do not contribute as much as directed-connection-cuts do. The latter ones yielded a significant speedup of the overall performance of the branch-and-bound algorithm. Regarding the small instances the running times to find proven optimal solutions could be reduced in the order of magnitude of 50% or more. Hence, the instance 11a (from Table 7.2) could be solved within 3 hours as opposed to 10 hours without the cutting planes.

The number of branch-and-bound nodes could consequently also be reduced, from approximately 81087 to 53501.

Different separation strategies have been investigated as well. In general it turned out to be most beneficial to add all cuts to the global model, and not only to the considered node and its subtree. Both cutting-planes have been separated in an aggregated way (by computing sums over all vehicles for all customer nodes) and in a disagreggated way. The latter way showed to be far superior to the first method.

The results of the exact methods helped to evaluate the performance of the VNS framework, in particular in early stages. Altough the presented exact algorithm is not competitive with state-of-the-art exact methods, it provided valuable insights for this work. In particular when combined with the heuristic algorithms it is an important way to assess the quality of the incumbent (heuristic) solutions. In order to be able to solve larger instances, the application of column-generation techniques will further improve the overall performance. As such approaches are not directly supported within ILOG CPLEX, this is beyond the scope of this work.

				B&C									SVNS		
Inst	n	t	m	f	T [s]	Nodes	twc	cec	dcc	tdc	pcc	gap	$\overline{f}$	f_{best}	$\overline{T}$ [s]
1a	48	4	2	1655.42	9826	53921	834	20	2822	834	15	0.00%	1661.04	1544.42	15
mip 1a	27	4	1	827.19	1	0	2	0	4	0	0	0.00%	827.19	827.19	1
mip 2a	57	4	2	1792.71	9043	32441	590	18	3065	34	0	0.00%	1811.42	1792.71	33
mip 3a	56	4	2	1484.35	844	2261	147	4	709	10	63	0.00%	1495.85	1484.35	24
mip 4a	54	4	2	1839.57	36000	90189	1505	54	6486	136	275	5.62%	1853.19	1839.57	32
mip 5a	57	4	2	1822.54	1366	2649	331	14	2572	8	125	0.00%	1826.54	1822.54	29
mip 6a	57	4	2	2085.97	24566	52738	1868	26	5764	156	65	3.65%	2063.33	2051.27	48
mip 7a	48	6	1	1435.38	3422	10281	189	8	1175	18	18	0.00%	1441.99	1435.38	24
mip 8a	44	6	1	1145.22	18	0	5	0	74	0	0	0.00%	1145.22	1145.22	2

Table 7.17: B&C Results

7.7 Experiments with the DSDVRPTW

For the application in operational practice the dynamic problem DSDVRPTW with a sliding planning horizon is most important. In this case planning is performed every day based on new customers. However, delivery times must be arranged with customers and cannot simply be determined based on algorithmic planning. Operational experience shows, that most customers will accept the proposed appointments, and only a small fraction of them asks for different delivery times. In such a continuous planning scenario it is thus crucial to propose appointments that already minimizes total logistics costs. If, on the other hand, no appointments are proposed and delivery times are just arranged based on the customers initial proposal, the total logistics costs might be significantly higher.

In this section we show that good appointment proposals have a positive impact on total logistics costs even tough further customer requests are unknown at the time the decision is made. In practice, many aspects have to be considered: customers may want to change already arranged appointments, in certain situations not only times but also a particular vehicle needs to be planned for a customer, and service-times, fleet-availabilities and even customer addresses could change. Although the SVNS is capable of dealing with all these challenges, it is difficult to demonstrate the effects in this complex scenario.

Here, we restrict the analysis of the DSDVRPTW to the following scenario: customer-driven appointment arrangements are simulated with a very quick execution of the SVNS, as opposed to logistics-driven appointment arrangements simulated by SVNS results obtained with reasonable runtimes.

The experiments are performed with a benchmark instance (6a from Table 7.2) with 288 customers and a fleet size of 28 vehicles with 4 different vehicle types. We split this instance into five intervals, each containing approximately 20% of all customers. We start the simulation without any time windows defined. In the first step, a solution is computed for the first 20% of customers, for which appointments are then "arranged" (in the above sense) by setting a tight time window. The remaining 80% of customers are considered to be not yet known in this simulation. Nevertheless, the obtained new instance with time windows for 20% of the customers is then resolved to obtain a lower bound for the logistics costs at the end of the planning horizon. This process is then continued with the next 20% of the customers, and so on and so forth.

In the experiments two configurations are compared: a very quick execution of the SVNS, simulating the customer-driven appointment arrangement, and one with longer running times, simulating the logistics-driven arrangement.

Table 7.18 and Figure 7.1 show the results, obtained for 8 runs of each variant. In every single run the customers are considered in different permutations respectively. The results are compared to the objective value f_{opt} obtained by solving the whole instance in one step. Obviously, the objective values in the simulation scenario are higher, when decisions are made under incomplete knowledge (of future customers). The results also show, that

even with such incomplete information, sophisticated planning decisions are crucial to obtain a relatively small gap between the theoretical optimum (i.e. customers have no influence on delivery times) and results obtained by iteratively shifting the planning horizon. We obtain additional costs of $\approx 30\%$ for the dynamic planning scenario, and $\approx 55\%$ costs for customer driven planning. This is a $\approx 25\%$ improvement compared to the case when no operation-research techniques are used for planning.

	percentage of apriori known customers				
customer driven	20%	40%	60%	80%	100%
fixed customers	1581.74	2930.79	4173.30	5374.65	6722.86
remaining customers	5110.71	5533.00	5911.90	6360.36	7013.74
logistic driven	20%	40%	60%	80%	100%
fixed customers	1638.94	2717.17	3719.96	4755.14	5859.69
remaining customers	5062.09	5364.69	5723.16	5905.05	6122.26

Table 7.18: Obtained objective values from the SVNS algorithm by simulating the customer-driven appointment arrangements and the logistics-driven appointment arrangements scenario. The objective value f_{opt} obtained by solving the whole instance in one step is 4532.

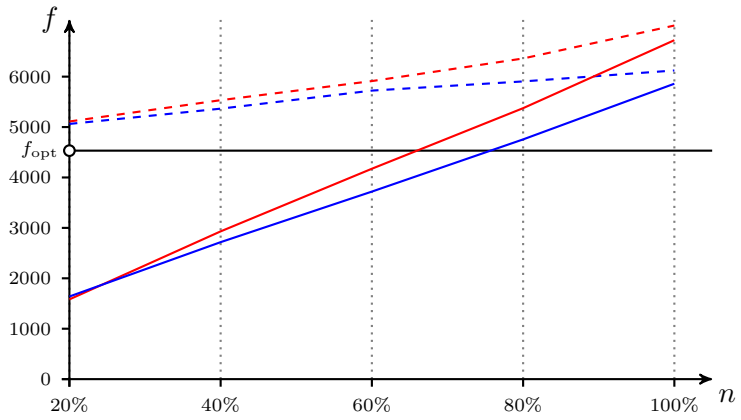


Figure 7.1: Simulation of customer-driven appointment arrangements (red) and logistics-driven appointment arrangements (blue). Solid lines indicate the obtained objective values from the two scenarios. Dashed lines indicates the obtained objective values when the remaining customers are known and scheduled as well.

7.8 Summary

The results presented in this chapter show the overall good performance of the VNS algorithm. It is important to mention, that all algorithms have been implemented within a complex algorithmic framework, and support many operations like fixing service times and vehicles for particular customers. Altough current state-of-the-art results could only be obtained in some cases, the average results clearly show that the algorithm is very competitive in this regard. Furthermore the obtained heuristic results show to have small variations, are thus reliable and robust and therefore well suited for practical applications.

CHAPTER 8

Conclusion

In this work I presented novel solution approaches to tackle the DSDVRPTW. Primary motivation for this work was to develop solution methods that are capable of solving real-world size instances, but also support all requirements for an operative setup.

The main part of this thesis consists of the presentation of an algorithmic framework to tackle difficult and large problem instances of the SDVRPTW. This goal has been primarily fulfilled with a VNS framework. One of the most important novelties of this approach is, that no construction algorithm is required prior to the neighborhood search. In contrast, the neighborhoods themselves are capable of constructing or enhancing solutions. Hence, no feasible solution is necessary as a starting point for the VNS algorithm. This is of importance, as the prerequisite of construction algorithms introduces several difficulties and problems. On the one hand, it may be very difficult to create feasible solutions by means of (more or less) sophisticated greedy steps. On the other hand it is an important requirement for practical purposes to be able to perform a warm-start, i.e. start the optimization process with a partial solution.

The presented algorithms also support further important requirements for practical applications. One important aspect is to compute solutions for planning horizons of many weeks. This planning horizon is then shifted day by day to include later dates. During this process new orders are considered within the planning horizon. Another important aspect is to be able to fix certain parts of the solution. For example, delivery times need to be arranged with customers at a certain point, and should not be changed afterwards. In some situations it might be also necessary to not only fix delivery times, but also the particular vehicle to supply this customer. Although these algorithmic components have been implemented, they are not presented in this work, as they are not really of academic interest.

In order to evaluate the solution quality of the VNS algorithms, certain exact approaches have been developed within this work. A mathematical programming formulation based

on Miller-Tucker-Zemlin inequalities is presented. This compact model is then enhanced with cutting planes that are separated within a B&C algorithm. Small and moderate sized instances can be solved to (proven) optimality with this approach.

Furthermore computational results are presented and compared to recent results published in the literature for certain benchmark instances. The heuristic algorithms showed a decent overall performance regarding robustness, solution quality and time. In many cases results close to the state-of-the-art results could be obtained.

Beyond the presented results, many computational experiments have been performed with real world instances provided by a customer of Destion. The preliminary results obtained within these experiments clearly indicate that the algorithmic framework fulfills all properties to be well suited for practical applications, and, in particular support all the additional requirements described above. A thorough description of this experimental setup and results is however beyond the scope of this work.

To demonstrate the dynamic aspects of the problem, the influence on logistic costs is analyzed if appointments with customers are proposed based on algorithmic planning or proposed by the customers. It turns out that proposals based on the SVNS algorithm yield to better overall logistic costs even if the planning horizon is shifted and therefore not all customers are known in advance.

Although many interesting results and insights have been obtained within this work, there are also many aspects that could be subject to further investigations. It would, for instance, be interesting to see how the proposed approach of constructing and enhancing solutions within the VNS framework would perform on other, similar VRP problems. It would also be very interesting to see how more shaking neighborhoods could possibly improve the overall performance of the search method. Regarding exact solution methods, techniques like column generation and more sophisticated cutting planes could also further improve the performance. These aspects are, however, beyond the scope of this thesis, and will be subject to future research.

List of Figures

List of Tables

List of Algorithms

Acronyms

ACO Ant Colony Optimization. 7, 11

B&B Branch & Bound. 10, 26, 27, 30, 32, 67

B&C Branch & Cut. 3, 4, 25, 30, 54, 56, 67, 69, 74–76

BIP Binary Integer Program. 26

BVNS Basic VNS. 14, 15

CI Cover Inequality. 34

COP Combinatorial Optimization Problem. 9–13, 26, 44

DFS Depth-First Search. 32

DSDVRPTW Dynamic Site-Dependent Vehicle Routing Problem with Time Windows. 17, 23, 27, 62, 70, 73, 75

DVRP Dynamic VRP. 6, 23

GA Genetic Algorithm. 7, 11

GVNS General VNS. 14

HVRP Heterogeneous VRP. 6–8

IP Integer Program. 26, 30

LP Linear Program. 25–27, 30–35, 67

LS Local Search. 7, 11–13

MDVRP Multiple Depots Vehicle Routing Problem. 6, 7

MIP Mixed Integer Program. 25–27, 30, 31, 33, 53

OP Optimization Problem. 9

PSO Particle Swarm Optimization. 11

PVRP Periodic VRP. 6, 7

RVNS Reduced VNS. 14

RVRP Rich VRP. 6, 7

SA Simulated Annealing. 11

SDCP Subset-Disjoint Cycle Problem. 46

SDMTPVRP Site-Dependent Multi-Trip Periodic Vehicle Routing Problem. 7

SDVRP Site-Dependent Vehicle Routing Problem. 2, 3, 6–8, 11, 17, 53, 54, 63, 64, 75, 76

SDVRPTW Site-Dependent Vehicle Routing Problem with Time Windows. xi, xiii, 4, 5, 7, 8, 24, 52–55, 62–66, 73, 75, 76

SSDVRPTW Static Site-Dependent Vehicle Routing Problem with Time Windows. 17, 20, 22, 23, 25, 37, 45, 54–56, 62

SVNS Skewed VNS. 14, 15, 53, 57–63, 65, 67, 69–71, 74

TS Tabu Search. 6, 7, 11

TSP Traveling Salesman Problem. 5

VND Variable Neighborhood Descent. 13, 14

VNS Variable Neighborhood Search. xiii, 3, 4, 7, 11, 13–15, 25, 37, 42, 50, 53, 57–61, 66, 68, 73, 74

VRP Vehicle Routing Problem. 1, 4–7, 74

VRPTW Vehicle Routing Problem with Time Windows. 5–7

Bibliography

[AAB08] Federico Alonso, MJ Alvarez, and John E Beasley. A tabu search algorithm for the periodic vehicle routing problem with multiple vehicle trips and accessibility restrictions. *Journal of the Operational Research Society*, 59(7):963–976, 2008.

[AMO93] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows: theory, algorithms, and applications*. Prentice hall, 1993.

[APSAL14] Pedro Amorim, Sophie N Parragh, Fabrício Sperandio, and Bernardo Almada-Lobo. A rich vehicle routing problem dealing with perishable food: a case study. *Top*, 22(2):489–508, 2014.

[Bal75] Egon Balas. Facets of the knapsack polytope. *Mathematical programming*, 8(1):146–164, 1975.

[BBMR10] Roberto Baldacci, Enrico Bartolini, Aristide Mingozzi, and Roberto Roberti. An exact solution framework for a broad class of vehicle routing problems. *Computational Management Science*, 7(3):229–268, 2010.

[Bel11] Slim Belhaiza. Hybrid variable neighborhood: Tabu search algorithm for the site dependent vehicle routing problem with time windows. Technical report, Department of Mathematics and Statistics, King Fahd University of Petroleum and Minerals, 2011.

[BG05a] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation science*, 39(1):104–118, 2005.

[BG05b] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation science*, 39(1):119–139, 2005.

[BM97] Jose Brandao and Alan Mercer. A tabu search algorithm for the multi-trip vehicle routing and scheduling problem. *European Journal of Operational Research*, 100(1):180–191, 1997.

[BR03] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[BRVN16] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300–313, 2016.

[CGL97] Jean-François Cordeau, Michel Gendreau, and Gilbert Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–119, 1997.

[CGS11] Paola Cappanera, Luís Gouveia, and Maria Grazia Scutellà. The skill vehicle routing problem. In *Network Optimization*, pages 354–364. Springer, 2011.

[CL01] Jean-François Cordeau and Gilbert Laporte. A tabu search algorithm for the site dependent vehicle routing problem with time windows. *INFOR-Information Systems and Operational Research*, 39(3):292–298, 2001.

[CLM04] Jean-François Cordeau, Gilbert Laporte, and Anne Mercier. Improved tabu search algorithm for the handling of route duration constraints in vehicle routing problems with time windows. *Journal of the Operational Research Society*, 55(5):542–546, 2004.

[CW64] Geoff Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

[Dan02] George B Dantzig. Linear programming. *Operations Research*, 50(1):42–47, 2002.

[DR59] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[EVR09] Burak Eksioglu, Arif Volkan Vural, and Arnold Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472–1483, 2009.

[HJMP00] Pierre Hansen, Brigitte Jaumard, Nenad Mladenović, and Anderson D. Parreira. Variable neighborhood search for weighted maximum satisfiability problem. Technical report, GERAD, Montreal, Canada, 2000.

[HJP75] Peter L Hammer, Ellis L Johnson, and Uri N Peled. Facet of regular 0–1 polytopes. *Mathematical Programming*, 8(1):179–206, 1975.

[HMBP10] Pierre Hansen, Nenad Mladenović, Jack Brimberg, and José A. Moreno Pérez. Variable neighborhood search. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, pages 61–86. Springer US, Boston, MA, 2010.

[HP02] Yu-Chi Ho and David L Pepyne. Simple explanation of the no-free-lunch theorem and its implications. *Journal of optimization theory and applications*, 115(3):549–570, 2002.

[JYP95] Jean-Marc Rousseau Jean-Yves Potvin. An exchange heuristic for routeing problems with time windows. *The Journal of the Operational Research Society*, 46(12):1433–1446, 1995.

[LD60] Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

[Lee04a] Jon Lee. *A first course in combinatorial optimization*, volume 36. Cambridge University Press, 2004.

[Lee04b] Jon Lee. *A first course in combinatorial optimization*, chapter Cutting Planes, pages 151–176. Volume 36 of [Lee04a], 2004.

[LKS15] Rahma Lahyani, Mahdi Khemakhem, and Frédéric Semet. Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research*, 241(1):1–14, 2015.

[MH97] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[MTFI+15] Jairo R Montoya-Torres, Julián López Franco, Santiago Nieto Isaza, Heriberto Felizzola Jiménez, and Nilson Herazo-Padilla. A literature review on the vehicle routing problem with multiple depots. *Computers & Industrial Engineering*, 79:115–129, 2015.

[MTZ60] Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[NW88a] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 605 Third Avenue, New York, N.Y. 10158-0012, 1988.

[NW88b] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*, chapter Strong Valid Inequalities and Facets for Structured Integer Programs, pages 259–291. In [NW88a], 1988.

[PGGM13] Victor Pillac, Michel Gendreau, Christelle Guéret, and Andrés L Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225(1):1–11, 2013.

[PR07] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007.

[PS82] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.

[RPH16] Ulrike Ritzinger, Jakob Puchinger, and Richard F Hartl. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, 54(1):215–231, 2016.

[Sav92] Martin WP Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA journal on computing*, 4(2):146–154, 1992.

[Sha02] Dushyant Sharma. *Cyclic exchange and related neighborhood structures for combinatorial optimization problems*. PhD thesis, Massachusetts Institute of Technology, 2002.

[Sol83] M. Solomon. Vehicle routing and scheduling with time window constraints: Models and algorithms. Technical report, College of Business Admin., Northeastern University, 1983.

[Sol87] Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[SQRE16] Juan Antonio Sicilia, Carlos Quemada, Beatriz Royo, and David Escuín. An optimization algorithm for solving the rich vehicle routing problem based on variable neighborhood search and tabu search metaheuristics. *Journal of Computational and Applied Mathematics*, 291:468–477, 2016.

[TO89] Paul Michael Thompson and James B Orlin. *The theory of cyclic transfers*. Citeseer, 1989.

[TP93] Paul M Thompson and Harilaos N Psaraftis. Cyclic transfer algorithm for multivehicle routing and scheduling problems. *Operations research*, 41(5):935–946, 1993.

[VCGP13] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1):475–489, 2013.

[WM97] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[Wol75] Laurence A Wolsey. Faces for a linear inequality in 0–1 variables. *Mathematical Programming*, 8(1):165–178, 1975.

[ZRM15] Esmat Zare-Reisabadi and S Hamid Mirmohammadi. Site dependent vehicle routing problem with soft time window: Modeling and solution approach. *Computers & Industrial Engineering*, 90:177–185, 2015.