FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Metaheuristics for the Districting and Routing Problem for Security Control

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computational Intelligence

by

Michael Prischink, BSc
Registration Number 0401740

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Assistance: Proj.-Ass. Dipl.-Ing. Christian Kloimüllner
Proj.-Ass. Dipl.-Ing. Benjamin Biesinger

Vienna, 3rd March, 2016

Michael Prischink

Günther Raidl

Erklärung zur Verfassung der Arbeit

Michael Prischink, BSc
Franz-Binder-Straße 47/2/69, 3100 St.Pölten

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, 3. März 2016

Michael Prischink

Danksagung

Ich möchte mich bei Günther Raidl, Christian Kloimüllner und Benjamin Biesinger für die hervorragende Betreuung dieser Diplomarbeit bedanken. Sie standen mir immer schnell und unermüdlich mit professionellen Ratschlägen und konstruktiver Kritik zur Verfügung. Bei Günter Kiechle und Fritz Payr von der CAPLAS GmbH möchte ich mich dafür bedanken, dass sie die dieser Arbeit zugrundeliegende Problemstellung an uns herangetragen haben und uns mit ihrem Feedback unterstützten.

Ganz besonders möchte ich meiner Familie und insbesondere meinen Eltern danken, ohne deren Unterstützung ein so sorgenfreies Studium nicht möglich gewesen wäre. Meiner Verlobten, Sabrina, die mich während meines Studiums stets unterstützte und zu diesem Abschluss ermutigte, bin ich ewig dankbar.

Kurzfassung

In dieser Arbeit wird das *Districting and Routing Problem for Security Control* eingeführt und als kombinatorisches Optimierungsproblem modelliert. Mehrere (Meta-)Heuristiken und exakte Methoden basierend auf *gemischt-ganzzahliger linearer Programmierung* werden zur Lösung des Problems verwendet. Die Ergebnisse der verschiedenen Ansätze werden anschließend analysiert und verglichen. Zum Abschluss wird ein Ausblick auf weitere Forschungsmöglichkeiten zu diesem neuen Problem gegeben.

Der private Sicherheitssektor ist ein stetig wachsendes Geschäft. Regelmäßige tägliche Sicherheitskontrollen sind ein essentieller und wichtiger Mechanismus um Diebstahl und Vandalismus in Firmengebäuden vorzubeugen. Typischerweise patrouillieren Mitarbeiter eines Sicherheitsdienstes durch eine Menge von Gebäuden, wobei jedes dieser Gebäude eine bestimmte Anzahl von Besuchen an allen oder nur an ausgewählten Tagen eines gegebenen Planungshorizonts benötigt und jeder Besuch in einem bestimmten Zeitfenster stattfinden muss. Ein wichtiges Ziel der Sicherheitsfirma ist daher, alle Gebäude in eine minimale Anzahl disjunkter Distrikte, d.h. Cluster, zu unterteilen, sodass für jeden Cluster und jeden Tag des Plannungszeitraums eine zulässige Route existiert, durch die alle notwendigen Objektbesuche abgedeckt werden. Jede Route ist begrenzt durch die tägliche Maximalarbeitszeit eines Mitarbeiters und muss die Zeitfenster aller Besuche einhalten. Je zwei Besuche des selben Gebäudes müssen einen vorgegebenen zeitlichen Mindestabstand einhalten. Wir nennen dieses Problem das *Districting and Routing Problem for Security Control*. In unserem heuristischen Ansatz teilen wir das Problem in einen Districting-Teil in dem jedes Gebäude einem Distrikt zugeteilt werden muss und einen Routing-Teil in dem zulässige Routen für jede Kombination von Distrikt und Planungsperiode gefunden werden müssen. Das Problem kann zwar in zwei Subprobleme zerlegt werden, diese können jedoch nicht unabhängig voneinander gelöst werden.

Der Districting-Teil des Problems wird gelöst, indem initiale Lösungen mittels einer *Districting Construction Heuristic* erzeugt werden und diese Lösungen durch einen *Iterative Destroy & Recreate* Algorithmus verbessert werden, indem versucht wird, die Anzahl der benötigten Distrikte zu minimieren. Um das Districting-Problem zu lösen, müssen viele Instanzen des Routing-Problems gelöst werden. Deshalb präsentieren wir ein Verfahren, mit dessen Hilfe die Gültigkeit einer gegebenen Route effizient überprüft werden kann. Initiale Lösungen für das Routing-Problem werden mittels einer adaptierten *Greedy Construction Heuristic* generiert, um gute Startlösungen für die darauffolgenden Verbesse-

rungsheuristiken zu erzeugen. Diese Lösungen werden anschließend mittels eines *Variable Neighborhood Descent* Ansatzes verbessert. Zusätzlich wird ein *gemischt-ganzzahliges lineares Programm* für den Routing-Teil vorgestellt. Die Ergebnisse unserer Tests zeigen, dass die Konstruktionsheuristiken Lösungen für das Routing-Problem erzeugen, die nahe an den unteren Schranken des exakten Algorithmus liegen und der *Iterative Destroy & Recreate* Algorithmus die Anzahl der Distrikte der Startlösungen, die von der *Districting Construction Heuristic* erzeugt wurden, signifikant reduzieren kann.

Abstract

In this thesis the *Districting and Routing Problem for Security Control* (DRPSC) is introduced and modeled as a combinatorial optimization problem. Multiple (meta-)heuristics and exact methods based on *mixed integer linear programming* are considered to practically solve the problem. The results of these different approaches are then analyzed and compared. Finally, an outlook on future work for this new problem is given.

The private security sector is a steadily growing business. Regular security controls on a day by day basis are an essential and important mechanism to prevent theft and vandalism in commercial buildings. Typically, security workers patrol through a set of objects where each object requires a particular number of visits on all or some days within a given planning horizon, and each of these visits has to be performed in a specific time window. An important goal of the security company is to partition all objects into a minimum number of disjoint clusters, called districts, such that for each cluster and each day of the planning horizon a feasible route for performing all the requested visits exists. Each route is limited by a maximum working time and has to satisfy the visits' time window constraints. Any two visits of an object must be separated by a minimum separation time. We call this problem the *Districting and Routing Problem for Security Control*. In our heuristic approach we split the problem into a districting part where objects have to be assigned to districts and a routing part where feasible routes for each combination of district and period have to be found. Although the problem can be decomposed, these parts cannot be solved independently.

The districting part of the problem is solved by generating initial solutions using a *districting construction heuristic* and improving the initial solutions by applying an *iterative destroy & recreate* algorithm trying to minimize the number of districts. In the course of solving the districting problem, feasible solutions for many instances of the routing problem have to be found. Therefore, we present an efficient method for checking the feasibility of a given route. Initial solutions to the routing problem are generated with a routing construction heuristic in a greedy fashion resulting in good starting solutions for the following improvement heuristics. These solutions are then improved using a *variable neighbourhood descent* approach. Additionally, an exact *mixed integer linear programming* model for the routing part is proposed. Computational results show that the *routing construction heuristics* is able to generate solutions close to the lower bounds provided by the exact algorithm and the *iterative destroy & recreate* algorithm is able to

reduce the number of districts significantly from the starting solutions, overall yielding very plausible solutions.

Contents

CHAPTER 1

Introduction

The area of private security control for protecting public and private property is a steadily growing business [47]. As in the area of private security control constant surveillance of an object might not be economically viable or even necessary, security firms have to send security guards to visit a large number of sites multiple times over the course of a day in order to fulfill their custodial duty. Therefore, security companies face the problem of having to schedule tours for their employees in order to cover all needed visits of all objects of customers under their guardianship. It is economically viable, to strive for generating as few tours as possible, since this reduces the number of employees needed for the patrolling business. The complexity of this task leaves a high potential for solving it by algorithmic techniques to minimize the number of employees needed to cover all customers.

Thus, we propose the *Districting and Routing Problem for Security Control* (DRPSC)[1] which consists of a districting part and a routing part. In the districting part all objects have to be partitioned into a minimum number of disjoint subsets, such that all objects of a single district can be serviced by a single security guard (or team of guards) within each working day of a planning horizon. Given such a partitioning, a routing problem has to be solved for each combination of district and day. We seek for a tour starting and ending at a central location which satisfies a maximum tour duration, e.g., the work shift duration, and the time window constraints for each visit of that period. In case multiple visits are required at an object in the same period, there typically has to be a separation time between consecutive visits to ensure a better distribution over time. For minimizing the number of districts, it is important to minimize the duration of the planned tours in order to incorporate as many objects into the resulting districts as possible, which shows the inseparability of the districting and routing part. Figure 1.1 schematically shows a problem instance and an example for a solution for the districting and routing problem.

[1] The problem specification was derived from real-world scenarios provided by CAPLAS GmbH

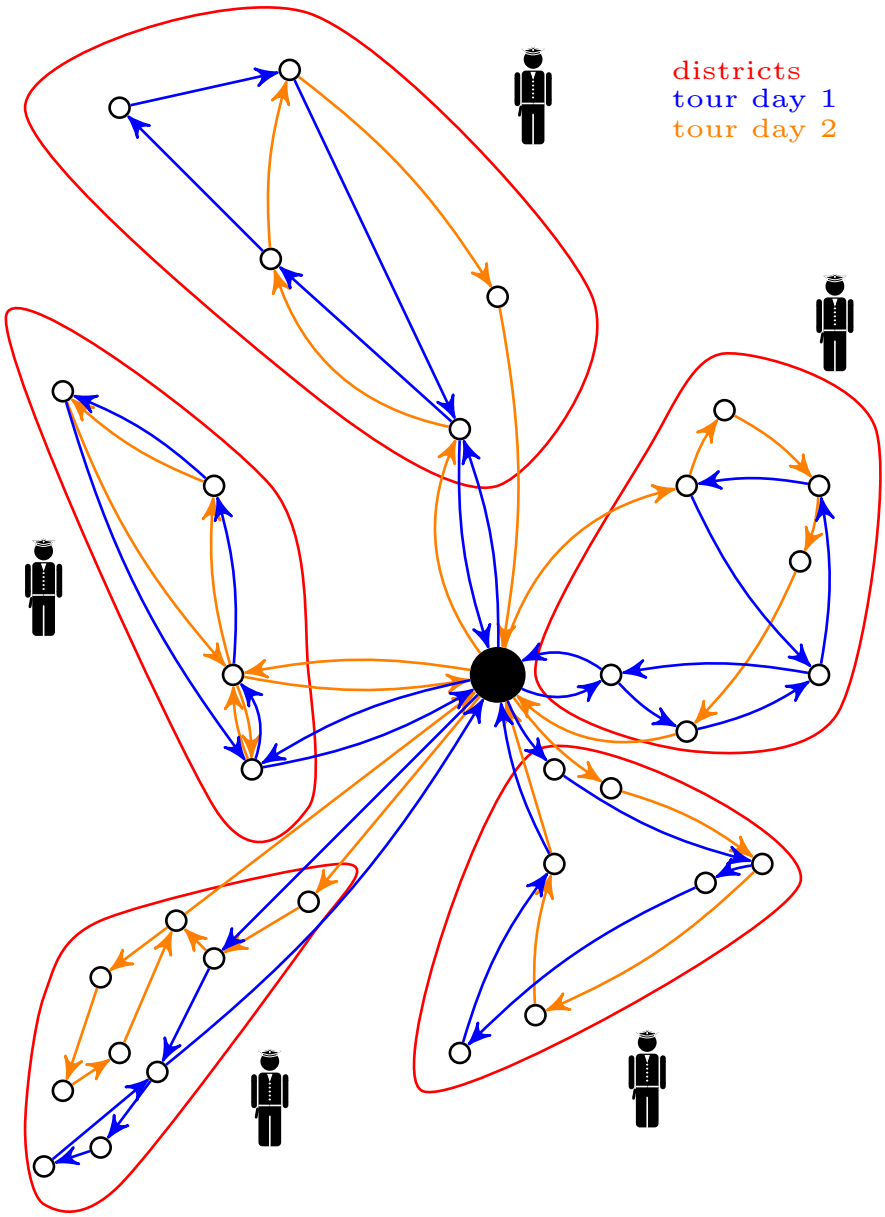

Figure 1.1: Example for a solution of an instance of the DRPSC: objects are clustered into districts and for each day of the planning horizon (e.g., 2 days) and for each district, a tour is given, that contains all visits, that have been requested.

We address the routing part of the problem by an exact mixed integer linear programming formulation (MIP) based on Miller-Tucker-Zemlin (MTZ) [28] inequalities and a routing construction heuristic in a greedy fashion with a subsequent variable neighborhood descent (VND). For the districting part an exact MIP formulation is stated and an iterative destroy & recreate (IDR) approach based on the route elimination algorithm by Nagata and Bräysy [32] for the vehicle routing problem with time windows is proposed. The starting solutions for the IDR are generated using a districting construction heuristic (DCH) based on a greedy insertion heuristic.

This thesis is structured as follows:

- The remainder of Chapter 1 gives a formal definition of the Districting and Routing Problem for Security Control with all its constraints. Additionally, the complexity of the problem is discussed.
- Chapter 2 gives a literature overview of problems related to the DRPSC and discusses the most important approaches to those problems with regard to solving the DRPSC.
- Chapter 3 describes the algorithms and heuristics used in this thesis. The concept of linear discrete optimization via mixed integer linear programming, the heuristic methods of greedy construction heuristics, variable neighborhood descent and regret heuristics and the route elimination algorithm are introduced.
- Chapter 4 explains in depth, how the routing part of the DRPSC was solved. Therefore, the means to efficiently checking the feasibility of a given route are described. Moreover, an exact mixed integer linear programming model and multiple strategies for greedy construction heuristics with a subsequent variable neighborhood descent are presented.
- In Chapter 5, the approaches to solve the districting part of the DRPSC are outlined. We give an exact approach in the form of a mixed integer linear programming model of the districting problem. Furthermore, multiple variants of a DCH combined with a regret heuristic for generating initial solutions are discussed. Finally, we show how an iterative destroy & recreate algorithm further minimizes the number of districts of these solutions.
- Chapter 6 explains how the test data was generated that has been used for the benchmarks. The results of the approaches for the routing and the districting part are presented and analyzed separately.
- Chapter 7 summarizes the findings of this thesis and gives an outlook on future work on the DRPSC.

The results of this thesis have been accepted at the 10th International Workshop on Hybrid Metaheuristics:

Michael Prischink, Christian Kloimüllner, Benjamin Biesinger, and Günther R. Raidl. Districting and routing for security control. In *Hybrid Metaheuristics, 10th Int. Workshop, HM 2016*, Lecture Notes in Computer Science. Springer, 2016. to appear

1.1 Problem Definition

This Section formalizes the Districting and Routing Problem for Security Control. We are given a set of objects $I = \{1, \ldots, n\}$ and a starting location 0, which we call in relation to the usual terminology in vehicle routing *depot*. There are p planning periods (days) $P = \{1, \ldots, p\}$, and for each object $i \in I$ a set of visits $S_i = \{i_1, \ldots, i_{|S_i|}\}$ is defined. Not all visits, however, have to take place in each period. The visits requested in period $j \in P$ for object $i \in I$ are given by subset $W_{i,j} \subseteq S_i$.

For each visit $i_k \in S_i$, $i \in I$, $k = 1, \ldots, |S_i|$, we are given its duration $t^{\text{visit}}_{i_k} \geq 0$ and a time window $T_{i_k} = [T^{\text{e}}_{i_k}, T^{\text{l}}_{i_k}]$, during which the whole visit including its visit time has to take place. The time windows of successive visits of an object may also overlap but visit i_k always has to start before a visit $i_{k'}$ with $k, k' \in W_{i,j},\ k < k'$ and they have to be separated by a minimum duration of t^{sep}. The maximum duration, i.e., the makespan, of each planned tour must not exceed a global maximum duration $t^{\max}$.

Next, we define underlying graphs on which our proposed algorithms operate. For each period $j \in P$ we define a directed graph $G^j = (V^j, A^j)$ where V^j refers to the set of visits requested at corresponding objects, i.e., $V^j = \bigcup_{i \in I} W_{i,j}$, and the arc set is: $A^j = \{(i_k, i'_{k'}) \mid i_k \in W_{i,j}, i'_{k'} \in W_{i',j}\} \setminus \{(i_k, i_{k'}) \mid i_k, i_{k'} \in W_{i,j}, k' \leq k\}$. We have arc weights associated with every arc in $A^j, j \in P$ which are given by $t^{\text{travel}}_{i,i'}$, the duration of the fastest connection from object i to object i'. We assume that the triangle inequality holds among these travel times. Let us further define the special nodes 0_0 and 0_1 representing the start and end of a tour and the augmented node set $\hat{V}^j = V^j \cup \{0_0, 0_1\}$, $\forall j \in P$. Accordingly, we add outgoing arcs from node 0_0 to all visits $i_k \in V^j$ and arcs from all visits $i_k \in V^j$ to node 0_1, formally, $\hat{A}^j = A^j \cup \{(0_0, i_k) \mid i_k \in V^j\} \cup \{(i_k, 0_1) \mid i_k \in V^j\}$. Consequently, we define the augmented graph $\hat{G}^j = (\hat{V}^j, \hat{A}^j)$.

The goal of the DRPSC is to assign all objects in I to a smallest possible set of districts $R = \{1, \ldots, \delta\}$, i.e., to partition I into δ disjoint subsets I_r, $r \in R$, with $I_r \cap I_{r'} = \emptyset$ for $r, r' \in R,\ r \neq r'$ and $\bigcup_{r \in R} I_r = I$, so that a feasible tour $\tau_{r,j}$ exists for each district I_r, $r \in R$ and each planning period $j \in P$. A tour $\tau_{r,j} = (\tau_{r,j,0}, \tau_{r,j,1}, \ldots, \tau_{r,j,l_{r,j}}, \tau_{r,j,l_{r,j}+1})$ with $\tau_{r,j,0} = 0_0, \tau_{r,j,l_{r,j}+1} = 0_1$, $l_{r,j} = \sum_{i \in I_r} |W_{i,j}|$, and $\tau_{r,j,1}, \ldots, \tau_{r,j,l_{r,j}} \in \bigcup_{i \in I_r} W_{i,j}$ has to start at the depot node 0_0, has to perform each visit $i_k \in W_{i,j}$ in the respective sequence for each object $i \in I_r$ exactly once, and finally has to return back to the depot, i.e., reach node (0_1). A tour $\tau_{r,j}$ is feasible if each visit $\tau_{r,j,u}$, $u = 0, \ldots, l_{r,j} + 1$ including its visit time can take place in its time window T_{i_k}, with waiting before a visit is allowed,

the minimum duration t^{sep} between visits of the same object is fulfilled, and the total tour duration, i.e., the makespan, does not exceed $t^{\max}$.

Note that the routing part can be solved for a given district I_r and period $j \in P$ separately and consists of finding a feasible tour $\tau_{r,j}$.

1.2 Complexity

It has been shown that deciding whether there exists a feasible tour for the TSPTW is an NP-complete problem [44]. The TSPTW is the problem of finding the shortest tour visiting all nodes of a given set of vertices exactly once and then returning to the starting node while visiting each node within its given time window. We show that any algorithm that solves the DRPSC also solves the TSPTW. We reduce the number of requested visits for each object to 1, relax the maximum allowed tour duration and reduce the number of periods to 1, i.e., $|S_i| = 1, \forall i$, $t^{\max} = \infty$ and $|P| = 1$. It is easy to see that the following decision problem of the DRPSC also decides the decision problem of the TSPTW. Does a feasible solution with ≤ 1 districts exist? This proves the NP-hardness of the DRPSC.

Many variants of vehicle routing problems are hard to solve in practice. Exact approaches are usually only able to solve smaller instances. In the literature many (meta-)heuristics have been applied to various vehicle routing problems with great success [23]. Therefore, we also propose a heuristic approach for the DRPSC which is able to solve real-world instances.

CHAPTER 2

Related Work

To the best of our knowledge there is no work covering all the aspects of the Districting and Routing Problem for Security Control as considered here. But the literature contains many routing and scheduling problems that share many properties of the DRPSC, which leads to promising approaches. In this chapter the most important of these related problems are introduced. Amongst these problems are the vehicle routing problem with time windows (VRPTW), the pickup and delivery problem with time windows (PDPTW), the traveling salesman problem with time windows (TSPTW), the periodic vehicle routing problem with time windows (PVRPTW) and especially its generalization, the fairly new periodic vehicle routing problem with time windows and time spread constraints (PVRPTS). These time spread constraints are somewhat similar to the separation time of the DRPSC.

2.1 Vehicle Routing Problem with Time Windows

The VRPTW is a well-studied problem. In decades of research many exact and heuristic methods have been proposed for solving it [46, 21, 40, 33, 48]. It consists of a number of customers that have to be serviced by a set of vehicles starting and ending at a depot. Each vehicle has a finite capacity and each customer has a demand to be serviced. Customers can only be serviced within their respective time windows. The objective of the VRPTW is to minimize the number of tours (or vehicles) needed to service all customers, the total distance covered while servicing all customers, or a combination of these objectives. A majority of the literature focuses specifically on minimizing the total length of each tour without taking the makespan of the planned tours into account [40, 33, 48], whereas the objective of the routing part of the DRPSC is to find a tour, whose makespan is less or equal to the maximum tour duration $t^{\max}$. A common approach is to focus on minimizing the number of needed routes first and only in a second step minimizing the travel time or makespan, e.g., by using a hierarchical objective

function [33, 40]. Nagata and Bräysy [32] propose a route minimization heuristic which in particular tries to minimize the number of routes needed to service all customers. They rely on a destroy-and-recreate heuristic which iteratively tries to delete routes while maintaining an ejection pool (EP). This EP stores all objects which are yet to be inserted. The algorithm tries to identify objects which are difficult to insert in one of the current routes and utilizes this information for choosing objects to be removed and re-inserted. As this approach produced excellent results for the VRPTW we adopt this basic idea of destroy-and-recreate here. Exact solution approaches for the VRPTW were proposed by Ascheuer et al. [1] who developed a branch-and-cut algorithm using several valid inequalities and were able to solve most instances with up to 50–70 nodes to proven optimality. Dash et al. [12] introduced a time bucket formulation and new cutting planes which could solve instances with up to 200 nodes. Baldacci et al. [2] introduce the ngL-tour relaxation. By using column generation as well as dynamic programming they are able to solve instances with up to 233 nodes to optimality and report new optimal solutions that have not been found previously. A current state-of-the-art method for heuristically solving several variants of the VRPTW is a hybrid genetic algorithm (GA) by Vidal et al. [48]. As many other approaches described in the literature [33, 40] they use a penalty function for handling infeasible routes, which is described in more detail in [33]. In the GA the initial solutions are created randomly but there are also more elaborate construction heuristics available: Solomon [46] proposes several algorithms for constructing only feasible solutions by extending the well-known savings heuristic, a nearest neighbor heuristic, and insertion heuristics using different criteria.

2.2 Pickup and Delivery Problem with Time Windows

The PDPTW is a generalization of the VRP [13]. A number of requests for transporting goods from pickup to delivery locations have to be fullfilled. The objective is to construct a number of routes that satisfy all these requests while corresponding pickup and delivery requests have to be on the same route and the delivery location must be visited after its corresponding pickup location. For this problem also time window constraints for picking up and delivering goods and capacity constraints for the vehicles have to be considered.

2.3 Traveling Salesman Problem with Time Windows

The TSPTW seeks for a minimum tour visiting a given set of nodes and returning back to the starting location. Additionally, time window constraints for each node have to be considered. For the TSPTW, the most common objective in the literature is to minimize the total travel time, rather than minimizing the makespan of the tour [35, 10, 25, 30]. As the practical difficulty of the problems usually increases when makespan minimization is considered, specialized algorithms have been developed for this purpose [8, 14]. The routing part of the DRPSC is similar to the TSPTW as the aim is to find a feasible tour of duration less than a prespecified value which is related to the minimization problem of the TSPTW. In the TSPTW, however, multiple visits of the same objects and a separation

time between them are not considered. Interestingly, López-Ibáñez et al. [26] showed that by adapting two state-of-the-art metaheuristics for travel time minimization of the TSPTW [25, 35] to makespan minimization it is possible to outperform the specialized algorithms. Ascheuer et al. [1] proposed a number of simple construction heuristics for the asymmetric TSPTW which could be adapted for the DRPSC which delivered quick but not always feasible solutions for the routing part.

2.4 Periodic Vehicle Routing Problem with Time Windows

The PVRPTW generalizes the VRPTW by adding a planning horizon of multiple days. Customers do not require daily service. Instead, there is a number of different patterns, i.e., combination of days, whereas service is only required on the days of one such pattern for each customer, resulting in different tours for each day of the planning horizon. A hybrid genetic algorithm combining population-based methods with two neighborhood-based metaheuristics resulted in very high quality solutions for previously published benchmark instances [34].

2.5 Periodic Vehicle Routing Problem with Time Windows and Time Spread Constraints

An interesting related problem, which also arises in the field of security, is the PVRPTS [27]. The idea behind the problem is to generate unpredictable tours for money transporters by scheduling two visits of the same customer at different times. The PVRPTS is therefore a generalization of the PVRPTW, where arrival times at a customer must vary over the course of multiple periods, i.e., the arrival times between any two visits in all periods must differ in more than a pre-specified value. A similar problem arises when solving the DRPSC, where two visits of the same object within one tour have to be separated by a given separation time. Michallet et al. [27] proposed a mixed integer linear model and a multi-start iterated local search for solving this problem.

CHAPTER 3

Methods

In this chapter the algorithms and heuristic methods, on which the approaches in this thesis for solving the DRPSC are based, are explained. The optimization technique of mixed integer linear programming was used in an approach to find exact solutions to the routing part of the DRPSC. Since these exact methods were unable to solve larger problem instances, different metaheuristic approaches were examined. First, different greedy construction heuristics were applied to the problem instances to generate good initial solutions both for the routing and the districting part within reasonable running times. Then, the local search strategy of variable neighborhood descent (VND) was used to increase the quality of these initial solutions. To further improve on these construction heuristics, we also integrated a regret heuristic into the construction heuristics. Finally, the so called route elimininination algorithm is introduced, which we adapted for the districting part of the DRPSC.

3.1 Mixed Integer Linear Programming

First, the basics of linear programming (LP) are introduced, following the book of Bertsimas and Tsitsiklis [4], and then (mixed) integer linear programming is explained.

3.1.1 Linear Programming

A general linear programming problem is the problem of minimizing or maximizing a linear cost function $f(\mathbf{x}) = c_1 \cdot x_1 + c_2 \cdot x_2 + \cdots + c_n \cdot x_n$ according to a cost vector $\mathbf{c} = (c_1, \ldots, c_n)$ and subject to linear constraints. Let M_1, M_2, M_3 be finite index sets and let N_1 and N_2 be index sets defining which of the variables from vector $\mathbf{x}$ are constrained to be either nonnegative or nonpositive. Then, a linear programming problem is stated

as follows:

$$\begin{aligned}
&\text{minimize} && \mathbf{c}'\mathbf{x} && && (3.1)\\
&\text{subject to} && \mathbf{a_i}'\mathbf{x} \geq b_i && \forall i \in M_1 && (3.2)\\
& && \mathbf{a_i}'\mathbf{x} \leq b_i && \forall i \in M_2 && (3.3)\\
& && \mathbf{a_i}'\mathbf{x} = b_i && \forall i \in M_3 && (3.4)\\
& && x_j \geq 0 && \forall j \in N_1 && (3.5)\\
& && x_j \leq 0 && \forall j \in N_2 && (3.6)
\end{aligned}$$

In linear programming the variables of the $\mathbf{x}$ vector are called *decision variables* and the term (3.1) is denoted as *objective function* whereas inequalities respectively equalities (3.2)-(3.4) are called *constraints*. Any vector $\mathbf{x}$ satisfying all constraints as well as all nonnegativity and nonpositivity restrictions is called feasible solution. There are programs for which there exists no feasible solution, one feasible solution or multiple feasible solutions, and linear programming formulations may also be unbounded which means that the value of the objective function can grow arbitrarily small. If there does not exist any feasible solution the problem is also said to be infeasible, if there exists only one feasible solution, this solution is the optimal solution at the same time. A feasible solution $\mathbf{x}^*$ that minimizes the objective function (3.1) is also called an *optimal solution* and the value of $\mathbf{c}'\mathbf{x}^*$ is then called the *optimal cost*. In case the linear program is unbounded the optimal cost is said to be $-\infty$.

General Form

We infer *transformation rules* which can be applied to every linear program:

objective function any objective function which maximizes the cost function can be rewritten to minimize it, i.e., $\max\{\mathbf{c}'\mathbf{x}\} \Leftrightarrow \min -\mathbf{c}'\mathbf{x}$

equalities any equality can be rewritten in the form of the two inequalities, i.e., $\mathbf{a_i}'\mathbf{x} = b_i \Leftrightarrow \mathbf{a_i}'\mathbf{x} \leq b_i \wedge \mathbf{a_i}'\mathbf{x} \geq b_i$

inequalities any inequality of the form $\mathbf{a_i}'\mathbf{x} \leq b_i$ can be rewritten in the form $-\mathbf{a_i}'\mathbf{x} \geq -b_i$

Thus, maximization problems can always be transformed into minimization problems and according to the transformation rules any constraint of a linear program can always be rewritten to be in the form of

$$\mathbf{a_i}'\mathbf{x} \geq b_i \tag{3.7}$$

which leads to the definition of the (compact) *general form* for linear programs:

$$\begin{aligned}
&\text{minimize} && \mathbf{c}'\mathbf{x} && (3.8)\\
&\text{subject to} && \mathbf{A}\mathbf{x} \geq \mathbf{b} && (3.9)\\
& && \mathbf{x} \in \mathbb{R}^n && (3.10)
\end{aligned}$$

Standard Form

A linear program of the form

$$\begin{aligned} &\text{minimize} && \mathbf{c}'\mathbf{x} && (3.11)\\ &\text{subject to} && \mathbf{A}\mathbf{x} = \mathbf{b} && (3.12)\\ & && \mathbf{x} \in \mathbb{R}^n \geq 0 && (3.13) \end{aligned}$$

is said to be in *standard form*. Any linear program stated in *general form* can be translated in *standard form*, and as already said that any linear program can be converted into general form we conclude that any linear program can also be converted into *standard form*. To this purpose we define the following rules to transform general-form linear programs into standard form:

eliminating free variables Any unrestricted variable x_j in general form can be rewritten as the difference of two nonnegative numbers $x_j^+ - x_j^-$.

eliminating inequality constraints Any inequality constraint of the form $\mathbf{a_i x} \geq b_i$ can be rewritten as $\mathbf{a_i x} + s_i = b_i$ with $s_i \geq 0$ where s_i is called *surplus variable*. Similarly, any inequality fo the form $\mathbf{a_i x} \leq b_i$ can be rewritten as $\mathbf{a_i x} - s_i = b_i$ with $s_i \geq 0$ where s_i is called *slack variable*.

Standard form is computationally more convenient and is used by practical solution methods for linear programs, such as the *simplex method* [11] or the *ellipsoid method* [50]. Today, most commercial and open-source solvers implement the simplex algorithm for solving linear programs.

3.1.2 (Mixed) Integer Linear Programming

Many real-world problems can be modeled by *(mixed) integer linear programs* (MIP). The definition of linear programs is extended by adding variables which are constrained to be integer. Here, we distinguish between general integer variables and binary variables. Binary variables are often useful in MIPs to model decisions, e.g., which facility to open, which arcs to traverse, in network design or routing problems. We infer the terms *integer linear program* (IP), *mixed integer linear program* (MIP), and *binary integer linear program* (BIP) where the former consists of decision variables constrained solely to be integer, MIPs have integer as well as continuous variables, and the latter consist only of binary decision variables. A mixed integer linear program is defined as follows:

$$\begin{aligned} &\text{minimize} && \mathbf{c_1 x_1} + \mathbf{c_2 x_2} && (3.14)\\ &\text{subject to} && A_1\mathbf{x_1} + A_2\mathbf{x_2} \leq \mathbf{b} && (3.15)\\ & && \mathbf{x_1} \geq 0 && (3.16)\\ & && \mathbf{x_2} \geq 0 \text{ and integer} && (3.17) \end{aligned}$$

When relaxing the integer constraints on the variables of vector $\mathbf{x_2}$, it is called *linear programming relaxation* (LP relaxation) of the MIP. When taking minimization problems into account, the solution to the LP relaxation provides a lower bound on the optimal solution to the MIP. Note, that rounding the optimal solution of the LP relaxation is not necessarily an optimal solution to the MIP.

A solution $\mathbf{x}^*$ with value $z^* = c(x^*)$ to a MIP of the form

$$\min\{c(\mathbf{x_1}, \mathbf{x_2}) \mid \mathbf{x_1} \in X_1 \subseteq \mathbb{Z}^n, \mathbf{x_2} \in X_2 \subseteq \mathbb{R}^n\} \tag{3.18}$$

is optimal, if there is a lower bound $\underline{z}$ and an upper bound $\bar{z}$ on the optimal cost value such that $\underline{z} = z = \bar{z}$.

The usual approach is to find steadily improving upper and lower bounds, such that the upper bounds decrease and lower bound increase through a run of the solving algorithm. Obviously, the algorithm can stop if equality (3.18) is fulfilled. Feasible solutions, for instance, can be used as bounds. If we are facing a maximization problem, then a lower bound to the problem could be a feasible solution of a construction- and/or a metaheuristic. These bounds are called *primal bounds*. On the other side, there is also the need of finding upper bounds for maximization problems and lower bounds for minimization problems. These bounds are called *dual bounds* and can be obtained by solving relaxations of the given problem to proven optimality or by obtaining any feasible solution of the dual of the problem.

Branch-and-Bound

The solution strategy of branch-and-bound is implemented by all available commercial and open-source MIP solvers. The idea behind this strategy is to generate smaller and easier solvable problems from the whole problem and solve these subproblems. If all subproblems are solved, that information can be used to get a solution to the overall problem. Most solvers generate a branch-and-bound tree where they fix some values of the variables, and solve the subproblems with these fixed values. However, if this mechanism would be used without exploiting any further information it would result in a complete enumeration of possible solutions to the problem which is not computationally tractable for most combinatorial optimization problems. Thus, the information on lower and upper bounds of the subproblems can be exploited and branches can be pruned according to the following strategies:

prune by optimality if the lower and upper bound for a given subproblem k is the same, this branch can be pruned, i.e., $\bar{z_k} = \underline{z}_k$

prune by bound consider we are given a maximization problem and the upper bound of subproblem k is smaller than the global lower bound we can prune this branch, i.e., $\bar{z_k} \leq \underline{z}$.

prune by infeasibility if a given subproblem is infeasible, this branch can be pruned.

There are many possible ways to apply and implement branch-and-bound, such as how to obtain good upper bounds in case of a maximization problem. Usually the LP-Relaxation is used resulting in LP-based branch-and-bound. There also exist multiple branching strategies and several options on how to examine the created branches or subproblems, respectively. The interested reader is referred to the book *Integer Programming* by Laurence A. Wolsey [49] which is a good literature and reference for integer programming method, techniques, solution approaches, decomposition mechanisms and more.

3.2 Greedy Construction Heuristics

For optimization problems *greedy construction heuristics* (GCH) have proven to be efficient in yielding mostly good results within a short amount of time. The idea of a GCH is to iteratively find a solution by choosing the locally best successor with the help of a predefined greedy evaluation function in building up a solution while never reverting any decision made earlier. Although there are many problems, where a greedy heuristic always finds an optimal solution, for many other problems these locally optimal choices do not lead to an optimal global solution [9].

3.3 Variable Neighborhood Search

Variable neighborhood search (VNS) was first introduced by Mladenović and Hansen [29] for solving combinatorial optimization problems. The observation, that a local optimum is often very close to local optima of other neighborhoods and the fact that a global optimum is a local optimum for all possible neighborhoods, led to the idea of systematically searching for a local optimum in multiple neighborhoods and escaping those local optima in a perturbation phase [17].

3.3.1 Variable Neighborhood Descent

The *variable neighborhood descent* (VND) method changes neighborhood structures deterministically. The VND finds a local optimum for an initial solution x with respect to $l_{\max}$ neighborhoods $N_1, \ldots, N_{l_{\max}}$ as presented in algorithm 1.

The VND searches the first neighborhood of a starting solution x for the best neighbor solution x' (line 3). If the best neighbor solution x' is better than the current solution x (line 4), make it the new incumbent solution (line 5) and restart the search at the first neighborhood structure (line 6). If the best neighbor solution does not improve on the current solution, move to the next neighborhood (line 8) and search that neighborhood for a better solution. The heuristic stops, when the last neighborhood $N_{k_{\max}}$ was searched and no further improvement was found (line 10). Since finding the best neighbor solution may be time-consuming, the alternative is to use a first improvement strategy for selecting a neighbor solution x'. Instead of selecting the best neighbor solution, the first neighbor solution x' that yields an improvement over the current solution x is chosen.

Algorithm 1 Variable neighborhood descent (VND)

Function VND(x, $l_{\max}$)

1: $l \leftarrow 1$
2: **repeat**
3: $x' \leftarrow$ choose a neighbor in $N_l(x)$
4: **if** solution x' is better than x **then**
5: $x \leftarrow x'$ // keep new best solution
6: $l \leftarrow 1$ // go to the first neighborhood
7: **else**
8: $l \leftarrow l + 1$ // move to the next neighborhood
9: **end if**
10: **until** $l = l_{\max}$

3.3.2 General Variable Neighborhood Search

The general *variable neighborhood search* (GVNS) uses a perturbation step in each iteration followed by a VND until a time limit is reached. A different neighborhood than for the VND is used for the so called shaking operation to escape local optima. The algorithm is shown in 2. For each iteration, before the VND is executed (line 5), the neighborhood is randomly changed in a shaking operation (line 4). If the new solution obtained by the VND is better than the current solution (line 6), it becomes the new incumbent solution (line 7) and the first neighborhood is used for the next iteration (line 8), otherwise the GVNS moves to the next neighborhood (line 10). When a given time limit is reached, the algorithm returns with the best solution found.

Algorithm 2 General VNS

Function GVNS(x, $l_{\max}$, $k_{\max}$)

1: **repeat**
2: $k \leftarrow 1$
3: **repeat**
4: $x' \leftarrow$ random neighbor from $N_k(x)$
5: $x'' \leftarrow \text{VND}(x', l_{\max})$
6: **if** solution x'' is better than x **then**
7: $x \leftarrow x''$ // keep new best solution
8: $k \leftarrow 1$ // go to the first neighborhood
9: **else**
10: $k \leftarrow k + 1$ // move to the next neighborhood
11: **end if**
12: **until** $k = k_{\max}$
13: **until** timelimit reached

3.4 Regret Heuristics

The problem of greedy heuristics is, that they often delay difficult decisions until the last iterations. For example for the VRPTW, the customers that are more difficult or expensive to insert into a route are often only inserted after most other customers are incorporated into a partial solution. This leaves less options for inserting those customers. The idea of regret heuristics is to add a look-ahead mechanism into the heuristic to prevent difficult decisions from being left over for the last iterations [37]. The pilot method is another approach that enhances a greedy method by looking ahead for each possible choice [18]. Regret heuristics have been used as a metric of how costly not inserting a specific unrouted customer immediately may become for the VRPTW with great success [38, 36].

Staying at the example of the VRPTW, before deciding which customer to add next to a route, each customer from the set of all unassigned customers U is given a regret value δ_i^k. This value denotes the difference in cost between inserting customer i in the best possible route and the k-th best route. For $k = 2$, the regret-2 heuristic chooses the customer i such that

$$i = \arg\max_{i \in U}(\delta_i^2 - \delta_i^1) \tag{3.19}$$

More generally a regret-q heuristic chooses customer i by selecting

$$i = \arg\max_{i \in U}\left(\sum_{h=2}^{q} \delta_i^h - \delta_i^1\right) \tag{3.20}$$

Ties between multiple customers with the same regret value are broken by inserting the customer with the lowest insertion cost. Customer i is inserted at the position of minimum cost in the best route.

3.5 Methods for Routing Problems

In this Section we will introduce a number of methods used for solving the routing problems presented in Chapter 2. Fischetti et al. [1] presented a number of greedy construction heuristics for routing problems reaching from very simple and fast (Section 3.5.1) to more sophisticated strategies (Sections 3.5.2, 3.5.3). The route elimination algorithm explained in Section 3.5.4 by Nagata and Bräysy [32] is an efficient route minimization heuristic originally developed for the VRPTW.

3.5.1 Sorting Heuristics

The simplest construction heuristics provide an initial solution by sorting all nodes of a tour depending on a sorting criterion. The resulting sequence of nodes is then checked for feasibility, i.e., if each node can be visited without violating its time window. This is generally a very fast heuristic.

- Sort the nodes by the start of their time window in increasing order and check the feasibility of the resulting sequence.
- Sort the nodes by the end of their time window in increasing order and check the feasibility of the resulting sequence.
- Compute the midpoint of the time windows of all nodes, order the nodes accordingly and check if the resulting sequence is feasible.

3.5.2 Nearest-Neighbor Heuristic

This construction heuristic greedily appends the node which increases the travel time or makespan by the smallest amount to the partial tour, whereas the selection is limited only to nodes leading to a feasible tour.

3.5.3 Insertion heuristics

Insertion heuristics incrementally add nodes to a partial tour, starting with an empty tour, until either all nodes are inserted or the insertion of the next node becomes infeasible. All unassigned nodes are inserted at the position in the partial tour which worsen the objective value the least. Two possible criteria for the insertion are:

- Select the node and insertion position by lowest increase in overall tour length while maintaining a feasible tour.
- Choose the node with the least feasible insertion positions and insert it at the insertion position with the lowest increase in the overall tour length.

The adoption of these strategies for the routing part of the DRPSC are discussed in Section 4.3.

3.5.4 Route Elimination Algorithm

Due to the twofold nature of the objective of the VRPTW, most heuristics minimize the number of routes first and only then minimize the route lengths [7, 37, 16, 3, 24, 31, 40]. Since minimizing the number of routes is often the most time consuming part of these two objectives, Nagata and Bräysy [32] suggest an efficient heuristic for minimizing the number of routes for the VRPTW. The basic idea of the so called route elimination algorithm is to start with a solution containing one route per customer and then iteratively eliminating routes from this initial trivial solution and reinserting the customers from these eliminated routes into the remaining routes, reducing the number of routes one at a time. For the difficult task of reinserting customers into the remaining routes, the use of an ejection pool (EP) [24] containing all unassigned customers is suggested. Furthermore, the algorithm allows the infeasible insertion of customers followed by a procedure to regain a feasible solution. The presented algorithm generated solutions

equal to or better than the previously best-known solutions for the well-known large-scale benchmark instances by Gehring and Homberger [15].

Algorithm 3 Function DeleteRoute(σ)

1: remove a randomly selected route from σ
2: initialize EP with customers from the removed route
3: initialize all penalty values $p_i \leftarrow 1$
4: **while** EP $\neq \emptyset$ and time $<$ maxTime **do**
5: remove customer i_{in} from EP with LIFO strategy
6: **if** $N_{\text{in}}^{\text{fe}}(i_{\text{in}}, \sigma) \neq \emptyset$ **then**
7: select random $\sigma' \in N_{\text{ej}}^{\text{fe}}(i_{\text{in}}, \sigma)$
8: $\sigma \leftarrow \sigma'$
9: **else**
10: $\sigma \leftarrow \text{Squeeze}(i_{\text{in}}, \sigma)$
11: **end if**
12: **if** i_{in} is not included in σ **then**
13: $p_{i_{\text{in}}} \leftarrow p_{i_{\text{in}}} + 1$
14: generate the set $N_{\text{ej}}^{\text{fe}}(i_{\text{in}}, \sigma)$ of feasible solutions by ejecting up to $k_{\max}$ customers from σ
15: select $\sigma' \in N_{\text{ej}}^{\text{fe}}(i_{\text{in}}, \sigma)$, minimizing $\sum_{l=1}^{k} = p_{i_{\text{out}}}^{l}$
16: $\sigma \leftarrow \sigma'$
17: add ejected customers $\{i_{\text{out}}^1 \dots i_{\text{out}}^k\}$ to the EP
18: $\sigma \leftarrow$ execute random local search moves on σ while remaining feasible
19: **end if**
20: **end while**
21: **if** EP $\neq \emptyset$ **then**
22: Restore σ to the initial state
23: **end if**
24: **return** σ

The procedure of eliminating one route of a feasible solution σ is shown in algorithm 3. After removing a route from the solution σ, the customers assigned to that route are added to the ejection pool (EP) (lines 1, 2) and the penalty values p_i for all customers are initialized (line 3). As long as the EP is not empty and the time limit has not been reached (line 4), the algorithm tries to reinsert the customers of the EP into the remaining routes. First, a customer from the EP is selected with a last-in first-out strategy (LIFO) and removed from the EP (line 5). If the set $N_{\text{in}}^{\text{fe}}(i_{\text{in}}, \sigma)$ of feasible partial solutions obtained by inserting i_{in} into all insertion positions of σ is not empty, i.e., there is at least one feasible insertion position for i_{in} in σ, then continue with a randomly selected solution σ' from this set as the incumbent solution else try to squeeze customer i_{in} into the current solution σ. The function *Squeeze* (line 10) tries to insert a customer into an existing partial solution, by allowing temporarily infeasible solutions and then trying to restore the feasibility of the solution by applying a local search with 2-opt* [39], intra-

and inter-route relocation and intra- and inter-route exchange [22]. If these methods fail, the penalty value $p_{i_{\text{in}}}$ of customer i_{in} is incremented and feasible candidate solutions $N_{\text{ej}}^{\text{fe}}(i_{\text{in}}, \sigma)$ including i_{in} are generated by removing up to $k_{\max}$ customers from the existing solution σ (lines 13, 14). The solution σ' generated by removing customers i_{out} with minimum total penalty is chosen from $N_{\text{ej}}^{\text{fe}}(i_{\text{in}}, \sigma)$ is the selected as the new incumbent solution (lines 15, 16). The ejected customers $\{i_{\text{out}}^1, \ldots, i_{\text{out}}^k\}$ are added to the EP. In a final perturbation step, the incumbent solution is changed by executing random local search moves (line 18) for a specified number of times, whereas infeasible moves are not allowed in this step.

CHAPTER 4

Solving the Routing Problem

An important factor when approaching the DRPSC is finding a practically efficient approach to the numerous underlying routing problems that have to be solved. This section shows the strategies that are embedded as a subcomponent into the approach for optimizing the districting and are used when the feasibility of a district needs to be checked. As already mentioned, this subproblem is similar to the well-known TSPTW which has been exhaustively studied in the literature. There is, however, one substantial and significant difference: Objects have to be visited several times per period and between every two visits of the same object there has to be a specific separation time. Nevertheless, many fruitful ideas of the literature can be adopted to our problem.

As a single routing problem is solved for each period $j \in P$ and each district $r \in R$ independently, we are given one graph $G_r^j = (V_r^j, A_r^j)$. The node set is defined as $V_r^j = V^j \cap \bigcup_{i \in I_r} W_{i,j}$ and the arc set as $A_r^j = A^j \setminus \{(i_k, i'_{k'}) \mid i_k \notin V_r^j \vee i'_{k'} \notin V_r^j\}$. Similarly, we define the augmented graph containing the tours' start and end nodes 0_0 and 0_1 as $\hat{G}_r^j = (\hat{V}_r^j, \hat{A}_r^j)$ where $\hat{V}_r^j = V_r^j \cup \{0_0, 0_1\}$ and $\hat{A}_r^j = A_r^j \cup \{(0_0, i_k) \mid i_k \in V_r^j\} \cup \{(i_k, 0_1) \mid i_k \in V_r^j\}$.

For computing the duration of a tour τ we first define the arrival and waiting times for each visit of the tour. Moreover, let us define the auxiliary function $\kappa : V_r^j \mapsto I$ which maps the visit $i_k \in V_r^j$, to its corresponding object $i \in I_r$, and the auxiliary function $\gamma : V_r^j \mapsto \mathbb{N}$ which maps visit $i_k \in V_r^j$, to its corresponding index in the set of visits for this particular object. For every visit $i_k \in V_r^j$, a_{i_k} denotes the arrival time at the object, whereas a_{0_0} and a_{0_1} denote the departure and arrival time for the depot nodes 0_0 and 0_1, respectively. Let $t_{\tau_u}^{\text{wait}} = \max(0, T_{\tau_u}^e - \max(a_{\tau_{u-1}} + t_{\tau_{u-1}}^{\text{visit}} + t_{\kappa(\tau_{u-1}),\kappa(\tau_u)}^{\text{travel}}, a_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t_{\kappa(\tau_u)_{\gamma(\tau_u)-1}}^{\text{visit}} + t^{\text{sep}}))$ denote the waiting time before a visit τ_u can be fulfilled. We aim at finding a feasible tour $\tau = (0_0, \tau_1, \ldots, \tau_l, 0_1)$, $\tau_1, \ldots, \tau_l \in V_r^j, l = |V_r^j|$ through all visits starting and ending at the depot such that the total tour duration $T(\tau) = a_{0_1} - a_{0_0}$ does not exceed $t^{\max}$.

In the next sections we will present an exact approach and multiple heuristics for solving this routing problem.

4.1 Mixed Integer Linear Programming Model

In the following we aim at formulating the routing problem for a single time period and district as a MIP model. Subtours are implicitly excluded through the computation of the arrival times as they define the ordering of the visits of the objects. Arrival times have to be checked in order to ensure that every visit complies with its time window. This formulation is used to evaluate the quality of the solutions to small instances of the routing problem of the metaheuristic approaches presented in this thesis.

The following compact mixed integer programming (MIP) model operates on the previously defined and reduced graph G_r^j and is based on Miller-Tucker-Zemlin (MTZ) [28] constraints. We use binary decision variables $y_{i_k,i'_{k'}}\ \forall(i_k, i'_{k'}) \in A_r^j$ which are set to 1 if the arc between the k-th visit of object i and the k'-th visit of object i' is used in the solution, and 0 otherwise. We model arrival times by additional continuous variables $a_{i_k}\ \forall i_k \in V_r^j$ and by these variables ensure compliance with the time windows and the elimination of subtours. For each district $r \in R$ and each period $j \in P$ we solve the following model:

$$\min \quad \sum_{i_k \in V_r^j} (t_{i_k}^{\text{wait}} + t_{i_k}^{\text{visit}}) + \sum_{(i_k,i'_{k'}) \in \hat{A}_r^j} (y_{i_k,i'_{k'}} \cdot t^{\text{travel}}_{\kappa(i_k),\kappa(i'_{k'})}) \tag{4.1}$$

$$\text{s.t.} \sum_{(i_k,i'_{k'}) \in \hat{A}_r^j} y_{i_k,i'_{k'}} = \sum_{(i'_{k'},i_k) \in \hat{A}_r^j} y_{i'_{k'},i_k} \qquad \forall i_k \in V_r^j \tag{4.2}$$

$$\sum_{(0_0,i_k) \in \hat{A}_r^j} y_{0_0,i_k} = 1 \tag{4.3}$$

$$\sum_{(i_k,0_1) \in \hat{A}_r^j} y_{i_k,0_1} = 1 \tag{4.4}$$

$$a_{i_k} - a_{i'_{k'}} + t^{\max} \cdot (1 - y_{i'_{k'},i_k}) \geq t^{\text{travel}}_{\kappa(i'_{k'}),\kappa(i_k)} + t^{\text{visit}}_{i'_{k'}} \qquad \forall i_k \in \hat{V}_r^j, (i_k, i'_{k'}) \in \hat{A}_r^j \tag{4.5}$$

$$a_{i_k} + t^{\text{travel}}_{0,\kappa(i_k)} \cdot (1 - y_{0_0,i_k}) \geq t^{\text{travel}}_{0,\kappa(i_k)} \qquad \forall (0_0, i_k) \in \hat{A}_r^j \tag{4.6}$$

$$t_{i_k}^{\text{wait}} + t^{\max} \cdot (1 - y_{i_k,i'_{k'}}) \geq a_{i'_{k'}} - a_{i_k} - t^{\text{travel}}_{\kappa(i_k),\kappa(i'_{k'})} - t_{i_k}^{\text{visit}} \qquad \forall i_k \in \hat{V}_r^j, (i_k, i'_{k'}) \in \hat{A}_r^j \tag{4.7}$$

$$a_{i_{k-1}} \leq a_{i_k} - t^{\text{sep}} \qquad \forall i_k, i_{k-1} \in V_r^j \tag{4.8}$$

$$\sum_{(i_k,i'_{k'}) \in \hat{A}_r^j} y_{i_k,i'_{k'}} = 1 \qquad \forall i_k \in V_r^j \tag{4.9}$$

$$T_{i_k}^e \leq a_{i_k} \leq T_{i_k}^l - t_{i_k}^{\text{visit}} \qquad \forall i_k \in V_r^j \tag{4.10}$$

$$y_{i_k,i'_{k'}} \in \{0,1\} \qquad \forall (i_k, i'_{k'}) \in \hat{A}_r^j \tag{4.11}$$

The objective function (4.1) minimizes the total makespan within which all object visits take place by summing up all visit times, travel times, and waiting times. Equalities (4.2) ensure that the number of ingoing arcs is equal to the number of outgoing arcs for each node $i_k \in V_r^j$. Equalities (4.3) and (4.4) ensure that there must be exactly one ingoing and outgoing arc for the depot in each period $j \in P$. Inequalities (4.5) are used to recursively compute the arrival times for every visit. If an edge $(i_k, i'_{k'})$ is not used, then the constraint is deactivated. These inequalities can also be lifted by replacing $t^{\max}$ with the term $(T^l_{i'_{k'}} - t^{\text{visit}}_{i'_{k'}}) - T^l_{i_k} - t^{\text{travel}}_{\kappa(i_k),\kappa(i'_{k'})} - t^{\text{visit}}_{i_k}$. Inequalities (4.8) model the minimum time required between two different visits of the same object, i.e., ensure the separation time t^{sep}. Inequalities (4.9) state that there must exist an ingoing and an outgoing arc for the k-th visit of object i, if this particular visit is requested in the considered period $j \in P$. It is ensured that every time window of every visit $i_k \in V_r^j$ is fulfilled in (4.10). In (4.11) the domain definitions for the binary edge-decision variables $y_{i_k,i'_{k'}}$ are given.

In the context of the districting problem we use this model only for checking feasibility which can usually be done faster than solving the optimization problem to optimality. To this end we replace the objective function by $\min\{0\}$ and add the following constraints for limiting the makespan to $t^{\max}$:

$$\sum_{i_k \in V_r^j} (t^{\text{wait}}_{i_k} + t^{\text{visit}}_{i_k}) + \sum_{(i_k,i'_{k'}) \in \hat{A}_r^j} (y_{i_k,i'_{k'}} \cdot t^{\text{travel}}_{\kappa(i_k),\kappa(i'_{k'})}) \leq t^{\max} \tag{4.12}$$

4.2 Feasibility of a Tour

When generating solution candidates for the routing problem, the task of checking whether or not a sequence of visits is feasible arises. This chapter shows how the non-trivial task of checking, if there exists a schedule for tour τ arriving at each requested visit τ_u within its given time window T_{τ_u}, respecting the separation time t^{sep} between multiple visits of the same object i and returning to the depot 0 within the maximum tour duration $t^{\max}$, is accomplished.

Given a sequence of visits τ, we first determine if a tour can be scheduled such that the time window constraints of all visits are satisfied. For this purpose, we compute the earliest possible arrival time a_{i_k} for each visit and minimize waiting times.

Since the tour τ starts at the depot at the earliest possible time, the departure at the depot a_{0_0} is set to 0. For each subsequent visit τ_u, the arrival time a_{τ_u} is the maximum of $T^e_{\tau_u}$ and the arrival time at the preceding visit $a_{\tau_{u-1}}$ including visit time $t^{\text{visit}}_{\tau_{u-1}}$ and travel time $t^{\text{travel}}_{\kappa(\tau_{u-1}),\kappa(\tau_u)}$ from the preceding visit's object $\kappa(\tau_{u-1})$ to the current visit's object $\kappa(\tau_u)$. The depot has no requested visit times, therefore we define $t^{\text{visit}}_{0_0} = t^{\text{visit}}_{0_1} = 0$. Furthermore, for each object i the separation time t^{sep} between visit i_k and i_{k-1} for all $k > 1$ has to be respected.

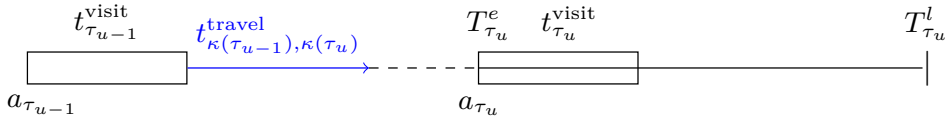

Figure 4.1: Early arrival at object $\kappa(\tau_u)$ resulting in waiting time before the start $T^e_{\tau_u}$ of the time window of visit τ_u

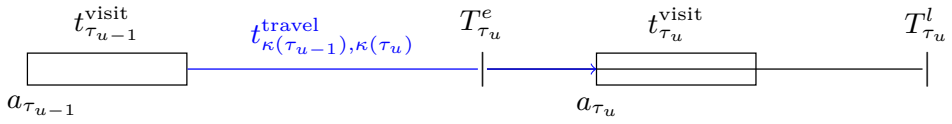

Figure 4.2: Arrival at $\kappa(\tau_u)$ after the start $T^e_{\tau_u}$ of the time window of visit τ_u due to long travel time

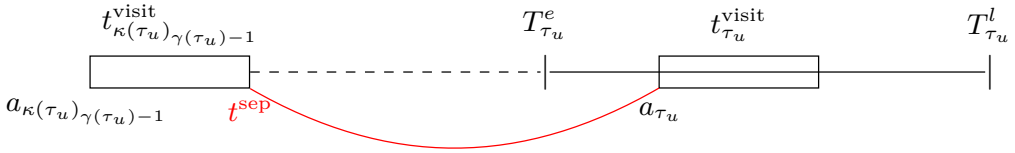

Figure 4.3: Separation time t^{sep} between two visits $\kappa(\tau_u)_{\gamma(\tau_u)-1}$ and τ_u of the same object has to be respected before visit τ_u can start

The three scenarios for computing the earliest arrival time at an object $\kappa(\tau_u)$ for visit τ_u are depicted in Figures 4.1 through 4.3. Satisfying the visit τ_{u-1} with duration $t^{\text{visit}}_{t_{u-1}}$ and travelling from object $\kappa(\tau_{u-1})$ to $\kappa(\tau_u)$ in time $t^{\text{travel}}_{\kappa(\tau_{u-1}),\kappa(\tau_u)}$ while still arriving before the start $T^e_{\tau_u}$ of the time window of visit τ_u at object $\kappa(\tau_u)$ results in a waiting time as shown in Figure 4.1. Arriving after the start $T^e_{\tau_u}$ of the time window of τ_u is presented in Figure 4.2. Finally, arriving early but having to respect the necessary separation time t^{sep} between two visits of the same object can be seen in Figure 4.3.

The earliest possible arrival time a_{τ_u} is computed as follows:

$$
\begin{aligned}
a_{0_0} &= 0 \\
a_{\tau_u} &= \begin{cases} \max\{T^e_{\tau_u}, a_{\tau_{u-1}} + t^{\text{visit}}_{\tau_{u-1}} + t^{\text{travel}}_{\kappa(\tau_{u-1}),\kappa(\tau_u)}\} \\ \qquad\qquad \text{for } u > 1, \gamma(\tau_u) = 1 \\ \max\{T^e_{\tau_u}, a_{\tau_{u-1}} + t^{\text{visit}}_{\tau_{u-1}} + t^{\text{travel}}_{\kappa(\tau_{u-1}),\kappa(\tau_u)}, a_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t^{\text{visit}}_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t^{\text{sep}}\} \\ \qquad\qquad \text{for } u > 1, \gamma(\tau_u) > 1 \end{cases} \\
a_{0_1} &= a_{\tau_l} + t^{\text{visit}}_{\tau_l} + t^{\text{travel}}_{\kappa(\tau_l),0}
\end{aligned}
$$

If for any arrival time a_{i_k} with $i_k \in V^j_r$ the following condition is violated, the sequence

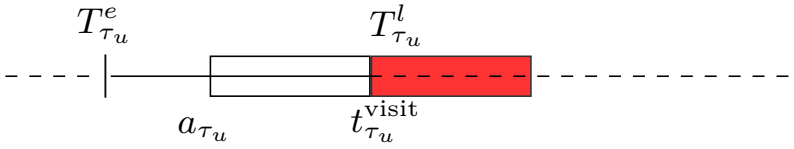

Figure 4.4: Late arrival, fulfilling visit i_k is impossible before the end $T^{\text{l}}_{i_k}$ of the time window

of visits is infeasible:

$$a_{i_k} + t^{\text{visit}}_{i_k} \leq T^{\text{l}}_{i_k} \tag{4.13}$$

The violation of this constraint is shown in Figure 4.4. Visit i_k can not be finished within its respective time window.

The resulting tour duration $T(\tau) = a_{0_1} - a_{0_0}$ might still be far from minimum at this point, since departing at the depot at time 0 might raise waiting times at some point of the tour. If one arrives at an object before the start $T^e_{\tau_u}$ of the time window of the requested visit, one would have to wait for the designated time window to start. To avoid this, Savelsbergh [45] proposed the so called *forward time slack* (FTS) for computing the maximum possible delay of the departure at the depot while retaining a feasible tour for the TSPTW. Since the time constraints of the DRPSC additionally contain the separation time, the FTS has to take multiple visits of the same object into account. The FTS $F(\tau_u, \tau_{u'})$ for the partial tour $\tau' = \tau_u, \ldots, \tau_{u'}$ adapted to our problem is

$$F'(\tau_u, \tau_{u'}) = \begin{cases} T^l_{\tau_u} - t^{\text{visit}}_{\tau_u} - a_{\tau_u} & \text{for } u = u' \\ F'(\tau_u, \tau_{u'-1}) - T^l_{\tau_{u'-1}} + T^l_{\tau_{u'}} - t^{\text{visit}}_{\tau_{u'}} - t^{\text{travel}}_{\kappa(\tau_{u'-1}),\kappa(\tau_{u'})} & \text{for } u' > 1, \gamma(\tau_{u'}) = 1 \\ \min\{F'(\tau_u, \tau_{u'-1}) - T^l_{\tau_{u'-1}} + T^l_{\tau_{u'}} - t^{\text{visit}}_{\tau_{u'}} - t^{\text{travel}}_{\kappa(\tau_{u'-1}),\kappa(\tau_{u'})}, & \\ \quad F'(\tau_u, \tau_{\kappa(\tau_{u'})_{\gamma(\tau_{u'})-1}}) - T^l_{\tau_{\kappa(\tau_{u'})_{\gamma(\tau_{u'})-1}}} + T^l_{u'} - t^{\text{visit}}_{\tau_{u'}} - t^{\text{sep}}\} & \\ & \text{for } u' > 1, \gamma(\tau_{u'}) > 1 \end{cases}$$

$$F(\tau_u, \tau_{u'}) = \min_{v=u,\ldots,u'} \{F'(\tau_u, \tau_v)\} \tag{4.14}$$

By simply delaying the departure at the depot by the FTS, the tour would always be scheduled for the latest possible arrival time at the depot. To avoid this, one also has to take the waiting times within the tour into account, when computing a schedule for a tour of minimum duration. Considering the tour duration of tour τ must not exceed $t^{\max}$, formally, if

$$T(\tau) - \min(F(0_0, 0_1), \sum_{u=1}^{l} t^{\text{wait}}_{\tau_u}) < t^{\max} \tag{4.15}$$

holds, the sequence τ of visits is a feasible tour, otherwise it is infeasible.

4.3 Routing Construction Heuristics

To provide initial solutions to the routing problem, a routing construction heuristic (RCH) based on an insertion heuristic by starting from a partial tour $\tau' = (0_0, 0_1)$ containing only the start and end nodes and iteratively adding all visits $i_k \in V_r^j$ to τ' was developed. A 2-step approach is used by ordering visits according to some criteria and then inserting them at the first feasible or best possible insertion position, respectively. For the insertion order we compute the flexibility value $flex$ of each visit $i_k \in V_r^j$, since visits with a lower flexibility value may be more difficult to insert as they need to be scheduled at a very specific time. Ties are broken randomly.

$$\mathit{flex}(i_k) = T^l_{i_k} - T^e_{i_k} - t^{\text{visit}}_{i_k} \tag{4.16}$$

$$\mathit{flex}(i_k^{(1)}) \leq \mathit{flex}(i_k^{(2)}) \leq \cdots \leq \mathit{flex}(i_k^{(|V_r^j|)}) \tag{4.17}$$

In a second phase we insert the ordered visits into the partial tour τ'. We start at the beginning of the tour, i.e., we try to insert visit $i_k^{(1)}$ after the starting node 0_0, and move backwards in the tour until node 0_1 is reached. For the first feasible insertion strategy, the heuristic inserts the current visit at the first feasible insertion position found, whereas for the best feasible insertion strategy, insertion costs are computed for each possible insertion position and the visit is inserted at the position of minimum cost. We define these costs as:

$$d_{i_k,u'} = \begin{cases} a_{\tau_{u'}} + t^{\text{visit}}_{\tau_{u'}} + t^{\text{travel}}_{\kappa(\tau_{u'}),\kappa(\tau_u)} - a_{\tau_u} & \text{if (4.19) and (4.20) hold} \\ \infty & \text{otherwise} \end{cases} \tag{4.18}$$

These insertion costs $d_{i_k,u'}$ determine the amount of time by which the visit τ_u has to be moved backwards in order to insert the new visit $\tau_{u'}$. Note, that $d_{i_k,u'}$ may also be negative, if the space for insertion of visit $\tau_{u'}$ is bigger than necessary. However, this is desirable as we use those insert positions more likely which have bigger gaps and smaller gaps are kept for later insertion operations. In Section 6.2 we compare both the first feasible and the best possible variant to each other in terms of solution quality and runtime.

We further maintain global variables for the *forward time slack* $F(\tau')$ and all arrival times a_{τ_u} of each partial tour τ' computed during the execution of the insertion heuristic for evaluating the feasibility of inserting object i_k at position u in tour τ. For an insertion

to be feasible, the latest allowed arrival time at visit $\tau_{u'}$ must be greater or equal to the earliest possible arrival at that visit considering the previous visit's earliest arrival, its visit time and the travel time between τ_{u-1} and $\tau_{u'}$. Furthermore, the earliest departure at $\tau_{u'}$ including the travel time between $\tau_{u'}$ and τ_u must be smaller or equal to the earliest arrival at τ_u delayed by the *forward time slack* of the partial tour from τ_u to the depot. Using the definition of the forward time slack in equality (4.14) and if inequality (4.13) holds, then the insertion is feasible if, in addition, also the following two inequalities hold:

$$T^l_{\tau_{u'}} - t^{\text{visit}}_{\tau_{u'}} \geq \max\{a_{\tau_{u-1}} + t^{\text{visit}}_{\tau_{u-1}} + t^{\text{travel}}_{\kappa(\tau_{u-1}),\kappa(\tau_{u'})}, a_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t^{\text{visit}}_{\kappa(\tau_u)_{\gamma(\tau_u)-1}} + t^{\text{sep}}\} \quad (4.19)$$

$$a_{\tau_{u'}} + t^{\text{visit}}_{\tau_{u'}} + t^{\text{travel}}_{\kappa(\tau_{u'}),\kappa(\tau_u)} \leq a_{\tau_u} + F(\tau_u, 0_1) \quad (4.20)$$

4.4 Variable Neighborhood Descent

If the solution found by the RCH is infeasible we additionally employ a VND to reduce the number of infeasibilities and possibly come to a feasible solution. First, we insert each infeasible visit i_k into the tour on the position u' where the costs $d_{i_k,u'}$ are minimum. We use a lexicographical penalty function to penalize infeasible tours where the first criterion is the number of time window violations and the second criterion is the duration of the route as proposed by López-Ibáñez et al. [25]. We use three common neighborhood structures from the literature and search them in a best improvement fashion in the following order:

Swap: This neighborhood considers all exchanges between two distinct visits (Figure 4.5).

2-opt: This is the classical 2-opt neighborhood for the traveling salesman problem where all edge exchanges are checked for improvement (Figure 4.6).

Or-opt: This neighborhood considers all solutions in which sequences of up to three consecutive visits are moved to another place in the same route (Figure 4.7).

If at some point during the algorithm the value of the penalty function is zero we terminate with a feasible solution.

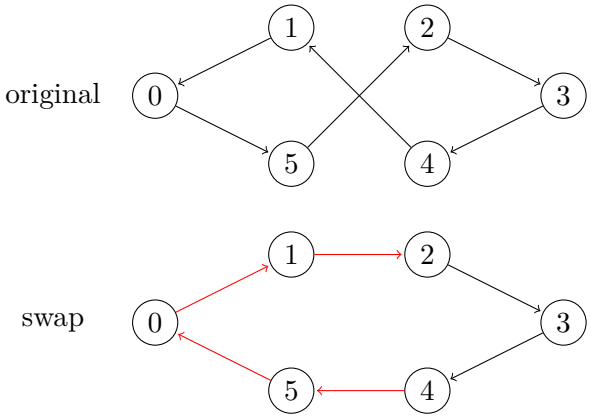

Figure 4.5: swap operation

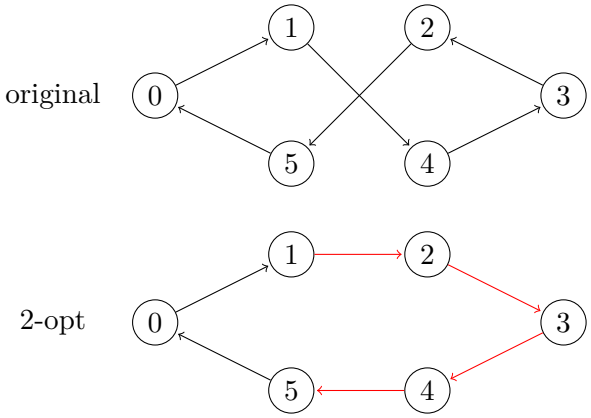

Figure 4.6: 2-opt operation

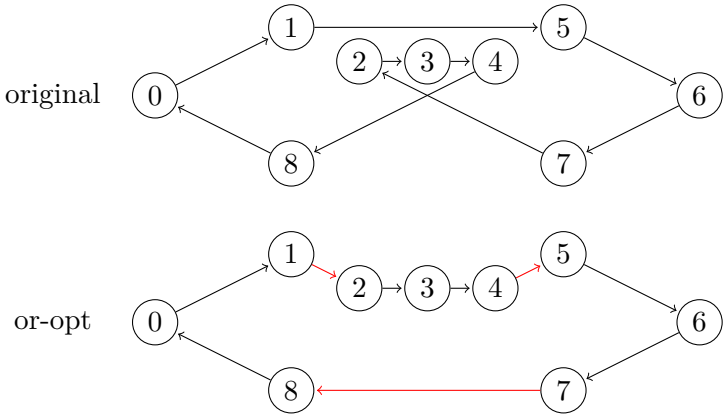

Figure 4.7: or-opt operation

CHAPTER 5

Solving the Districting Problem

In the previous section we introduced a fast heuristic for efficiently testing feasibility of a given set of objects by building a single tour for each period through all requested visits of these objects. In the districting part of the DRPSC we face the problem of intelligently assigning objects to districts such that the number of districts is minimized. For checking the feasibility of this assignment the previously introduced RCH and VND are used. The MIP model representing the districting problem is given in Section 4.1. We propose a districting construction heuristic for generating initial results and an iterative destroy & recreate (IDR) algorithm for solving the districting problem of the DRPSC by iteratively reducing the number of districts.

5.1 Mixed Integer Linear Programming Model

We model the problem in the following by a compact mixed integer programming (MIP) formulation. This formulation shall serve as a starting point for a decomposition approach in future work. As for the routing model we also rely on Miller-Tucker-Zemlin constraints here to exclude subtours from consideration. We introduce the following decision variables:

$$f_r = \begin{cases} 1 & \text{district } r \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

$$x_{i,r} = \begin{cases} 1 & \text{object } i \text{ is assigned to district } r \\ 0 & \text{otherwise} \end{cases}$$

$$y_{j,r,i_k,i'_{k'}} = \begin{cases} 1 & \text{the arc between the } k\text{-th visit of object } i \text{ and the } k'\text{-th visit} \\ & \text{of object } i' \text{ used in route } r \text{ of period } j \\ 0 & \text{otherwise} \end{cases}$$

Additionally, we introduce the following variables to model the arrival and departure times which are used later on to ensure compliance with given time windows:

- a_{j,r,i_k} : Denotes the arrival time at object $i \in I$ in period $j \in P$ for visit $i_k \in W_{i,j}$ and district $r \in R$

Using the variables defined above, the compact MIP formulation is stated as follows:

$$\min \sum_{r \in R} f_r \tag{5.1}$$

$$\text{s.t. } x_{i,r} \leq f_r \qquad \forall i \in I, r \in R \tag{5.2}$$

$$\sum_{r \in R} x_{i,r} = 1 \qquad \forall i \in I \tag{5.3}$$

$$\sum_{(i_k, i'_{k'}) \in \hat{A}^j} y_{j,r,i_k,i'_{k'}} = \sum_{(i'_{k'}, i_k) \in \hat{A}^j} y_{j,r,i'_{k'},i_k} \qquad \forall i_k \in V^j, r \in R, j \in P \tag{5.4}$$

$$\sum_{(0_0, i_k) \in \hat{A}^j} y_{j,r,0_0,i_k} = 1 \qquad \forall j \in P, r \in R \tag{5.5}$$

$$\sum_{(i_k, 0_1) \in \hat{A}^j} y_{j,r,i_k,0_1} = 1 \qquad \forall j \in P, r \in R \tag{5.6}$$

$$y_{j,r,i_k,i'_{k'}} \leq x_{\kappa(i_k),r} \qquad \forall (i_k, i'_{k'}) \in \hat{A}^j, j \in P, r \in R, i_k \neq 0_0 \tag{5.7}$$

$$y_{j,r,i_k,i'_{k'}} \leq x_{\kappa(i'_{k'}),r} \qquad \forall (i_k, i'_{k'}) \in \hat{A}^j, j \in P, r \in R, i'_{k'} \neq 0_1 \tag{5.8}$$

$$a_{j,r,i_k} - a_{j,r,i'_{k'}} + t^{\max} \cdot (1 - y_{j,r,i'_{k'},i_k}) \geq t^{\text{travel}}_{\kappa(i'_{k'}),\kappa(i_k)} + t^{\text{visit}}_{i'_{k'}} \qquad \forall (i'_{k'}, i_k) \in \hat{A}^j, j \in P, r \in R \tag{5.9}$$

$$a_{j,r,i_k} + t^{\max} \cdot (1 - y_{j,r,0_0,i_k}) \geq t^{\text{travel}}_{0,\kappa(i_k)} \qquad \forall (0_0, i_k) \in \hat{A}^j, j \in P, r \in R \tag{5.10}$$

$$a_{j,r,i_{k-1}} \leq a_{j,r,i_k} - t^{\text{sep}} \qquad \forall i_k, i_{k-1} \in V^j, j \in P, r \in R \tag{5.11}$$

$$a_{j,r,0_1} - a_{j,r,0_0} \leq t^{\max} \qquad \forall j \in P, r \in R \tag{5.12}$$

$$\sum_{r \in R} \sum_{(i_k, i'_{k'}) \in \hat{A}^j} y_{j,r,i_k,i'_{k'}} = 1 \qquad \forall i_k \in V^j, j \in P \tag{5.13}$$

$$T^e_{i_k} \leq a_{j,r,i_k} \leq T^l_{i_k} - t^{\text{visit}}_{i_k} \qquad \forall i_k \in V^j, j \in P, r \in R \tag{5.14}$$

$$f_r \in \{0,1\} \qquad \forall r \in R \tag{5.15}$$

$$x_{i,r} \in \{0,1\} \qquad \forall i \in I, r \in R \tag{5.16}$$

$$y_{j,r,i_k,i'_{k'}} \in \{0,1\} \qquad \forall (i_k, i'_{k'}) \in \hat{A}^j, j \in P, r \in R \tag{5.17}$$

Objective function (5.1) minimizes the number of needed districts. By inequalities (5.2) it is ensured that an object can only be assigned to a district if the corresponding district is used. Equalities (5.3) state that every object must be assigned to exactly one district, and equalities (5.4) ensure that the number of ingoing arcs is equal to the number outgoing arcs for every object, in every district and each period. The departure of the the depot

Algorithm 4 Districting Construction Heuristic

```
1:  init: R ← {1}, U ← sort(I)
2:  for all i ∈ U do
3:     inserted ← false
4:     for all r ∈ R do
5:        if insert(i, r) then
6:           inserted ← true
7:           break
8:        end if
9:     end for
10:    if notinserted then
11:       r' ← create |R| + 1-th new empty district
12:       R ← R ∪ {r'}
13:       insert(i, r')
14:    end if
15: end for
```

has to have exactly one outgoing arc (5.5) and the arrival at the depot has to have exactly one ingoing arc (5.6). We connect assignment variables with arc-selection variables by inequalities (5.7) and (5.8). Inequalities (5.9)-(5.11) are used to compute arrival times for every object visit and each period where this visit is requested. Inequalities (5.9) compute the arrival time at station i if an arc $(i'_{k'}, i_k) \in A^j$ between the k'-th visit of i' and the k-th visit of i exists. Equalities (5.10) set the start time of the departure at the depot for each period. Inequalities (5.11) model the minimum waiting between two different visits of the same object, i.e., the so called *separation time*. Inequalities (5.12) ensure that each tour for a given period and district does not exceed the maximum time budget $t^{\max}$, i.e., the makespan has to be lower or equals the maximum time budget for a single tour. Equalities (5.13) make sure that an outgoing arc for the k-th visit of object i exists if it is requested, i.e., $i_k \in W_{i,j}$. Inequalities (5.14) ensure that all given time windows are satisfied. Domain definitions for the variables are stated in (5.15)-(5.17).

5.2 Districting Construction Heuristic

Starting with one district, objects are iteratively added to the existing districts R. Whenever adding an object i to any of the available districts in R would make the assignment infeasible, i is added to a newly created district r'. The overall DCH is shown in Algorithm 4 and explained below.

First, the set of districts R is initialized with the first empty district 1 and the set of objects I is sorted by extending the flexibility values as defined in equation (4.16) from visits to objects. All objects are sorted by the sum of their flexibility values $\sum_{j \in P} \sum_{i_k \in W_{i,j}} \mathit{flex}(i_k)$ in ascending order. As in the RCH, the resulting set U is denoted as the set of unscheduled visits. The DCH terminates when all $i \in U$ have been

scheduled (line 2) and as a consequence, all requested visits have been inserted successfully and we obtain a feasible solution to the DRPSC. The insertion of object i into district r (lines 5 and 13) is accomplished by checking for each scheduled visit $i_k \in W_{i,j}$ if i_k can be feasibly inserted into the particular district r (for the definition of feasibility of a tour see also Section 4.2). In line 5 the DCH inserts i either into the first feasible or into the best possible insert position, as described in Section 4.3. The *insert* function returns *false*, if no feasible insertion position is found for at least one $i_k \in W_{i,j}$, $\forall j \in P$. It returns *true*, if a feasible insertion position is found for each visit $i_k \in W_{i,j}$, $\forall j \in P$. If the loop over all districts (line 4) terminates without finding any feasible insertion position the variable *inserted* stays *false* and a new empty district is created in line 11. The proposed constructive algorithm will terminate with a feasible solution after $|U|$ iterations.

5.3 Regret Heuristics

Instead of simply trying to assign objects $i \in I$ to districts in the order of their creation, as seen in Algorithm 4, additionally a regret-q heuristic proposed by Pillac et al. [36] for the VRPTW, is developed to determine the regret value for not inserting object i at a district r. Let $d_{i_k,u,j,r}$ be the cost of inserting visit i_k into the tour of period j of district r at position u and

$$\delta_i^1 = \min_{r \in R} \left\{ \max_{j \in P} \left\{ \sum_{i_k \in W_{i,j}} \min_{u=1,\dots,l} \{d_{i_k,u,j,r}\} \right\} \right\} \qquad \forall i \in I \tag{5.18}$$

be the minimum cost of all districts' $r \in R$ maximum increase of all periods' $j \in P$ tour duration for inserting all visits $i_k \in W_{i,j}$. These values $\delta_i^{(1,\dots,p)}$ $\forall i \in I$ define a ranking based on the insertion costs in ascending order for inserting object i into district $r \in R$ where δ_i^p references the p-th best district for insertion of object $i \in I$. Thus, in the sum of equalities (5.18) the costs for the best insertions of the visits of object i are considered. Then, we take the maximum of the aggregated insertion costs over all periods $j \in P$. The insertion costs for every district are then ranked where lower insertion costs are better. The best district for assignment of object i is the district maximizing $\sum_{p=2}^{q}(\delta_i^p - \delta_i^1)$ which is also referred to as the maximum regret value where δ_i^p is the p-th best district. Object i is then assigned to the district resulting in δ_i^1. This way, we always use the object $i \in I$ for insertion which has the highest regret value of all objects.

5.4 Iterative Destroy & Recreate

Nagata et al. [32] proposed a route elimination algorithm for reducing the number of vehicles needed in the VRPTW. We apply the basic idea to the districting problem. The algorithm starts with the initial assignment where every customer is reached by a separate route. Then, one district $r \in R$ is chosen for elimination at a time, maintaining all now unassigned objects in an ejection pool (EP). Then, the algorithm tries to assign all objects of the EP to the remaining districts $R \setminus \{r\}$. If the EP becomes emtpy, the

Algorithm 5 District elimination algorithm

1: **init:** $EP \leftarrow \emptyset,\ c_i \leftarrow 0\ \forall i \in I$
2: choose a district $r^{\text{del}} \in R$ for deletion
3: $R \leftarrow R \setminus \{r^{\text{del}}\}$
4: $EP \leftarrow EP \cup \{i \mid i \in I_{r^{\text{del}}}\}$
5: **while** $EP \neq \emptyset \wedge$ termination criterion not met **do**
6: $\quad i^{\text{ins}} \leftarrow \arg\max_{i \in EP} \{c_i\}$
7: $\quad R_f \leftarrow$ feasible districts for assignment of i^{ins}
8: $\quad c_{i^{\text{ins}}} \leftarrow c_{i^{\text{ins}}} + |R| - |R_f|$
9: $\quad$ **if** $R_f \neq \emptyset$ **then**
10: $\quad\quad$ assign object i^{ins} to a randomly chosen feasible district $r \in R_f$
11: $\quad$ **else**
12: $\quad\quad$ select random district $r^{\text{ins}} \in R$
13: $\quad\quad$ assign object i^{ins} to district r^{ins}
14: $\quad\quad$ call VND for district r^{ins} (see Section 4.4)
15: $\quad\quad$ **while** $\exists$ an infeasible tour for any period of district r^{ins} **do**
16: $\quad\quad\quad i^{\text{del}} \leftarrow \arg\min_{i \in I_{r^{\text{ins}}}} \{c_i\}$
17: $\quad\quad\quad I_{r^{\text{ins}}} \leftarrow I_{r^{\text{ins}}} \setminus \{i^{\text{del}}\}$
18: $\quad\quad\quad EP \leftarrow EP \cup \{i^{\text{del}}\}$
19: $\quad\quad\quad$ call VND for district r^{ins} (see Section 4.4)
20: $\quad\quad$ **end while**
21: $\quad$ **end if**
22: **end while**

number of districts was successfully reduced and another district is chosen for elimination. We adapt this idea to the DRPSC and use the result of the DCH described in Section 5.2 as initial solution.

Let the assignment of an object $i \in I$ to a district $r \in R$ be feasible if and only if a feasible tour can be scheduled for all assigned visits of all objects for each period. Let c_i be a penalty value of object $i \in I$ denoting failed attempts of inserting object i into a district. Each time a visit cannot be inserted, this penalty value is increased by one, revealing objects which are difficult to assign to one of the available districts.

If the EP becomes empty, a feasible assignment of objects to districts is found. Subsequently, another iteration is started, destroying a district and reassigning its objects to the remaining ones. The district elimination algorithm is shown in Algorithm 5.

First, the EP is initialized to the empty set and the penalty values of all objects are set to 0. Starting with the solution provided by DCH a district is chosen for elimination in line 2. One of the following strategies is applied uniformly at random for selecting a district for elimination:

Minimum number of scheduled visits: This implies that only a minimum number

of visits has to be reinserted to regain a feasible solution.

Shortest tour duration: Selecting a district where the maximum tour duration over all periods is minimum can be promising because this district might lead to a district with visits of shorter durations resulting in easier insert operations.

Maximum waiting times: Selecting a district with a loose schedule may indicate less or shorter visits, making them easier to reinsert.

After deleting a district all objects of this district are moved to the EP (line 4). As long as the EP contains objects, we try to assign each object to one of the remaining districts. An object with maximum penalty value is chosen for the next assignment (line 6). For the chosen object i^{ins} the feasible districts for assignment are computed. If there is at least one feasible district for an assignment of object i^{ins} (line 9) we assign the object to such a district uniformly at random (line 10). If it is not possible to feasibly assign the object i^{ins} to any of the remaining districts we randomly choose a district for assignment (line 11). Then, we apply the VND described in Section 4.4 trying to make the district feasible. If this is not possible and the assignment is still infeasible we iteratively try to remove objects with lowest penalty values from this district r^{ins} in the following loop (line 15), remove them from district r^{ins} (line 17), and finally add them to the *EP* (18). Then again, we call the VND from Section 4.4 trying to make the resulting tour from the actual assignment feasible. After an iteration of the outer loop the object with highest penalty value of the *EP* has been inserted and other objects previously assigned to this district may have been added to the *EP*. The idea behind this approach is to insert difficult objects first and temporarily remove easy to insert objects from the solution to reinsert them later. When the EP is empty, a new best assignment with one district less is found. This algorithm iterates until a termination criterion, e.g., a time limit is met.

CHAPTER 6

Results & Benchmarks

To evaluate our proposed algorithms, computational tests are performed on a set of benchmark instances. As the DRPSC is a new problem we created new instances[1] based on the characteristics of real-world scenarios provided by an industry partner.

The algorithm is implemented in C++ using Gurobi 6.5 for solving the MIP. For each combination of configuration and instance we performed 20 independent runs for the IDR while for the routing part only one run was performed since these algorithms are deterministic. All runs were executed on a single core of an Intel Xeon processor with 2.54 GHz. The IDR algorithm is terminated after a maximum of 900 CPU seconds. The MIP model for the routing part was executed by Gurobi and was aborted after 3600 CPU seconds. A Wilcoxon signed-rank test with an error level of 5% was done for any pair of non-deterministic approaches.

In the first set of experiments the routing part of the DRPSC is examined more closely to evaluate RCH in comparison to the MIP model. Then, several configurations of our proposed algorithms for the whole problem are investigated.

6.1 Test Instances

The benchmark instances were generated using distance matrices from TSPlib instances. The depot was selected by taking the node for which the total distance to all other nodes is a minimum. Visits and the time windows were added using a custom parameterized algorithm as follows:

- v: between 1 and v visits are assigned to each period and node of the original TSP instance uniformly at random.

[1] https://www.ac.tuwien.ac.at/research/problem-instances/#Districting_and_Routing_Problem_for_Security_Control

- α: small time windows of length within the set $\{5, \ldots, 30\}$ in minutes, chosen uniformly at random, are assigned with probability α to each visit.
- β: medium time windows of length of 2,3,4 or 5 hours, chosen uniformly at random, are assigned with probability β to each visit.

Large time windows are unrestricted and assigned to visits with probability $1 - \alpha - \beta$. Small and medium time windows of visits of the same object do not overlap and the visit time is chosen uniformly at random from 3 to 20 minutes in steps of 1 minute. A feasibility check is applied to all generated visits of a single object, i.e., there must be a feasible route satisfying all visits of a single object, to prevent the generation of infeasible instances. It is easy to see, that there can be no feasible solution to the districting problem, if a district containing only a single object has no feasible route. For all benchmark instances t^{sep} is set to 60 minutes and $t^{\max}$ is set to 10 hours for the districting results. A planning horizon spans 7 days.

6.2 Results for the Routing problem

First, the methods for the routing part are evaluated on a separate set of benchmark instances. Very small instances for which a solution containing only one district could be generated were tested. The routing algorithms used the requested visits for day 1 of the planning period as input for finding a feasible tour. In Table 6.1 the MIP model is compared to RCH, and RCH with the subsequent VND, denoted by RCH-VND. For this comparison the objective is changed to minimize the makespan of a specific tour instead of a lexicographical objective function which also considers the penalty value of a tour. Therefore, the maximum tour duration constraint is relaxed and the resulting makespan is given in minutes in the column *obj*. In the first four columns the instance parameters are specified. Sequentially, the instance name, the number of objects $|I|$, the maximum number of nodes of all objects $|V| = \sum_{i \in I} |S_i|$, the percentage of small (α), and medium time windows (β) and the maximum number of visits per objects v is given. For the RCH and RCH-VND we give the objective value (makespan in minutes) and the time needed for solving the instance. Then, the upper bound (UB), the lower bound (LB), the final optimality gap, and the time spent by Gurobi for solving the MIP model is shown. In the two remaining columns we present the relative gap between the MIP and RCH-VND $\Delta_{\text{MIP}} = (obj_{\text{RCH-VND}} - LB)/obj_{\text{RCH-VND}}$ as well as the relative gap between RCH and RCH-VND $\Delta_{\text{RCH}} = (obj_{\text{RCH}} - obj_{\text{RCH-VND}})/obj_{\text{RCH-VND}}$. We performed only one run for each instance because the methods applied to routing part are all deterministic.

In Table 6.1 we see that the MIP model was able to solve easier instances, i.e., instances with very few visits, no small time windows and only up to 50% medium time windows, to optimality, but soon had very high running times. Only very few instances with a number of visits of up to 23 were solved to optimality and the gap could only be reduced below 5% for 2 larger instances. The runtime for those instances varied widely between a few seconds and over 30 minutes. The gap between lower and upper bound of the MIP

Table 6.1: Results of the MIP, RCH, and RCH-VND for the routing part.

Instance						**RCH**		**RCH-VND**		**MIP**				**Rel. Difference**	
name	$\|I\|$	$\|V\|$	α	β	v	*obj*	t[s]	*obj*	t[s]	UB	LB	Gap	t[s]	Δ_{MIP}	Δ_{RCH}
burma14_01	13	19	0	0.2	2	495.80	< 0.01	333.49	0.03	**332.62**	332.62	0.00%	2025.56	0.26%	48.67%
burma14_02	13	21	0	0.2	2	525.49	0.01	440.86	0.05	**433.42**	421.91	2.66%	3600.00	4.30%	19.20%
burma14_03	13	19	0	0.2	2	607.11	< 0.01	397.14	0.03	**395.15**	387.89	1.84%	3600.00	2.33%	52.87%
burma14_04	13	19	0	0.2	2	409.54	< 0.01	273.28	0.01	**272.93**	272.93	0.00%	2318.17	0.13%	49.86%
burma14_05	13	17	0	0.5	2	372.63	< 0.01	312.70	0.02	**311.10**	311.10	0.00%	2.09	0.51%	19.16%
burma14_06	13	21	0	0.5	2	416.55	< 0.01	370.61	0.04	**360.71**	360.71	0.00%	3.35	2.67%	12.40%
burma14_07	13	20	0	0.5	2	386.02	< 0.01	342.00	0.02	**336.25**	336.25	0.00%	10.74	1.68%	12.87%
burma14_08	13	26	0	0.5	3	644.19	< 0.01	**573.70**	< 0.01	856.00	285.76	66.62%	3600.00	50.19%	12.29%
burma14_09	13	21	0	0.7	2	786.76	< 0.01	**612.87**	0.01	893.99	349.92	60.86%	3600.00	42.91%	28.37%
burma14_10	13	26	0.1	0.5	3	921.46	0.01	**872.60**	0.01	966.80	587.94	39.19%	3600.00	32.62%	5.60%
ulysses16_01	15	29	0	0.5	3	878.46	0.01	**717.16**	0.01	743.01	489.23	34.16%	3600.00	31.78%	22.49%
ulysses16_02	15	31	0	0.5	3	942.42	0.01	696.11	0.02	**664.55**	631.60	4.96%	3600.14	9.27%	35.38%
ulysses16_03	15	24	0.1	0.5	2	897.10	< 0.01	**849.15**	< 0.01	1126.33	450.89	59.97%	3600.00	46.90%	5.65%
ulysses16_04	15	29	0.1	0.5	3	838.78	0.01	**838.78**	0.01	1032.56	458.31	55.61%	3600.00	45.36%	0.00%
ulysses16_05	15	33	0.1	0.5	4	998.92	< 0.01	**729.55**	0.01	1001.00	611.42	38.92%	3600.00	16.19%	36.92%
ulysses16_06	15	23	0.1	0.7	2	814.06	< 0.01	692.76	0.01	**684.19**	684.19	0.00%	146.12	1.24%	17.51%
ulysses16_07	15	25	0.1	0.7	3	897.20	0.01	897.20	0.01	897.20	846.49	5.65%	3600.00	5.65%	0.00%
ulysses16_08	15	29	0.1	0.7	3	880.46	0.01	**868.43**	0.01	950.60	642.82	32.38%	3600.00	25.98%	1.38%
ulysses16_09	15	32	0.2	0.5	3	1042.15	0.01	895.57	0.01	**892.83**	801.35	10.25%	3600.00	10.52%	16.37%
ulysses16_10	15	28	0.2	0.7	3	1014.79	< 0.01	**835.11**	0.01	940.39	811.87	13.67%	3600.00	2.78%	21.52%
gr17_01	16	39	0	0.5	4	897.97	0.01	**759.32**	0.01	861.37	484.02	43.81%	3600.00	36.26%	18.26%
gr17_02	16	29	0	0.7	3	799.80	0.01	**642.57**	0.01	795.58	403.96	49.22%	3600.00	37.13%	24.47%
gr17_03	16	34	0	0.7	3	929.87	0.01	**813.72**	0.02	1010.92	437.85	56.69%	3600.00	46.19%	14.27%
gr17_04	16	43	0	0.7	4	862.50	0.01	**763.70**	0.02	983.55	496.14	49.56%	3600.00	35.04%	12.94%
gr17_05	16	24	0.1	0.5	2	709.42	< 0.01	**575.97**	0.01	828.98	379.97	54.16%	3600.00	34.03%	23.17%
gr17_06	16	33	0.1	0.5	3	896.88	0.01	**622.95**	0.02	893.58	468.21	47.60%	3600.00	24.84%	43.97%
gr17_07	16	37	0.1	0.5	4	1105.38	0.01	**859.87**	0.01	1205.00	358.63	70.24%	3600.00	58.29%	28.55%
gr17_08	16	21	0.1	0.7	2	746.57	< 0.01	**639.42**	0.01	813.48	285.02	64.96%	3600.00	55.42%	16.76%
gr17_09	16	25	0.2	0.7	2	874.32	< 0.01	**855.20**	< 0.01	952.60	357.71	62.45%	3600.00	58.17%	2.24%
gr17_10	16	37	0.2	0.7	4	921.85	0.02	894.37	0.02	**865.85**	865.78	0.01%	541.73	3.20%	3.07%
gr21_01	20	28	0.1	0.5	2	869.75	< 0.01	**622.22**	0.01	845.92	383.88	54.62%	3600.00	38.30%	39.78%
gr21_02	20	36	0.1	0.5	3	894.25	0.01	**731.75**	0.02	1031.00	518.16	49.74%	3600.00	29.19%	22.21%
gr21_03	20	29	0.1	0.7	2	884.47	< 0.01	**715.57**	0.01	1029.30	448.25	56.45%	3600.00	37.36%	23.60%
gr24_01	23	33	0	0.5	2	863.72	0.01	**558.10**	0.01	777.98	489.04	37.14%	3600.00	12.37%	54.76%
gr24_02	23	32	0	0.7	2	806.15	0.01	**688.95**	0.01	875.87	423.26	51.68%	3600.00	38.56%	17.01%
gr24_03	23	42	0	0.7	3	830.57	0.01	**703.32**	0.02	878.75	552.71	37.10%	3600.00	21.41%	18.09%
fri26_01	25	39	0.1	0.5	2	802.53	0.01	**742.98**	0.01	**742.98**	418.54	43.67%	3600.00	43.67%	8.01%
fri26_02	25	44	0.1	0.7	3	872.90	0.01	**803.42**	0.03	953.97	549.52	42.40%	3600.00	31.60%	8.65%
fri26_03	25	36	0.2	0.7	2	899.25	0.01	**782.80**	0.02	1122.99	470.71	58.08%	3600.00	39.87%	14.88%

also increased with the instance size. For many of the larger instances the gap could not be reduced below 50% within the running time of one hour. When looking at the relative gap between the RCH and RCH-VND (Δ_{RCH}), we can conclude that the VND improved greatly on the objective value of 36 of the 39 instances tested with only a minor increase in running time. RCH-VND yielded reasonable solutions with objective values close to the LB of the MIP for 9 out of the 10 smallest instances with 23 or less visits. Moreover, for those instances where the relative gap between the MIP and RCH-VND is greater than 10%, the MIP also has a relatively larger gap between UB and LB. When looking at those instances, the objective value of the RCH-VND is still well below the upper bound of the MIP for almost all instances. The running times of the RCH-VND did not increase noticeably for larger instances. Figure 6.1 is giving a graphical overview of the different objective values of the exact and heuristic approach.

As we require a fast method for deciding if a route is feasible within the districting problem, we conclude that RCH-VND is a reasonable choice for using it within the algorithms for solving the districting problem.

6.3 Results for the Districting Problem

The results for the districting problem are analyzed by comparing the different construction heuristics with each other, followed by different configurations for the IDR and finally, results of the districting construction heuristics combined with the iterative destroy & recreate algorithm are investigated.

6.3.1 Comparison of the Districting Construction Heuristics

The comparison of the different districting construction heuristics is shown in table 6.2. DCH^{b} denotes the columns with the results for the best possible insertion strategy, DCH^{f} the first feasible insertion strategy (see Section 4.3) and SCH a simple construction heuristic proposed by Nagata and Bräysy [33]. For the SCH each object is put into a separate district which results in a trivial initial solution candidate. The first feasible insertion strategy generated solutions with a better objective value than the best feasible insertion strategy on 3 problem instances, the latter resulted in better solutions for 22 problem instances. The DCH^{b} was able to outperform the DCH^{f} by up to 3 districts on all of the very large instances with more than 1000 visits. The results for the SCH are obviously worse than the other two strategies, since the objective value always reflects the number of objects of the problem instance. Comparing the running times of the different construction heuristics reveal, that the DCH^{f} was faster than the DCH^{b} on all but the smallest problem instances with running times of or below 0.1 seconds. It is apparent, that the SCH was the fastest construction heuristic for all problem instances. The DCH^{b} took up to 12.1 seconds for solving instance rat575, $\alpha = 10$, $\beta = 50$, $v = 4$, which was almost three times longer than the runtime of 4.1 seconds of the DCH^{b} on the same problem instance.

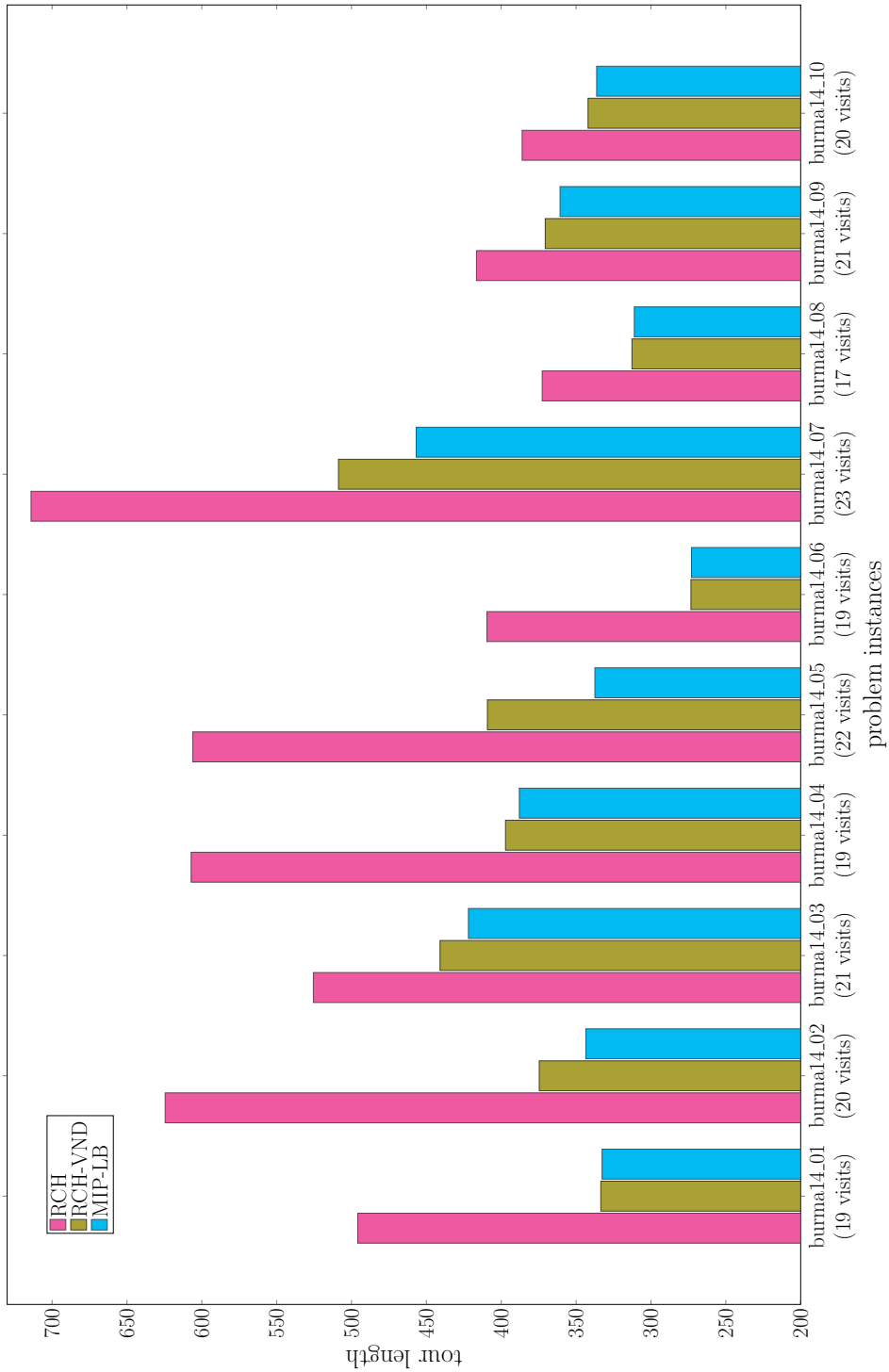

Figure 6.1: Tour lengths computed by the routing construction heuristic with and without the variable neighborhood descent heuristic compared to the lower bound of the mixed integer linear programming model.

Table 6.2: Results of the different districting construction heuristics.

Instance						**DCH**^{b}		**DCH**^{f}		**SCH**	
name	$\|I\|$	$\|V\|$	α	β	v	obj	t^*[s]	obj	t^*[s]	obj	t^*[s]
d198	197	297	10	50	2	11	0.3	11	0.2	197	<0.1
d198	197	472	10	50	4	**16**	0.9	17	0.3	197	<0.1
d198	197	288	20	70	2	**13**	0.2	14	0.2	197	<0.1
d198	197	500	20	70	4	20	0.5	20	0.2	197	<0.1
gil262	261	389	10	50	2	12	1	12	0.5	261	0.1
gil262	261	635	10	50	4	19	1.7	19	0.6	261	0.1
gil262	261	404	20	70	2	15	0.6	15	0.4	261	0.1
gil262	261	682	20	70	4	**20**	1.1	21	0.5	261	0.1
gr137	136	202	10	50	2	19	0.1	**18**	0.1	136	<0.1
gr137	136	357	10	50	4	**30**	0.2	31	0.1	136	<0.1
gr137	136	203	20	70	2	**19**	0.1	20	0.1	136	<0.1
gr137	136	335	20	70	4	33	0.1	33	0.1	136	<0.1
gr202	201	306	10	50	2	15	0.4	15	0.2	201	<0.1
gr202	201	495	10	50	4	23	0.7	**22**	0.3	201	<0.1
gr202	201	308	20	70	2	15	0.3	15	0.2	201	<0.1
gr202	201	499	20	70	4	**25**	0.4	26	0.3	201	<0.1
gr96	95	145	10	50	2	**12**	<0.1	13	<0.1	95	<0.1
gr96	95	247	10	50	4	**22**	0.1	23	0.1	95	<0.1
gr96	95	145	20	70	2	15	<0.1	**14**	<0.1	95	<0.1
gr96	95	235	20	70	4	25	0.1	25	0.1	95	<0.1
lin318	317	459	10	50	2	22	0.7	22	0.5	317	0.1
lin318	317	769	10	50	4	**34**	1.4	35	0.7	317	0.1
lin318	317	485	20	70	2	**26**	0.6	28	0.4	317	0.1
lin318	317	799	20	70	4	**39**	0.9	42	0.6	317	0.1
pr107	106	158	10	50	2	**16**	0.1	17	<0.1	106	<0.1
pr107	106	168	10	50	4	6	0.3	6	0.1	106	<0.1
pr107	106	162	20	70	2	**18**	0.1	19	<0.1	106	<0.1
pr107	106	222	20	70	4	8	0.1	8	0.1	106	<0.1
pr299	298	446	10	50	2	27	0.5	27	0.4	298	0.1
pr299	298	726	10	50	4	**41**	0.9	42	0.6	298	0.1
pr299	298	444	20	70	2	**30**	0.4	31	0.4	298	0.1
pr299	298	769	20	70	4	**50**	0.7	51	0.5	298	0.1
pr439	438	661	10	50	2	51	1.2	51	1	438	0.1
pr439	438	1075	10	50	4	**25**	6.7	26	2.4	438	0.1
pr439	438	673	20	70	2	**57**	1	59	0.9	438	0.1
pr439	438	1039	20	70	4	**26**	3.3	27	1.6	438	0.1
rat575	574	845	10	50	2	24	4.5	24	2.8	574	0.2
rat575	574	1446	10	50	4	**35**	12.1	36	4.1	574	0.2
rat575	574	870	20	70	2	27	3	27	2.2	574	0.2
rat575	574	1388	20	70	4	**37**	6	40	2.9	574	0.2

6.3.2 Comparison of the District Deletion Strategies

The IDR uses four different strategies for selecting a district for deletion as explained in Section 5.4. These strategies are tested by running the IDR with the simple construction heuristic (SCH). This guarantees far more iterations of the IDR algorithm than with any other construction heuristic as the SCH generates as many districts as there are objects in the problem instance. This should make differences in the performance of the various district deletion strategies most obvious. Table 6.3 shows the different run configurations with regard to the selected deletion strategy whereas $\text{IDR}^{\text{del}^{\text{rand}}}$ shows the results of selecting deletion strategies for each iteration uniformly at random; $\text{IDR}^{\text{del}^{\text{visits}}}$ shows the results of deleting the district with the least amount of scheduled visits, $\text{IDR}^{\text{del}^{\text{tour}}}$ shows the results of deleting the district with the shortest maximum tour duration and $\text{IDR}^{\text{del}^{\text{wait}}}$ deletes the district with maximum waiting time. The results do not show that any of the proposed district deletion strategies to be consistently better than the others. Each strategy was outperforming the other three strategies only on some problem instances resulting in the same objective value for most instances. This suggests, that the method for selecting a district for deletion might not be as important as the following repair mechanisms for recreating feasible routes for the remaining districts.

6.3.3 Comparison of VND Neighborhoods

Repairing an infeasible solution during the recreation phase of the iterative destroy & recreate algorithm for the districting problem is accomplished by using three different neighborhood structures in a VND approach as described in Section 4.4. Since the neighborhood structures are applied using a fixed order, resulting in more runs for the first and the second neighborhood structure than for the third, their relative success rate was compared. The columns $sr[\%]$ in table 6.4 show the relative success rate for each VND neighborhood structure. The success rate is computed by dividing the number of successful improvements within that neighborhood structure divided by the number of total runs of that neighborhood structure. Columns *sd* show the standard deviation of the success rate for the 20 runs of each instance. The swap neighborhood structure was by far the most successful one for all tested instances followed by the or-opt neighborhood structure. The success of the 2-opt neighborhood structure was inferior with rates below 3%. Further examination of these results led to the observation that many generated 2-opt and or-opt neighbors have to be dismissed as infeasible prematurely. This is because the massive reordering of visits after applying the neighborhood moves of or-opt and especially 2-opt often results in an invalid visit order for a single object, since visit i_k has to be satisfied before i_{k+1} as mentioned in Section 1.1. Since the swap move only changes the order of two visits at a time, its performance suffered the least from this constraint.

6.3.4 Results of the Iterative Destroy & Recreate Algorithm

For testing the proposed districting construction heuristics in combination with the iterative destroy & recreate algorithm for the DRPSC we used three different configurations.

Table 6.3: Results of the different district deletion strategies.

Instance							**IDR**$^{\text{del}^{\text{rand}}}$			**IDR**$^{\text{del}^{\text{visits}}}$			**IDR**$^{\text{del}^{\text{tour}}}$			**IDR**$^{\text{del}^{\text{wait}}}$		
name	runs	$\|I\|$	$\|V\|$	α	β	v	*obj*	*sd*	t^*[s]	*obj*	*sd*	t^*[s]	*obj*	*sd*	t^*[s]	*obj*	*sd*	t^*[s]
d198	20	197	297	10	50	2	13.0	0.0	39	13.0	0.0	41	13.0	0.0	55	13.0	0.0	43
d198	20	197	472	10	50	4	18.0	0.0	112	18.0	0.0	118	18.0	0.0	209	18.0	0.0	152
d198	20	197	288	20	70	2	13.0	0.0	282	13.0	0.0	245	13.0	0.0	274	13.0	0.0	239
d198	20	197	500	20	70	4	21.0	0.0	366	21.0	0.0	335	21.0	0.0	407	21.0	0.0	286
gil262	20	261	389	10	50	2	11.0	0.0	130	11.0	0.0	41	11.0	0.0	46	11.0	0.0	51
gil262	20	261	635	10	50	4	17.0	0.0	374	17.0	0.0	240	17.0	0.0	374	17.0	0.0	290
gil262	20	261	404	20	70	2	12.0	0.0	74	12.0	0.0	121	12.0	0.0	117	12.0	0.0	91
gil262	20	261	682	20	70	4	19.0	0.0	404	19.0	0.0	413	19.0	0.0	433	19.0	0.0	482
gr137	20	136	202	10	50	2	18.0	0.0	387	19.0	0.0	87	18.0	0.0	550	18.0	0.0	253
gr137	20	136	357	10	50	4	27.0	0.0	310	27.0	0.0	281	27.0	0.0	253	**26.2**	0.4	593
gr137	20	136	203	20	70	2	**18.0**	0.0	379	19.0	0.0	156	19.0	0.0	178	19.0	0.0	133
gr137	20	136	335	20	70	4	28.0	0.0	362	28.0	0.0	336	28.0	0.0	366	28.0	0.0	359
gr202	20	201	306	10	50	2	17.0	0.0	37	17.0	0.0	25	17.0	0.0	57	17.0	0.0	57
gr202	20	201	495	10	50	4	24.0	0.0	236	24.0	0.0	190	24.0	0.0	254	24.0	0.0	260
gr202	20	201	308	20	70	2	16.0	0.0	448	16.0	0.0	305	17.0	0.0	43	16.0	0.0	404
gr202	20	201	499	20	70	4	25.0	0.0	542	25.0	0.0	501	25.0	0.0	402	25.0	0.0	414
gr96	20	95	145	10	50	2	12.0	0.0	153	12.0	0.0	385	12.0	0.0	154	12.0	0.0	271
gr96	20	95	247	10	50	4	18.0	0.0	92	18.0	0.0	179	18.0	0.0	129	18.0	0.0	126
gr96	20	95	145	20	70	2	13.0	0.0	41	12.0	0.0	437	12.0	0.0	447	12.0	0.0	474
gr96	20	95	235	20	70	4	20.0	0.0	243	20.0	0.0	107	20.0	0.0	142	20.0	0.0	234
lin318	20	317	459	10	50	2	29.0	0.0	132	29.0	0.0	155	29.0	0.0	133	29.0	0.0	29
lin318	20	317	769	10	50	4	42.0	0.0	543	42.0	0.0	291	42.0	0.0	279	42.0	0.0	331
lin318	20	317	485	20	70	2	31.0	0.0	521	31.0	0.0	467	31.6	0.5	175	31.0	0.0	518
lin318	20	317	799	20	70	4	48.0	0.0	470	48.0	0.0	476	48.0	0.0	545	**47.9**	0.3	384
pr107	20	106	158	10	50	2	14.0	0.0	43	**13.0**	0.0	253	**13.0**	0.0	268	14.0	0.0	54
pr107	20	106	168	10	50	4	4.0	0.0	56	4.0	0.0	122	4.0	0.0	109	4.0	0.0	179
pr107	20	106	162	20	70	2	15.0	0.0	133	15.0	0.0	120	15.0	0.0	83	15.0	0.0	135
pr107	20	106	222	20	70	4	6.0	0.0	223	6.0	0.0	269	6.0	0.0	441	6.0	0.0	214
pr299	20	298	446	10	50	2	31.0	0.0	368	31.0	0.0	493	31.0	0.0	250	31.0	0.0	277
pr299	20	298	726	10	50	4	45.0	0.0	352	44.8	0.4	318	45.0	0.0	421	**44.4**	0.5	538
pr299	20	298	444	20	70	2	**31.0**	0.0	564	32.0	0.0	482	32.0	0.0	339	32.0	0.0	408
pr299	20	298	769	20	70	4	50.0	0.0	477	50.0	0.0	568	50.0	0.0	519	51.0	0.0	419
pr439	20	438	661	10	50	2	64.0	0.0	466	64.0	0.0	309	64.0	0.0	381	64.1	0.3	583
pr439	20	438	1075	10	50	4	27.0	0.0	304	27.0	0.0	288	27.8	0.4	230	27.0	0.0	391
pr439	20	438	673	20	70	2	67.5	0.5	424	68.0	0.0	416	**67.0**	0.0	563	68.0	0.0	505
pr439	20	438	1039	20	70	4	28.0	0.0	224	28.0	0.0	336	28.0	0.0	586	28.0	0.0	543
rat575	20	574	845	10	50	2	25.0	0.0	467	25.0	0.0	488	25.1	0.3	587	25.0	0.0	381
rat575	20	574	1446	10	50	4	41.0	0.0	188	**39.0**	0.0	585	40.0	0.0	564	39.4	0.5	577
rat575	20	574	870	20	70	2	28.0	0.0	283	28.0	0.0	374	**27.3**	0.5	567	28.0	0.0	390
rat575	20	574	1388	20	70	4	44.0	0.0	229	44.0	0.0	394	44.0	0.0	173	44.0	0.0	229

Table 6.4: Results of the different VND neighborhoods.

Instance							**swap**		**2-opt**		**or-opt**	
name	runs	$\|I\|$	$\|V\|$	α	β	v	$sr[\%]$	sd	$sr[\%]$	sd	$sr[\%]$	sd
d198	20	197	297	10	50	2	53.93	0.07	2.41	0.02	14.39	0.10
d198	20	197	472	10	50	4	51.95	0.07	1.49	0.01	13.57	0.04
d198	20	197	288	20	70	2	51.73	0.10	1.68	0.02	12.13	0.16
d198	20	197	500	20	70	4	45.56	0.21	0.72	0.01	9.54	0.17
gil262	20	261	389	10	50	2	55.01	0.04	2.06	0.01	10.42	0.06
gil262	20	261	635	10	50	4	52.40	0.02	1.33	0.03	12.79	0.08
gil262	20	261	404	20	70	2	50.80	0.02	1.27	0.00	8.79	0.03
gil262	20	261	682	20	70	4	50.34	0.05	0.90	0.01	8.75	0.12
gr137	20	136	202	10	50	2	46.56	0.05	1.56	0.02	19.48	0.05
gr137	20	136	357	10	50	4	38.41	0.03	0.13	0.00	16.14	0.02
gr137	20	136	203	20	70	2	38.01	0.10	0.61	0.01	9.37	0.04
gr137	20	136	335	20	70	4	33.05	0.10	0.04	0.00	9.90	0.04
gr202	20	201	306	10	50	2	52.87	0.03	2.04	0.01	16.09	0.03
gr202	20	201	495	10	50	4	47.35	0.20	0.61	0.02	13.92	0.21
gr202	20	201	308	20	70	2	48.74	0.08	1.66	0.02	13.07	0.15
gr202	20	201	499	20	70	4	42.33	0.03	0.45	0.00	10.19	0.03
gr96	20	95	145	10	50	2	48.64	0.12	1.90	0.02	19.50	0.10
gr96	20	95	247	10	50	4	38.11	0.04	0.13	0.00	13.62	0.07
gr96	20	95	145	20	70	2	43.70	0.18	0.96	0.03	12.30	0.11
gr96	20	95	235	20	70	4	32.46	0.56	0.02	0.00	8.62	0.14
lin318	20	317	459	10	50	2	51.70	0.01	2.34	0.02	16.14	0.04
lin318	20	317	769	10	50	4	47.42	0.03	0.77	0.01	14.47	0.05
lin318	20	317	485	20	70	2	44.81	0.01	1.15	0.00	11.05	0.02
lin318	20	317	799	20	70	4	42.16	0.02	0.33	0.00	8.60	0.02
pr107	20	106	158	10	50	2	43.65	0.02	1.66	0.01	13.45	0.03
pr107	20	106	168	10	50	4	56.03	0.75	1.06	0.04	7.07	0.30
pr107	20	106	162	20	70	2	28.33	0.18	0.27	0.01	5.09	0.08
pr107	20	106	222	20	70	4	43.69	0.70	0.51	0.05	8.48	0.17
pr299	20	298	446	10	50	2	49.05	0.02	1.96	0.01	15.75	0.03
pr299	20	298	726	10	50	4	45.68	0.07	0.54	0.00	13.54	0.02
pr299	20	298	444	20	70	2	45.67	0.03	1.35	0.01	10.54	0.02
pr299	20	298	769	20	70	4	42.17	0.06	0.25	0.01	8.58	0.04
pr439	20	438	661	10	50	2	41.72	0.10	0.92	0.01	12.47	0.04
pr439	20	438	1075	10	50	4	52.80	0.33	1.51	0.09	12.18	0.79
pr439	20	438	673	20	70	2	38.06	0.03	0.59	0.01	7.80	0.04
pr439	20	438	1039	20	70	4	49.80	0.01	0.63	0.02	9.98	0.00
rat575	20	574	845	10	50	2	53.71	0.20	2.21	0.05	11.54	0.32
rat575	20	574	1446	10	50	4	52.34	0.07	1.51	0.07	10.06	0.29
rat575	20	574	870	20	70	2	52.66	0.08	1.67	0.02	8.73	0.10
rat575	20	574	1388	20	70	4	49.15	0.07	1.30	0.01	7.93	0.05

Table 6.5: Results of the DCH and IDR for the districting part.

Instance							IDR-SCH			IDR-DCH^{f}				IDR-DCH^{b}			
name	runs	$\|I\|$	$\|V\|$	α	β	v	*obj*	*sd*	t^*[s]	obj_f	t_f[s]	*obj*	*sd*	t^*[s]	obj_b	t_b	*obj*	*sd*	t^*[s]
st70_1	20	69	105	0.1	0.7	2	3.0	0.0	11	6	< 0.1	3.0	0.0	5	5	0.1	3.0	0.0	9
st70_2	20	69	91	0.1	0.7	2	3.0	0.0	4	6	< 0.1	3.0	0.0	6	5	0.1	3.0	0.0	6
st70_3	20	69	106	0.1	0.7	2	3.0	0.0	11	6	< 0.1	3.0	0.0	13	5	0.1	3.0	0.0	11
rd100_1	20	99	152	0.1	0.5	2	6.0	0.0	8	9	< 0.1	6.0	0.0	10	9	0.1	6.0	0.0	14
rd100_2	20	99	160	0.1	0.5	2	6.0	0.0	16	8	0.1	6.0	0.0	14	8	0.1	6.0	0.0	6
rd100_3	20	99	152	0.1	0.5	2	6.0	0.0	3	7	0.1	6.0	0.0	2	8	0.1	6.0	0.0	7
tsp225_1	20	224	334	0.2	0.7	2	11.0	0.0	47	13	0.2	11.0	0.0	64	13	0.4	11.0	0.0	121
tsp225_2	20	224	341	0.2	0.7	2	11.0	0.0	46	13	0.3	11.0	0.0	67	13	0.5	11.0	0.0	36
tsp225_3	20	224	332	0.2	0.7	2	10.0	0.0	226	12	0.3	10.0	0.0	257	12	0.5	10.0	0.0	177
gr48_1	20	47	120	0.2	0.7	4	5.0	0.0	6	8	< 0.1	5.0	0.0	4	7	< 0.1	5.0	0.0	14
gr48_2	20	47	115	0.2	0.7	4	5.0	0.0	12	7	< 0.1	5.0	0.0	7	7	< 0.1	5.0	0.0	10
gr48_3	20	47	125	0.2	0.7	4	4.0	0.0	527	7	< 0.1	4.0	0.0	438	6	< 0.1	4.0	0.0	488
berlin52_1	20	51	133	0.0	0.7	4	5.0	0.0	3	6	< 0.1	5.0	0.0	2	7	0.1	5.0	0.0	7
berlin52_2	20	51	130	0.0	0.7	4	5.0	0.0	5	7	< 0.1	**4.0**	0.0	142	6	0.1	5.0	0.0	1
berlin52_3	20	51	140	0.0	0.7	4	5.0	0.0	19	7	< 0.1	5.0	0.0	14	6	0.1	5.0	0.0	16
ft70_1	20	69	167	0.1	0.5	4	8.0	0.0	12	12	< 0.1	8.0	0.0	12	10	0.1	8.0	0.0	18
ft70_2	20	69	180	0.1	0.5	4	8.0	0.0	33	11	< 0.1	8.0	0.0	15	11	0.1	8.0	0.0	18
ft70_3	20	69	144	0.1	0.5	4	7.0	0.0	41	9	< 0.1	7.0	0.0	36	9	0.1	7.0	0.0	19
ch150_1	20	149	360	0.2	0.5	4	11.0	0.0	589	15	0.2	**11.0**	0.0	480	15	0.4	11.2	0.4	881
ch150_2	20	149	402	0.2	0.5	4	12.0	0.0	342	17	0.2	12.0	0.0	338	15	0.4	12.0	0.0	387
ch150_3	20	149	357	0.2	0.5	4	11.0	0.0	655	14	0.2	11.0	0.0	520	13	0.4	11.0	0.0	808

The iterative destroy & recreate algorithm was tested with both the DCH with the first feasible (IDR-DCH^{f}) and the best possible insertion strategy (IDR-DCH^{b}). Both configurations are compared with the IDR-SCH, where the simple construction heuristic is used. The random district deletion strategy was chosen for all runs.

In Table 6.5 the results of the experiments are shown. Columns *obj* show the average objective value, i.e., the minimum number of districts at the end of the optimization after the full run of the IDR algorithm, while columns obj_f and obj_b show the average number of districts after the respective construction heuristic. Columns t^* show the median time in seconds after which the best solution has been found during the run of the IDR algorithm while t_f and t_b show the median time after which the respective construction heuristic has found an initial solution. Columns *sd* show the standard deviation of the objective value for the 20 runs of a single instance.

We observe that for most instances the final objective value of the IDR is the same for all three configurations. There are, however, differences for the construction heuristics alone and the DCH^{b} for most but not all instances yielded better results but needed more time. The IDR-SCH works surprisingly well and was able to find good results in about the same amount of time as the other two (more sophisticated) configurations. Specifically, for the instances st70_1, st70_2 and st70_3 the IDR algorithm was able to cut the objective value of the DCH^{f} into half. Overall, the standard deviation was zero for almost all instances, showing that the IDR is very robust over the course of

multiple runs. Figure 6.2 shows the differences in the objective values and runtime of the construction heuristics and the IDR.

Figure 6.2: Number of districts computed by the first feasible and best feasible construction heuristics, without and with improvement of the iterative destroy & recreate algorithm for selected instances. Running times for complete runs of all construction heuristics, each followed by the IDR algorithm.

CHAPTER 7

Conclusions and Future Work

In this work we introduced a new vehicle routing problem which originates from the security control sector. The goal of the Districting and Routing Problem for Security Control is to partition a set of objects under surveillance into disjoint clusters such that for each period a route through all requested visits can be scheduled satisfying complex time window constraints. As the objects may require multiple visits there needs to be a minimum separation time between each two visits which imposes an interesting additional challenge. The proposed heuristic solution approach starts with a district construction heuristic followed by an iterative destroy & recreate algorithm. The latter works by iteratively destroying districts and trying to insert the resulting unassigned objects into the other districts.

The computational results for the routing problem show that the proposed MIP model is able to solve smaller instances to optimality. The variable neighborhood descent was able to improve the initial solutions of the routing construction heuristics significantly. Furthermore, the results of the RCH-VND came close to the lower bound of the MIP results for all but a few instances. For those instances, where the RCH-VND did not come close to the MIP, the MIP was also far from closing the gap between the lower and upper bound, making the difficulty of solving these instances apparent.

On the one hand the tests showed that none of the different deletion strategies for removing a district from an interim solution during the destroy phase of the iterative destroy & recreate algorithm outperformed the others on more than a few specific problem instances each. On the other hand a substantially superior success rate was observed for the swap neighborhood structure of the VND during the recreation step of the IDR algorithm. The greedy construction heuristic with a first feasible insertion strategy yielded good starting solutions while the same heuristic with a best feasible insertion strategy resulted in slightly better initial solutions for many problem instances. Overall the iterative destroy & recreate algorithm improved on the initial solution, generated by by the different construction heuristics. For the districting problem the results revealed

that the quality of the initial solutions of the districting problem has only a minor influence on the final solution quality of the IDR algorithm.

7.1 Future Work

For further evaluation of the heuristics proposed in this thesis, enhancing the MIP models, which are presented in Sections 4.1 and 5.1 for the routing and the districting problem, respectively, with decomposition techniques like branch & cut, column generation or Benders decomposition is desirable. Especially for the districting part, solutions of a MIP-based approach could provide more information about the solution quality of the presented variants of the IDR algorithm.

Another interesting aspect for further analyzing the proposed approaches for solving the DRPSC is the change in solution quality for different problem instances regarding the relation between travel times and visit times of objects. One could investigate, which approaches are specifically well suited for problem instances with rather long travel times combined with very short visit times and vice versa.

As the feasibility check for a district is time-consuming a caching mechanism to prevent checking the same assignment of objects to a district all over again seems promising. This could even be extended to checking if such an assignment is a subset of a previously evaluated assignment. Since by the nature of the DRPSC, a subset of objects always has a feasible route if any superset of these objects results in a feasible route and for a set of objects without a feasible route, any superset can immediately be considered infeasible as well. An efficient way to prevent checking duplicate solutions multiple times and subsequently convert them into new solutions was introduced by Raidl and Hu [42] with solution archives. These trie-based complete solution archives have been used for improving different algorithms and heuristics with great success [20, 19, 43, 6, 5] and a similar approach could also benefit our proposed algorithm.

Using inter-route neighborhood structures which exchange objects of two or more distinct districts in large neighborhood search heuristics seems promising. Adaptive large neighborhood search (ALNS) has already been successfully applied to many different vehicle routing problems [37] and might yield good results for the DRPSC, by steering the destroy and repair heuristics used by the ALNS towards reducing the number of districts.

Finally, holding on to the hard time constraints introduced with the DRPSC might often be counterproductive for real-world applications. Since small delays in the actual execution of scheduled tours may occur from time to time anyways, softened time constraints for object's time windows, separation times and/or maximum working time may result in solutions of practical relevance.

List of Algorithms

List of Figures

List of Tables

Bibliography

[1] Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel. Solving the asymmetric travelling salesman problem with time windows by branch-and-cut. *Mathematical Programming*, 90(3):475–506, 2001.

[2] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 24(3):356–371, 2012.

[3] Russell Bent and Pascal Van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4):515–530, 2004.

[4] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[5] Benjamin Biesinger, Bin Hu, and Günther Raidl. A hybrid genetic algorithm with solution archive for the discrete (r|p)-centroid problem. *Journal of Heuristics*, 21(3):391–431, 2015.

[6] Benjamin Biesinger, Christian Schauer, Bin Hu, and Günther R. Raidl. Enhancing a genetic algorithm with a solution archive to reconstruct cross cut shredded text documents. In Roberto Moreno-Díaz, Franz Pichler, and Alexis Quesada-Arencibia, editors, *Computer Aided Systems Theory – EUROCAST 2013*, volume 8111 of *Lecture Notes in Computer Science*, pages 380–387. Springer, 2013.

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

[8] Chi-Bin Cheng and Chun-Pin Mao. A modified ant colony system for solving the travelling salesman problem with time windows. *Mathematical and Computer Modelling*, 46(9):1225–1235, 2007.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[10] Rodrigo Ferreira Da Silva and Sebastián Urrutia. A general VNS heuristic for the traveling salesman problem with time windows. *Discrete Optimization*, 7(4):203–211, 2010.

[11] George B. Dantzig. Linear programming. In *Problems for the Numerical Analysis of the Future*, Proceedings of Symposium on Modern Calculating Machinery and Numerical Methods. UCLA, 1948.

[12] Sanjeeb Dash, Oktay Günlük, Andrea Lodi, and Andrea Tramontani. A time bucket formulation for the TSP with time windows. *INFORMS Journal on Computing*, 24(1):132–147, 2012.

[13] Yvan Dumas, Jacques Desrosiers, and Francois Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, 1991.

[14] Luca Maria Gambardella, Éric Taillard, and Giovanni Agazzi. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. Technical report, 1999.

[15] Hermann Gehring and Jörg Homberger. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In *Proceedings of EUROGEN99*, volume 2, pages 57–64. Springer, 1999.

[16] Hermann Gehring and Jörg Homberger. Parallelization of a two-phase metaheuristic for routing problems with time windows. *Journal of Heuristics*, 8(3):251–276, 2002.

[17] Michel Gendreau and Jean-Yves Potvin. *Handbook of metaheuristics*, volume 2. Springer, 2010.

[18] Holger Höller, Belén Melián, and Stefan Voß. Applying the pilot method to improve VNS and GRASP metaheuristics for the design of SDH/WDM networks. *European Journal of Operational Research*, 191(3):691–704, 2008.

[19] Bin Hu and Günther R. Raidl. An evolutionary algorithm with solution archive for the generalized minimum spanning tree problem. In R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, editors, *Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I*, volume 6927 of *Lecture Notes in Computer Science*, pages 287–294. Springer, 2012.

[20] Bin Hu and Günther R. Raidl. An evolutionary algorithm with solution archives and bounding extension for the generalized minimum spanning tree problem. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 393–400. ACM Press, 2012.

[21] Brian Kallehauge, Jesper Larsen, Oli BG Madsen, and Marius M Solomon. *Vehicle routing problem with time windows*. Springer, 2005.

[22] Gerard AP Kindervater and Martin W.P. Savelsbergh. Vehicle routing: handling edge exchanges. In Emile Aarts and Jan K. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 10, pages 337–360. John Wiley & Sons, New York, NY, USA, 1st edition, 1997.

[23] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.

[24] Andrew Lim and Xingwen Zhang. A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 19(3):443–457, 2007.

[25] Manuel López-Ibáñez and Christian Blum. Beam-aco for the travelling salesman problem with time windows. *Computers & Operations Research*, 37(9):1570–1583, 2010.

[26] Manuel López-Ibáñez, Christian Blum, Jeffrey W Ohlmann, and Barrett W Thomas. The travelling salesman problem with time windows: Adapting algorithms from travel-time to makespan optimization. *Applied Soft Computing*, 13(9):3806–3815, 2013.

[27] Julien Michallet, Christian Prins, Lionel Amodeo, Farouk Yalaoui, and Grégoire Vitry. Multi-start iterated local search for the periodic vehicle routing problem with time windows and time spread constraints on services. *Computers & Operations Research*, 41(0):196–207, 2014.

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

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

[30] Nenad Mladenović, Raca Todosijević, and Dragan Urošević. An efficient GVNS for solving traveling salesman problem with time windows. In *EURO Mini Conference*, volume 39 of *Electronic Notes in Discrete Mathematics*, pages 83–90. Elsevier, 2012.

[31] Yuichi Nagata. Efficient evolutionary algorithm for the vehicle routing problem with time windows: Edge assembly crossover for the VRPTW. In *IEEE Congress on Evolutionary Computation*, pages 1175–1182. IEEE, 2007.

[32] Yuichi Nagata and Olli Bräysy. A powerful route minimization heuristic for the vehicle routing problem with time windows. *Operations Research Letters*, 37(5):333–338, 2009.

[33] Yuichi Nagata, Olli Bräysy, and Wout Dullaert. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, 37(4):724–737, 2010.

[34] Phuong Khanh Nguyen, Teodor Gabriel Crainic, and Michel Toulouse. A hybrid generational genetic algorithm for the periodic vehicle routing problem with time windows. *Journal of Heuristics*, 20(4):383–416, 2014.

[35] Jeffrey W Ohlmann and Barrett W Thomas. A compressed-annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, 19(1):80–90, 2007.

[36] Victor Pillac, Christelle Gueret, and Andrés L Medaglia. A parallel matheuristic for the technician routing and scheduling problem. *Optimization Letters*, 7(7):1525–1535, 2013.

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

[38] Jean-Yves Potvin and Jean-Marc Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, 1993.

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

[40] Eric Prescott-Gagnon, Guy Desaulniers, and Louis-Martin Rousseau. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, 54(4):190–204, 2009.

[41] Michael Prischink, Christian Kloimüllner, Benjamin Biesinger, and Günther R. Raidl. Districting and routing for security control. In *Hybrid Metaheuristics, 10th Int. Workshop, HM 2016*, Lecture Notes in Computer Science. Springer, 2016. to appear.

[42] Günther R. Raidl and Bin Hu. Enhancing genetic algorithms by a trie-based complete solution archive. In Peter Cowling and Peter Merz, editors, *Evolutionary Computation in Combinatorial Optimisation – EvoCOP 2010*, volume 6022 of *Lecture Notes in Computer Science*, pages 239–251. Springer, 2010.

[43] Mario Ruthmair and Günther R. Raidl. A memetic algorithm and a solution archive for the rooted delay-constrained minimum spanning tree problem. In R. Moreno-Díaz et al., editors, *Proceedings of the 13th International Conference on Computer Aided Systems Theory: Part I*, volume 6927 of *Lecture Notes in Computer Science*, pages 351–358. Springer, 2012.

[44] Martin W.P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4(1):285–305, 1985.

[45] Martin W.P. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *ORSA Journal on Computing*, 4(2):146–154, 1992.

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

[47] Ronald van Steden and Rick Sarre. The growth of private security: Trends in the european union. *Security Journal*, 20(4):222–235, 2007.

[48] 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.

[49] Laurence A Wolsey. *Integer programming*, volume 42. Wiley New York, 1998.

[50] D.B. Yudin and A.S. Nemirovskii. Informational complexity and efficient methods for the solution of convex extremal problems. *Ékonomika i Matematicheskie Metody*, 12(0):357–369, 1976.