Wissenschaftliche Arbeit zur Erlangung des Grades
Master of Science
an der Technischen Universität München

A Metaheuristic Approach for the Vehicle Routing Problem With Load-dependent Travel Times

Referent:	Prof. Dr. Stefan Minner Lehrstuhl für Logistics and Supply Chain Management Technische Universität München
Betreuer:	Dr. Pirmin Fontaine
Studiengang:	Master in Management & Technology
Eingereicht von:	Manuel Freytag Aßlkofenerstraße 14 85560 Ebersberg Tel.: +49 1573 4988846 Matrikelnummer: 03643418
Eingereicht am:	01.05.2019

We propose an adaptive large neighborhood search heuristic for the Vehicle Routing Problem with Load-Dependent Travel Times (VRPLDTT), which is an extension to the Vehicle Routing Problem with Time Windows (VRPTW). This formulation assumes the use of cargo bikes as transportation mode and recognizes the load and slope dependency of their driving speed. We avoid a large number of recalculations by discretizing the load representation. Our algorithm is able to improve both the diversification and intensification of the original framework. This is achieved through adjustments to the weighting procedure and a mechanism to adapt the neighborhood size based on the stagnation of the search procedure. Benchmarking against both standardized VRPTW and VRPLDTT instances shows that the implementation is competitive for problem sizes with up to 400 customers. Finally, we introduce new VRPLDTT benchmarking instances with up to 200 customers and generate first solutions for future comparison.

Table of Contents

List of Figures

List of Tables

Abbreviations

β-HI	β-Hybrid Insertion.
ALNS	Adaptive Large Neighborhood Search.
ALNSLDTT	Adaptive Large Neighborhood Search with Load Dependent Travel Times.
BKS	Best Known Solution.
CART	Classification and Regression Trees.
CNV	Cumulated Number of Vehicles.
CTD	Cumulated Travel Distance.
D-GI	Deep Greedy Insertion.
DeSR	Demand Similarity Removal.
DiSR	Distance Similarity Removal.
DR	Demand Removal.
GI	Greedy Insertion Heuristic.
k-RI	k-Regret Insertion.
LNS	Large Neighborhood Search.
MAD	Median Average Deviation.
MIP	Mixed-Integer Program.
MSE	Mean Squared Error.
NPR	Node Pair Removal.
R-GI	Random Greedy Insertion.
RR	Random Removal.
RRR	Random Route Removal.
SA	Simulated Annealing.
SR	Shaw Removal.

TR	Time Removal.
VRP	Vehicle Routing Problem.
VRPLDTT	Vehicle Routing Problem with Load-Dependent Travel Times.
VRPTW	Vehicle Routing Problem with Time Windows.
WiSR	Window Similarity Removal.
WoR	Worst Removal.

1 Introduction

1.1 Current challenges of urban logistics

Globalization and increasing urbanization present great challenges for both infrastructure and logistics in large cities and metropolises. According to an United Nations report, approximately 50% of the global population is expected to live in cities by 2050. That is an estimated increase of 20% compared to 2014 (DESA et al., 2014). Together with a population growth of 30%, this will increase the urban population by 50% by 2050. (DESA, 2017).

Pollution limits are being repeatedly exceeded, because of the increase in city traffic that results from urbanization. This issue receives growing attention from governments and the public. For example, the European Commission sued multiple countries for exceeding the allowed limits of nitrogen dioxide (NO_2) and particulate matter (PM_{10}) in May of 2018 (European Commission, 2018). The Chinese government released a new action plan for 2020 to reduce air pollution in Chinese cities as well (Chinese State Council, 2018). Improving the environmental balance in urban logistics can be an integral component of tackling this problem.

Growing public awareness and increasing necessity of urban logistics benefits funding and research popularity in recent years. Urban logistics can be divided into two fields: Human transport systems and city logistics. This thesis focuses on a problem that can be attributed to the latter. The discipline of city logistics is only loosely defined, but is commonly associated with the goals of improving efficiency and environmental friendliness of urban freight transport (Bektas et al., 2015).

Dekker et al. (2012) documents several findings and accomplishments of environmentally oriented logistics. A popular technique is the use of multi-modal transport chains. The use of heterogeneous transport modes in a supply chain allows the logistics operator to combine their different strengths. One of the current developments in multi-modal transportation is the idea of a two-tier system as introduced by Crainic et al. (2009). This is an extension of the original concept and uses transportation modes like trucks and trains to transport the goods to major facilities located at city limits. These goods are then redistributed with the use of more flexible modes, like cars to smaller strategically placed satellite facilities within city bounds. Finally, last mile delivery is performed with highly flexible and economically friendly transportation modes such as cargo bikes (Wrighton and Reiter, 2016).

1.2 The Vehicle Routing Problem with Load Dependent Travel Times

The last mile delivery phase of the previously described two-tier model is mathematically equivalent to the well researched Vehicle Routing Problem (VRP) first introduced by Dantzig and Ramser (1959) as the "Truck Dispatching Problem". A solution to this problem answers the question: Which route should every vehicle take to serve all customers with minimal effort? Traditionally, it is assumed that all vehicles start from one designated depot and have a maximum capacity they can carry. However, several relaxations of additional factors, like traffic or demand uncertainty, limit the application of the VRP in realistic scenarios.

The simplicity of the traditional VRP results in a lack of applicability, provoking a wide range of problem extensions (Lin et al., 2014). Chapter 3 reviews the most important extensions related to this thesis in more detail. One of the core limitations of the traditional VRP is that customers commonly have preferences in which time frame they receive their goods. This became increasingly relevant with the rise of just-in-time (JIT) production and customer scheduled delivery in retail. We adopt this requirement, because it fits into our use-case of last mile delivery. An additional core assumption of the VRP is that all vehicles travel with constant speed and ignores that the speed may depend on the time of day, the weight of the load or the gradient of a route. With the use of cargo bikes as the main means of transport, however, these dependencies should not be neglected.

This was the main motivation for Fontaine (2019) to introduce the Vehicle Routing Problem with Load-Dependent Travel Times (VRPLDTT). He discovered that ignoring load dependencies can lead to situations where time windows are missed, implicating a breach of the service level agreement on the side of the forwarding agent. Furthermore, the consideration of this dependency can decrease the total travel time of last mile delivery by up to 23% if cargo bikes are used as transportation modes.

1.3 Contributions of this thesis

Optimally solving variants of the VRP is often often deemed too slow and expensive in a realistic scenario for this approach to be feasible. This is why the number of peer reviewed articles about heuristic approaches outnumbered papers on exact solutions between 2009 and 2016 in a ratio of 5:1 (Braekers et al., 2016). The VRPLDTT is no exception to that. Motivated by the combinatoric explosion in complexity of exact approaches, this thesis aims to introduce a viable meta-heuristic alternative to existing general use Mixed-Integer Program (MIP) solvers. The final algorithm is based on the well known Adaptive Large Neighborhood Search (ALNS) framework.

Additionally, we introduce new aspects to the ALNS that (to the best of our knowledge) have never been applied to VRPs. Most importantly, the implementation in this thesis incorporates temperature dependent shake-up, purposeful diversification and execution time normalized operator weighting. The only standardized VRPLDTT problem instances available have been introduced by Fontaine (2019) and are targeted towards exact approaches. This is why we also introduce new problem instances with up to 200 customers. Our aim is to provide a basis for future comprehensive performance benchmarking.

The thesis starts by defining the VRPLDTT in Chapter 2, accompanied by a discussion about its complexity and the resulting relevance of a meta-heuristic approach to solve it. Afterwards, the most relevant literature regarding the ALNS and VRPLDTT is reviewed in Chapter 3. Our applied methodology will be detailed in Chapter 4, which further includes a description of regression-tree based parameter tuning technique. The proposed heuristic is finally tested and benchmarked in an extensive computational analysis, accompanied by a discussion about the behavior, strengths and weaknesses.

2 Problem Definition

After introducing the VRPLDTT intuitively in Section 1.2, this chapter defines it comprehensively. We first define all parameters of the VRPLDTT and describe the linear formulation afterwards. This is followed by an enumeration of all assumptions on cyclists, vehicles and physical constants used to compute the load dependent travel speed.

2.1 Problem formulation

The VRPLDTT is an optimization problem, implying that one or more objectives are either minimized or maximized. Let $\acute{s}$ be the optimal solution defined by a set of routes in the solution space S. The costs associated with solution s is $f(s)$. Traditionally, VRPs are minimization issues with the goal to:

$$\text{Find } \acute{s} \in S \text{ such that } f(\acute{s}) \leqslant f(s) \quad \forall s \in S. \tag{2.1}$$

Common cost functions in VRPs are the total route length, the number of vehicles used to serve all customers or a custom cost function incorporating fixed and variable vehicle costs. The VRPLDTT the minimizes total travel time T.

2.1.1 Problem parameters

A problem instance of the VRPLDTT is defined as a graph $G(N, A)$ where $N = \{0, ..., n\}$ is the set of nodes and $A = \{(0, 0), ..., (i, j)\}$ the set of arcs connecting each node $i, j \in N$. Each node can be either a depot or customer. The VRPLDTT assumes one central depot $N_0 = \{0\}$ thus declaring the set of customers as $N_1 = \{1, ..., n\}$.

Each customer i is associated with a non-negative demand q_i and the time window $[a_i, b_i]$ in which he must be served. A customer i can be visited at earliest on a_i and at latest on b_i. Serving a customer i requires s_i time units.

Every customer must be served by a vehicle $k \in K = \{0, ..., m\}$ which is limited by its capacity Q_k. The VRPLDTT assumes a homogeneous fleet thus resulting in $Q_k = Q \quad \forall k \in K$. A vehicle can travel the distance d_{ij} between two nodes i and j in t_{ij}^l time units. This is dependent on the current load $l \in L$ of the vehicle.

2.1.2 Mixed integer formulation

We will use the deterministic, discrete and linear arc-flow based formulation proposed in Fontaine (2019) to define the objective and constraints of the VRPLDTT.

This formulation assumes discrete load buckets $l \in L$, each comprising a load interval $[p^l, r^l]$ where p^1 = 0 and $r^{|L|}$ = Q. The size of each bucket can be set to a precision value that is fitting for the problem at hand. A naive way of bucketing could be based on weight units like gram or kilogram.

The binary decision variable x_{ij} is set to 1 if a vehicle travels on an arc (i, j), else it is set to 0. The continuous decision variables y_i and f_{ij} determine the arrival time of a vehicle for all customers i and the load of a vehicle when traveling from node i to node j, respectively. All travel times t^l_{ij} are mapped by fitting the binary decision variable z^l_{ij} to 1, if the load level l is used on arc (i, j). Furthermore, a lower bound M_{ij} defines the earliest possible start time when travelling from node i to j, in order to reduce computational complexity.

Appendix A offers a comprehensive summary of the complete notation. The linear model using the Fontaine (2019) notation is defined and described in the following:

Objective Function

$$\min \sum_{(i,j)\in A} \sum_{l\in L} t^l_{ij} z^l_{ij} \tag{2.2}$$

Constraints

$$\sum_{j\in N} x_{0j} \leq m \tag{2.3}$$

$$\sum_{j\in N} x_{ij} = 1 \qquad \forall i \in N_1 \tag{2.4}$$

$$\sum_{i\in N} x_{ij} = 1 \qquad \forall j \in N_1 \tag{2.5}$$

$$\sum_{j\in N} f_{ji} - \sum_{i\in N} f_{ij} = q_i \qquad \forall i \in N_1 \tag{2.6}$$

$$q_j x_{ij} \leq f_{ij} \leq (Q - q_i) x_{ij} \qquad \forall (i, j) \in A \tag{2.7}$$

$$y_i - y_j + s_i + \sum_{l\in L} t^l_{ij} z^l_{ij} \leq M_{ij} x_{ij} \qquad \forall i \in N, j \in N_1, i \neq j \tag{2.8}$$

$$a_i \leq y_i \leq b_i \qquad \forall i \in N_1 \tag{2.9}$$

$$\sum_{l\in L} z^l_{ij} = x_{ij} \qquad \forall (i, j) \in A \tag{2.10}$$

$$\sum_{l\in L} p^l z^l_{ij} \leq f_{ij} \leq \sum_{l\in L} r^l z^l_{ij} \qquad \forall (i, j) \in A \tag{2.11}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A \tag{2.12}$$

$$f_{ij} \geq 0 \qquad \forall (i, j) \in A \tag{2.13}$$

$$z^l_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A, l \in L \tag{2.14}$$

$$m \in \mathbb{N}_0 \tag{2.15}$$

The objective is described in (2.2) and minimizes the sum of all vehicle travel times. Please note that all potential waiting or service times are disregarded. Constraint (2.3) ensures that the maximum number of vehicles m cannot be exceeded. Constraints (2.4) and (2.5) enforce that each customer is visited exactly once and that the depot has to be the final destination of a route. Each customer demand must be satisfied, therefore, if a customer i was visited by a vehicle, its load must decrease by q_i. This is done in the flow balancing constraint (2.6). The constraint (2.7) guarantees that no flow along an arc (i, j) can ever exceed the maximum capacity Q of the homogeneous vehicles. The arrival time at a customer y_i is set by (2.8). Fontaine (2019) suggests a smart lower bound of $M_{ij} = \max\{0, b_i + s_i + t^1_{ij} - a_j\}$. The arrival time y_i is limited by the time window, which is modeled in constraint (2.9). Constraints (2.10) and (2.11) activate the correct load level for each arc with a flow bigger than 0, which implicates two things. First, the flow is bigger than 0 if a vehicle travels along an arc. Secondly, the vehicle must travel in a speed corresponding the current load on the arc. Finally, the bounds of all decision variables are declared in (2.12) - (2.15).

2.2 Computing the load dependent travel times

The main contribution of the VRPLDTT is acknowledging that load and slope of a sub-tour will impact the travel speed required by a vehicle. This effect might be negligible for cars and trucks, but is considerable when using cargo bikes. Fontaine (2019) showed that it severely impacts quality and feasibility of time window constrained problem instances.

The speed of a cyclist depends on the power input, and the forces acting on both cyclist and bike. This chapter names the taken assumptions on both components. All assumptions on the speed calculation are adopted from Fontaine (2019) in full. Please refer to his article for an in-depth analysis of the effect of certain parameter values on the relevance of the formulation.

2.2.1 Power input

Similarly to Fontaine (2019), we assume that every cargo bike is equipped with an electrical motor, which can supply additional power of 250 W. Additionally, the European Union commands that the power surplus must stop after 25 km/h is surpassed. Differing specifications require a dedicated drivers licence and conclusively are ignored in this thesis. Fontaine (2019) also states that a well trained cyclist is able to generate 100 W for a longer period of time. The total available power to transport goods on a cargo bike therefore sums up to 350 W.

2.2.2 Power consumption

Parameters and assumptions

We are interested in the resulting maximum speed, given the available power input of 350 W. The maximum speed is reached if the power input is in equilibrium to the forces acting upon the cyclist and his bike. Bikes at Work (2012) declare that a cyclist faces four main forces: The air resistance (F_D), the rolling resistance (F_R), the gravity given positive slope (F_G) and frictional force (F_F). All additional forces are assumed to be negligible. For simplicity, friction is fixed to be 5% of the total power input.

The total force acting upon the cargo bike is dependent on the velocity v of the cyclist, slope h of the street and mass of vehicle m and . The mass of a vehicle itself is composed of the mass of the cyclist ($m \approx 80$ kg), the bicycle ($m \approx 60$ kg) and the load carried ($m \in [0, ..., 150]$ kg).

For simplification reasons we assume that the arc (i, j) can be travelled on a straight line with constant slope. This enables us to compute the slope h through the difference of elevation e_{ij} and the distance d_{ij} of two nodes i and j (Equation 2.16).

$$h_{ij} = \frac{e_{ij}}{\sqrt{d_{ij}^2 - e_{ij}^2}} \tag{2.16}$$

It is important to note that the use of real routing results can lead to differing distance values for d_{ij} and d_{ji}. Therefore, the distance matrix describing the lengths d_{ij} of all arcs $(i, j) \in A$ is not necessarily symmetric. Additionally, all values h_{ij} of the slope matrix must abide to $h_{i,j} = -h_{j,i}$. These properties implicate that $t_{ij}^{\acute{l}}$ does not have to be equal to $t_{ji}^{\acute{l}}$ for a given load level $\acute{l}$.

Resistances

The three remaining main forces F_D, F_R and F_G can be formulated as follows:

$$\text{Air resistance:} \quad F_D = \frac{\rho\, C_d\, A}{2} v^2 \tag{2.17}$$

$$\text{Rolling resistance:} \quad F_R = C_r\, m\, g\, cos(\arctan(h)) \tag{2.18}$$

$$\text{Gravity resistance:} \quad F_G = m\, g\, sin(\arctan(h)) \tag{2.19}$$

with the air density being $\rho = 1.18$, the coefficient of drag $C_d = 1.18$, the cross-sectional front area of the bicycle $A = 0.83m^2$, the coefficient of rolling resistance $C_r = 0.01$ and the gravity acceleration constant $g = 9.81m/s$.

Please consult Fontaine (2019) for additional information about these physical constants and their value settings.

2.2.3 Velocity calculation

The previous chapter states that it is possible to describe the velocity v for a given power input P in the equilibrium of power input and total resistance as follows:

$$P = (F_D + F_R + F_G)v/0.95 \tag{2.20}$$

This equation can be solved for the maximum speed v, given values for P, m and h. In theory it is possible that a cargo bike reaches a velocity faster than 25 km/h. However, we cap the maximum speed at 25 km/h, because of the concerns that a faster speed might cause steering issues for a cargo bike.

2.3 Complexity of the VRPLDTT

We finalize the problem description by reasoning about the complexity of the VRPLDTT and the resulting implications on the solution approach. The widespread $\mathcal{O}$ notation is used to quantify the worst case complexity of an algorithm throughout this thesis (Cormen et al., 2009). This merely describes the number of atomic steps needed for an algorithm to terminate at the worst case. The final execution time additionally depends on the time of one atomic step, which can be influenced by both hardware and implementation.

Determining the mathematical worst case complexity of a problem is crucial in operations research. Knowing the complexity of an optimal solution to a problem allows researchers to weight the importance of heuristic alternatives. However, determining the exact $\mathcal{O}$ complexity of a problem is often difficult. Therefore, a common approach is, to attribute problems into the complexity classes P or NP (Cormen et al., 2009).

Let n be an arbitrary dependent and k an arbitrary independent variable. All problems in the complexity class P can be solved in at most $\mathcal{O}(n^k)$ steps. In other words: Problems in P are *solvable* in polynomial time. Problems in NP, however, are only *verifiable* in polynomial time. A problem is called NP-hard if it is *at least as hard as the hardest problem in* NP and NP-complete when it belongs to both the NP, and $NP - hard$ complexity class. One of the biggest mathematical question of our time is whether $P = NP$. Proving this equation would implicate that all problems, which are *verifiable* in polynomial time, are also *solvable* in polynomial time. At the time of this thesis no conclusive mathematical proof for either $P = NP$ or $P \neq NP$ exists. However, it is widely expected that $P \neq NP$, because of the large number of NP-complete problems and the lack of success to reduce any to a problem in P (Cormen et al., 2009). Therefore, we currently assume that every NP-complete problem is only solvable with exponential-time algorithms.

Exponential worst-time complexity scaling of an algorithm realistically limits the

maximum feasible problem size. Even the currently polynomially scaling computational power (Moore's law) is not sufficient to counteract this problem. Figure 2.1 visualizes the improvement of computational power given Moore's law (gray line) and compares it to a exponential scaling curve with a basis of only 1.06 (black line). We can observe that computational improvement might speed up smaller instances significantly, but offers limited benefit after exceeding a threshold n.

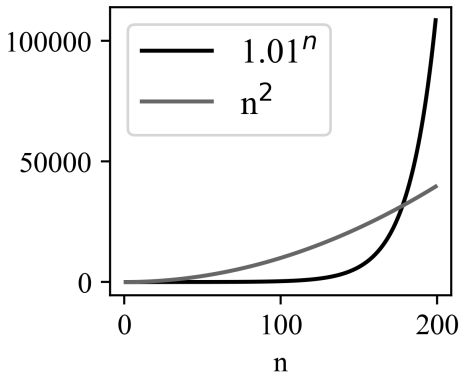


Figure 2.1: Example of polynomial and exponential scaling functions

An important property of NP-complete problems is that each problem can be reduced to any other problem in NP (Karp, 1972). Conclusively, a problem is guaranteed to be NP-complete if it can be reduced to a different NP- complete problem. This method is also known as proof by reduction (Cormen et al., 2009).

We refer to Kallehauge (2008) and their proof that the Vehicle Routing Problem with Time Windows (VRPTW) is NP-complete in the strong sense. Relaxing the load dependency $l \in L$ in the VRPLDTT will fix the vehicle load, thus implicating a fixed vehicle speed as well. The set indicator l looses its purpose and enables a transformation of t^l_{ij} into t_{ij}. Therefore, travel time t is now directly proportional to the distance. This simple relaxation reduces the linear model proposed Fontaine (2019) to a valid arc-based formulation of the VRPTW as described in Kallehauge (2008). Conclusively, the VRPLDTT is NP-complete and thus as "hard" to solve.

Nevertheless, this classification only declares the scaling of computational complexity. Practical applicability of exact solution approaches depends on problem size of realistic use cases as well. An empirical indicator of the speed of general purpose solvers is the computational study by Fontaine (2019). The therein proposed maximum instance size of 20 customers raises doubts on the applicability of exact approaches in realistic scenarios. We argue that this validates the use of heuristics as necessary and relevant.

3 Review of Literature and Research

The VRPLDTT is part of the family of vehicle routing problems. Laporte (2009) distinguishes in the of research solution methods between exact methods and heuristics. Exact methods solve the problem at hand to optimality while heuristics merely return feasible (preferably good) solutions. We discussed in Section 2.3 why exact algorithms are of secondary interest in this thesis and will focus on a heuristic approach. For details on exact methods, the interested reader might refer to Toth and Vigo (2014).

The term heuristic is derived from the Greek word *heuriskein* ($\epsilon\upsilon\rho\iota\sigma\kappa\epsilon\iota\nu$ - "to find") and refers to the ability to find a good solution by strategically searching the solution space. Heuristics are attributed with the prefix *Meta* ($\mu\epsilon\tau\acute{\alpha}$ - "beyond, in an upper level") if the basic procedure is context independently applicable (Bianchi et al., 2009). Siarry (2016) provides a comprehensive overview of traditional meta-heuristic approaches and their core concepts. Please refer to Cuevas et al. (2018) for a summary of current developments, where we observe a common trend towards population based cooperative behaviour. However, most of these frameworks have not been applied in the context of VRP yet. The meta-heuristic framework used in this thesis is ALNS.

The following chapter begins with an integration of the VRPLDTT into the existing literature and continues with a research summary describing the development of the ALNS. Finally, applications of the ALNS to the VRP are reviewed.

3.1 The VRPLDTT and similar problems

Chapter 1 and 2 describe the VRPLDTT as a problem extension of the VRPTW introduced by Fontaine (2019). Its base assumption is the load and slope dependency of vehicle travel times. This formulation is specifically aimed at applications in urban logistics, where cargo bikes are regarded as an important alternative transportation mode.

The idea of dynamic travel times in VRPs has been researched by multiple authors previous to the introduction of the VRPLDTT, for example in the time-dependent VRP (TDVRP). It assumes that the vehicle speed depends on the time of day in order to model high traffic periods and traffic congestion (Malandraki and Daskin, 1992; Jabali et al., 2012; Huang et al., 2017).

Load dependency in VRPs is also not a new concept and even the traditional capacitated VRP already constrains the maximum capacity of vehicles. Several models additionally incorporate load dependencies into their objective function. Zachariadis et al. (2015), for example, employs the load-dependent VRP (LDVRP) to minimize the product of distance travelled and the gross weight carried along this distance. The most popular VRP variant using a load dependent objective, however, is emission routing (Suzuki, 2011). Its goal is to minimize vehicle emissions based on a load dependent emission function. This can be extended to pollution routing, by additionally incorporating time windows (Bektaş and Laporte, 2011). Most recently Liu and Jiang (2018) introduced a new application of the pollution routing, which minimizes load dependent costs in a toll-by-weight scheme applied in China.

A small subset of publications incorporate both dynamic travel times and load dependencies. Ehmke et al. (2016) minimize the time-dependent emission problem and Franceschetti et al. (2017) tackles the time-dependent pollution routing problem.

To the best of our knowledge Fontaine (2019) is the first to introduce load-dependent travel times into the family of VRPs. Additionally, we know of no author that introduced an exact or heuristic solution approach dedicated towards the VRPLDTT.

3.2 Adaptive Large Neighborhood Search (ALNS) in vehicle routing problems

Laporte (2009) distinguishes meta-heuristic frameworks into local search, population search and learning mechanisms. The heuristic implemented in this thesis is based on the ALNS framework, which can be attributed to the first. This section introduces all core concepts and components building the foundation of ALNS and closes with a review of the current state of ALNS in the research of VRPs.

3.2.1 Neighborhoods

A local search procedure starts from an initial solution s and searches for an improving, neighboring solution $\acute{s}$. Let N be a function that maps a solution to a set of other solutions $N(s)$, called *neighborhood*. Multiple neighborhoods can be defined for a given solution s. A neighborhood is defined by a set of *moves* (Labadie et al., 2016), which are selected through an operator. A solution is called *globally optimal* if it fulfills the condition of formula 2.1 introduced in section 2.1. We can also apply the optimality paradigm to $N(s)$ to find solutions which are globally optimal in $N(s)$, but might only be *locally optimal* in the set of all solutions S (Pisinger and Ropke, 2010).

The concept of neighborhoods is visualized in Figure 3.1 which shows a solution space for a single vehicle visiting three customers $N_1 \in \{1, 2, 3\}$. Each solution is represented by the order in which the vehicle visits the customers. Let $s = [1\ 2\ 3]$ serve as the current solution. A neighborhood is constituted by removing a customer from its current position. Reinserting the customer into the solution represents a move. The visualization shows three possible neighborhoods depending on the removed customer. We can also observe that neighborhoods can overlap, because different moves might lead to identical solutions. The final important observation is that there might be solutions that are not included in any neighborhood, given current position and neighborhood function.

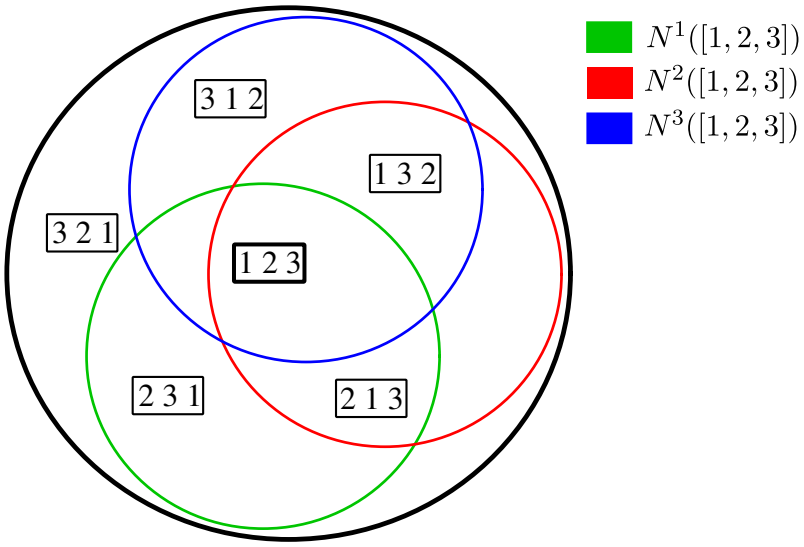


Figure 3.1: Example solution space for one vehicle visiting three customers.

3.2.2 Neighborhood search

The default approach of local search is to iteratively generate neighborhoods $N(s)$ and move from the previous local optimum s to the new local optimum $\acute{s}$ of $N(s)$. Traditionally, a *steepest descent* approach is employed, accepting $\acute{s}$ as s if the cost of a new found local optimum $\acute{s}$ is better than the currently accepted solution s. The goal is to find the local optimum $\acute{s} \equiv \hat{s}$ where $\hat{s}$ is the global optimum in S. After the stoppage criterion is met, which is often a fixed number of iterations, the search procedure terminates. However, there is no guarantee that the search for $\acute{s} \equiv \hat{s}$ was successful.

It is important to create neighborhoods that have a high chance of finding a $\acute{s} \neq s$ to avoid stagnation. Therefore, the most important success factor in local search is the neighborhood function generating all possible $N(s)$. A poorly defined neighborhood function might limit the subspace of solutions $\hat{S}$ that can ever be reached. Let $f(s)$ be a function returning the quality of a solution. The pseudocode of a rudimentary local search implementation is represented in Algorithm 1.

Algorithm 1 General local search procedure

```
1: function LOCALSEARCH()
2:     s ← initSolution()

3:     while stop != True do
4:         ś ← move(s)
5:         if f(ś) < f(s) then
6:             s ← ś
7:         end if
8:         stop ← checkStoppage()
9:     end while
10: end function
```

3.2.3 Large Neighborhood Search (LNS)

A common problem of the local search approach is that it reaches a local optimum $\acute{s} \not\equiv \hat{s}$, which is the global optimum for all $N(s)$. This state is commonly referred to as *trapped*.

An example of a trapped state can be derived from the visualization in Figure 3.1, where (f([1 2 3]) $\leq s$) $\forall\ s\ \in$ {[3 1 2], [1 3 2], [2 1 3], [2 3 1]} and f([3 2 1]) $<$ f([1 2 3]). Under the stated model assumptions, it would be impossible to find the global optimum [2 3 1].

Shaw (1998) was the first to introduce the concept of *large neighborhoods*, which are defined as not polynomial in n, with n being the number of customers (Pisinger and Ropke, 2010). Large neighborhoods often ensue the condition:

$$\bigcup_{N(s)\in O(s)} \equiv S \qquad \forall s \in S \tag{3.1}$$

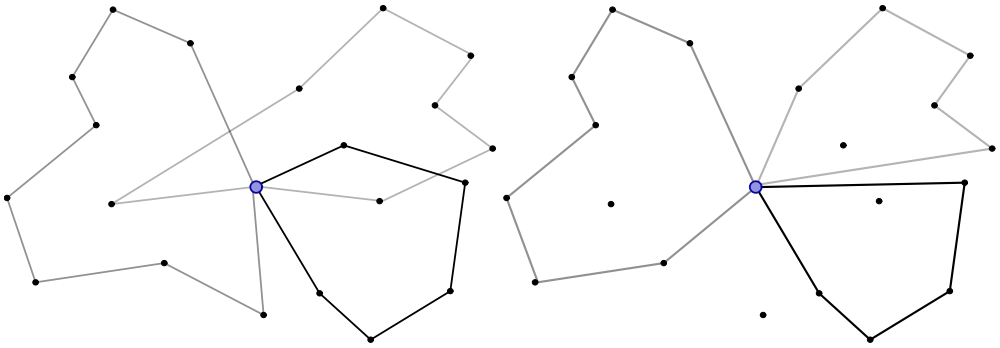
where $O(s)$ is the set of neighborhoods given s. In other words: Every existing solution can be reached from any position in the solution space, thus avoiding a trapped state. Traditionally, these neighborhoods are not explored in full because of their complexity.

Our previously introduced example in Figure 3.1 uses the *destroy and recreate* principle to generate neighborhoods. Its basic idea is that a neighborhood $N(s)$ is implicitly defined by partially destroying a solution. This neighborhood then consists of all solutions that can be created after reinserting all removed customers. The Large Neighborhood Search (LNS) proposed by Shaw (1998) extends on this idea by removing multiple customers from the initial state. Neighborhood complexity increases exponentially with every additionally removed customer, thus qualifying as "large". It is easily derivable that the criterion 3.1 is fulfilled, because $N(s) \equiv S$ when the number of removed customers is n.

A destroy and recreate iteration with multiple removed customers is visualized in Figure 3.2. Each of the three sub-figures show a differing solution state for the same problem instance, corresponding to the three stages: Initial, after destruction and after reparation. Every dot represents a node, wherein the blue dot is the depot.

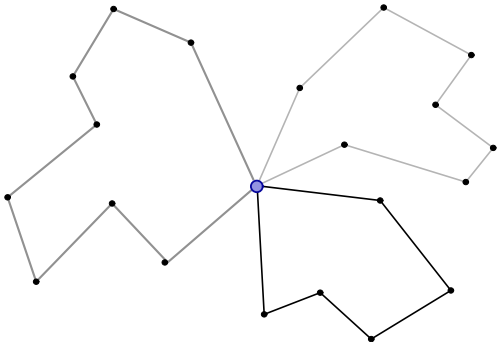
Recreating the solution to optimality is equivalent to searching the space of possible reparations. Finding the best solution within the neighborhood can also be interpreted as a separated optimization problem. Shaw (1998) employs a constraint programming approach to solve this sub-problem to optimality in every iteration. He additionally introduces a hybrid destruction operator to generate neighborhoods.

Optimally solving the resulting sub-problem ensures that $f(s_i) \leq f(s_{i+1})$. However, the side effect of searching in large neighborhoods is that the complexity of one iteration is significantly larger. Conclusively, the number of iterations is less compared to traditionally local search procedures. One alternative to optimal reparations first adopted by Ropke (2003) is to use heuristic methods. The use of reparation heuristics implicates that the condition $f(s_i) \leq f(s_{i+1})$ is not guaranteed.



(a) Initial state

(b) Destroyed state



(c) Repaired state

Figure 3.2: Visualization of the destroy and recreate principle

3.2.4 Adaptive Large Neighborhood Search (ALNS)

Ropke and Pisinger (2006) refined the LNS in several ways. They continued the use of heuristic repair algorithms and replaced the steepest decent with a Simulated Annealing (SA) search framework. In addition, multiple repair and destroy operators are combined in a roulette wheel mechanism, which incorporates adaptive weights. The weighting scheme benefits the selection of operators that yield improving results. Summarized, their underlying hypothesis is that a combination of multiple operators avoids cases where $N(s)_{i+1} \subseteq N(s)_i$ while prioritizing movements into promising neighborhoods $N(s) \in S$.

The ALNS meta-heuristic serves as baseline for the algorithm implemented in this thesis. All core concepts like the SA framework and the wheel mechanism will be detailed in Chapter 4.

3.2.5 Current state of ALNS in VRPs

The ALNS has already been applied on a broad range of routing problems, in spite its relative recent development. Ropke and Pisinger originally introduced the ALNS on the Pick-up and Delivery Problem with Time Windows (PDPTW), but later expanded the concept to a variety of VRPs including the VRPTW (Pisinger and Ropke, 2007).

Application on new domains

Applications of the ALNS on other routing problem variants include Adulyasak et al. (2012) on the Production Routing Problem (PRP), Masson et al. (2013) on the Pick-up and Delivery Problem with Transfers (PDPT), Hemmelmayr et al. (2012) on the Two-Echelon Vehicle Routing Problem (2E-VRP) and Aksen et al. (2014) on a Selective and Periodic Inventory Routing Problem (SPIRP). These implementations focus mainly on domain specific adaptations of operators and operator selection.

The ALNS has also been successfully applied to problems with dynamic travel times and load dependency as discussed in Section 3.1. Demir et al. (2012) implemented the meta-heuristic framework on the problem of Pollution Routing in a two-stage approach, first solving a VRPTW and applying a speed-optimization routine retroactively. A similar approach was applied by Franceschetti et al. (2017) on the Time-Dependent Pollution Routing Problem (TDPRP).

Framework adjustments

Laporte et al. (2010) applied the ALNS on the capacitated arc-routing problem with stochastic demands (CARPSD) and incorporated the record-to-record algorithm as

search framework. Additionally, new operators were introduced and infeasibility was managed adaptively. Coelho et al. (2012) applied a variant of the ALNS on the Inventory-Routing Problem with Transshipments (IRPT) and adapted the framework so that only one operator (destroy or repair) is applied in each iteration. This was only possible because the consideration of transshipments guaranteed feasibility even if a sub-set of customers was not considered in a solution.

Influence of previous applications on this thesis

Our implementation also incorporates new domain specific adjustments like the specialized operator hybrid insertion suggested by Franceschetti et al. (2017) and an improved weighting mechanism explicitly encouraging diversification. One of the biggest shortcomings of the roulette wheel mechanism is that slow but high quality repair heuristics are inherently favored over faster alternatives. Therefore, our operator rewards are normalized on iteration time instead of iteration count. Another key difference to existing ALNS applications is that we allow a degree of infeasibility and manage the penalty weights by an adaptive diversity management mechanism, similar to Laporte et al. (2010) or Vidal et al. (2013). Finally, we introduce a new component that we call *incremental shake-up*, which adjusts the average number of removed customers in each iteration based on the number of iterations without improvements.

4 Methodology

The VRPLDTT is solved with a heuristic approach, adapting the ALNS framework introduced by Ropke and Pisinger (2006). Section 3.2 explained the development and general concept behind this meta-heuristic, which this chapter builds on to provide an indepth explanation of our implementation.

Every ALNS variant centers around a local search framework to guide the *destroy and recreate* principle. Ropke and Pisinger (2006) proposed Tabu Search or Guided Local Search as suitable candidates, but finally opted for SA. This is currently the most popular local search framework for ALNS, which is also why we applied it in this thesis. Algorithm 2 depicts the pseudocode of the ALNS adaptation employed in this thesis. We term this implementation as Adaptive Large Neighborhood Search with Load Dependent Travel Times (ALNSLDTT) and repeatedly reference to specific code-lines in squared brackets throughout this chapter (e.g. the first line as [1]).

First, all components must be initialized to their user defined parameter values [2-4]. SA is an improvement heuristic and thus requires an initial solution to iteratively optimize. This initial solution is then repeatedly destroyed and repaired until the stoppage criterion is reached [5-36]. The next section will explain the SA specific components, specifically the adopted acceptance and stoppage criteria, in detail.

All other main components of the ALNS are then explained throughout the rest of the chapter. This includes in particular the search space (Section 4.2), solution initialization (Section 4.4), all destroy operators (Section 4.5), repair operators (Section 4.6) and the roulette wheel mechanism (Section 4.7). The chapter ends with a description of the employed CART parameter tuning technique.

4.1 Simulated Annealing (SA)

Simulated Annealing (SA) is a variant of the Metropolis-Hastings algorithm and based of the intuition on the physical annealing process in metallurgy. It incorporates the logic of steepest descent but extends it with the potential to accept a solution by chance. Let $\Delta F(s, \acute{s})$ be the quality difference of two solutions defined as $\Delta F(s, \acute{s})$ = f($\acute{s}$) - f(s). The probability of randomly accepting a worse solution decreases with iteration count and increasing $\Delta F(s, \acute{s})$. This initially guidelines the algorithm to adjust its sampling distribution to correspond to the quality of all

Algorithm 2 ALNSLDTT

function ALNS($T, c, It_{NI}, \acute{z}$)
  Randomly initialize s ▷ [1] Initialize components
  Initialized destroy and repair wheels W^-, W^+

  $time_s, it_{NI}, L \leftarrow$ now(), 1, T, []
  while (now() - $time_s$) > $\acute{z}$ **or** $it_{NI} > It_{NI}$ **do**

    Choose random destroy operator N^- from W^-
    Choose random repair operator N^+ from W^+
    Create $\acute{s}$ by applying N^- and N^+

    $\sigma_s \leftarrow 0$ ▷ [2.2] Evaluate acceptance
    if f($\acute{s}$) < $f(\hat{s})$ **and** $\acute{s}$ is feasible **then** ▷ New best solution
      s, $\hat{s} \leftarrow \acute{s}$
      $it_{NI} \leftarrow 1$
      $mr \leftarrow \lceil log_\lambda n \rceil$
      $\sigma_s \leftarrow (\sigma_s + \sigma_1)$
    else ▷ Evaluate current solution
      $it_{NI} + +$
      $mr \leftarrow \lceil log_\varsigma it_{NI}\ log_\lambda n \rceil$

      if $f(\acute{s}) < fs$ **then**
        $\sigma_s \leftarrow (\sigma_s + \sigma_2)$
      else ▷ No new current solution
        $\sigma_s \leftarrow (\sigma_s + \sigma_3)$
      end if

      $\nu \leftarrow e^{-\frac{f(\acute{s})-f(s)}{t}}$
      if rnd() < ν **then**
        $s \leftarrow \acute{s}$
      end if
    end if

    if $\acute{s}$ **not in** L **then** ▷ [2.2] Evaluate diversity
      L.append($\acute{s}$)
      $\sigma_s \leftarrow (\sigma_s + \sigma_4)$
    end if
    $\sigma_s \leftarrow (\sigma_s$ + ν getDiversity($\acute{s}$) σ_5)

    Normalize σ_s and update weights of W^- and W^+
    Update penalty weights for infeasibility
    T $\leftarrow$ T * c
  end while
  return $\hat{s}$
end function

solution regions. The goal is to find the most promising region and with increasing iteration count explore it in-depth.

4.1.1 Acceptance criterion

The SA is initialized with temperature T [4], which is iteratively discounted by a cooling rate of $c < 1$ [35]. In each iteration a new solution $\acute{s}$ is created in that both a destroy and repair operator are applied to s [6-8]. The operator selection mechanism adapts based on the performance of previous iterations and is detailed in Chapter 4.7.

A solution $\acute{s}$ is accepted if $\Delta F(s, \acute{s}) > 0$ [10] or with a probability of ν by chance ([24]). This can be formalized as follows:

$$\nu = \begin{cases} 1 & if \Delta F(s, \acute{s}) \geq 0 \\ exp\left(-\frac{f(\acute{s}) - f(s)}{t}\right) & if \Delta F(s, \acute{s}) < 0 \end{cases} \tag{4.1}$$

Tuning T and c is important to find a suitable trade-off between intensification and diversification. Both parameters should be adjusted based on problem characteristics and the expected number of iterations. Adjusting initial temperature T mainly defines the accepted degree of deviation $\Delta F(s, \acute{s})$ during local search. This value is highly problem specific and depends on the number of customers and average distance of an arc. The main responsibility of the cooling rate is to define the length of both diversification and intensification periods. Larger instances might require additional iterations for convergence and thus a smaller c.

Figure 4.1 visualizes the changing probability of accepting a solution by chance with increasing iteration count. The graphic shows four lines, each corresponding to a specific cooling rate, which implicitly defines the temperature T_i at a given iteration. We assume that $\Delta F(s, \acute{s})$ = 1 and T = 1. In the example of Figure 4.1, random acceptance given c = 0.999 is only in the first ∼3000 iterations realistic, then highly unlikely. Decreasing c prolongs the phase of randomized acceptance.

It is important to note that probabilistic acceptance relaxes the role of the destroy operator as the only component with trapping avoidance responsibility. However, it also relaxes the condition $f(s) < f(\acute{s})$ and implicates that s is not necessarily the best solution found after procedure termination. Therefore, it is important to track the best found solution separately in $\hat{s}$ [11]. We must also note that, in contrast to s and $\acute{s}$, $\hat{s}$ must obey constraints 2.2 - 2.15 at all times. Additional information on scenarios, where s and $\acute{s}$ might be permissible given a constraint relaxation, is provided in Section 4.2.

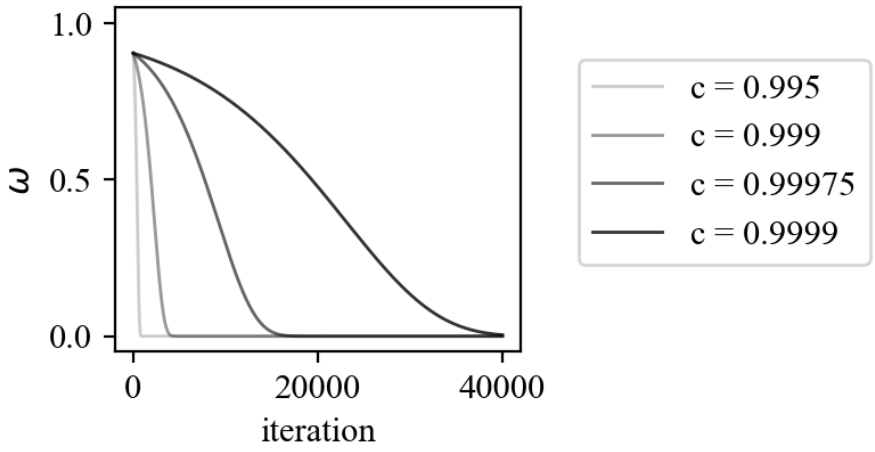


Figure 4.1: Changing acceptance probability of f($\acute{s}$) - f(s) = 1.

4.1.2 Stoppage criterion

The ALNS as introduced by Ropke and Pisinger (2006) stops the search procedure after a pre-determined number of iterations. Depending on the problem size, either 25 000 or 50 000 iterations have been suggested. Problem instances with more customers draw greater added value from additional iterations than instances with fewer customers. This lead Ropke and Pisinger (2006) to suggest a size based guideline of T = 20 000 + 50 $|N_1|$.

However, fixing the number of iterations preemptively has two apparent drawbacks. The first one is that a stagnant solution might be found but the algorithm cannot terminate prematurely. This is wasteful of time and resources. The second drawback is that it can be difficult to anticipate the exact time it takes for the ALNS to terminate, as this depends on hardware, operator selection and the problem instance. Therefore, the time to perform the defined number of iterations might exceed the time one is willing to invest into a solution.

These draw-backs motivate us to apply a two-fold stoppage logic [5]. The main stoppage criterion in our implementation is a fixed maximum number of iterations without improvement IT_{NI}. After the current number of iterations without improvement it_{NI} exceeds the set threshold IT_{NI}, a stagnant solution can be assumed and the search procedure should end. However, the algorithm terminates prematurely if a set execution time threshold $\acute{z}$ is exceeded and returns the best solution found until this point.

4.2 Search space

A solution is considered *infeasible* if one of the constraints 2.3-2.15 is not satisfied. Nevertheless, recent research shows that the relaxation of one or more constraints can simplify the search through the solution space by enabling short-cuts and new patterns (Glover and Hao, 2011). This in return can favor diversity and avoid trappings in local optima. Several highly competitive meta-heuristic implementations use this approach (Vidal et al., 2012; Laporte et al., 2010; Hemmelmayr et al., 2012).

Increasing the search space by enabling infeasible solutions during the local search is an extension of the original framework of Ropke and Pisinger (2006). To the best of our knowledge, in an ALNS this was first applied by Laporte et al. (2010). Typically optimal solutions for the VRPLDTT exist on the boundary of 2.7 and 2.8. Therefore, a relaxation of load and time-windows appears to be appropriate.

Let $r \in R(s)$ be the set of all routes within a solution. Each route starts from the depot N_0, visits a sequence of n_r customers $i \in N_1$ and returns to the depot again. The route r can be described by its total load $q(r) = \sum_{i \in r} q_i$ and total driving time $T(r) = \sum_{i \in r} l_i t^l_{ij}$.

Time-window infeasibility $\tau(r)$ of a route r is defined as:

$$\tau(r) = \sum_{i \in r} (\max\{0, a_i - y_i\} + \max\{0, y_i - b_i\}) \tag{4.2}$$

Infeasible solutions should be penalized in the quality evaluation $f(s)$ to disincentivize their frequency. Let ω^D be the penalty for exceeding vehicle capacity and ω^W the penalty for serving customers outside of their time window. The quality of a route r is defined as follows:

$$f(r) = T(r) + \omega^D \tau(r) + \omega^W \{0, q(r) - Q\} \tag{4.3}$$

The penalized objective function $f(s)$ is simply the sum of the penalized quality of all routes within a solution $f(s) = \sum_{r \in R(s)} f(r)$. It is important to note, however, that the global best solution $\hat{s}$ must be feasible at all times, because it will be returned in the event of premature termination [10].

4.2.1 Adaptive weights

A suitable value for ω^D and ω^W is highly problem specific and depends on the number of customers, average travel time within the network and the constriction of the problem. Defining penalty weights without prior knowledge of the problem is non-trivial. Additionally, the suitability of weights might also change during the local search procedure.

Adaptive weights can solve this problem by adjusting themselves during runtime.

Let ξ^{PAR} be the real percentage and ξ^{REF} the target percentage of infeasible solutions during the last k runs. Weights are adapted every k iterations if the number of infeasible solutions generated is not within 5% of the target percentage. The employed weight adjustment is described in Algorithm 3 and uses k = 100 [34]. It is important to note that the weight increasing and decreasing adjustment speeds differ slightly. This enables the procedure to approach an equilibrium in infinity, if one exists, and is based of the procedure introduced in Vidal et al. (2013).

Algorithm 3 Update penalty weights

```
1:  function UpdatePenaltyWeights(ξ^PAR)
2:      if k iterations passed then
3:          if ξ^PAR - ξ^REF ≥ 0.05 then
4:              ω^D ← 0.85 ω^D
5:              ω^W ← 0.85 ω^W
6:          else if ξ^PAR − ξ^REF ≤ −0.05 then
7:              ω^D ← 1.2 ω^D
8:              ω^W ← 1.2 ω^W
9:          end if
10:     end if
11: end function
```

4.2.2 Infeasibility limits

The use and pre-computation of discrete load levels l requires information about the maximum load $q(r) \leq Q + \max\{\tau(r)|r \in R(s)\}$ that can occur during local search. We use an upper bound U to limit the maximum allowed infeasibility so that $\tau(r) \leq U \; \forall r \in R(s)$. However, U must be large enough so that a construction heuristic is guaranteed to find a solution in the relaxed solution space, if it allocates a customer i to any route r that obeys $Q + U \leq q(r) + q_i$.

Three naive upper bounds fulfill this property, the first being $U = \sum_{i \in N_1} q_i$. This upper bound would enable any vehicle to serve every customer. However, depending on the problem instance, this might be inefficient and requires heavy pre-computation. A stricter upper bound $U = Q$ can be identified if one considers a problem fundamentally unsolvable, if a single customer requests more than vehicle capacity Q. Lastly, a solution in the relaxed solution space is guaranteed if $U = \max\{q_i|i \in N_1\}$. If a construction heuristic cannot find a solution fulfilling $q(r) \leq Q + max\{q_i|i \in N_q\}$), all vehicles exceed their maximum capacity. This means that $Q \leq q(r) \; \forall r \in R(s)$ and that the unrelaxed solution space is empty.

There is no obvious optimal bound U. A greater U enables a larger relaxed search space but requires more pre-processing, while a smaller U is computationally more efficient but might be restrictive in the search behavior. We decide for $U = \max\{q_i|i \in N_1\}$.

4.2.3 Size of destruction

The size of the search space within an iteration is defined by the number of removed customers. To avoid stagnation, this number of customers is randomly generated from a Gaussian distribution with mean value rm and standard deviation $\frac{rm}{2}$. This differs from other implementations, which draw uniformly from a value interval. Bounds of the interval in previous implementations are either linearly (e.g Ropke and Pisinger (2006)) or logarithmically dependent on the number of customers in an instance (Franceschetti et al. (2017)).

However, Ropke and Pisinger (2006) already acknowledged that the insertion heuristics are not sufficiently exhaustive to generate high quality results from heavily destroyed solutions. This implementation therefore adopts the logarithmic dependency on the number of customers and thus avoids high rm values for large scale instances.

Nonetheless, insufficient disruption might prohibit the local search to escape from local minima. A possible indicator can be an increasing number of iterations without improvement it_{NI}. First experiments have shown that the number of iterations necessary for intensification is often smaller than 500. This motivates the introduction of a new concept called *incremental shake-up*, which increases the average neighborhood size with increasing it_{NI} [17]. However, if the number of removed customers increases, the search space increases exponentially as well. A logarithmic adjustment seems therefore appropriate to adapt the number of search attempts to the size of the search space.

In conclusion, the average number of removed customers rm can be described as:

$$rm = \lceil log_{\varsigma} it_{NI} \; log_{\lambda} n \rceil \tag{4.4}$$

Parameters ς and λ are adjusted during the parameter tuning procedure and might also be problem specific.

4.3 Solution reevaluation

Removing and reinserting a customer requires frequent reevaluation of the quality function $f(s)$ described in the previous section. Operators frequently use the quality and feasibility status of a route to inform their decision policies. A recalculation of $f(s)$ after each decision is therefore often required.

The great number of reevaluations leads to the evaluation routine accounting for a significant proportion of the algorithm's total time. One of the ways to improve the efficiency is to minimize the number of recalculations to the necessary minimum.

Let $\acute{r}$ be an altered route in a solution $R(s)$. Adjusting route $\acute{r}$ can only change the

quality and feasibility status of $\acute{r}$, all other $r \in R(s) \setminus \acute{r}$ are independent and do not have to be reevaluated.

Adding a customer to a route implies that a vehicle has to pick up additional cargo at the depot and transport it through all stops to the new customer. Conversely, removing a customer relieves the additional load until the point of change. Both operations affect the weight of the vehicle for all edges (i, j) up to the point of change. By definition, this also leads to a change in the load dependent travel times t^l_{ij}. Additionally, changes to the route introduce a time-shift for all customers after the insertion position, caused by potential detours.

We exemplify the effect of both operations on the quality and feasibility of a route in Figure 4.2. It displays all changes in loads l, travel times t^l_{ij} and arrival times y_i in three sub-figures, similarly to Figure 3.2. Each sub-figure represents one of the states *initial*, *destroyed* and *repaired*. A vehicle starts at the depot (blue circle) and visits a set of customers $i \in r$ (white circles) until it returns to the depot. All blue values are deviations from the initial state and red values indicate infeasibility. The notation is described in the upper right corner. It is important to note that the formulation of the VRPLDTT does not consider waiting times in the objective, thus does not optimize on it. Therefore, we accept high waiting times and start each route at the earliest start time possible, that is $a_{r_1} - t^l_{0,r_1}$.

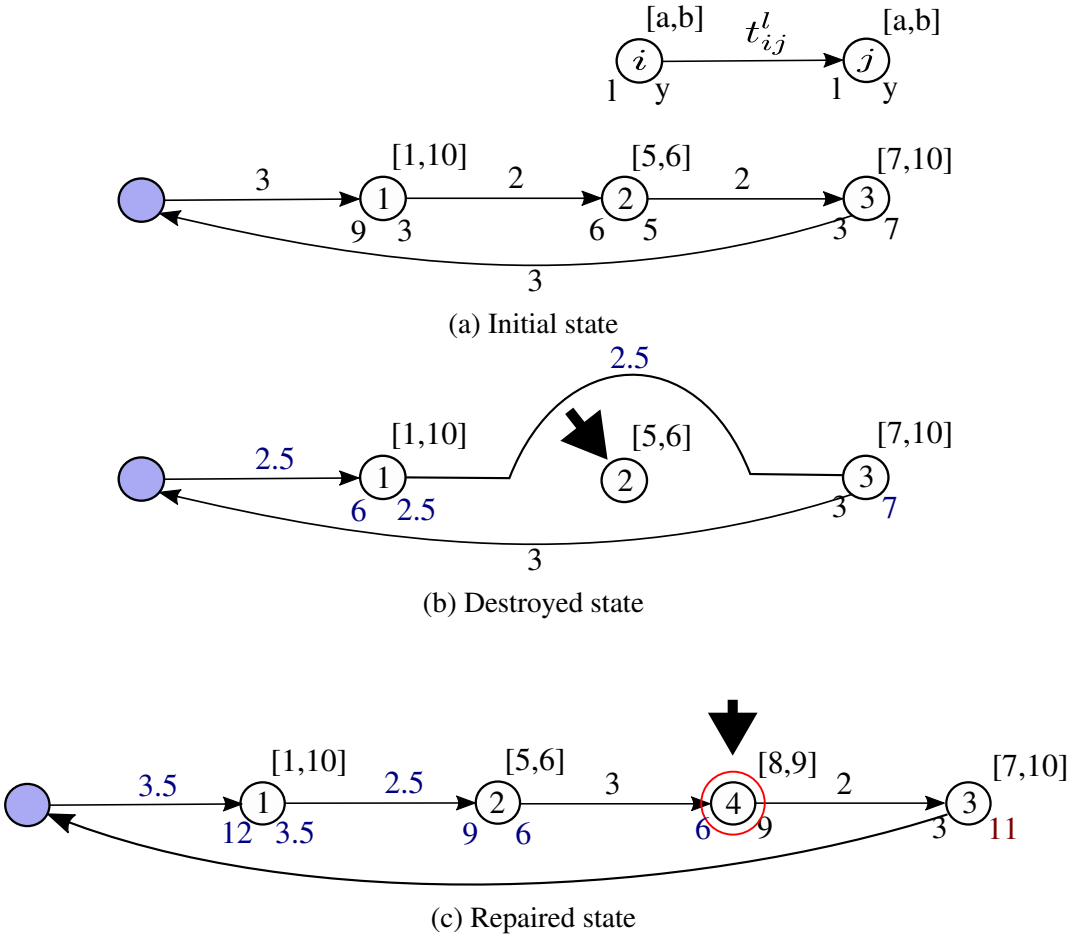


Figure 4.2: Visualization of re-computations and resulting value changes.

Requiring the re-computation of all y_i within a route represents a key difference to other VRP variants, in which y_i typically changes for all customers visited after the position of change. The described shift of arrival times additionally requires a revalidation of the solution feasibility.

4.4 Solution initialization

The ALNS algorithm is based on the destroy and repair paradigm, which inherently assumes that an initial solution exists that can be destroyed.

An initial solution can be generated by construction heuristics [2], whose main properties are simplicity and speed. Construction heuristics incrementally build a solution with the help of simple customer selection and insertion rules in each iteration. Solutions created by these heuristics often do not possess competitive quality but can serve as a starting point.

A good starting solution positions the search framework in a promising part of the solution space. Nonetheless, initial tests indicate that the search framework needs less than 200 iterations to find a promising sub-space for most problems. This puts the computational benefit of a comprehensive construction heuristic into question. Additionally, optimized initialization might also lead to a position near a local optimum and restricts the local search framework in its initial endeavors.

Taking these observations into account, the only relevant property of an initialization technique seems to be the generation of a valid starting solution. Our proposed meta heuristic generally allows infeasible solutions via a soft-constraint. This guarantees a solution in the relaxed solution space under the assumption that all conditions previously described in Section 4.2.2 are met.

Examples of construction heuristics are the savings heuristic as introduced by Solomon (1987a) and insertion heuristics like the Greedy- and k-Regret heuristics as detailed in Section 4.6. The latter was applied by Ropke and Pisinger (2006) and has the advantage that code of insertion operators can be reused. The simplest technique, however, is to generate a solution at random.

Finding a valid solution for tightly constrained problems at random is very unlikely, but the relaxation of the solution space eliminates this disadvantage. We apply the following random initialization for its simplicity. Allocate each customer in random order to a randomly chosen vehicle. If it is not possible to assign a customer based on vehicle capacity, he may be assigned to the next.

4.5 Removal operators

The basic framework of ALNS iteratively destroys unfavorable patterns within the initial solution and repairs the solution to feasibility afterwards. In the context of VRPs, destroying a solution generally removes customers from their current route.

As mentioned in Section 3.2.5, a large number of ALNS variants have been implemented and a multitude of specialized operators have been tested. An explicit analysis and description of each operator exceeds the scope of this thesis. Nevertheless, we were able to identify four reoccurring strategies:

1. Randomization based removal
2. Metric based removal
3. Relatedness based removal
4. Historic based removal

The following section briefly introduces each of the categories and subordinates the implemented operators. Every operator has been selected based on problem specific dependencies and promising results of other authors (Ropke and Pisinger, 2006; Hemmelmayr et al., 2012; Demir et al., 2012; Franceschetti et al., 2017). However, this is not supposed to imply that the selection of operators in this thesis is the best possible combination.

Throughout this section we assume that $r\bar{m}$ is the exact number of customers to be removed, drawn from the Gaussian distribution as defined in Section 4.2.3

Most removal operators incorporate a degree of randomization to avoid stagnation, which is addressed in Section 4.5.5 separately.

4.5.1 Randomization based removal

The simplest form of removal is completely randomized since no metric calculation is required. Its obvious effect is diversification, as it can create new and surprising neighborhoods. The two most popular random removal operators are Random Removal (RR) and Random Route Removal (RRR).

Random Removal (RR)

The RR randomly selects and removes customers from the solution and can ensure trapping avoidance (Equation 3.1) in infinite iterations. The worst time complexity of this operator is $\mathcal{O}(n)$ (Demir et al., 2012).

Random Route Removal (RRR)

RRR is the only operator applied in this thesis where the number of removed customers is not guaranteed to be $r\bar{m}$. The operator selects one route at random and removes all customers contained in this route. It is important to note that the operator's effectiveness is reduced if the number of vehicles exceeds the number of final routes significantly. This could lead to multiple selections of empty routes, rendering the operator useless. The RRR is specialized in reducing the number of routes and most effective if fleet size minimization is within the objective. The worst time complexity of this operator is $\mathcal{O}(1)$ (Demir et al., 2012).

4.5.2 Metric based removal

Metric based removal operators prioritize customers based on a key performance indicator, approximating the quality of the customer position within the solution. This operator type is one of the most common and focuses often on highly problem-specific attributes.

Demand Removal (DR)

The Demand Removal (DR) is a specialized operator for the VRPLDTT and simply prioritizes the removal of customers based on their demand. Its solution independence distinguishes it from other metric based removals, in that it does not evaluate the quality of customer positions, but removes customers with a prioritization-worthy characteristic. We use this operator given that the positioning of high demand customers strongly impacts the quality and feasibility of a solution. Therefore, their placement should be given special emphasis. After sorting the demand values once, the worst time complexity of this operator is $\mathcal{O}(1)$.

Time Removal (TR)

The second metric based removal operator is Time Removal (TR), which approximates the quality of the positioning within a solution via the travel time from the customer to its route neighbors. Its basic hypothesis is that customers may be misplaced if they cause a major detour in the current route.

Let i_{prec} and i_{suc} be the preceding and succeeding nodes of customer i within the current solution. First compute all neighboring travel times $t_{i_{prec}i}(r) + t_{ii_{suc}}(r)\ \forall i \in N_1$, sort them and finally remove the highest ranked customers. All components of this routine are linear in worst case, except for the complexity of the applied sorting algorithm (here $\mathcal{O}(n\log(n))$) (ISO, 2017).

Worst Removal (WoR)

A more elaborate approach to metric based removal is to approximate the set of removed customers based on the the change in objective value. Let $\Delta F(\acute{s}, s)$ be defined as $\Delta F(\acute{s}, s)$ = f(s) - f($s \setminus i$). In other words, $\Delta F(\acute{s}, s)$ is the change of the objective value after a customer has been removed from the solution. It is important to note that the asymmetry of the distance matrix enables infrequent scenarios in which $\Delta F(\acute{s}, s) \leq 0$.

This approach is equivalent to a greedy removal heuristic and requires extensive reevaluation as detailed in Section 4.3. The Worst Removal (WoR) is expected to perform poorly on its own because of quick stagnation, but is optimized for fast intensification in new solution areas.

The general procedure of the WoR is identical to the TR, therefore the worst time complexity is also $\mathcal{O}(n \log(n))$. Nevertheless, the evaluation of each removal decision is much more time-consuming, which is why this operator is significantly slower in practice.

4.5.3 Relatedness based removal

A common problem of methodologies such as random or metric based removal is that the customer context is not considered. Removing a highly heterogeneous set of customers could limit the ability to swap them with other positions. Consequently, the likelihood is higher that customers will be reinserted at the location they were removed from.

One option suggested by Shaw is to quantify homogeneity in the removal set and include it in the decision process. This is achieved by calculating a relatedness measure of all customer node pairs $R(i, j)$, an idea closely related to clustering. Clustering optimizes on a distance measure to form groups of instances with the objective to maximize cluster coherency. Distance measures typically combine multiple attributes, which in the case of the VRPLDTT can be both customer and solution related.

One of the biggest differences between traditional clustering and relatedness removal is that clustering aims to form consistent and replicable groups. This limits the number of neighborhoods that clustering algorithms could produce. Therefore, relatedness removal should always incorporate a randomization component.

Shaw Removal (SR)

The oldest and one of the most frequently adopted relatedness removal operator is Shaw Removal (SR) introduced by Shaw (1997) with the LNS.

It starts with the initialization of a removal list L, after which a random customer is selected, removed and inserted into the removal list. In every following iteration a random customer is selected from L and used as reference node i. All $R(i, j)$ for the remaining customers j are computed and the customer with the biggest proximity to i is added to L

The originally proposed relatedness metric by Shaw (1997) (adopted to VRPLDTT) is:

$$R(i,j) = \frac{1}{d_{ij}(r) + V_{ij}} \tag{4.5}$$

where $d_{ij}(r)$ is the normalized distance from node i to j and V_{ij} the binary indication whether i and j are assigned to the same vehicle.

Ropke and Pisinger (2006) extend this formulation to include time windows and demand of a customer. They furthermore use a weight-based combination of attributes to enable subjective prioritization.

Adapted to the VRPLDTT, the relatedness metric could be defined as follows:

$$\begin{aligned} R(i,j) = {} & \varphi^d d_{ij}(r) + \\ & \varphi^v(V_{ij}) + \\ & \varphi^w(|a_i - a_j| + |b_i - b_i|) + \\ & \varphi^q(|q_i - q_j|) \end{aligned} \tag{4.6}$$

where φ describes the subjective weight of each component. Every component is normalized to the scale [0,1] to avoid instance dependencies of the calibration.

The subjective weights are adopted from Ropke (2003) as φ_d = 9, φ_w = 3, φ_q = 2, φ_v = 5. The worst time complexity of the operator is $\mathcal{O}(n!)$. All steps of the SR are again summarized in pseudocode in Algorithm 4 in which C_{cand} is the list of removal candidates.

Expert similarity removals (DiSR, WiSR, DeSR)

Different weight combinations can change the behavior of SR significantly. A simple and popular technique is to define one weight as 1 and all others as 0, in order to create an expert removal strategy.

Given the relatedness measure defined in Equation 4.7, four expert classifiers are conceivable:

1. Distance Similarity Removal (DiSR) with φ_d = 1, all others 0
2. Window Similarity Removal (WiSR) with φ_w = 1, all others 0
3. Demand Similarity Removal (DeSR) with φ_q = 1, all others 0
4. Vehicle Similarity Removal (VeSR) with φ_v = 1, all others 0

Algorithm 4 Shaw removal operator (SR)

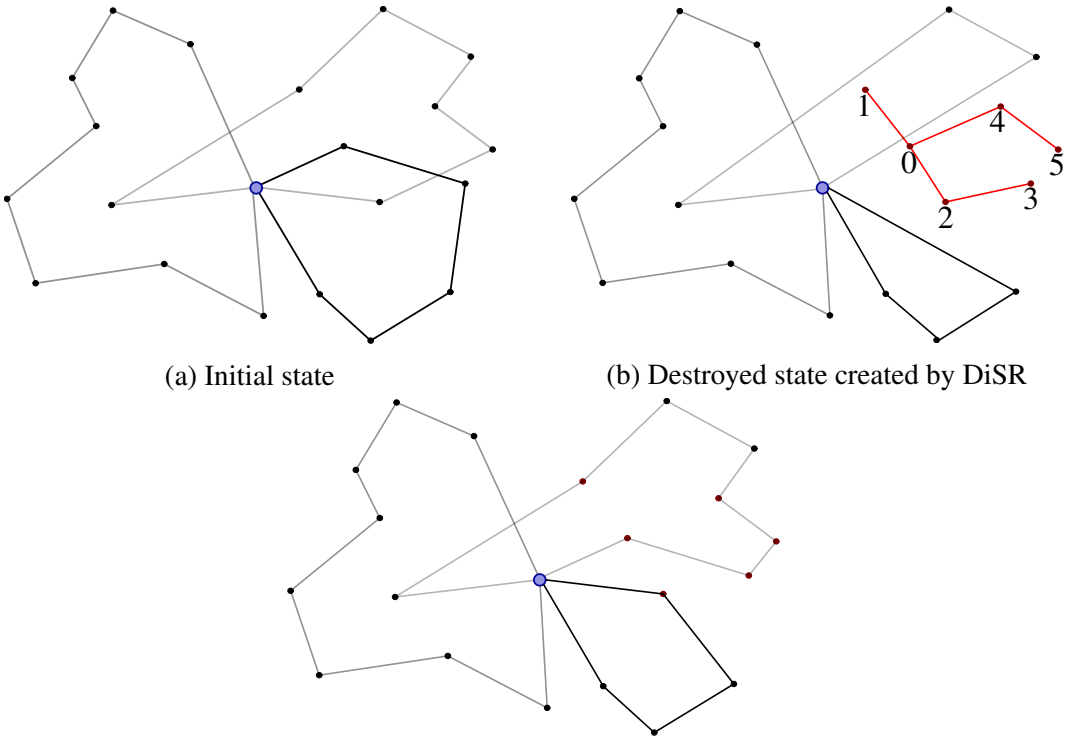
```
1: function ShawRemovalOperator(s, rm)
2:     L ← []                                   ▷ Initialize L with random C
3:     C_cand ← N_1
4:     Get random customer c from C_cand and append to L
5:     Remove c from s and C_cand
6:     Get random r̄m based on rm
7:     for r̄m -1 iterations do                  ▷ Find next candidate
8:         Get random customer c̄ from L

9:         ĉ ← None
10:        v_best ← ∞
11:        for c in s do
12:            v ← R_{c,ĉ}
13:            if v ≤ v_best then
14:                v_best ← v
15:                ĉ ← c
16:            end if
17:        end for

18:        L.append(ĉ)                          ▷ Process selected candidate
19:        Remove ĉ from s
20:    end for
21:    return L
22: end function
```

However, the functionality of VeSR is mostly redundant to RRR and is thus ignored. All other operators are adopted in this implementation.

Figure 4.3 depicts the destroy and repair principle with the selection of DiSR as removal operator. We chose this variant because of its visualization simplicity and to convey the general idea of the SR. The customer distribution and procedure states are identical to Figure 3.2. The depot is visualized with the blue circle, red nodes are customers marked for removal and all other dots represent customers that remain in the solution. Each number below a red dot in sub-figure (b) indicates the iteration in which the customer was selected for removal.

4.5.4 History based removal

So far all mentioned operator types act under the hypothesis that the removal decision is independent from previous iterations. History based removal relaxes this hypothesis and expects that information from previous iterations can be used to improve removal decisions. Thereby, the information storage is usually a matrix that logs a quality measure of all possible edges.



(a) Initial state

(b) Destroyed state created by DiSR

(c) Repaired state

Figure 4.3: Visualization of the destroy and recreate principle with DiSR as removal operator.

Node Pair Removal (NPR)

The Node Pair Removal (NPR) logs the best value $f(s)$ of any solution containing that edge. If the values of the incoming and outgoing edges $(i, j) \in A$ of each customer are summed, a potential of their positioning can be obtained. Customers who have historically only encountered poor quality solutions in their current positioning are removed. This approach assumes that certain customer allocations are viable in the short-term but block successful solution patterns. It therefore acts in the spirit of least regret.

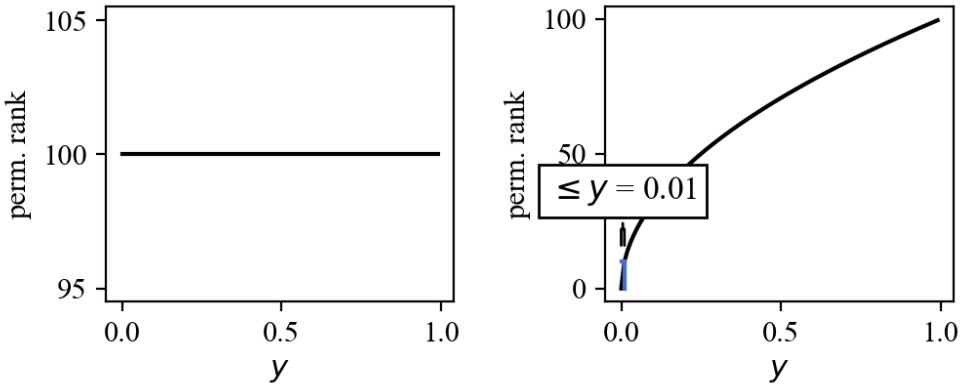
To calculate the quality of a customer based on the adjacent edges, then sort them by quality and remove the first $r\bar{m}$ is an identical to the procedure of TR. The worst case complexity therefore $\mathcal{O}(n \log(n))$ as well.

4.5.5 Permutation

All applied removal operators, with exception of random removal and route removal, use a quality score to rank customers. The customers with the worst rank are then removed to destroy the solution. However, this might lead to a rapid stagnation because of a limited diversity of created neighborhoods. A common approach is to apply a degree of randomness in the selection procedure of removal candidates (Ropke and Pisinger, 2006; Hemmelmayr et al., 2012; Franceschetti et al., 2017). We adopt the randomization scheme of Franceschetti et al. (2017) and skew the rank

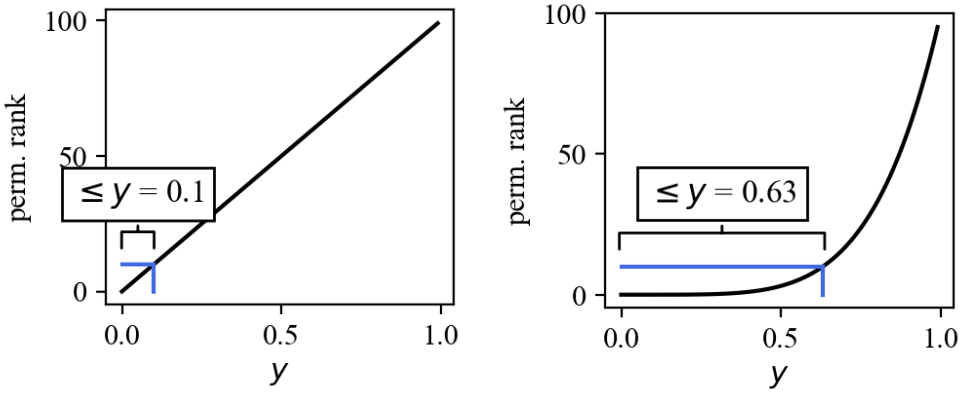
of each candidate by multiplying it with the factor y^ρ where y is a random number uniformly drawn from $y \in [0, 1]$ and ρ is an user defined fixed input parameter to indicate accepted randomness.

Figure 4.4 visualizes the impact of randomness on the final ranking. The black line shows the permuted rank of a customer with 100th best quality score based on the random variable y. All four sub figures plot the permutation behavior with changing ρ. The bracket indicates the minimum value y needed to achieve a permuted rank of 10. We can observe that a value of $\rho = 0$ leads to no permutation of the candidate ranking. In this case the selection is only dependent on the quality score. With increasing ρ the value of y needed to achieve the minimum acceptance threshold decreases. The convexity of the permutation function for $\rho > 1$ forces all early y values to a small rank. This ensures a high consistency for the best ranks and allows a bigger permutation in later ranks.



(a) Rank permutation for $\rho = 0$

(b) Rank permutation for $\rho = 0.5$



(c) Rank permutation for $\rho = 1$

(d) Rank permutation for $\rho = 5$

Figure 4.4: Impact of randomness on the final ranking based on changing ρ and y.

4.6 Insertion operators

Contrary to the large number of removal operators, the number of insertion operators presented in literature is comparatively low. Two main heuristics were applied

(in slightly modified form) in most implementations: Greedy Insertion and k-Regret Insertion. Our application supports a total of 5 variations of these insertion heuristics, which are described in more detail below.

Greedy Insertion Heuristic (GI)

The basic idea of Greedy Insertion Heuristic (GI) is to select a customer in the order of their removal and insert them in the position that minimizes the quality decrease of a solution. It is assumed that the order of prioritization provided by the removal operators indicates for which customer the correct placement is most important. Inserting a customer and reevaluating a solution is time-consuming and can be accelerated by only recalculating the quality of the changed route.

More formally, let $\Delta F(r, \acute{r})$ = f($\acute{r}_{ip}$) - f(r) be the cost of inserting a customer i into position p at route r. We set $\Delta F(r, \acute{r}) = \infty$ and skip all other insertion positions in that route if the insertion of a customer at position p exceeds the maximum allowed capacity limit $Q + U$. Let $\hat{pos}$ and $\Delta\hat{F}$ be the best insertion position and its corresponding objective value increase respectively. Algorithm 5 can then describe GI in pseudocode.

Algorithm 5 Greedy insertion (GI)

```
1:  function GreedyInsertion(s, L)
2:      for i in L do                                  ▷ Iterate through customers
3:          pôs ← (0,0)
4:          ΔF̂ ← ∞
5:          for r in s do
6:              for p in r do                          ▷ Get best insertion position
7:                  ŕ ← insert(i, (r,p))
8:                  if τ(r) ≤ U then
9:                      ΔF(r, ŕ) ← f(ŕ) - f(r)
10:                 else
11:                     ΔF(r, ŕ) ← ∞
12:                     break
13:                 end if

14:                 if ΔF(r, ŕ) < ΔF̂ then
15:                     pôs ← (r, p)
16:                     ΔF̂ ← ΔF(r, ŕ)
17:                 end if
18:             end for
19:         end for
20:         s ← insert(i, s, pôs)                      ▷ Do insertion
21:     end for
22: end function
```

Deep Greedy Insertion Heuristic (D-GI)

Deep Greedy Insertion (D-GI) is a modification of the GI and does not respect the order of the removal list. Instead, the best position is calculated for each remaining candidate in each iteration and the customer with the overall least insertion cost is inserted. This requires even more recalculations and there is less variance in the generated solutions.

It is important to note that the final implementation can minimize the number of recalculations by reusing intermediate results of previous iterations for routes that have not been altered.

Random Greedy Insertion Heuristic (R-GI)

The second implemented variant of the GI is called Random Greedy Insertion (R-GI) and disregards the order of the removal list L as well. Instead it selects customers from L for reinsertion at random. Conversely to the D-GI this has the effect of added diversification during solution generation. However, the procedure is comparable in speed with the standard greedy insertion.

β-Hybrid Insertion Heuristic (β-HI)

A relatively new hybrid variant of the GI was introduced by Franceschetti et al. (2017). This operator takes the removal list L and reverses it with a probability of 50%. It then tries to insert the list of removed customers L in one position if $|L| \leq \beta$. This includes the insertion in an empty route. If either no route can obey $\tau(r) \leq U$ after the insertion or $|L| > \beta$ the operator falls back to the procedure of GI. According to Franceschetti et al. (2017) this operator improves the myopic behavior of the local search routine. We adopt the suggested threshold of β=3 from the original article.

Regret Insertion (k-RI)

One of the biggest drawbacks of the greedy insertion is that it does not consider the potential impact of an insertion on positional alternatives of other customers. Additionally, the D-GI in particular tends to postpone difficult insertion decisions. The k-Regret Insertion (k-RI) tries to circumvent this problem by prioritizing the insertion of customers, based on their k best insertion alternatives. A first application of the k-RI on the VRP was introduced by Potvin and Rousseau (1993).

Let $\Delta F(s, \acute{s})^k$ be the cost of the kth best insertion alternative for customer i in solution s. The k-RI tries to find customer i where:

$$i := \arg\max_{i \in N_1} \sum_{h=2}^{k} (\Delta F(s, \acute{s})^h - \Delta F(s, \acute{s})^1) \tag{4.7}$$

and inserts it into his best position. This measure identifies the aggregated difference of the k best insertion alternatives to the best position and selects the decision that has the worst alternatives. Similarly to Ropke and Pisinger (2006) we compare only alternatives of two different routes.

This operator is the computationally most demanding, as it not only requires computation of all candidates in each iteration but also sorting of the alternatives to compute the regret measure. Similarly to the D-GI, intermediate results from previous iterations should be used to increase the computational speed.

4.7 Roulette wheel

All destroy and repair operators are managed within a roulette wheel mechanism, which was one of the main extensions from LNS to ALNS and allowed the combination of multiple operators. Destroy and repair operators are managed in separate wheels with identical functionality. In each iteration, both one destroy and one repair operator are randomly selected by the wheel and then applied on s. A performance score π_o defines the selection probability of every operator and adapts in regular intervals.

The following sections explain the reward mechanism and operator weight updating procedure in more detail.

4.7.1 Reward scheme

Ropke and Pisinger (2006) identified three scenarios in which an operator should be rewarded and attributed each scenario with a value σ :

σ_1: Reward for improving the global best solution $\hat{s}$

σ_2: Reward for finding a new unique solution with better quality than s

σ_3: Reward for finding a new unique solution with worse quality than s

This reward scheme serves as inspiration for our implementation, but is refined in several ways. First, we separate the aspect of finding a unique solution into a designated weight σ_4 for easier interpretation. Weight σ_3 can now be interpreted as a penalty, if the quality of $\acute{s}$ is worse than the quality of the current solution s and best solution $\hat{s}$. Secondly, we introduce a new weight σ_5 that rewards solutions located in less explored solution areas [32]. Exploration should only be incentivized

in the first phase of the algorithm, which is why this reward is weighted by the SA acceptance criterion defined in Equation 4.1. The implemented penalties and rewards can be summarized as:

σ_1: Reward for improving the global best solution $\hat{s}$

σ_2: Reward for improving the current solution s

σ_3: Penalty for neither improving $\hat{s}$ or s

σ_4: Reward for finding a new unique solution

σ_5: ν weighted diversity reward.

Rewards σ_1, σ_2 and σ_3 are mutually exclusive, but σ_4 and σ_5 can be awarded for every solution. All visited solutions can be efficiently tracked with a hash-map.

4.7.2 Purposeful diversification

The original ALNS implementation of Ropke and Pisinger (2006) benefits finding new solutions that have not been visited before. However, this mechanism mainly incentivizes a thorough intensification in later stages, rather than early exploration. Acceptance of a solution in the early stages of local search is highly randomized and most solutions are unique due to the small comparison pool up to this point. Only with decreasing randomization does this tendency reverse.

The hypothesis on which we base our approach is that the exploration phase can be conducted more effectively by documenting in which solution areas extensive searching has already taken place. Therefore, we introduce a matrix P which logs the frequency of use for every arc $(i, j) \in A$ throughout the search procedure. The diversity of a route is defined as:

$$f_{DIV}(r) = \sum_{(i,j)\in A(r)} 1 - \frac{P_{ij}}{it} \tag{4.8}$$

where it is the current iteration in which diversity is calculated. Diversity of a solution is the sum of diversity of its routes $f_{DIV}(s) = \sum_{r\in R(s)} f_{DIV}(r)$.

It is important to note that this reward might discourage the selection of operators specialized in fine tuning solutions and could thus discourage intensification later on. The reward $f_{DIV}(s)$ is therefore weighted by the SA criterion ν to form σ_5. Its benefit is twofold: First, this approach reduces the relevancy of diversity with increasing temperature. Additionally, it identifies diversity of good solutions as more relevant. The diversification is therefore *purposeful*.

4.7.3 Execution time normalization

To the best of our knowledge, each ALNS implementation for the VRP rewards operators on an iterational basis. One of the biggest drawbacks of this method is that it inherently prefers complex but slower methods to fast and simple heuristics.

There are two potential shortcomings of this tendency. First it might limit the diversification potential because of the reduced iteration count, one of the main historic motivations for the ALNS. The second is that multiple iterations with faster operators might ultimately scan the search space more efficiently than one single complex operation. Conclusively, it does not optimize the quality per time unit.

Based on this realization, we normalize the weight $\hat{\sigma}_{it} = \frac{\sigma_{it}}{t_{it}}$ where it is the current iteration and t_{it} its duration. It is important to note that a high precision time measurement can be prone to measurement noise. The execution time of an operator in this thesis is measured in milliseconds.

4.7.4 Weight adjustment

The rewards attributed in each iteration are tracked and used to adapt the selection probability in periodic steps [33]. All operators o are initialized with weights $\pi_o = 1$, which also serves as allowed minimum weight.

Previous implementations set a fixed update interval r (often 100 iterations) to avoid fitting on random behavior. This disregards the fact that the mean sample size of scores for each operator decreases with increasing $|\Theta|$. We counteract this inconsistency by multiplying the interval size r with the number of operators in a wheel $|\Theta^W|$.

After each segment of $r\ |\Theta^W|$ iterations all weights are updated by combining the previous weight π_o with the achieved average score in a smoothing function, a technique frequently used in forecasting. The reaction factor ζ herein controls the speed at which the new scores are adapted into weights. Let $\bar{\sigma}_o$ be the sum of rewards and $\bar{n}_o$ the number of applications of operator o during a segment. Algorithm 6 then describes the pseudocode to perform the updating routine of operator weights π_o.

Algorithm 6 Update wheel weights

```
1: function UpdateWeights
2:     if r * |Θ^W| iterations passed then
3:         for o in Θ do
4:             π_o ← ζ σ̄_o / n̄_o + (1 − ζ)π_o
5:         end for
6:     end if
7: end function
```

4.8 Parameter tuning

The previous sections presented several functionality components of the ALNSLDTT, tunable with various parameters. Depending on their setting, the behavior and performance of the algorithm can change significantly.

Parameter tuning is the process of configuring all parameters of an algorithm so that the performance is optimized. The optimality of the parameters often hinges on the instance size and structure. Finding the optimal subset of parameter values can be viewed as an overarching optimization problem.

However, heuristic implementations often incorporate components with random behavior. The tuning optimization problem in these instances is not deterministic but stochastic. A common approach to address this complication is to solve each parameter combination k times and optimize on the mean value. Depending on the runtime of an iteration and the standard deviation of the values this might be realistically infeasible. Therefore, rudimentary rule of thump heuristics are frequently applied.

Talbi (2009) distinguishes between online and offline parameter tuning techniques. Offline parameter tuning determines all parameter settings $x_i \in X$ prior and computes the performance for them during the routine. The results of the process are often insufficient in situations where the parameter domain is continuous, since the pre-selection of steps inherently limits its precision. On the other hand, this can enable the analysis of interactions between different parameters retrospectively.

The approach of online parameter tuning is much more autonomous and diverse. The term encompasses a variety of stochastic, statistical and heuristic methods that all share the commonality of autonomously finding a result on specified value domains (Talbi, 2009).

Eiben and Smit (2011) evaluated multiple online tuning techniques and reference promising results for a Classification and Regression Trees (CART) based domain reduction. The basic idea is to create decision trees and use the rules of the best branches to narrow down the value domains iteratively. This procedure is implemented and applied in our thesis based on its promising results, the implicit empiricism and the good scalability of the procedure. The following sections will explain the routine in detail.

4.8.1 Classification and Regression Trees (CART)

The original application of CART is in machine learning, in which new instances are classified given all rules defined in the decision tree. These rules are generated by identifying the features and split positions that exhibit the highest entropy on the label. It is important to note that the CART implementation can only perform binary splits.

A splitting criterion is used to quantify this entropy. The most popular criterion for numerical dependent variables is the Mean Squared Error (MSE) (Breiman, 2017).

Let N_m be the set of all instances in partition m and MSE(x_m) define its impurity given feature x as predictor. Additionally, let y_i and $\hat{y}_i$ define the real and predicted values of the dependent variable within the partition respectively. The impurity MSE(x) of a split into two partitions $m \in [1, 2]$ is defined as:

$$\text{MSE}(x_m) = \frac{1}{|N_m|} \sum_{i \in N_m} (y_i - \hat{y}_i)^2 \tag{4.9}$$

$$\text{MSE}(x) = \sum_{m \in [1,2]} MSE(x_m) \tag{4.10}$$

For all independent features $x \in X$, sort them, calculate the $MSE(x)$ for all split positions and perform the split on the feature with minimum impurity. This procedure results in two new branches and recursively repeats the splitting process for all partitions created. It terminates when no partition obeys the minimum split criteria, or no split decision results in a branch with less impurity. In the case of CART, the minimum split criteria typically are a minimum number of instances attributed to a node either before or after performing a split.

Figure 4.5 visualizes an example of a CART tree and its corresponding decision surface. This decision tree predicts the performance values of the ALNSLDTT based on parameters λ and ς and requires a minimum leaf size of 500. It is evident that the area of best performance in this example is λ = [2.6, 6] and ς = [9.5, 26] with a MSE = 918,77.

4.8.2 The CART tuning routine

The example in the previous section demonstrated that the CART can also be used to assess and isolate promising solution spaces. Their rules might also be used to define traits of well-performing heuristic configurations.

Let B_{og} define the potential value ranges for all parameters $x \in X$. Goal of the parameter tuning procedure is to reduce these boundaries B_{og} in order to fit the best sub-space B_{opt}. Let x_i be a randomly generated configuration given B_{og}.

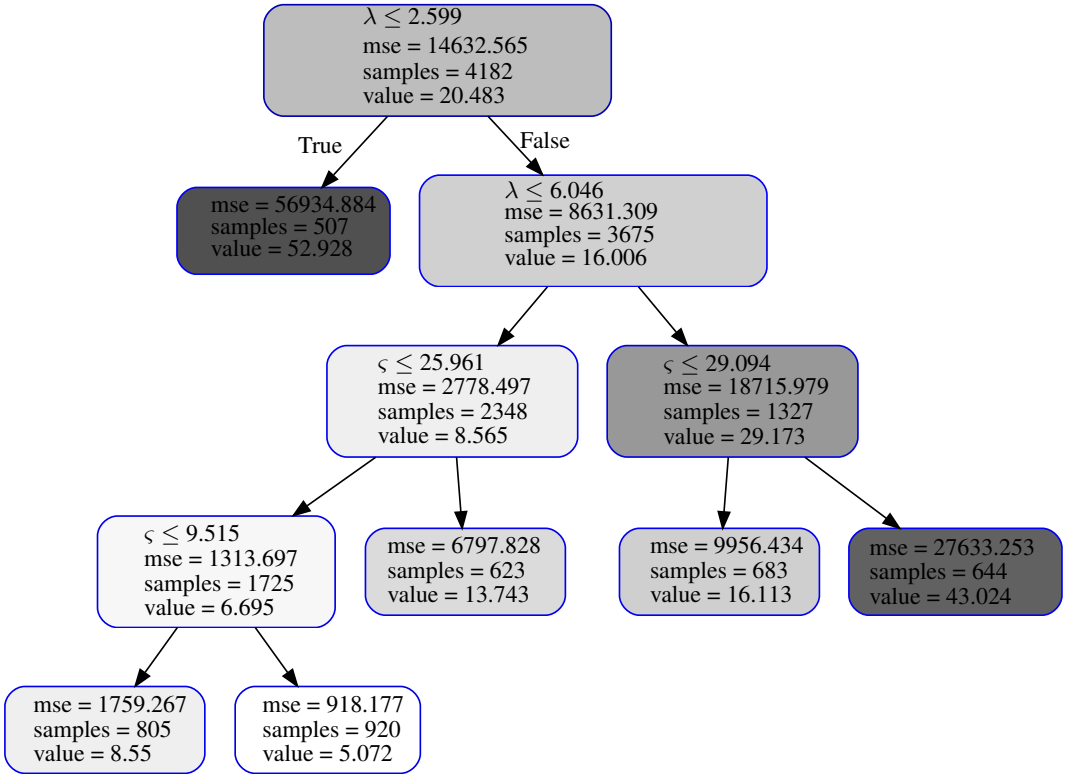
The procedure starts by generating n_{init} random parameter configurations $x_i \in X$ and solves one instance for each configuration to determine their quality $y_i \in Y$. After generating a sufficiently large training set, a CART tree is trained on Y with independent variables X.

The best branch and its corresponding rule-set can be determined recursively and used to create the intermediate B_{opt}. Subsequently, n_{iter} new x_i are randomly generated and added to X, obeying the new bounds B_{opt}. After their evaluation, a new CART tree is trained and used to find B_{iter} given B_{og}.

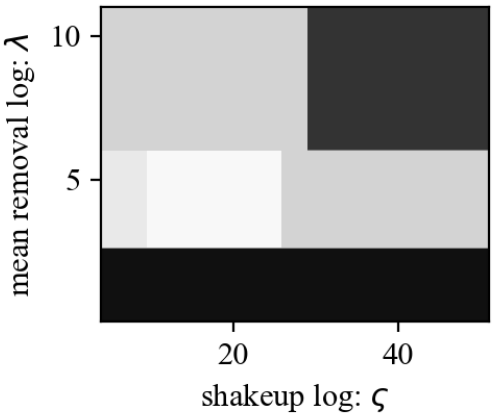
The newly isolated sub-space is compared to B_{opt} and checked for significant deviations. Afterwards B_{iter} replaces B_{opt}. If B_{opt} is consistent over κ iterations, the procedure is stopped and B_{opt} is returned. We refer to two value domains as consistent if each lower and upper bound deviates from the other by a maximum of 5%. The pseudocode in Algorithm 7 formalizes the described procedure.

The granularity and precision of all sub-spaces is defined by the minimum split criteria, however, a small minimum branch size might lead to over-fitting. A noteworthy difference of this approach compared to other online parameter tuning approaches is that it returns a value range instead of a clearly defined parameter value. Additionally, some value ranges might not be significantly reduced if the procedure is not confident in the decisiveness of their corresponding parameter.

Finally we might add that this parameter tuning approach inherits the *curse of dimensionality* from the CART, which (grossly simplified) implies that an increasing feature count requires a larger sample size to maintain consistent accuracy.



(a) Example CART tree for ∼4200 instances between $\lambda \in [1.05, 10]$ and $\varsigma \in [5, 50]$



(b) Decision surface of the CART tree in (a)

Figure 4.5: Example of a simple CART tree and the corresponding partitioned decision surface.

Algorithm 7 CART parameter tuning

```
1:  function CartTuning(κ, B_og, n_init, n_iter)
2:      X ← []                                          ▷ Build initial training set
3:      Y ← []
4:      for i in n_init do
5:          x_i ← getParameters(B_og)
6:          y_i ← metaheuristic(parameter).solve()
7:          X.append(x_i)
8:          Y.append(y_i)
9:      end for
10:     tree ← CART.fit(X)
11:     B_opt, λ ← getBestRules(tree)

12:     it ← 0                                          ▷ Iteratively update from reduced bounds
13:     while it < κ do
14:         for i in n_iter do
15:             x_i ← getParameters(B_new)
16:             y_i ← ALNSLDTT(x_i).solve()
17:             X.append(x_i)
18:             Y.append(y_i)
19:         end for
20:         tree ← CART.fit(X)
21:         B_iter, λ ← getBestRules(tree)
22:         if rules change significantly then
23:             B_opt ← B_iter
24:             it ← 0
25:         else
26:             it ++
27:         end if
28:     end while

29:     return B_opt                                    ▷ Return new reduced bounds
30: end function
```

4.8.3 Categorical parameters

A limiting factor in the application of CART is that it can only fit on numerical data by design. Parameters with categorical value domains (e.g. operator names) cannot be processed by the algorithm without previous value transformation. Our implementation of the CART tuning algorithm recognizes categorical data and performs a *one-hot encoding* to transform them into numerical features. This procedure (also frequently known as *dummy encoding*) creates a new feature for each possible category and assigns the value 1 if the instance belongs to the corresponding category, 0 otherwise.

Our implementation also recognizes that some categorical parameters might be multiple choice, for example, $|\Theta|$ might be bigger than one. Therefore, we relax the condition that all dummy columns of one feature must sum up to one.

4.8.4 Value standardization

Tuning the parameters on a single instance risks over-fitting on its properties. Therefore, all evaluations are done on an instance randomly drawn from a pool.

This requires standardization of all y_i generated by solving a specific instances to reduce bias towards network properties, instance size and solution outliers. Given a value x_i, the standardized score z_i can be computed by:

$$z_i = \frac{x_i - \bar{X}}{\text{sd}(X)} \tag{4.11}$$

where $\bar{X}$ is a the mean of the sample distribution and sd(X) its standard deviation. The standardized value is frequently called z-score and indicates the number of standard deviations a value deviates from the mean.

However, Iglewicz and Hoaglin (1993) criticize that it is still easily confused by outliers and suggest using the modified z-score as a replacement. The modified z-score is defined as:

$$M_i = \frac{x_i - \tilde{X}}{\text{MAD}(X)} \tag{4.12}$$

where $\tilde{X}$ describes the median of the given sample size and MAD(X) its Median Average Deviation (MAD) defined as MAD(X) = median($|X_i - \bar{X}|$). We use the modified z-scores of all solution values as standardized dependent variable. A drawback of standardization is that each new instance increases the degrees of freedom of the parameter tuning routine. It is important to find a suitable trade-off between robustness of the estimation through large sample size and good generalization capabilities through instance pool size.

5 Implementation and Setup of the Computational Analysis

5.1 Implementation software and hardware

The algorithm is implemented in C++, compiling with Visual Studio (2017) into a Dynamic Link Library (DLL). The compiled executable is then embedded into the parameter tuning approach implemented in Python (3.6.5). All computations are performed on a 12 core Intel Xeon CPU E7-4830 v3 (2.10 GHz) and 50 GB memory. The ALNSLDTT implementation is not natively parallelized, thus one iteration is executed on a single core.

We used the Bing Maps API to generate all distance values of our newly introduced benchmark instances and the Google Maps API for map visualization. Additionally, our implementation incorporates some algorithmic components of the the Boost and STD library of C++, as well as the Scikit-learn, SciPy, and NumPy libraries in Python (Nakariakov, 2013; ISO, 2017; Pedregosa et al., 2011; Jones et al., 2001–; Oliphant, 2006–).

5.2 Datasets

Research on meta-heuristics is commonly compared with the help of benchmarking data-sets. A big limitation of heuristics is that their performance cannot be proven to be optimal, neither can a performance gap be computed without performance reference. Two types of reference values are commonly used: Optimal solutions generated once with high computational effort or Best Known Solution (BKS) of previously applied meta-heuristics.

No standardized benchmark data-set has been introduced for the VRPLDTT yet and thus limits a comprehensive performance comparison. We solve the data-set of Fontaine (2019) and show that our algorithm consistently finds optimal solutions for problem instances with up to 20 customers. The algorithm is also tested on well known benchmark sets for the simpler VRPTW and indicates competitiveness to current state of the art heuristics.

We further introduce new benchmarking data-sets and first BKS for the VRPLDTT. These instances are, similarly to Fontaine (2019), based on cities with notable al-

titude differences. However, these problem instances include distance and slope information for up to 200 customers.

5.2.1 Datasets for the VRPTW

The chapter prologue already indicated the use of popular VRPTW benchmark sets introduced in Solomon (1987a) and Gehring and Homberger (1999), to prove competitive performance of our implementation. These datasets are explained in more detail during this section.

Solomon VRPTW instances

Solomon (1987a) was one of the first to introduce a set of benchmarking instances for the VRPTW. The instance sizes reach a maximum number of 100 customers and most have been solved to optimality. However, new implementations still frequently report performance on his data-sets to prove competitiveness. Traditionally, all VRPTWs are solved on the objective to minimize the number of vehicles used and adopt the total travel distance only as secondary objective. Nevertheless, optimal solutions for the total travel times are also available (Solomon, 1987b). Solomon considers three cases of customer distributions:

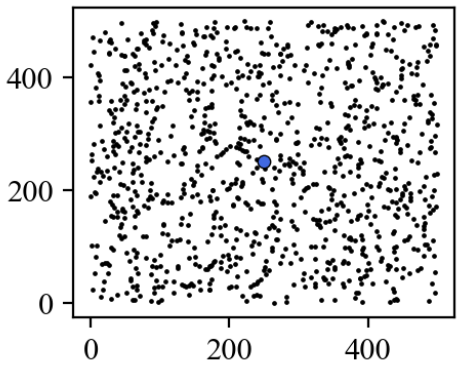
1. **Random (R)**: Customers are randomly placed on a coordinate grid.
2. **Clustered (C)**: A number of centroids are randomly placed on the coordinate grid. Customers are then clustered around these centroids.
3. **Random-Clustered**: 50% of the customers are clustered around randomly placed centroids while the rest is randomly positioned on the grid.

He further distinguishes between instances with denominator 1, indicating short traveling horizon ($\sim$5-10 customers per route) and 2 indicating a long traveling horizon ($\sim$30 customers per route). Every case combination is finally discriminated into six settings with increasingly tight time windows.

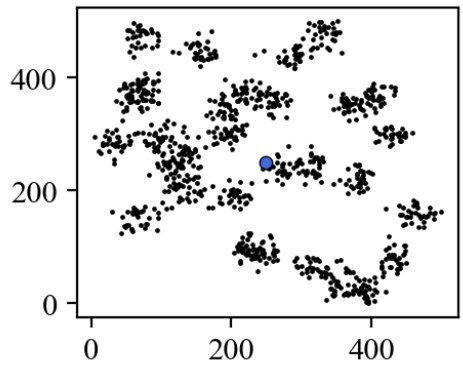
Gehring and Homberger VRPTW instances

Several heuristics have been introduced that are able to solve the benchmark-instances of Solomon (1987a) consistently near optimality. The capability of these benchmarking sets to distinguish between those approaches is therefore limited. This is why Gehring and Homberger (1999) introduced new large scale problem instances with 200, 400, 600, 800 and 1000 customers respectively. All of their customer, demand and time-window generation strategies are identical to Solomon (1987a). The only difference is that the time-window gradation is divided in 10 instead of 6 steps.

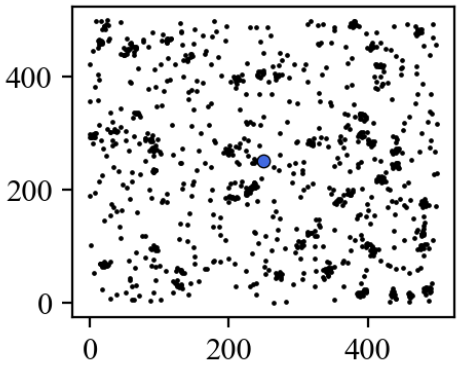
Figure 5.1 visualizes customer distribution scenario R1, C2, RC2 of Gehring and Homberger (1999), one for each distribution strategy. The centric blue dot represents the depot.



(a) Random customer distribution (R1)



(b) Clustered customer distribution (C2)



(c) Random-Clustered customer distribution (RC1)

Figure 5.1: Visualization of the three distribution types R, C and RC introduced by Solomon (1987b)

5.2.2 Datasets for the VRPLDTT

Fontaine VRPLDTT instances

Fontaine (2019) was the first author to introduce the VRPLDTT problem and corresponding data-sets. He selected six cities based on their altitude profile: Fukuoka (Japan), Madrid (Spain), Pittsburgh (USA), Seattle (USA) as well as Sydney (Australia) and positioned a depot and 20 customers evenly distributed across each city. The distance matrix and altitude data is based on real routing requested in Google Maps. Each customer is attributed a demand q_i which is randomly drawn from a uniform distribution $q_i \in [5, 30] \forall i \in N_1$. Additionally, customer time windows are randomly placed from beginning to end of the planning horizon of 120 minutes. The average length of each time window is set to 55 minutes. Finally, the service time for every customer is constant at 5 minutes. Fontaine (2019) solved all instances to optimality. It is important to note that all solutions are reported with a load bucket size of 10.

New VRPLDTT instances

Due to the current lack of suitable medium or large scale VRPLDTT data-sets, we have decided to introduce new benchmarking instances for future comparability. The characteristics of all new problem instances are based on the suggestions of Fontaine (2019) and adopt cities, time-window assumptions and load distributions. One central time-window setting is generated for all test instances. This is paired with a set of four standardized demand scenarios.

The main difference to Fontaine (2019) is that customers were not positioned manually but generated from random draws within given geo-coordinate limits of each city. Due to API request limits, instance sizes are at maximum 200 customers per city. A second key difference to Fontaine (2019) is that we do not assume a symmetric distance matrix. The final disparity to Fontaine is that the load bucket size reported on is 1 instead of 10.

The customer distributions of all new benchmarking instances are visualized in Appendix B. All performance values reported for instances with less than 200 customers use only the first n entries of a datasets.

5.3 Pre-processing

Pre-processing is used to generate data which would be inefficient or redundant to compute during runtime. We pre-compute two data aspects:

1. Travel times t_{ij}^{l} defined by distances, elevations and load levels. We name the set of travel times *time-cube*.
2. Customer disparity matrices describing the difference of customers by demand, time-windows and position.

The following sections explain both pre-processing components in more detail and provide additional context about the use and necessity of this information within the algorithm.

5.3.1 Time-cube

Operators often require multiple recalculations of the solution quality, which depends on the travel time from customer i to customer j. This need for intensive reevaluation is the main added complexity when comparing heuristic approaches for the VRPTW and VRPLDTT. It is computational expensive to recalculate the load dependent travel speed and -time dynamical. Pre-processing of t_{ij}^{l} values is therefore crucial.

Pre-computing load dependent travel times for demand values with a value domain of $\mathbb{R}^{+}$ is impossible without discretization, due to the infinite load values. We solve

this by designing load buckets $L = \{1, ..., l, ...\}$, covering all load levels of a fixed range $[p^l, r^l]$ as was suggested in Section 2.1.1. Each bucket is of uniform size ls to obey condition $ls = (r^l - p^l) \ \forall \ l \in L$ and all loads allocated to a bucket are substituted to be $(p^l + r^l)/2$. Using a high granularity (e.g. the lowest unit used for mass measuring) will ensure high solution accuracy.

This definition of discrete buckets enables the pre-calculation of all travel times for distance values d_{ij}. It is assumed that the distance- and slope matrices are provided as user input. The velocity v_{ij}^l can be determined by simplifying and solving equation 2.20, but is returned in $\frac{m}{s}$. The total travel time therefore is defined as:

$$t_{ij}^t = \frac{d_{ij}}{3.6 \ v_{ij}^t} \tag{5.1}$$

It is important to add that this pre-calculation step is not necessary for generalized problems such as the VRPTW. Problems without load dependent travel times can substitute t_{ij}^l with d_{ij} by simply defining the load granularity to be $Q + U$.

5.3.2 Disparity matrices

The SR operator removes customers from a solution based on their disparity to other customers in their route. The degree of disparity between customers can be defined on multiple parameters, some of which might be fixed for a solution instance. These fixed disparity values can be pre-computed. The difference dimensions recognized and pre-computed in this implementation are customer demand, time windows and geographical position.

Positional distance requires little pre-processing, because we can leverage the travel distances d_{ij}. Discrepancy in customer demand q_i, start times a_i and end times b_i are determined through the euclidean distance for all customer pairs (i, j).

The SR also assumes normalized values, which is why we perform a *min-max normalization* on all disparity matrices during pre-processing, as defined in Equation 5.2.

$$\hat{x} = \frac{x - min\{x : x \in X\}}{max\{x : x \in X\} - min\{x : x \in X\}} \tag{5.2}$$

5.4 Setup of the parameter tuning

Parameter grouping and domains

The parameter tuning is performed in a split and greedy setup. We split all parameters into functionality buckets and tune these buckets in the order of their assumed priority. Results from previous tuning iterations serve as input for later iterations.

The final grouping of all buckets is listed in Table 5.1. It also includes information about the starting value domain of each parameter and the default values used in absence of intermediate results. All default values are either adopted from Ropke and Pisinger (2006) and Franceschetti et al. (2017) or defined through initial undocumented testing.

The applied tuning approach is limited to uniformly distributed draws from the given value range. We define continuous numerical domains for all parameters possible, and discrete domains if this assumption does not seem suitable. The bucket order in Table 5.1 is based on estimated performance priority. Operator selection is performed last to minimize the influence of insufficiently optimized parameters.

CART parameter setting

Most CART parameters are adopted from the default configuration proposed by Pedregosa et al. (2011). However, we adjust the minimum split criteria to increase tuning robustness. A split is only allowed if the node to be split contains at least 20 instances and each node created at least 10.

Both n_{init} and n_{iter} are adjusted in each tuning bucket to fit the number of parameters and the complexity of their value domains.

Instance selection

Six differing instances are selected to prevent over-fitting on a single problem setting. The final selection of instances is $P_i \in \text{Fu1}_{50}$, Fu2_{100}, Ma3_{200}, Ma2_{50}, Pi1_{200}, Pi3_{100}. Two instances of each 50, 100 and 200 customer cases are selected to guarantee diversity. Similarly, two instances for each Fukuoka, Madrid and Pittsburgh are used and demand case 1, 2 and 3 are each selected twice as well.

Table 5.1: Tuning setup: Parameter grouping and value domains for tuning

Parameter	Group	Value domain	Initial value
c	Local search acceptance	{0.99, 0.995, 0.999, 0.99975, 0.9999}	-
T	Local search acceptance	{100, 1, 0.1, 0.01, 0.001, 0.0001}	-
ς	Local search radius	[5, 50]	20
λ	Local search radius	[1.05, 10]	2
ρ	Operator randomness	{0, 0.15, 0.5, 1, 5}	0.15
σ_1	Operator selection	[0, 100]	33
σ_2	Operator selection	[0, 100]	13
σ_3	Operator selection	[0, 100]	9
σ_4	Operator selection	[0, 100]	9
σ_5	Operator selection	[-100, 0]	0
r	Wheel memory	{1, 10, 50, 100, 250, 1000}	10
ζ	Wheel memory	[0, 1]	0.2
ϱ	Infeasibility	[0, 0.9]	0.2

5.5 Setup of the computational analysis

Section 6.2 analyzes the algorithm with the goal to verify and visualize its behavior. Especially the custom components shake-up and operator reward normalization, as well as the general search pattern are of interest.

Insights are generated on individual solution iterations for the Fu1_{200} and only serve as exemplary tendency.

5.6 Setup of benchmarking

We identify two core criteria that our implementation must achieve in order to have relevance.

1. Competitive performance on small instances (compare w. exact approaches)
2. Competitive performance on large instances (compare w. other heuristics)

The first can be indicated by investigating whether the ALNSLDTT is capable of consistently finding optimal solutions for instances solved by Fontaine (2019).

However, no explicitly optimized heuristics for the VRPLDTT currently exists besides our implementation that would enable us to prove the second criterion. We therefore compare the performance of the algorithm with heuristics for the simpler but well researched VRPTW. Our BKS performance comparison is inspired by the setup of Vidal et al. (2013) and includes reports from a series of popular VRPTW heuristics.

The BKS of the following implementations are presented and compared with our results.

1. **PisR**: ALNS (Ropke and Pisinger, 2006)
2. **NB-100** and **NB-f(n)**: Hybrid Genetic Algorithm based on EAX crossover of Nakariakov (2013) with a population size of 100 and f(n) = $\frac{n}{20000}$
3. **PDR**: Branch-and-Price based LNS (Prescott-Gagnon et al., 2009)
4. **RTI**: Arc-guided Evolutionary Algorithm (Repoussis et al., 2009)
5. **HGSADC**: Hybrid Genetic Search with Advanced Diversity Control (Vidal et al., 2013)

It is important to note that our implementation is not configured for the VRPTW and optimizes on the objective of travel time minimization. This differs from the applied objective of fleet minimization and therefore limits the comparability severely. However, we assume that an indication of a good baseline performance is possible nonetheless.

Finally, we calculate initial BKS for the newly introduced instances to enable future comparisons.

6 Results

All results are presented and discussed in this chapter. Similar to the structure of Chapter 5, we start with the results of parameter tuning (Section 6.1), then proceed to analyse the most important innovations (Section 6.2) and finally compare the performance with those of other meta-heuristic implementations (Section 6.3).

6.1 Parameter tuning results

6.1.1 Local search acceptance

The first component to be tuned is the acceptance behavior of our SA framework. It consists of the initial temperature T and cooling rate c, which define the degree and length to which $\acute{s}$ is accepted as s if $f(\acute{s}) \geq f(s)$. The size of the initial training set is 1 800 and the size of each additional iteration 180. We will use a tuple based notation with the form (T, c) for conciseness.

The CART tuner returns (0.9999, 0.01) as the suggested combination after 1 980 iterations. All parameter combinations and their performances are visualized in Appendix C. The collection of box-plots supports the result of the tuning approach and attests to the highest potential of the selected combination (encoded as 03). Other notable alternatives are (0.99975, 0.01) and (0.9999, 0.001) with comparable median performances. Nevertheless, the best solution created with these combinations is worse, which indicates a lower potential.

6.1.2 Local search radius

The ALNS is based on the *destroy and recreate* paradigm which implicitly defines the sizes of neighborhoods by the degree of solution destruction. We mentioned in Section 4.5 that the number of removed customers is randomly selected. However, its distribution depends on the instance size n, the previous number of iterations without improvement it as well as on the two parameters λ and ς. These parameters define the corresponding log transformation and will be tuned in this section.

We selected the log domains to be $\lambda \in [1.05, 5]$ and $\varsigma \in [5, 50]$ based on the expected sizes of n and it respectively. The tuning parameters are i_{init} = 3 000 and i_{iter} = 300, and the tuning procedure reaches a stable state after 4 200 iterations. The domains returned are $\lambda \in [3.35, 3.35]$ and $\varsigma \in [9.74, 10.66]$.

Appendix D visualizes the generated decision space of the tree learner. This visualization is the more precise representation of the example in Figure 4.5. White areas indicate exceptional and black areas bad performance.

We can observe that the learner evaluates regions poorly that result in either a consistently small or very high rm. Most combinations in $\lambda \in [1.8, 3.5]$ and $\varsigma \in [3.5, 15]$ indicate good ratings. All further calculations are performed with $\lambda = 3.35$ and $\varsigma = 10$. This parameter combination setting seems robust, as it is both in the center of the broader high performance region and the smaller recommendation interval.

6.1.3 Operator randomness

Random behavior in an algorithm is a popular approach to guarantee diversification. There are three main components in our implementation that induce randomness, the first being operators acting exclusively on random guidelines. Our second component is the draw procedure of $r\bar{m}$ using λ and ς, which were tuned in the previous section. Last but not least, randomness can be induced by permuting operator decisions with random noise. This behavior can be configured with the parameter ρ and is subject of this section.

We chose a discrete selection $\rho \in \{0, 0.15, 0.5, 1.0, 2.0, 5.0\}$ as value domain to showcase performance change with non-linear randomization adjustments. The initial training set n_{init} is of size 1 800, the iteration of size n_{iter} 180 and the tuning procedure reached a stable state after computing 1 980 instances.

Results of the tuning procedure are visualized in Figure 6.1 and reveal that both a small and large degree of randomization negatively impact the performance stability. However, it seems difficult to exactly pinpoint the perfect degree of randomization, as the average performance difference appears to be only marginal. Our final selection is $\rho = 0.15$ based on previous settings of Franceschetti et al. (2017).

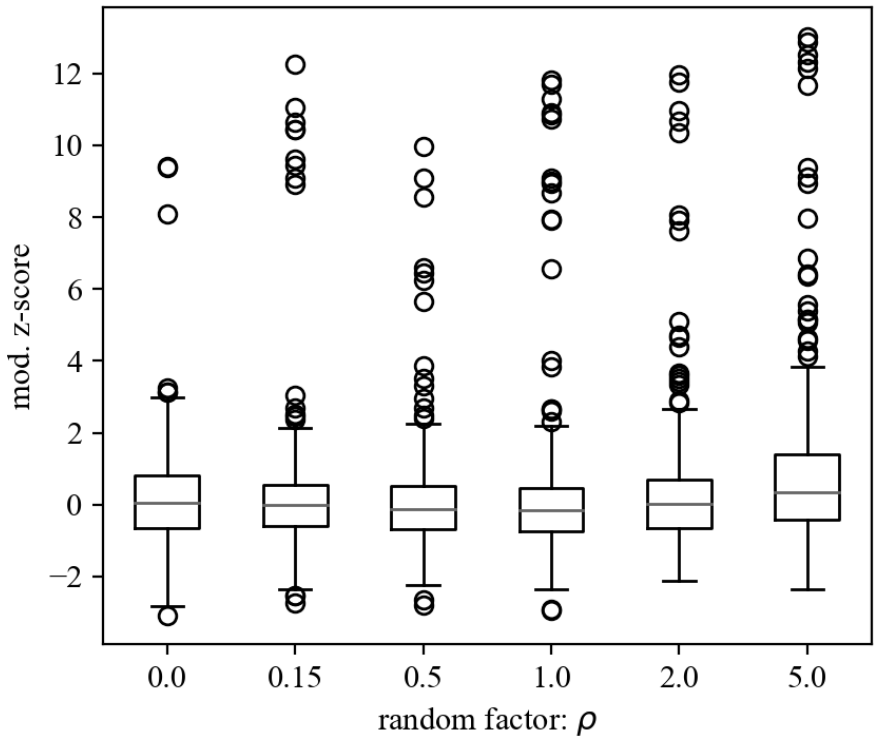


Figure 6.1: Box-plots visualizing the performance of the discrete ρ domain.

6.1.4 Operator weighting

The next parameter group consists of the reward and penalty values assigned to operators, in order to determine the draw probabilities within the wheel mechanism. Optimized weightings can prioritize suitable operators for the current stage of the search procedure. However, if the wheel mechanism is too reactive it might overfit on randomness. Parameters defining the reward strategy are: σ_1, σ_2, σ_3, σ_4 and σ_5 as defined in Section 4.7.

This is the largest bucket of our tuning procedure and therefore requires the longest iteration count. We set the size of the initial training set to 10 000 and the improvement iteration size to 500. Nevertheless, the parameter tuning routine was not able to determine a stable state after 27 000 iterations.

Appendix E visualizes the z-score distributions of all training sets based on the number of customers. It is evident that instances with 50 or 100 customers regularly generate identical solution values and thus suggest high consistency. We conclude that smaller instances have hardly any performance discrimination potential. Subsequently, all instances with less than 200 customers are substituted by their large scale counterparts for the rest of the procedure.

Parameter tuning suggestions

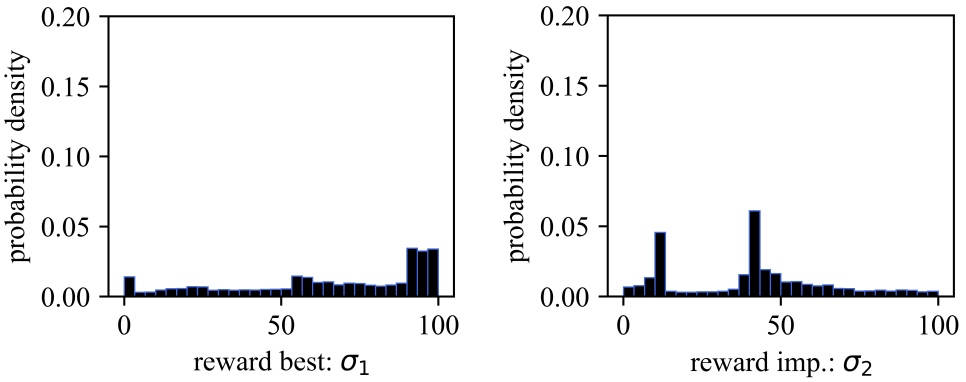
However, even substituting all instances with less than 200 customers yields no stable state in acceptable time. We interrupted the parameter tuning after 20 000 iterations and used the results generated until that point. The intermediate results for all weights can be found in Table 6.1. A certain degree of indecisiveness for σ_1 σ_3 and σ_4 is confirmed by the relatively wide domain breadth.

Table 6.1: Reduced σ domains after interrupted parameter tuning.

Parameter	Value domain	Initial value
σ_1	[0, 55.38]	33
σ_2	[97.34, 100]	13
σ_3	[53, 100]	9
σ_4	[6.91, 54.31]	9
σ_5	[-100, -77.60]	0

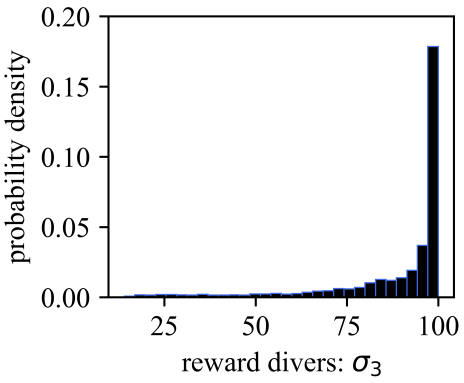
Parameter value frequency during tuning

Figure 6.2 visualizes the draw frequency within each domain during the iterative stabilization phase. It is surprising that the modes of σ_1 and σ_2 are positioned outside the value domains returned by the parameter tuning. However, this might be explained by the relatively low frequency modes of these parameters, which symbolize uncertainty. The target region of the remaining three parameters is clearly defined.

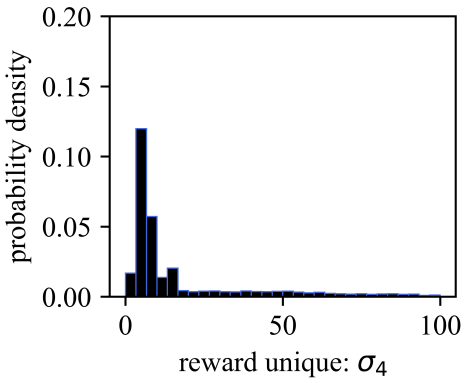


(a) Draw frequency of values for σ_1

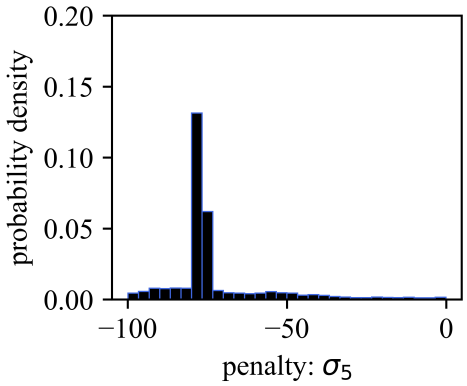
(b) Draw frequency of values for σ_2



(c) Draw frequency of values for σ_3



(d) Draw frequency of values for σ_4



(e) Draw frequency of values for σ_5

Figure 6.2: Value frequency of σ during the last 10 000 iterations

Modified z-score distributions for parameter values

An illustration of the z-scores distribution across the value domains in Appendix F cannot highlight any trend. It therefore questions the certainty of the value domains of σ_3, σ_4 and σ_5. However, this representation is not suitable to visualize possible interdependencies between weights. Yet even if one represents relative weight trade-offs of all operator pairs (Appendix G), no unambiguous pattern is evident.

Both z-score visualizations indicate that the selection wheel is of minor importance for the performance of the algorithm. Nevertheless, we cannot reject the possibility of a non-linear pattern, exceeding the visualization capabilities of these graphs. We decide to follow the recommendations of the CART and substitute the uncertainties in Table 6.1 with the modes in Figure 6.2. This results in the final values σ_1 = 50, σ_2 = 100, σ_3 = 90, σ_4 = 7 and σ_5 = -80.

6.1.5 Wheel memory

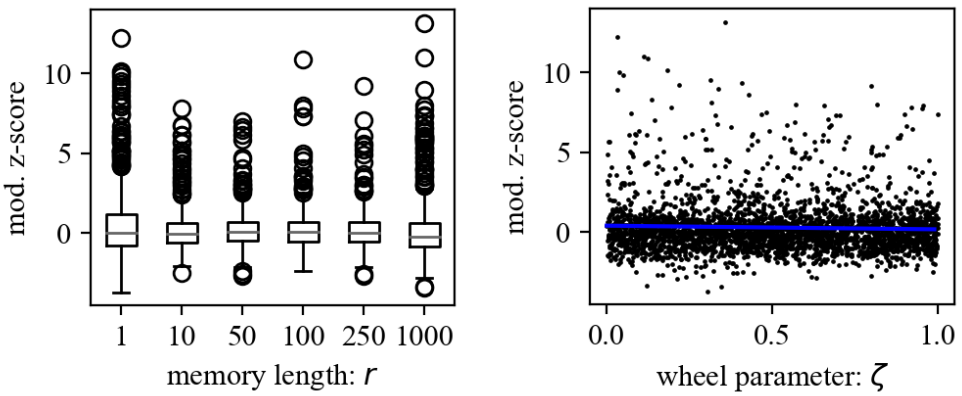
The next functionality group of parameters is the wheel memory, configurable through the length of the roulette wheel memory r and the wheel parameter ζ. Both define the discount on weights of previous iterations in tandem.

We set the domain of the memory length r as $r \in \{1, 10, 50, 100, 250, 1000\}$ to be able to map both short and long response times. The domain for the memory parameter is $\zeta \in [0, 1]$, wherein a value of 1 suggests that all previous update-cycles are disregarded. Contrary, setting ζ to 0 leads to the wheel weights being fixed at their initial value (here $w_o = \frac{1}{|O|} \; \forall \; o \in O$).

The parameter tuning routine is performed with n_{init} = 3 000 and n_{iter} = 300. A stable state with r= 1 000 and $\zeta \in [0.34, 0.35]$ was determined after ∼5 000 iterations. This parameter results in a notably slower adjustment speed compared to the setting of Ropke and Pisinger (2006).

Visualizing the modified z-scores of the considered value domains in Figure 6.3 reveals that the median is mostly consistent for changing ζ and r settings. The only visible trend is that performance inconsistency is an issue for very small (r = 1) and very large memory lengths (r = 1 000). This increase in inconsistency leads to outliers but also resulted in the highest quality solutions achieved during the tuning procedure. The full scatter distribution of ζ given a specific r is plotted in Appendix H and solidifies the expectation generated through Figure 6.3, which is that ζ is of marginal importance. It is only in the case of r = 1 that the correct selection of ζ has a decisive effect by reducing outliers notably.

Taking the suggestions of the regression tuner and the visualizations into account, we select ζ = 0.35 and r = 10. This diverges from the parameter tuning suggestions of r = 1000, but is closer to the original parameter setting introduced by Ropke and Pisinger (2006). We hypothesize that r = 1000 might be too slow to adjust for cases with a faster stoppage criteria. First signs of this are visible in Appendix H. Therefore, we take inspiration from *Occam's razor* and select the simplest model possible to describe the desired behavior.



(a) Box-plots of the r domain.

(b) Scatter plot of the ζ value domain.

Figure 6.3: Modified z-score visualization for the selected r and ζ value domains

6.1.6 Infeasibility

The last parameter is ϱ, defining the desired percentage of infeasible solutions created during the local search process. We suspect that the relatively small n_{init} = 1 800 and n_{iter} = 180 are sufficient. The regression tuning technique terminated at iteration 1 980 with $\varrho \in [0.643, 0.645]$.

Figure 6.4 shows the modified z-score distribution of independently and uniformly drawn ϱ values. We can identify that a minimum value of $\varrho >$ ~0.3 is beneficial for the average performance. This supports the claim of Laporte et al. (2010) that infeasible solutions can help escape local optima. We additionally hypothesize that the observation is coherent with the proof that an optimal solution to an optimization problem in canonical form must be on one of the extreme points of the solution space (Murty, 1983). Forcing the search routine on the brink of infeasibility enables a more efficient search of these extreme points and increases the chance of finding a global optimal solution.

We decide to adopt the suggestion returned by our parameter tuning routine and define ϱ to be 0.65.

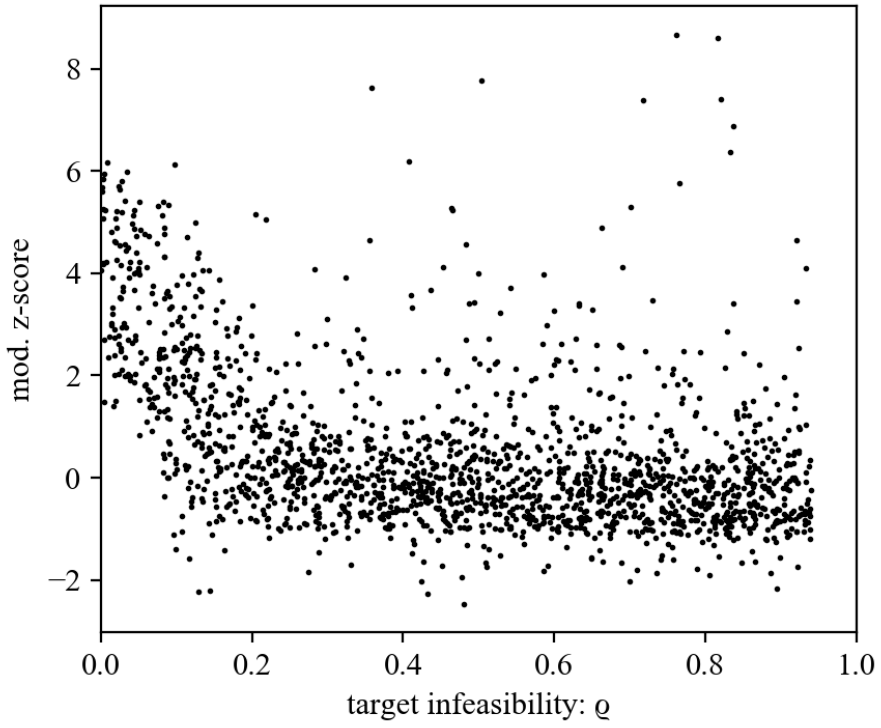


Figure 6.4: Scatter plot of the modified z-score distribution given changing $\varrho \in [0, 1]$

6.1.7 Operator selection

Up until now we assumed that the selection wheel mechanism is capable of adjusting the search procedure to instance inherent biases. However, it is ill advised to

include sub-par performing operators into the search procedure with the assumption that the weighting procedure avoids their selection implicitly. This would neglect potential noise in the selection routine, which significantly increases with the number of operators. We aim to find the smallest operator subset Θ that yields robust search behavior by combining operators whose neighborhoods compliment each other.

Appendix I and J illustrate the average performance of the ALNS after including a specific destroy or insertion operator to the set of operators Θ. Based on these visualizations, we might assume that the inclusion of operators RRR, DR, WoR, WiSR or DeSR provokes below-average results. Additionally, only 2-RI and 3-RI seem to have a noticeable positive effect on the average performance. However, no interdependencies between operators can be mapped in this visualization.

The result of the applied CART parameter tuning technique is summarized in Appendix K. Two evaluations are particularly noteworthy in relation to Appendices I - J. First, it seems counter-intuitive that the insertion operator with the second best impact (3-RI) is not explicitly included. A possible explanation might be that it offers a high functionality overlap with 2-RI and is thus beneficial in cases where 2-RI is missing. Secondly, it seems perplexing that the RR was not considered significant.

Our declared objective is to find the smallest possible model that guarantees robust solutions. Conclusively, we should select all operators with explicit include verdict by the CART and ignore all others. Nevertheless, we suspect that omitting a purely random component drastically reduces the general ability to come up with surprising solutions. The addition of RR is especially important considering it ensures that the trapping avoidance condition 3.1 holds true. The final selection of operators is thus RR, DR, NPR, SR, DiSR, 2-RI, 5-RI and β-Hybrid Insertion (β-HI).

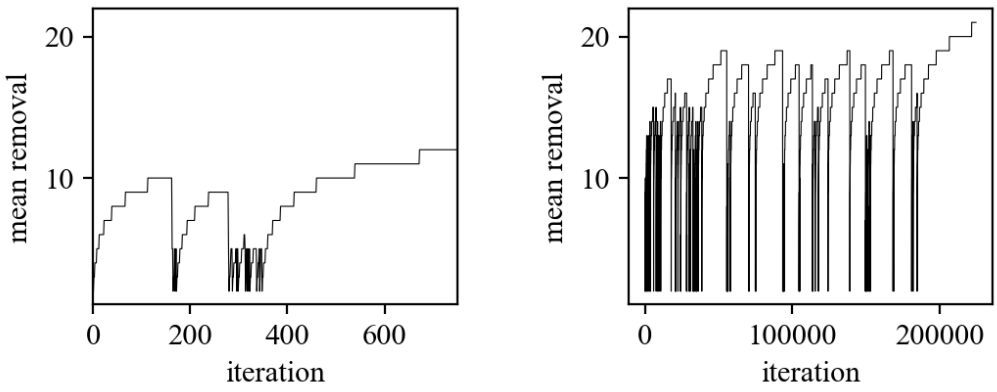
6.2 Computational analysis

After all parameters have been tuned, the behavior of the algorithm can be analyzed. The final parameter vector represents the results of our tuning routine: (c, T, ς, λ, ρ, σ_1, σ_2, σ_3, σ_4, σ_5, r, ζ, ϱ) = (0.9999, 0.01, 10, 3.35, 0.15, 50, 100, 90, 7, -80, 10, 0.35, 0.65). A tabular representation of the final selection is given in Appendix L. The first central observation for all solution attempts is that the number of final iterations is much higher than the 50 000 proposed by Ropke and Pisinger (2006) and can even reach up to 500 000. This indicates both that the new weighting procedure favoring simpler heuristics moves in smaller steps, but also that the solution space is not sufficiently explored after the iteration limit proposed by Ropke and Pisinger (2006).

6.2.1 Incremental shake-up

One of the key innovations of our implementation is the concept of incremental shake-up. The aim is to incrementally increase the size of neighborhoods in order to improve the directed intensification. In practice, this effect should only be triggered in later iterations with increasing stagnation.

We visualize an exemplary development of the average number of removed customers rm in Figure 6.5. The parameter values defined in Section 6.1 enable at minimum $rm = 2$ and at maximum $rm = 21$ for an instance of size $n = 200$. The shake-up speed increases logarithmically, which corresponds to the idea of increasing the number of search iterations with growing neighborhoods. However, the previous expectation that the shake-up procedure impacts the search behavior only in the later stages does not seem to be verified. The average number of removed customers rises to 15 in less than ∼5 000 iterations.



(a) Change of rm for the initial 750 iterations (b) Change of rm during local search

Figure 6.5: Change of rm during a search procedure

We additionally expect that the shake-up procedure will lead to an increasing difference between s and $\acute{s}$, especially throughout the intensification phase. Therefore, we visualize the Hamming distance of both for an exemplary run in Figure 6.6.

The Hamming distance is a measure to quantify solution similarity and is computed by comparing all incoming and outgoing edges for each customer in a solution. Every edge that differs increases the Hamming distance by 1. Accordingly, a higher number means increased difference and can reach a maximum of $2n$. Conversely, a distance value of 0 implies that both solutions are identical.

The expectation of increasing diversity with progressing shake-up seems to be verified given Figure 6.6. It is noteworthy that the upper bound of the Hamming distances is relatively constant. One explanation could be that some of the created neighborhoods overlap strongly after being centered around the most promising solution space.

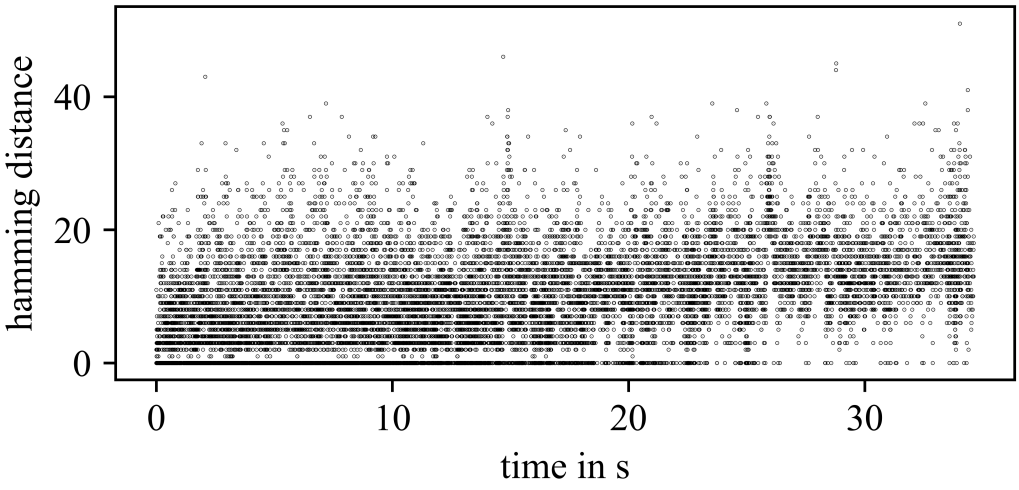


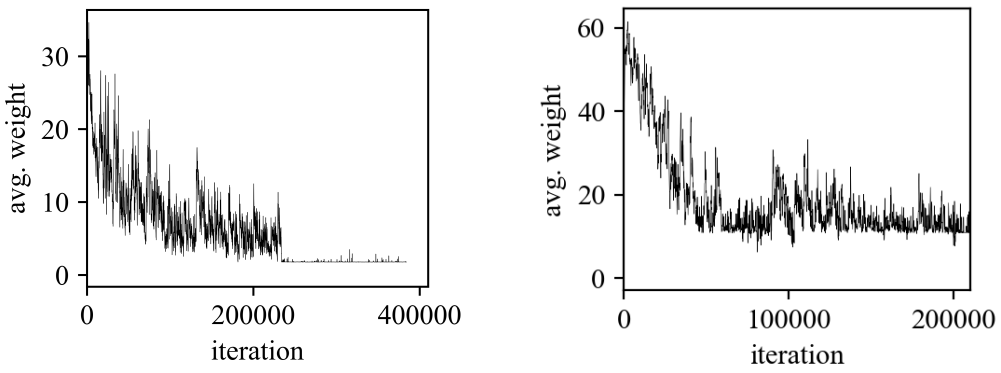
Figure 6.6: Hamming distance between current solution s and new solution $\acute{s}$.

6.2.2 Impact of time normalized reward weighting

We further want to investigate the impact of the new operator rewarding scheme in comparison to the prevalent iteration based concepts.

Figure 6.7 represents the average weight of all operators for both weighting alternatives throughout one exemplary solution run. The average weight decreases in both variants, which could be explained by the reducing SA factor ν. This factor discounts diversification (σ_3) and random acceptance (σ_2), and both new best (σ_1) as well as unique solutions (σ_4) become increasingly unlikely in the later stages.

A noticeable difference of both variants, however, can be detected in the minimum average weight. The iteration based variant does not seem to approach the minimum weight of 1. After a closer examination it appears that this can be traced back to the WoR which had a continuous weighting of 100 throughout the procedure. We hypothesize that two operators have entered a cycle in which the WoR regularly led differing solution aspects back to a known alternative with superior quality. This in return awarded the WoR with σ_2.



(a) Avg. time normalized operator weights (b) Avg. iter. normalized operator weights

Figure 6.7: Average operator weights during a search procedure.

Operator selection tendencies using iteration normalization

The hypothesis of a potential feedback loop is further supported by the representation in Appendices M - N. These appendices show the selection probability of different operators given an iteration based reward scheme, an approach employed for example in Ropke and Pisinger (2006).

We can observe relatively even distributed selection probabilities in the first 30 000 iterations, with light tendencies towards individual operators. This pattern changes with increasing number of iterations however and WoR and SR become prevalent. In later stages of the algorithm RR, RRR, DR and TR are almost completely ignored.

This observation is consistent with both assumptions that slow and verbose operators are preferred and that this preference then quickly returns the solution routine to the previous position.

An interesting phenomenon is that randomizing components gain popularity over a short period, but flatten out again after $\sim \Delta 30\,000$ iterations. A possible relation might be identified in the simultaneous increase of the 2-RI operator and the near disappearance of β-RI. It is also important to note that the selection probabilities of insertion operators seem less volatile with iteration based weighting.

Operator selection tendencies using time normalization

Weighting of operators changes drastically with a time normalized scheme. Both Appendix O and P present the selection probabilities of the operators using time normalization.

The previous dominance of the WoR is absent until all other operators are unable to exceed the minimum weight of 1. However, its growing popularity in the final phase might indicate solution circulation. Removal operators RR and RRR in particular benefit from the new scheme during the initial phase. As the discounting of the diversification bonus progresses, WiSR and SR gain popularity.

The volatility of the insertion operator weights is higher than in the iteration based weighting scheme. It is also important to note that the β-HI is hardly considered, whereas D-GI and 2-RI are popular despite their complexity.

We conclude that normalizing the weighting scheme of operators based on iteration time can be an important component to add operator selection diversity. It is, however, important to note that this visualization is based on a single run and might therefore not be representative.

6.2.3 Search behavior: Quality, diversity and acceptance

The dynamics of operator selection, solution acceptance, incremental shake-up, allowed infeasibility and other core components are difficult to distinguish from each other. However, we will try to illustrate how the quality and diversity of s, $\acute{s}$ and $\hat{s}$ changes throughout the adapted search behavior in our implementation.

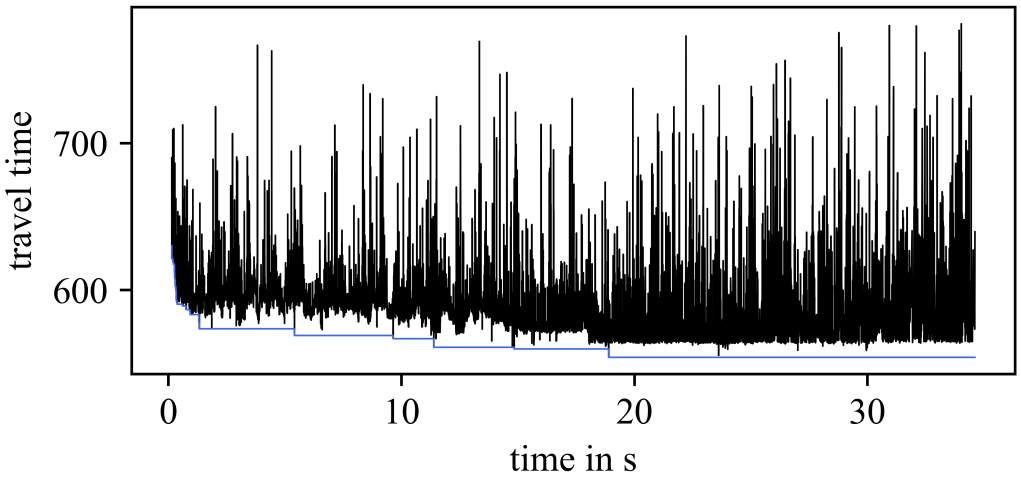
Diversity and quality of new solutions $\acute{s}$

We start by assessing the quality and diversity of new solutions $\acute{s}$. Both are illustrated in Figure 6.8, where sub-figure (a) represents the total travel time of the newly created solutions and (b) the diversity through Hamming distance. The blue line indicates the best best known solution value until that point.

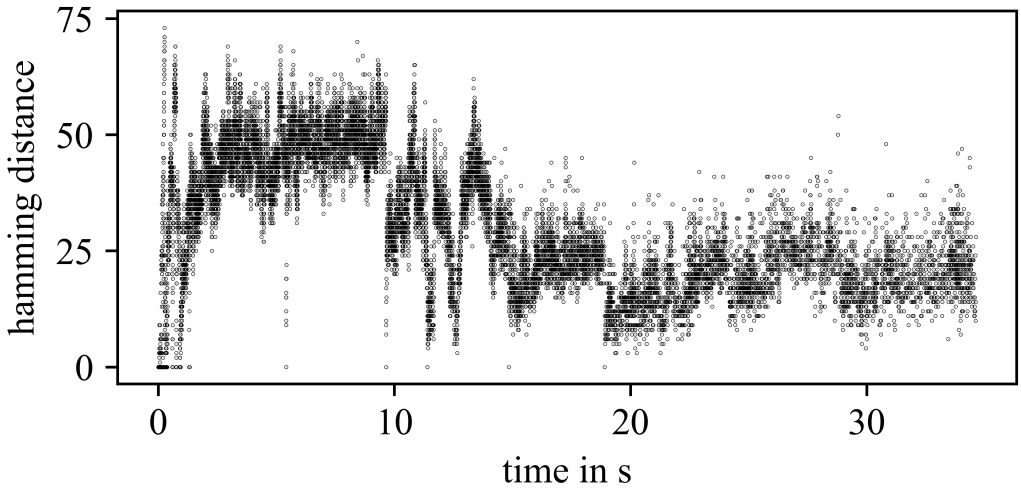
Graph (a) shows that the quality density is higher in the first iterations, but the average solution quality reduces throughout approximately half of the lifecycle. After that time the final and best solution $\hat{s}$ is determined. It is at that time as well that the quality density decreases and the number of diversity shifts in graph (b) reduces drastically.

Both effects can be attributed to the shake-up procedure, which additionally reverses the trend of decreasing solution diversity. Nevertheless, the search procedure always returns to solutions of comparable quality throughout the intensification phase. Additionally, the diversity does not increase at the same speed and stagnates at a moderate height.

We interpret the results to suggest that the idea of shake-up works as it was intended. The search space during intensification increases consistently, but remains centered around the most promising region. This might lead to multiple revisited solutions and limited solution diversity in the final stages. However, it also increases the chances of finding the corresponding local minimum.



(a) Travel time of the new solution over time



(b) Hamming distance between global best and new solution.

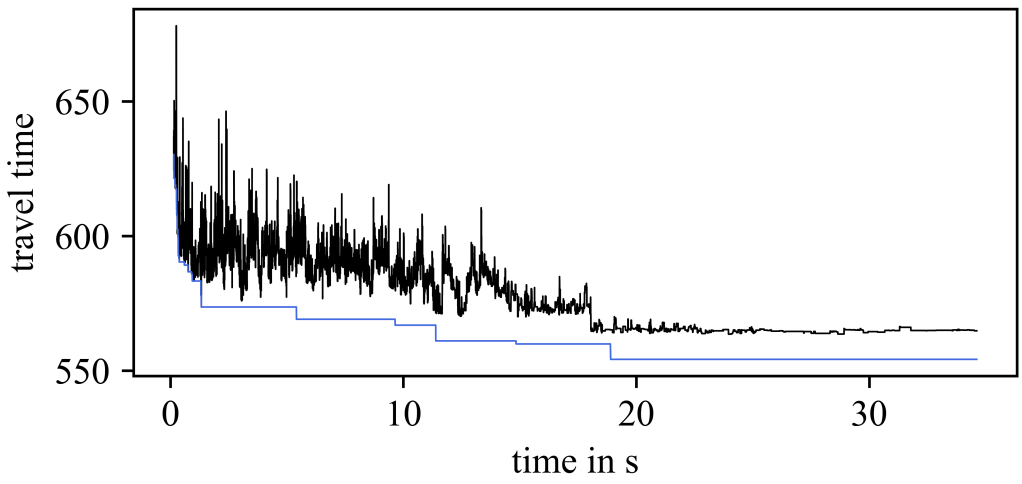
Figure 6.8: Diversity and quality of all generated solutions over time.

Diversity and quality of the current base solution s

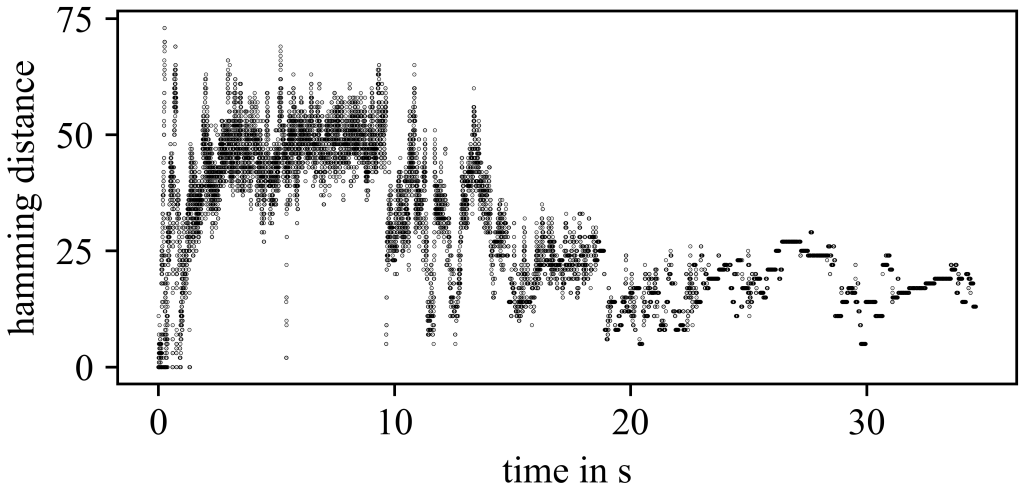
Finally, we are interested in the behavior of the currently accepted solution s. Thereby we expect to verify the anticipated acceptance behavior of the SA framework.

Similar to the previous section, the quality and diversity of s is shown in Figure 6.9. We can identify two general trends in (a). Firstly, the average quality of the currently accepted solution increases consistently and secondly, its standard deviation decreases. This observation is consistent with the fundamental concept of the exploration and intensification phases underlying the SA.

However, it is intriguing that the diversity of the accepted solution in later stages is comparably high, although the quality threshold for acceptance is strict. This suggests a multitude of visited solutions with similar quality.



(a) Travel time of the current solution over time



(b) Hamming distance between global best and current solution.

Figure 6.9: Diversity and quality of the current solution over time.

Conclusion of the computational analysis

The analysis of individual components confirmed that the expected effects of shake-up and time normalization are recognizable. We continue our assumption that the directed intensification increases consistency, despite not finding a new best solution in our example.

However, it is surprising how quickly rm is growing. The resulting destruction pattern might be difficult for simple insertion heuristics to recover, especially with increasing n. In addition, as the average weighting of operators decreases, individual operators seem to be disproportionately promoted.

Both findings point towards the need of other measures to prevent early stagnation. This could be achieved, for example, by permutation of the insertion operators, such as applied in Hemmelmayr et al. (2012).

6.3 Benchmarking

The following chapter provides an comprehensive computational benchmark analysis to enable the quality assessment of our implementation as introduced in 5.6. Throughout this section we report the performance on two variants: The first including all operators and the second only considering the reduced operator sub-set created in 6.1.7.

It is important to note that most articles are not precise on the measurement and aggregation of the reported execution times. In the following we assume that an experiment contains the calculation of all instances in one run. The execution time represented is the time needed for one experiment.

6.3.1 Benchmarking on Fontaine

This section summarizes the performance of the ALNSLDTT on the small scale instances of Fontaine (2019). Table 6.2 and 6.3 represent the average results of both considered operator sets created during 10 solution attempts. The data is divided into 5 groups, one for each city and the reported total travel time for all demand scenarios is averaged within that group for conciseness. Both tables report the optimal solution derived by Fontaine (2019) as well as the average and best solution values derived throughout the benchmarking procedure. Secondary evaluation criteria like number of iterations, execution time and number of vehicles used are reported as well. The row indexed CS reports the cumulated sum across all groups.

Table 6.2: Summary of the average performance of 10 runs with *all* operators on Fontaines instances (n=20)

Inst	**BKS**	**Avg**	**Gap%**	**Best**	**Gap%**	**Iters**	**Time (s)**	**Nr. Veh.**
	Opt.					Avg.	Avg.	Avg.
Fu	32.78	32.79	0.03	32.79	0.03	15559.0	15.56	5.33
Ma	35.74	35.74	0.01	35.74	0.01	17965.1	17.96	5.0
Pi	40.22	40.22	0.0	40.22	0.0	20104.92	20.1	5.0
Se	33.62	33.62	0.0	33.62	0.0	17919.73	17.92	4.83
Sy	23.8	23.8	0.01	23.8	0.0	16091.1	16.09	5.17
CS	166.16	166.17	0.06	166.17	0.05	87639.85	87.64	25.33

Table 6.3: Summary of the average performance of 10 runs with *selected* operators on Fontaines instances (n=20)

Inst	**BKS**	**Avg**	**Gap%**	**Best**	**Gap%**	**Iters**	**Time (s)**	**Nr. Veh.**
	Opt.					Avg.	Avg.	Avg.
Fu	32.78	32.79	0.03	32.79	0.03	15559.0	15.56	5.33
Ma	35.74	35.74	0.01	35.74	0.01	17965.1	17.96	5.0
Pi	40.22	40.22	0.0	40.22	0.0	20104.92	20.1	5.0
Se	33.62	33.62	0.0	33.62	0.0	17919.73	17.92	4.83
Sy	23.8	23.8	0.01	23.8	0.0	16091.1	16.09	5.17
CS	166.16	166.17	0.06	166.17	0.05	87639.85	87.64	25.33

It is evident that the ALNSLDTT finds the optimal solution for all instances in every repetition for both operator subsets. We suspect that the small deviation of the Fukuoka instances is due to an issue with floating point arithmetic. Regrettably,

a comparison on the calculation times is not possible, as these values were not reported by Fontaine (2019). Nevertheless, these results indicate competitiveness with exact solution procedures on small instances. We would like to refer the reader to Appendix Q and R for a non-aggregated view on the results.

6.3.2 Benchmarking on Solomon

A summary of the results obtained for the Solomon instances is shown in Table 6.5.

Contrary to the previous comparison, we omit the average and only present the best solutions generated within 10 runs. The results are averaged within six groups, each corresponding to a different customer distribution pattern. For every algorithm we report both the average number of vehicles in use and the average total travel distance. Entries containing the best average total travel distance within a row are highlighted in bold numbers. For each algorithm we furthermore present the Cumulated Number of Vehicles (CNV), the Cumulated Travel Distance (CTD) and the execution time in minutes. Please note that the CTD is equivalent to the cumulated travel time within the VRPTW.

All best solutions of the ALNSLDTT are at least as good or better when compared to any of the implementations considered. The variant containing only a selection of operators hereby was able to generate the overall best solution four times, the variant containing all only once.

However, it is important to emphasize that this is likely due to the modified objective function. A strong indication of this is also the CNV, which is significantly larger than reported in any other. Conclusively the total travel time improvements might be achieved through additional but underutilized routes.

Appendix S and T present all non-aggregated results and compare them to all known optimal solutions reported by Solomon (1987b). However, it is widely assumed that these results might be inaccurate due to rounding issues.

The measured time to generate the results is significantly higher compared to other implementations. Nevertheless, we would like to state that an average total runtime of the ALNSLDTT to solve one problem instance is less than 30 seconds and therefore within the scope of applicability.

All in all the results promise viability through both competitive results and execution time. However, is is not possible to conclusively state that the ALNSLDTT is better than alternative algorithms.

Table 6.5: Benchmarking selected operators Solomon (n=100)

Inst	**n**	**PisR**	**PDR**	**RTI**	**NB-100**	**NB-f(n)**	**HGSADC**	**ALNSLDTT**	
Runs		10	5	3	1	5	5	10 (all)	10 (sel.)
R1	100	11.92 | 1212.4	11.92 | 1210.3	11.92 | 1210.8	11.92 | 1210.3	11.92 | 1210.3	11.92 | 1210.7	13.28 | 1180.3	**13.25 | 1179.9**
R2	100	2.73 | 957.7	2.73 | 955.7	2.73 | 952.7	2.73 | 952.1	2.73 | 951.0	2.73 | 951.5	5.27 | 882.7	**5.27 | 879.5**
C1	100	10.0 | 828.4	10.0 | 828.4	10.0 | 828.4	10.0 | 828.4	10.0 | 828.4	10.0 | 828.4	10.0 | 828.4	**10.0 | 828.4**
C2	100	3.0 | 589.9	3.0 | 589.9	3.0 | 589.9	3.0 | 589.9	3.0 | 589.9	3.0 | 589.9	3.0 | 589.9	**3.0 | 589.9**
RC1	100	11.5 | 1385.8	11.5 | 1384.1	11.50 | 1384.3	11.50 | 1384.7	11.5 | 1384.1	11.5 | 1384.1	**13.04 | 1341.2**	12.88 | 1341.9
RC2	100	3.25 | 1123.5	3.25 | 1119.4	3.25 | 1119.7	3.25 | 1119.4	3.25 | 1119.2	3.25 | 1119.2	5.41 | 1010.06	**5.92 | 1005.2**
CNV		405	405	405	405	**405**	405	479	483
CTD		57 332	57 240	57 216	57 205	57 187	57 196	54 854	**54 782**
Time (min)*		10x2.5 min	5x30 min	3x17.9 min	3.2 min	5x5.0 min	5x2.68 min	10x26min	10x26min
Processor		P4-3G	Opt-2.3G	P4-3G	Opt-2.4G	Opt-2.4G	Xe-2.93G	Xe-2.1G	Xe-2.1G

6.3.3 Benchmarking on Gehring & Homberger

All results for the datasets of Gehring and Homberger (1999) are summarized in Tables 6.6-6.8. The aggregation and presentation of these results is identical to the tables in the previous section.

Multiple algorithmic tendencies are consistent with the insights gathered from the Solomon benchmarking. First, the ALNSLDTT is able to improve the current BKS of the total travel distance in multiple instances. Secondly, this improvement likely originates from the use of multiple additional underutilized vehicles. Generally, the variant with a reduced set of operators outperforms the variant including all.

However, our implementation is only able to set an improved CTD for instances of up to 400 customers. Competitiveness seems to decrease with increasing complexity. A possible explanation could be that all parameters are optimized for an instance size of up to 200. This could lead to a unreasonable high responsiveness of the algorithm. Parameters that impact the responsiveness and algorithmic search speed are c, T, ς, λ, r and ζ. Especially the high responsiveness of the shake-up procedure might cause severe disruptions of these large scale solutions which might not be recoverable with relatively simple insertion heuristics. This might be further investigated in future research.

Contrary to the decreasing competitiveness in solution values is the increasing competitiveness of solution speed. The previously slow execution time for small instances seems to scale better than most of the other algorithms in comparison. It is particularly atypical that the average calculation time for instances with 800 customers is longer than the average calculation time for 1000 customers. We associate this behavior with the degrading solution competitiveness, which might imply a premature termination of the algorithm. Based on our shake-up hypothesis, large disruptions could lead to limited diversity in the created solutions. Additionally, the absence of randomization in the insertion algorithms leads to a high consistency and repetition of potential insertion flaws. However, we cannot disregard the possibility of a server inconsistency during the computational analysis either.

Table 6.6: Best known solutions Gehring & Homberger 200-400

Inst	**n**	**PisR**	**PDR**	**RTI**	**NB-100**	**NB-f(n)**	**HGSADC**	**ALNSLDTT**	
Runs		1	5	3	1	5	5	5 (all)	5 (sel.)
R1	200	18.2 | 3631.23	18.2 | 3615.69	18.2 | 3640.11	18.2 | 3615.15	**18.2 | 3612.36**	18.2 | 3613.16	19.5 | 3618.18	19.8 | 3618.37
R2	200	4.0 | 2949.37	4.0 | 2937.67	4.0 | 2941.99	4.0 | 2930.04	4.0 | 2929.41	4.0 | 2929.41	6.6 | 2712.50	**8.0 | 2662.37**
C1	200	18.9 | 2721.52	18.9 | 2718.77	18.9 | 2721.90	18.9 | 2718.44	18.9 | 2718.41	18.9 | 2718.41	9.7 | 2679.91	**19.7 | 2679.52**
C2	200	6.0 | 1832.95	6.0 | 1831.59	6.0 | 1833.36	6.0 | 1831.64	6.0 | 1831.64	**6.0 | 1831.59**	6.4 | 1835.04	6.9 | 1835.26
RC1	200	18.0 | 3212.28	18.0 | 3192.56	18.0 | 3224.63	18.0 | 3182.48	**18.0 | 3178.68**	18.0 | 3180.48	19.0 | 3187.00	18.8 | 3186.96
RC2	200	4.3 | 2556.87	4.3 | 2559.32	4.3 | 2554.33	4.3 | 2536.54	4.3 | 2536.22	4.3 | 2536.20	6.7 | 2322.36	**7.2 | 2313.40**
CNV		694	694	694	694	**694**	694	779	804
CTD		169 042	168 556	169 163	168 143	168 067	168 092	163 550	**162 959**
Time (min)		5x7.7min	5x53min	90min	4.7min	5x4.1min	5x45.16min	5x45min	5x40min
R1	400	36.4 | 8540.04	36.4 | 8420.52	36.4 | 8514.11	36.4 | 8413.23	36.4 | 8403.24	**36.4 | 8402.57**	38.0 | 8573.49	37.8 | 8535.14
R2	400	8.0 | 6241.72	8.0 | 6213.48	8.0 | 6258.82	8.0 | 6149.49	8.0 | 6148.57	8.0 | 6152.92	10.0 | 6093.40	**13.5 | 5825.42**
C1	400	37.6 | 7290.16	37.6 | 7182.75	37.6 | 7273.90	37.6 | 7179.71	37.6 | 7175.72	37.6 | 7170.47	39.0 | 7059.69	**39.0 | 7046.06**
C2	400	12.0 | 3844.69	**11.9 | 3874.58**	11.7 | 3941.70	11.7 | 3898.02	11.7 | 3899.00	11.6 | 3952.95	12.3 | 3899.70	12.8 | 3889.59
RC1	400	36.0 | 8069.30	36.0 | 7940.65	36.0 | 8088.46	36.0 | 7931.66	36.0 | 7922.23	**36.0 | 7907.14**	37.1 | 8086.73	37.2 | 8046.68
RC2	400	8.5 | 5335.09	8.6 | 5269.09	8.4 | 5516.59	8.4 | 5293.74	8.4 | 5297.86	8.5 | 5215.21	10.9 | 5090.12	**12.6 | 5013.01**
CNV		1385	1385	1381	1381	1381	**1381**	1473	1529
CTD		393 210	389 011	395 936	388 548	388 466	388 013	388 031	**383 559**
Time (min)*		5x15.8min	5x89min	180min	34.0min	5x16.2min	5x34.1min	5x75min	5x62min
Processor		P4-3G	P4-2.8G	Opt-2.3G	P4-3G	Opt-2.4G	Xe-2.93G	Xe-2.1G	Xe-2.1G

Table 6.7: Best known solutions Gehring & Homberger 600-800

Inst	**n**	**PisR**	**PDR**	**RTI**	**NB-100**	**NB-f(n)**	**HGSADC**	**ALNSLDTT**	
Runs		5	5	3	1	5	5	5 (all)	5 (sel.)
R1	600	54.5 | 18888	54.5 | 18252	54.5 | 18781	54.5 | 18194	54.5 | 18186	**54.5 | 18023**	56.0 | 18644	55.8 | 18437
R2	600	11.0 | 12619	11.0 | 12808	11.0 | 12804	11.0 | 12319	11.0 | 12330	11.0 | 12352	13.3 | 12443	**19.4 | 11832**
C1	600	57.5 | 14065	57.4 | 14106	57.3 | 14236	**57.4 | 14054**	57.4 | 14067	57.4 | 14058	58.6 | 14222	58.8 | 14132
C2	600	17.5 | 7801 6	17.5 | 7632	17.4 | 7729	17.4 | 7601	17.4 | 7605	17.4 | 7594	19.3 | 7653	**19.3 | 7545**
RC1	600	55.0 | 16594	55.0 | 16266	55.0 | 16767	55.0 | 16179	55.0 | 16183	**55.0 | 16097**	56.0 | 16606	56.3 | 16521
RC2	600	11.6 | 10777	11.7 | 10990	11.4 | 11311	11.4 | 10591	11.4 | 10586	11.5 | 10511	14.4 | 10438	**18.5 | 10237**
CNV		2068	2071	**2066**	2067	2067	2068	2176	2281
CTD		802 681	800 797	816 326	789 420	789 592	**786 373**	800 086	787 080
Time		646.9min	5x105min	270min	80.4min	5x25.3min	5x99.4min	5x101min	5x78min
R1	800	72.8 | 32316	72.8 | 31797	72.8 | 32734	72.8 | 31486	72.8 | 31492	**72.8 | 31311**	74.0 | 32956	74.9 | 33040
R2	800	15.0 | 20353	15.0 | 20651	15.0 | 20618	15.0 | 19873	15.0 | 19914	15.0 | 19933	16.3 | 21052	**25.5 | 19797**
C1	800	75.6 | 25193	75.4 | 25093	75.2 | 25911	75.3 | 24990	75.2 | 25151	**75.4 | 24876**	77.6 | 25648	77.5 | 25357
C2	800	23.7 | 11725	23.5 | 11569	23.4 | 11835	**23.4 | 11438**	23.4 | 11447	23.3 | 11475	26.0 | 12350	26.7 | 12021
RC1	800	73.0 | 29478	72.0 | 33170	72.0 | 33795	72.0 | 31020	72.0 | 31278	**72.0 | 29404**	74.0 | 30116	74.1 | 29809
RC2	800	15.7 | 16761	15.8 | 16852	15.5 | 17536	15.4 | 16438	15.4 | 16484	15.4 | 16495	18.5 | 16796	**23.3 | 16384**
CNV		2758	2745	2739	2739	**2738**	2739	2908	3020
CTD		1 358 291	1 391 344	1 424 321	1 352 478	1 357 695	**1 334 963**	1 389 220	1 364 102
Time (min)*		5x22.7min	5x129min	360min	126.8min	5x27.6min	5x215min	10x93min	10x87min
Processor		P4-3G	P4-2.8G	Opt-2.3G	P4-3G	Opt-2.4G	Xe-2 G	Xe-2.1G	Xe-2.1G

Table 6.8: Best known solutions Gehring & Homberger 1000

Inst	**n**	**PisR**	**PDR**	**RTI**	**NB-100**	**NB-f(n)**	**HGSADC**	**ALNSLDTT**	
Runs		1	5	3	1	5	5	5 (all)	5 (sel.)
R1	1000	92.2 | 50751	91.9 | 49702	91.9 | 51414	91.9 | 48287	91.9 | 48369	**91.9 | 47759**	93.8 | 50785.33	93.7 | 49911.63
R2	1000	19.0 | 29780	19.0 | 30495	19.0 | 30804	**19.0 | 28913**	19.0 | 29003	19.0 | 29076	20.3 | 32075.76	31.9 | 29272.49
C1	1000	94.6 | 41877	94.3 | 41783	94.2 | 43111	94.1 | 41683	94.1 | 41748	**94.1 | 41572**	98.4 | 43821.25	97.7 | 42747.32
C2	1000	29.7 | 16840	29.5 | 16657	29.3 | 16810	**29.1 | 16498**	29.1 | 16534	28.8 | 16796	33.5 | 18618.70	33.4 | 17621.84
RC1	1000	90.0 | 46752	90.0 | 45574	90.0 | 46753	90.0 | 44743	90.0 | 44860	**90.0 | 44333**	91.7 | 47437.11	94.4 | 48228.00
RC2	1000	18.3 | 25090	18.5 | 25470	18.4 | 25588	**18.3 | 23939**	18.3 | 24055	18.2 | 24131	20.9 | 26049.08	27.3 | 24394.65
CNV		3438	3432	3428	3424	3424	**3420**	3586	3784
CTD		2 110 925	2 096 823	2 144 830	2 040 661	2 045 720	**2 036 700**	2 187 872	2 121 759
Time (min)*		5x26.6min	5x162min	450min	186.4min	5x35.3min	5x349min	5x68.5min	5x78.3min
Processor		P4-3G	P4-2.8G	Opt-2.3G	P4-3G	Opt-2.4G	Xe-2 G	Xe-2.1G	Xe-2.1G

6.3.4 Benchmarking on new instances

Finally, we want to provide the first results for the newly introduced instances in order to improve the comparability of possible subsequent approaches. The results using all operators are summarized in Table 6.9, while Table 6.10 represents the results for the reduced operator sub-set. Both tables report the average performance grouped by the city and instance size. We state both the average solution quality and the best solution of all 10 iterations. The latter is adopted as the current BKS and used as basis to compute the gap between the average. Finally we report the secondary criteria execution time, iterations and the number of vehicles used.

Similar to the VRPTW instances, the variant with the selected operators performs better on average. Furthermore, the solution speed for VRPLDTT instances is similar to the time needed for VRPTW instances. Last but not least, an average gap of $\sim$0.6% indicates high consistency.

Interestingly, the number of iterations for the 200 customer instances is around 60 000. This figure contrasts with the previous observation in our component analysis, in which a significantly larger iteration count was common. However, we already mentioned in that section that these observations are solely based on one instance and therefore may not be universally valid. A detailed overview of all results on instance basis is presented in Appendices U and V.

Table 6.9: Average performance summary on new instances (all ops.)

Inst	**n**	**BKS** Best 10	**Avg**	**Gap%**	**Iters** Avg. 10	**Time (s)** Avg. 10	**Nr. Veh.** Avg. 10
Fu	50	292.98	294.09	0.38	22457.13	22.46	6.37
Fu	100	567.3	572.52	0.92	35974.87	35.97	13.0
Fu	200	1024.26	1034.59	1.0	66990.83	66.99	24.83
Ma	50	43.79	43.81	0.04	31463.7	31.46	6.63
Ma	100	82.41	82.94	0.64	35210.2	35.21	12.63
Ma	200	149.78	151.45	1.12	60576.17	60.58	25.03
Pi	50	120.76	121.15	0.32	19119.3	19.12	6.37
Pi	100	217.35	218.84	0.68	36400.53	36.4	12.23
Pi	200	385.54	388.56	0.78	60671.9	60.67	24.63
Se	50	116.99	117.67	0.58	20184.63	20.18	6.33
Se	100	203.32	204.21	0.44	30303.83	30.3	12.73
Se	200	367.98	370.78	0.77	54985.2	54.99	24.77
Sy	50	244.87	247.18	0.94	20377.27	20.38	6.47
Sy	100	456.57	461.82	1.15	33156.07	33.16	12.37
Sy	200	807.91	816.49	1.06	69354.8	69.36	24.87
CS		5081.81	5126.12	10.84	597226.43	597.22	219.27

Table 6.10: Average performance summary on new instances (sel. ops.)

Inst	**n**	**BKS** Best 10	**Avg**	**Gap%**	**Iters** Avg. 10	**Time (s)** Avg. 10	**Nr. Veh.** Avg. 10
Fu	50	293.02	294.02	0.34	21382.73	21.38	6.3
Fu	100	568.33	572.09	0.66	31946.67	31.95	13.07
Fu	200	1022.06	1032.06	0.97	62944.27	62.94	24.83
Ma	50	43.79	43.8	0.04	28029.3	28.03	6.73
Ma	100	82.41	82.63	0.27	36943.87	36.94	12.5
Ma	200	149.65	150.72	0.71	56255.63	56.26	25.13
Pi	50	120.75	120.93	0.15	21152.4	21.15	6.33
Pi	100	217.37	218.73	0.62	35051.37	35.05	12.23
Pi	200	382.52	387.02	1.19	55904.83	55.9	24.5
Se	50	117.09	117.58	0.42	20471.33	20.47	6.17
Se	100	203.63	204.09	0.23	31083.2	31.08	12.53
Se	200	366.38	368.75	0.65	52855.7	52.86	24.57
Sy	50	244.87	247.97	1.26	26136.9	26.14	6.57
Sy	100	456.49	459.84	0.73	33663.43	33.67	12.47
Sy	200	802.99	810.92	0.99	65307.53	65.31	24.77
CS		5071.34	5111.14	0.616		579.14	218.7

7 Conclusion

The growing relevance of cargo bikes as a last mile transportation mode increasingly necessitates the consideration of its load and slope dependent travel speed.

The VRPLDTT incorporating this aspect is like most VRPs NP-hard. Therefore, the application of exact approaches is limited in realistic scenarios and heuristic approaches are currently indispensable. Potential interested parties are cities, retailers and delivery services that either participate in or regulate the urban last mile delivery process.

Our algorithm presented in this paper is based on the ALNS meta-heuristic and incorporates state of the art components from previous implementations while eliminating historical weaknesses. New concepts add purposeful diversification, increasingly thorough intensification and performance per time unit optimization. These components have been analyzed during a comprehensive computational analysis, which proved that these can be a sensible extension to the existing framework.

We were able to reliably find optimal solutions for VRPLDTT instances with up to 50 customers and guarantee a high quality for instances with up to 100 customers. The implementation also demonstrated competitive results for the VRPTW, although a deviating target function prevents a one on one BKS comparison. However, the success of the implementation decreases with an increasing number of customers and is no longer competitive for instances with 500 or more customers.

Despite these promising results, we had to recognize that the new concepts in particular benefit from new and surprising solutions. Insertion heuristics that both have limited solution variance and are too weak to recover heavily destroyed solutions limit the added value of the incremental shake-up drastically. We would therefore like to encourage future authors to continue the use of random permutation in both destroy and insertion operators. Additionally, we identified the possibility that some operators are disproportionately frequent selected during all stages of the local search. One potential extension to avoid this shortcoming could be a conditional tabu list, but this and other approaches should be subject of future research.

Bibliography

Adulyasak, Yossiri, Jean-François Cordeau, Raf Jans. 2012. Optimization-based adaptive large neighborhood search for the production routing problem. *Transportation Science* **48**(1) 20–45.

Aksen, Deniz, Onur Kaya, F Sibel Salman, Özge Tüncel. 2014. An adaptive large neighborhood search algorithm for a selective and periodic inventory routing problem. *European Journal of Operational Research* **239**(2) 413–426.

Bektas, T., T.G. Crainic, T. Van Woensel. 2015. From managing urban freight to smart city logistics networks. Project report. URL `https://eprints.soton.ac.uk/380790/`. [Online; accessed 12.12.2018].

Bektaş, Tolga, Gilbert Laporte. 2011. The pollution-routing problem. *Transportation Research Part B: Methodological* **45**(8) 1232–1250.

Bianchi, Leonora, Marco Dorigo, Luca Maria Gambardella, Walter J Gutjahr. 2009. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing* **8**(2) 239–287.

Bikes at Work. 2012. How much weight can a bicycle carry. URL `https://www.bikesatwork.com/blog/how-much-weight-can-a-bicycle-carry`. [Online; accessed 01.12.2018].

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

Breiman, Leo. 2017. *Classification and regression trees*. Routledge.

Chinese State Council. 2018. Three-year action plan for winning the blue sky war. URL `http://www.gov.cn/zhengce/content/2018-07/03/content_5303158.htm`. [Online; accessed 08.12.2018].

Coelho, Leandro C, Jean-François Cordeau, Gilbert Laporte. 2012. The inventory-routing problem with transshipment. *Computers & Operations Research* **39**(11) 2537–2548.

Cormen, Thomas H, Charles E Leiserson, Ronald L Rivest, Clifford Stein. 2009. *Introduction to algorithms*. MIT press.

Crainic, Teodor Gabriel, Nicoletta Ricciardi, Giovanni Storchi. 2009. Models for evaluating and planning city logistics systems. *Transportation science* **43**(4) 432–454.

Cuevas, Erik, Daniel Zaldívar, Marco Pérez-Cisneros. 2018. *Advances in Metaheuristics Algorithms: Methods and Applications*, vol. 775. Springer.

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

Dekker, Rommert, Jacqueline Bloemhof, Ioannis Mallidis. 2012. Operations research for green logistics–an overview of aspects, issues, contributions and challenges. *European Journal of Operational Research* **219**(3) 671–679.

Demir, Emrah, Tolga Bektaş, Gilbert Laporte. 2012. An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research* **223**(2) 346–359.

DESA, UN. 2017. World population prospects, the 2017 revision. *New York United Nations Department of Economic & Social Affairs* .

DESA, UN, et al. 2014. World urbanization prospects, the 2011 revision. *Population Division, Department of Economic & Social Affairs, United Nations Secretariat* .

Ehmke, Jan Fabian, Ann Melissa Campbell, Barrett W Thomas. 2016. Vehicle routing to minimize time-dependent emissions in urban areas. *European Journal of Operational Research* **251**(2) 478–494.

Eiben, Agoston E, Selmar K Smit. 2011. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation* **1**(1) 19–31.

European Commission. 2018. Air quality: Commission takes action to protect citizens from air pollution. URL `http://europa.eu/rapid/press-release_IP-18-3450_en.htm`. [Online, accessed 28.04.2019].

Fontaine, Pirmin. 2019. The vehicle routing problem with load-dependent travel times for cargo bike routing. Internal pre-print of the Technical University of Munich [accessed 01.01.2019].

Franceschetti, Anna, Emrah Demir, Dorothée Honhon, Tom Van Woensel, Gilbert Laporte, Mark Stobbe. 2017. A metaheuristic for the time-dependent pollution-routing problem. *European Journal of Operational Research* **259**(3) 972–991.

Gehring, Hermann, Jörg Homberger. 1999. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. *Proceedings of EUROGEN99*, vol. 2. Citeseer, 57–64.

Glover, Fred, Jin-Kao Hao. 2011. The case for strategic oscillation. *Annals of Operations Research* **183**(1) 163–173.

Hemmelmayr, Vera C, Jean-François Cordeau, Teodor Gabriel Crainic. 2012. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & operations research* **39**(12) 3215–3228.

Huang, Yixiao, Lei Zhao, Tom Van Woensel, Jean-Philippe Gross. 2017. Time-dependent vehicle routing problem with path flexibility. *Transportation Research Part B: Methodological* **95** 169–195.

Iglewicz, Boris, David Caster Hoaglin. 1993. *How to detect and handle outliers*, vol. 16. Asq Press.

ISO. 2017. *ISO/IEC 14882:2017 Information technology — Programming languages — C++*. 5th ed. URL `https://www.iso.org/standard/68564.html`.

Jabali, Ola, T Van Woensel, AG De Kok. 2012. Analysis of travel times and co2 emissions in time-dependent vehicle routing. *Production and Operations Management* **21**(6) 1060–1074.

Jones, Eric, Travis Oliphant, Pearu Peterson, et al. 2001–. SciPy: Open source scientific tools for Python. URL `http://www.scipy.org/`. [Online; accessed 17.04.2019].

Kallehauge, Brian. 2008. Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers & Operations Research* **35**(7) 2307–2330.

Karp, Richard M. 1972. Reducibility among combinatorial problems. *Complexity of computer computations*. Springer, 85–103.

Labadie, Nacima, Christian Prins, Caroline Prodhon. 2016. *Metaheuristics for Vehicle Routing Problems*. John Wiley & Sons.

Laporte, Gilbert. 2009. Fifty years of vehicle routing. *Transportation Science* **43**(4) 408–416.

Laporte, Gilbert, Roberto Musmanno, Francesca Vocaturo. 2010. An adaptive large neighbourhood search heuristic for the capacitated arc-routing problem with stochastic demands. *Transportation Science* **44**(1) 125–135.

Lin, Canhong, King Lun Choy, George TS Ho, Sai Ho Chung, HY Lam. 2014. Survey of green vehicle routing problem: past and future trends. *Expert Systems with Applications* **41**(4) 1118–1138.

Liu, Ran, Zhibin Jiang. 2018. A constraint relaxation-based algorithm for the load-dependent vehicle routing problem with time windows. *Flexible Services and Manufacturing Journal* 1–23.

Malandraki, Chryssi, Mark S Daskin. 1992. Time dependent vehicle routing problems: Formulations, properties and heuristic algorithms. *Transportation science* **26**(3) 185–200.

Masson, Renaud, Fabien Lehuédé, Olivier Péton. 2013. An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Science* **47**(3) 344–355.

Murty, Katta G. 1983. *Linear programming*. Springer.

Nakariakov, Sergei. 2013. The boost c++ libraries: Generic programming .

Oliphant, Travis. 2006–. NumPy: A guide to NumPy. USA: Trelgol Publishing. URL `http://www.numpy.org/`. [Online; accessed 19.04.2019].

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research* **12** 2825–2830.

Pisinger, David, Stefan Ropke. 2007. A general heuristic for vehicle routing problems. *Computers & operations research* **34**(8) 2403–2435.

Pisinger, David, Stefan Ropke. 2010. Large neighborhood search. *Handbook of metaheuristics*. Springer, 399–419.

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

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

Repoussis, Panagiotis P, Christos D Tarantilis, George Ioannou. 2009. Arc-guided evolutionary algorithm for the vehicle routing problem with time windows. *IEEE Transactions on Evolutionary Computation* **13**(3) 624–647.

Ropke, S. 2003. A local search heuristic for the pickup and delivery problem with time windows. *Technical paper. Copenhagen, Denmark: DIKU, University of Copenhagen* .

Ropke, Stefan, David Pisinger. 2006. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science* **40**(4) 455–472.

Shaw, Paul. 1997. A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK* .

Shaw, Paul. 1998. Using constraint programming and local search methods to solve vehicle routing problems. *International conference on principles and practice of constraint programming*. Springer, 417–431.

Siarry, Patrick. 2016. *Metaheuristics*. Springer.

Solomon, Marius M. 1987a. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research* **35**(2) 254–265.

Solomon, Marius M. 1987b. Vrptw benchmark problems. URL `http://w.cba.neu.edu/~msolomon/problems.htm`. [Online; accessed 28.04.2019].

Suzuki, Yoshinori. 2011. A new truck-routing approach for reducing fuel consumption and pollutants emission. *Transportation Research Part D: Transport and Environment* **16**(1) 73–77.

Talbi, El-Ghazali. 2009. *Metaheuristics: from design to implementation*, vol. 74. John Wiley & Sons.

Toth, Paolo, Daniele Vigo. 2014. *Vehicle routing: problems, methods, and applications*. SIAM.

Vidal, Thibaut, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, Walter Rei. 2012. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research* **60**(3) 611–624.

Vidal, Thibaut, Teodor Gabriel Crainic, Michel Gendreau, Christian Prins. 2013. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & operations research* **40**(1).

Wrighton, Susanne, Karl Reiter. 2016. Cyclelogistics–moving europe forward! *Transportation Research Procedia* **12** 950–958.

Zachariadis, Emmanouil E, Christos D Tarantilis, Chris T Kiranoudis. 2015. The load-dependent vehicle routing problem and its pick-up and delivery extension. *Transportation Research Part B: Methodological* **71** 158–181.

Appendix

A Appendix: Notation of the linear model describing the VRPLDTT

Table 7.1: Notation of the linear model describing the VRPLDTT

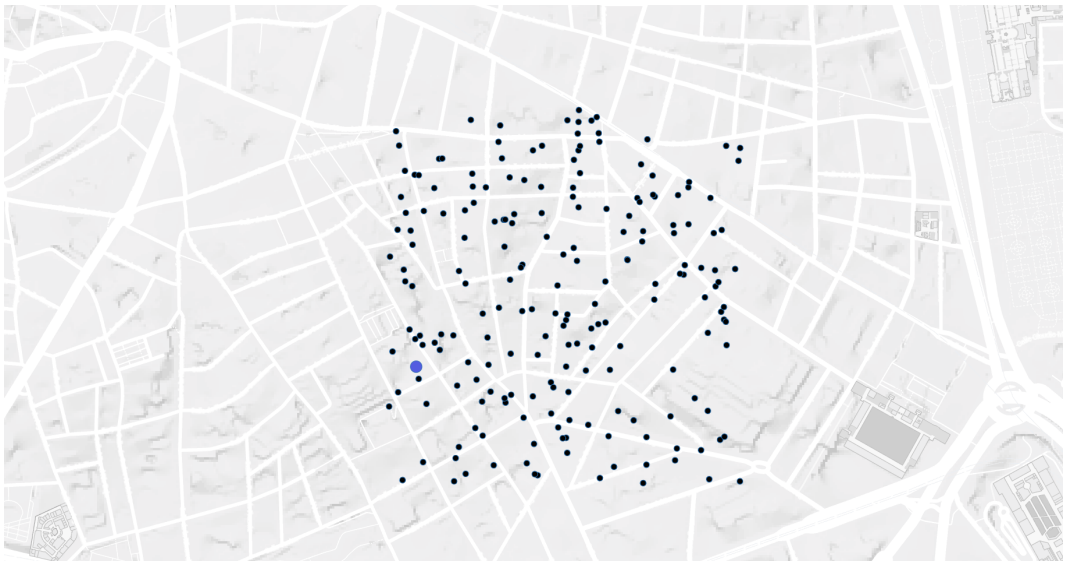
Indices		Description
$i, j \in$	$N = \{0, ..., n\}$	Set of nodes i or j with 0 being the depot
	$N_1 = \{1, ..., n\}$	Set of customer nodes
$(i, j) \in$	A	Set of arcs (i, j)
	$K = \{0, ..., m\}$	Set of homogeneous vehicles
$l \in$	L	Set of discrete load levels

Variables	Description
x_{ij}	Binary decision variable indicating if a vehicle travels on arc (i, j)
y_i	Visiting time of customer i
z_{ij}^l	Binary decision variable indicating if a vehicle travels from node i to node j with load l
f_{ij}	Continuous decision variable indicating the flow on arc (i, j)

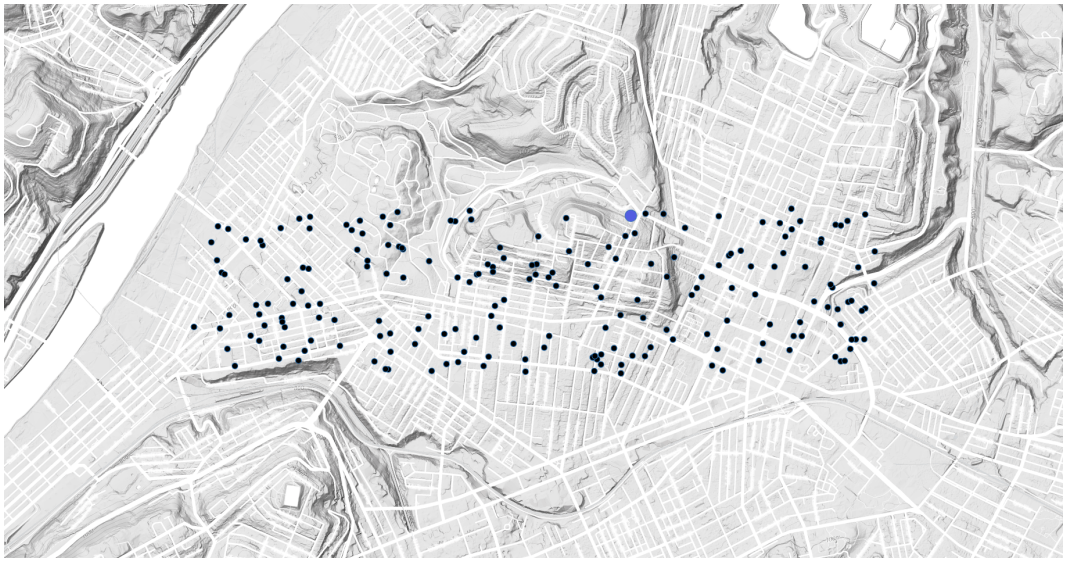
Parameters	Description
M_{ij}	Lower bound of the visiting time between node i and j
Q	Max. capacity of a vehicle
q_i	Requested demand of customer i
s_i	Service times of customer i
t_{ij}^l	Travel time for $arc(i, j)$ to node j with load l
$[a_i, b_i]$	Time window at which customer i must be visited a_i is the earliest and b_i is the latest possible time
$[p^l, r^l]$	Load interval defining the load bucket l This is needed for the speed calculation p^l is the lower bound and r^l the upper bound

B Appendix: Maps of all new benchmarking instances

(a) Customer distribution Fukuoka

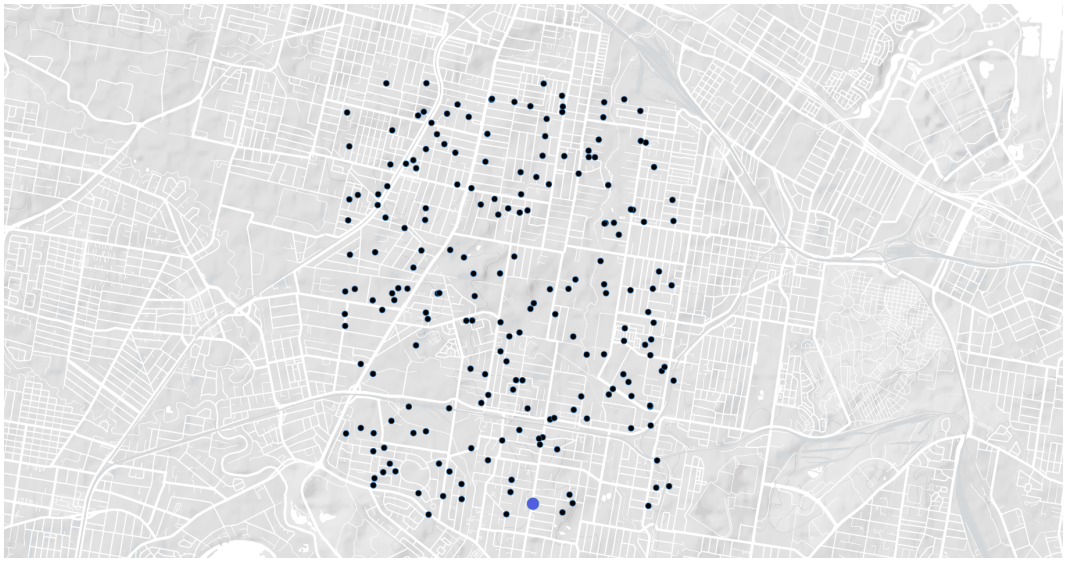


(b) Customer distribution Madrid



(c) Customer distribution Pittsburgh

(d) Customer distribution Seattle



(e) Customer distribution Sydney

Figure 7.1: Maps of all new benchmarking instances

C Appendix: Performance of all T and c combinations

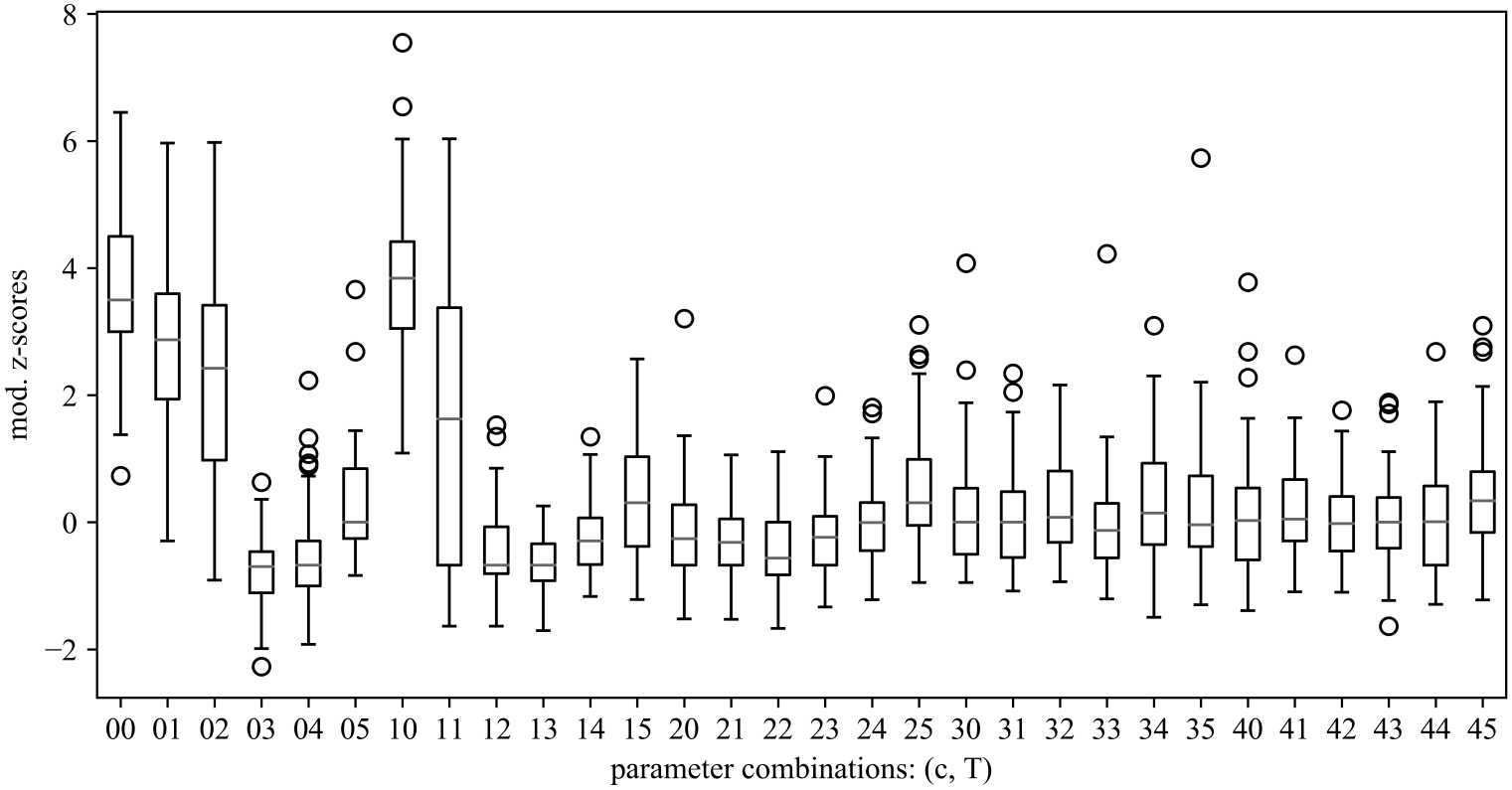


Figure 7.2: All box-plots for all parameter combinations of c and T. Combinations are described in code. The first position is $c = \{0.9999, 0.99975, 0.999, 0.995, 0.99\}$ second is $T = \{100, 1, 0.1, 0.01, 0.001, 0.0001\}$

D Appendix: Decision space for changing ς and λ combinations

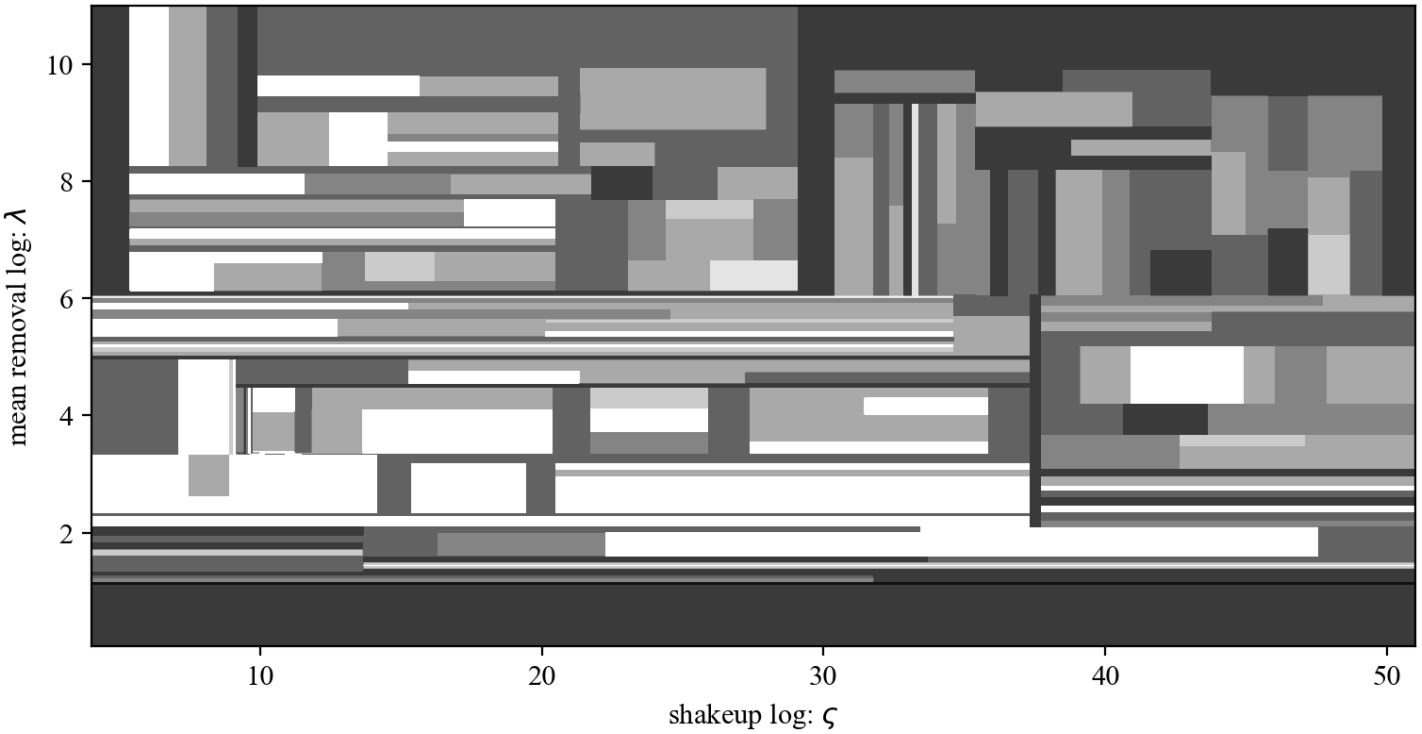
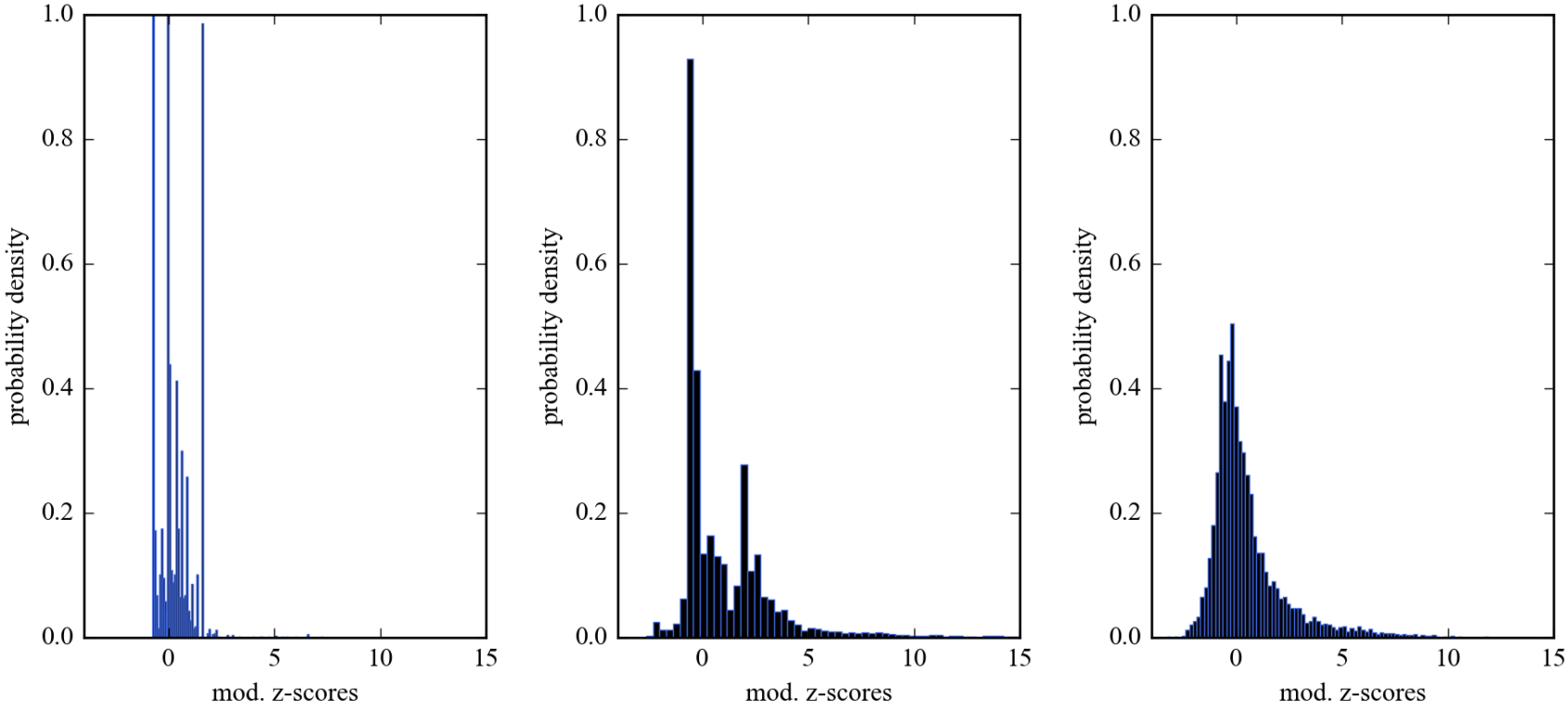


Figure 7.3: Visualization of the decision space of the tree learner. Bright areas indicate good performance and dark areas bad performance.

E Appendix: Performance consistency for differing instance sizes



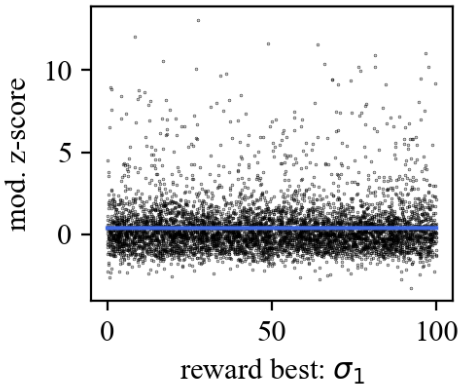
(a) Instance size: 50 customers

(b) Instance size: 100 customers

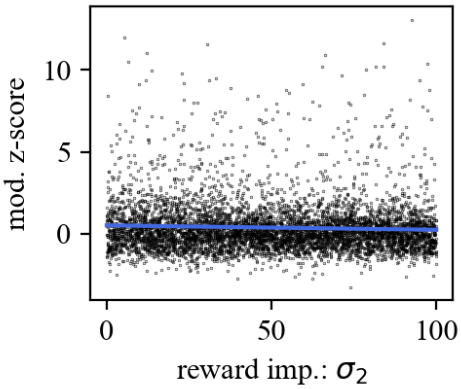
(c) Instance size: 200 customers

Figure 7.4: Normalized histograms showing the modified z-score distributions of different instance sizes. The median of each distribution is 0. The amount of blue correlates with the bucket size of the histogram.

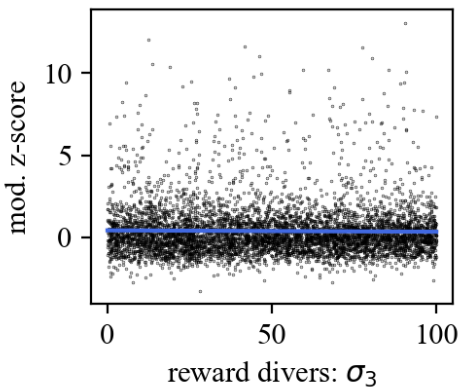
F Appendix: Operator weights (σ) performance distributions



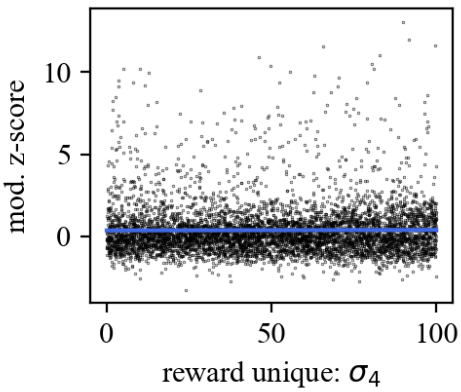
(a) σ_1: Reward for new best solution



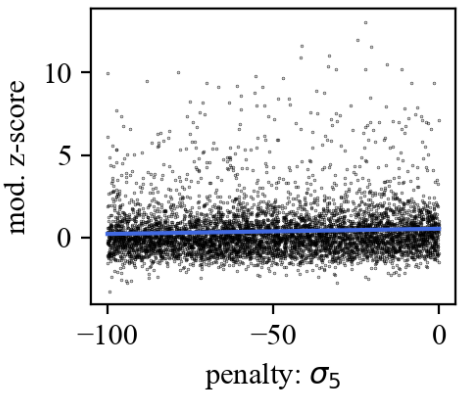
(b) σ_2: Reward for new better solution



(c) σ_3: Reward for high diversity



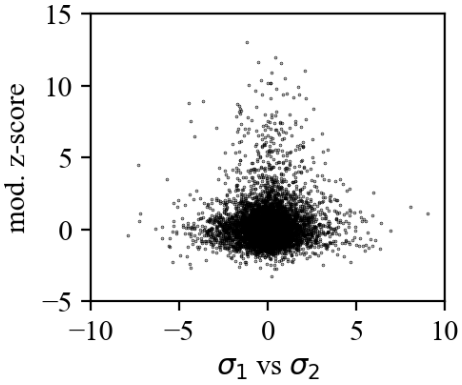
(d) σ_4: Reward for new unique solution



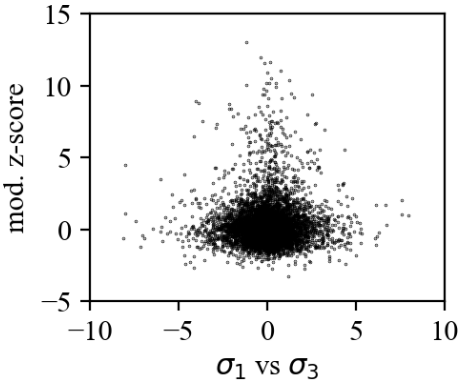
(e) σ_5: Penalty for worse solution

Figure 7.5: Modified z-score distributions of all σ settings with random draw. The blue line is a regression line of first order.

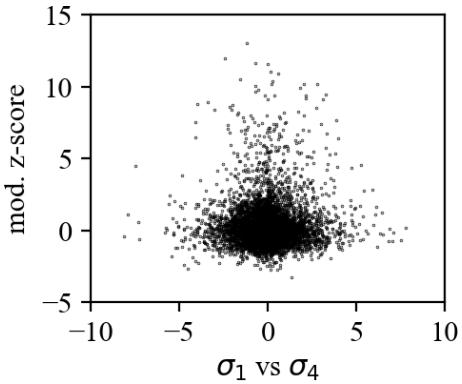
G Appendix: Operator weights (σ) relativity analysis



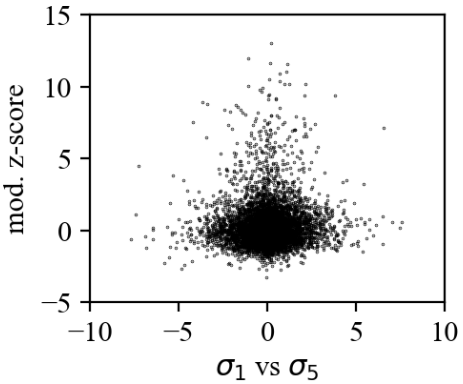
(a) Reward best (σ_1) **vs** Reward imp. (σ_2)



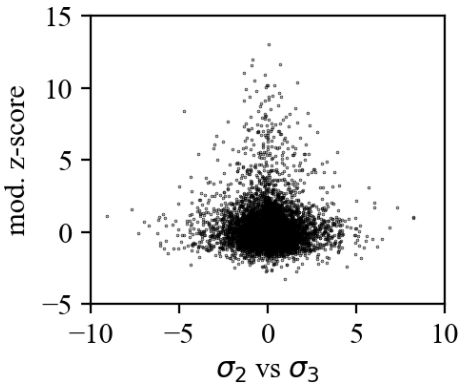
(b) Reward best (σ_1) **vs** Reward div. (σ_3)



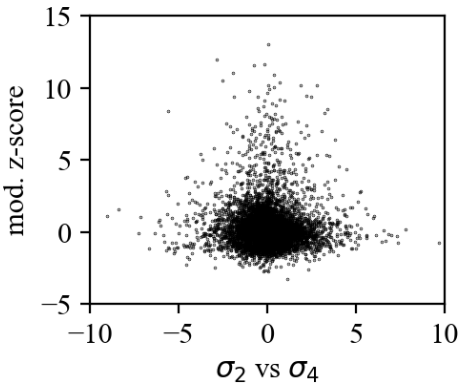
(c) Reward best (σ_1) **vs** Reward uniq. (σ_4)



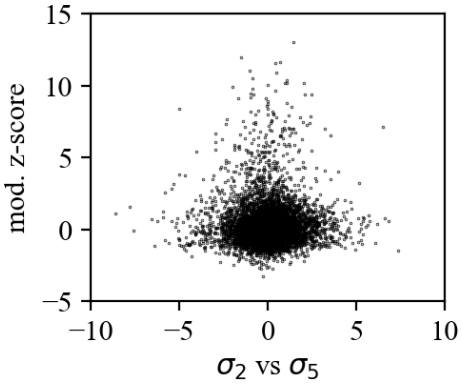
(d) Reward best (σ_1) **vs** Penalty (σ_5)



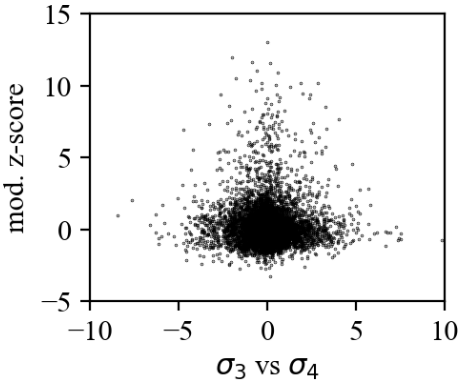
(e) Reward imp. (σ_2) **vs** Reward div. (σ_3)



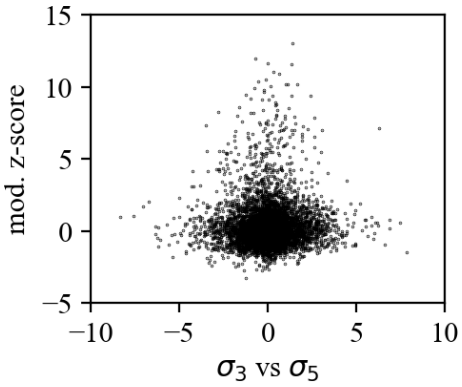
(f) Reward imp. (σ_2) **vs** Reward uniq. (σ_4)



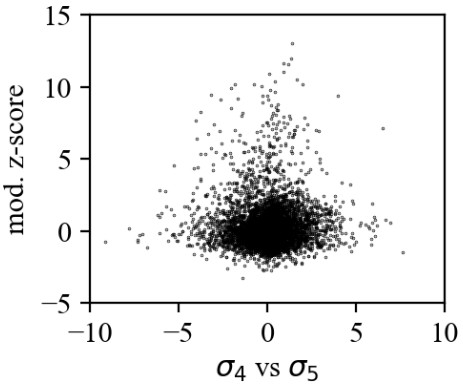
(g) Reward imp. (σ_2) **vs** Penalty (σ_5)



(h) Reward div. (σ_3) **vs** Reward uniq. (σ_4)



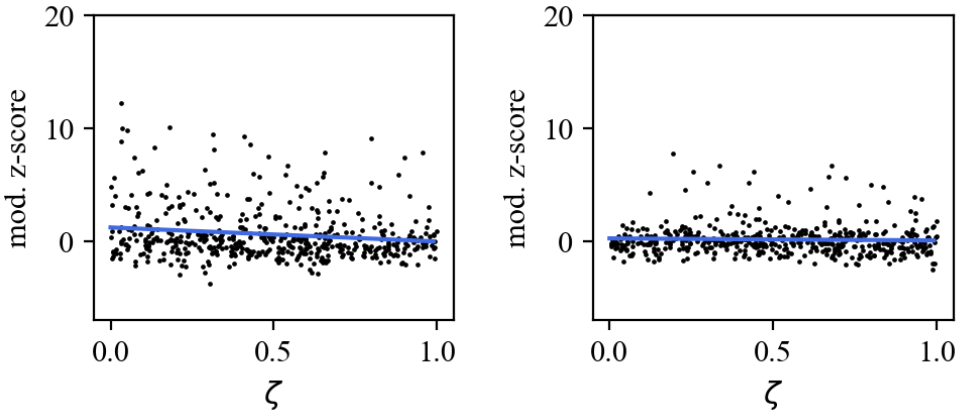
(i) Reward div. (σ_3) **vs** Penalty (σ_5)



(j) Reward uniq. (σ_4) **vs** Penalty (σ_5)

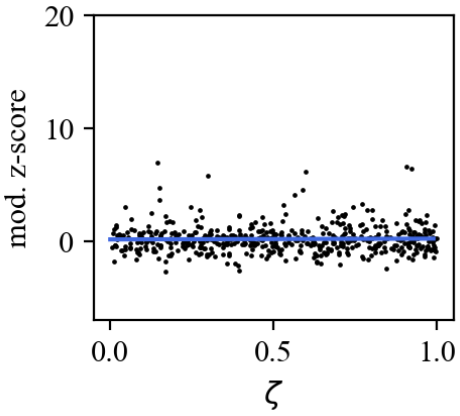
Figure 7.6: Modified z-score distributions of all σ vs σ values. The relativity values are log transformed. All solutions are from the same experiment as appendix F.

H Appendix: Performance of ζ and r combinations

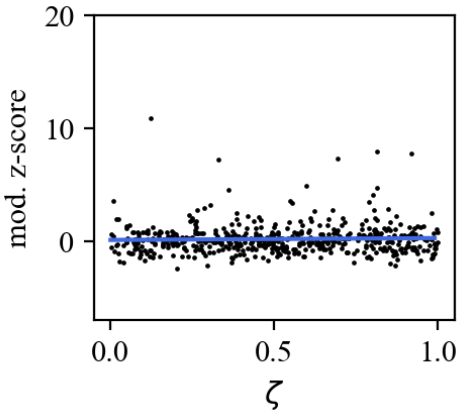


(a) $r = 1$

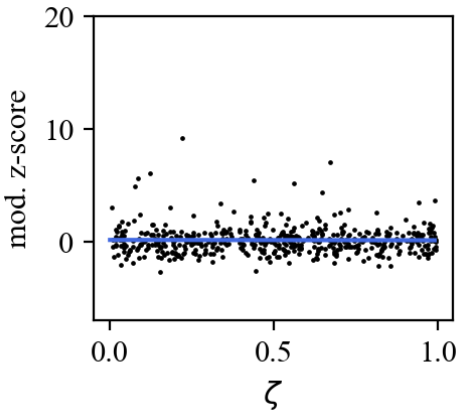
(b) $r = 10$



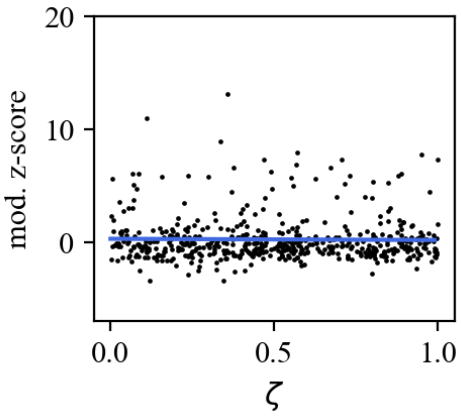
(c) $r = 50$



(d) $r = 100$



(e) $r = 250$



(f) $r = 1000$

Figure 7.7: Modified z-score distributions given differing wheel memory lengths r. The blue line is a regression line of first order.

I Appendix: Heuristic performance after selecting a specific destroy operator

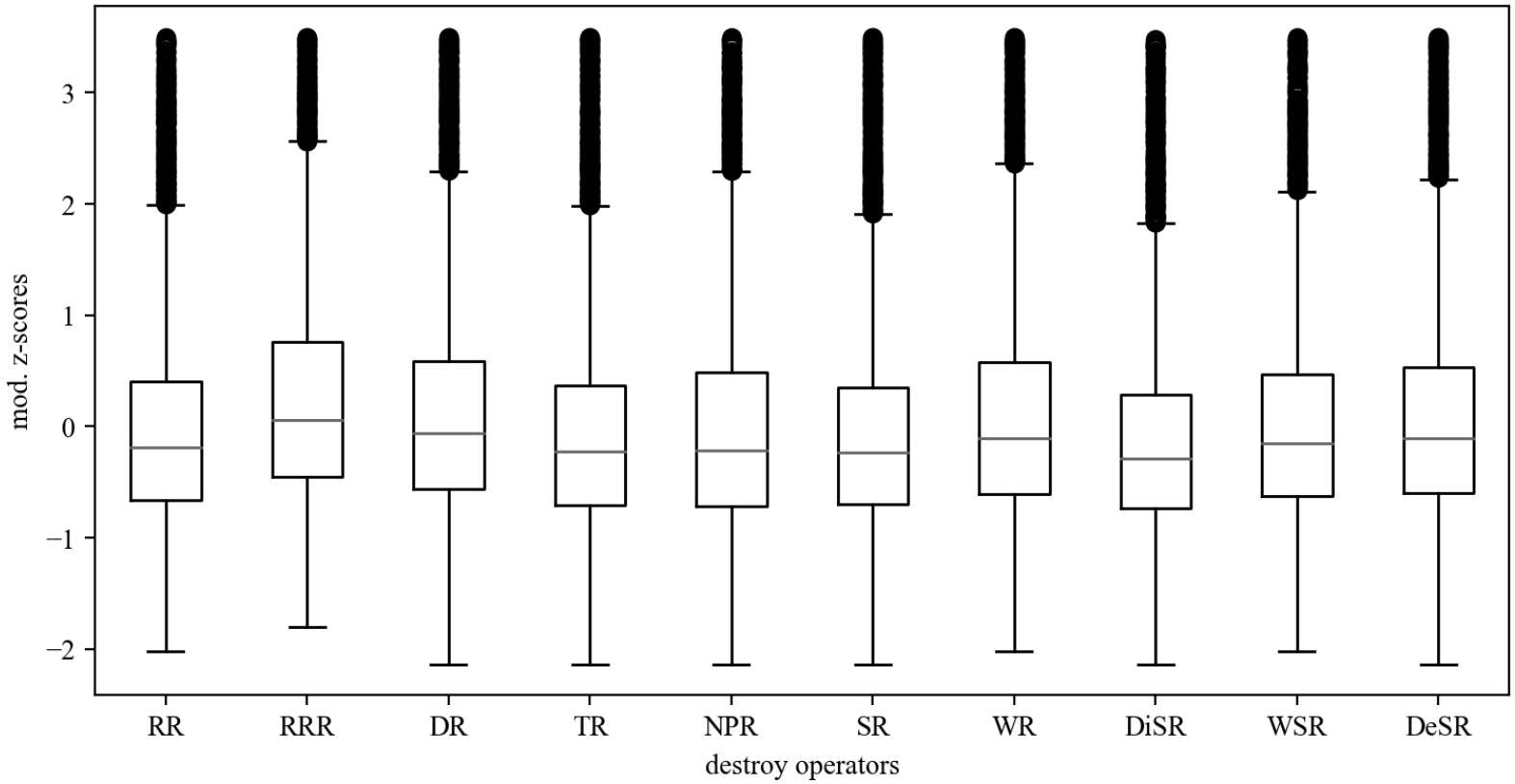


Figure 7.8: Box-plots indicating the average performance of the implementation after selecting a destroy operator. Outlier with mod. z-score > 3.5 have been removed.

J Appendix: Heuristic performance after selecting a specific repair operator

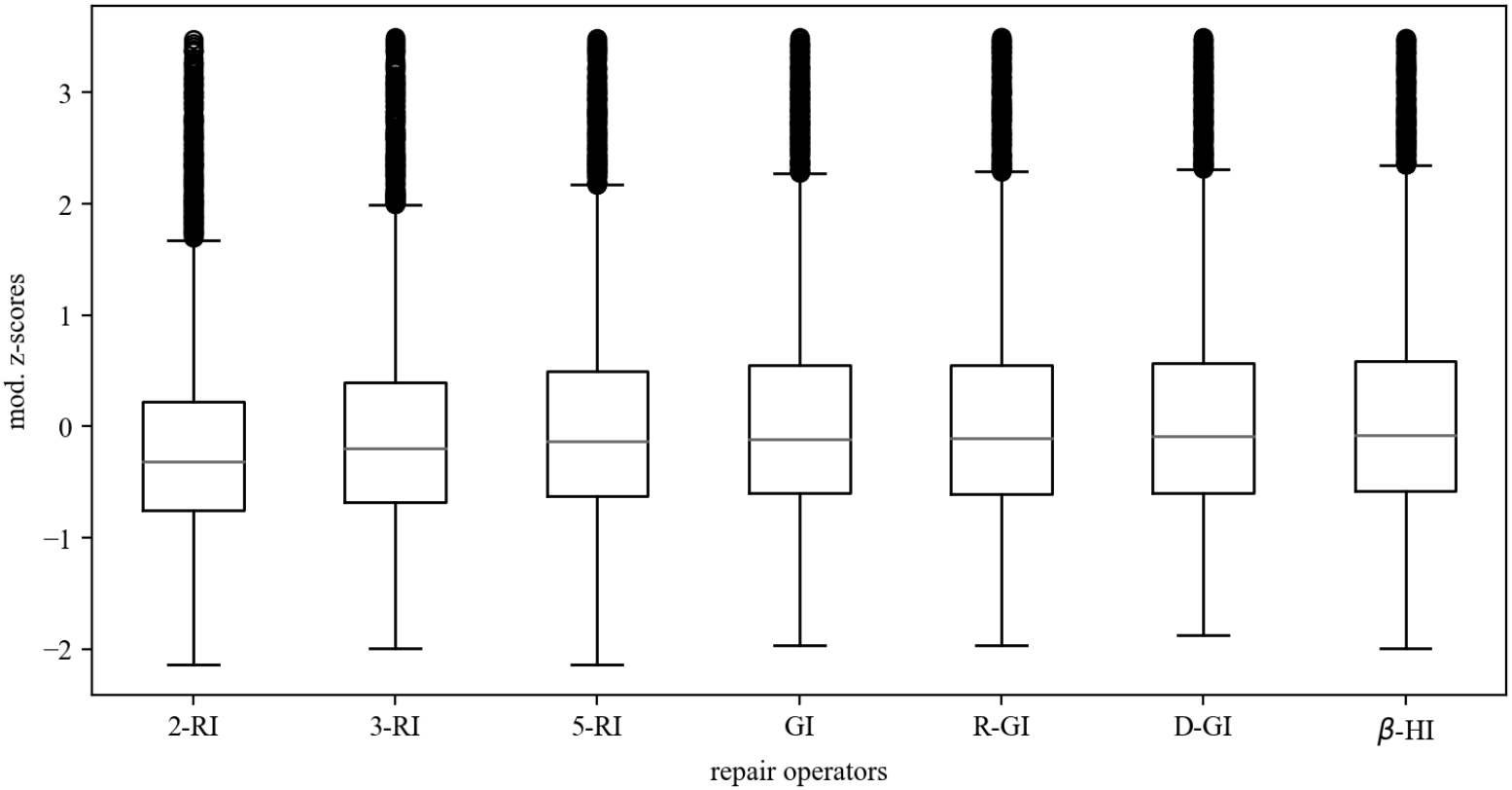


Figure 7.9: Box-plots indicating the average performance of the implementation after selecting a repair operator. Outlier with mod. z-score > 3.5 have been removed.

K Appendix: Final operator verdict of the CART mechanism

Table 7.2: Selection returned by the regression tree based tuning procedure. Possible selection verdicts are + (include), / (no verdict) and - (exclude).

Destroy operators	**Verdict**
Random Removal (RR)	/
Random Route Removal (RRR)	-
Demand Removal (DR)	+
Time Removal (TR)	-
Node Pair Removal (NPR)	+
Shaw Removal (SR)	+
Worst Removal (WoR)	/
Distance Similarity Removal (DiSR)	+
Window Similarity Removal (WiSR)	/
Demand Similarity Removal (DeSR)	/

Insertion operators	**Verdict**
2-Regret Insertion (2-RI)	+
3-Regret Insertion (3-RI)	/
5-Regret Insertion (5-RI)	+
Greedy Insertion Heuristic (GI)	/
Random Greedy Insertion (R-GI)	/
Deep Greedy Insertion (D-GI)	-
β-Hybrid Insertion (β-HI)	+

L Appendix: Tabular representation of the results of the parameter tuning routine

Table 7.3: Final parameter selection after regression tree tuning

Operator type		**Value**	
Destroy operators		RR, DR, NPR, SR, DiSR	
Repair operators		2-RI, 5-RI, β-HI	
Parameter		**Group**	**Value**
Cooling rate	c	Local search acceptance	0.9999
Initial temperature	T	Local search acceptance	0.01
Shake-up log	ς	Local search radius	10
Mean removal log	λ	Local search radius	3.35
Operator randomness	ρ	Operator randomness	0.15
New best reward	σ_1	Operator selection	50
New current reward	σ_2	Operator selection	100
Diversity reward	σ_3	Operator selection	90
New unique reward	σ_4	Operator selection	7
Penalty	σ_5	Operator selection	-80
Memory length	r	Wheel memory	10
Learning rate	ζ	Wheel memory	0.35
Infeasibility percentage	ϱ	Infeasibility	0.65

M Appendix: Removal operator probability given iteration normalized weighting

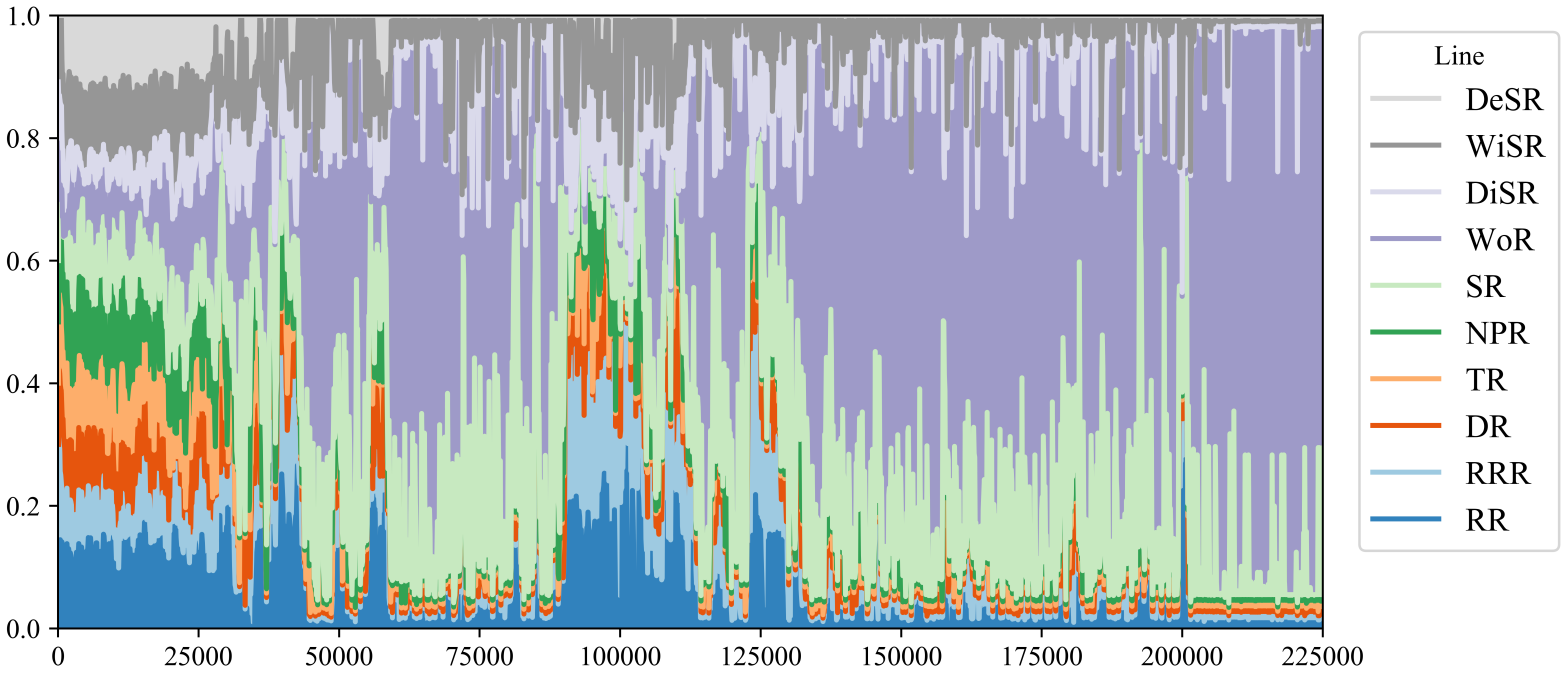


Figure 7.10: Removal operator probability over time given iteration dependent weighting.

N Appendix: Insertion operator probability given iteration normalized weighting

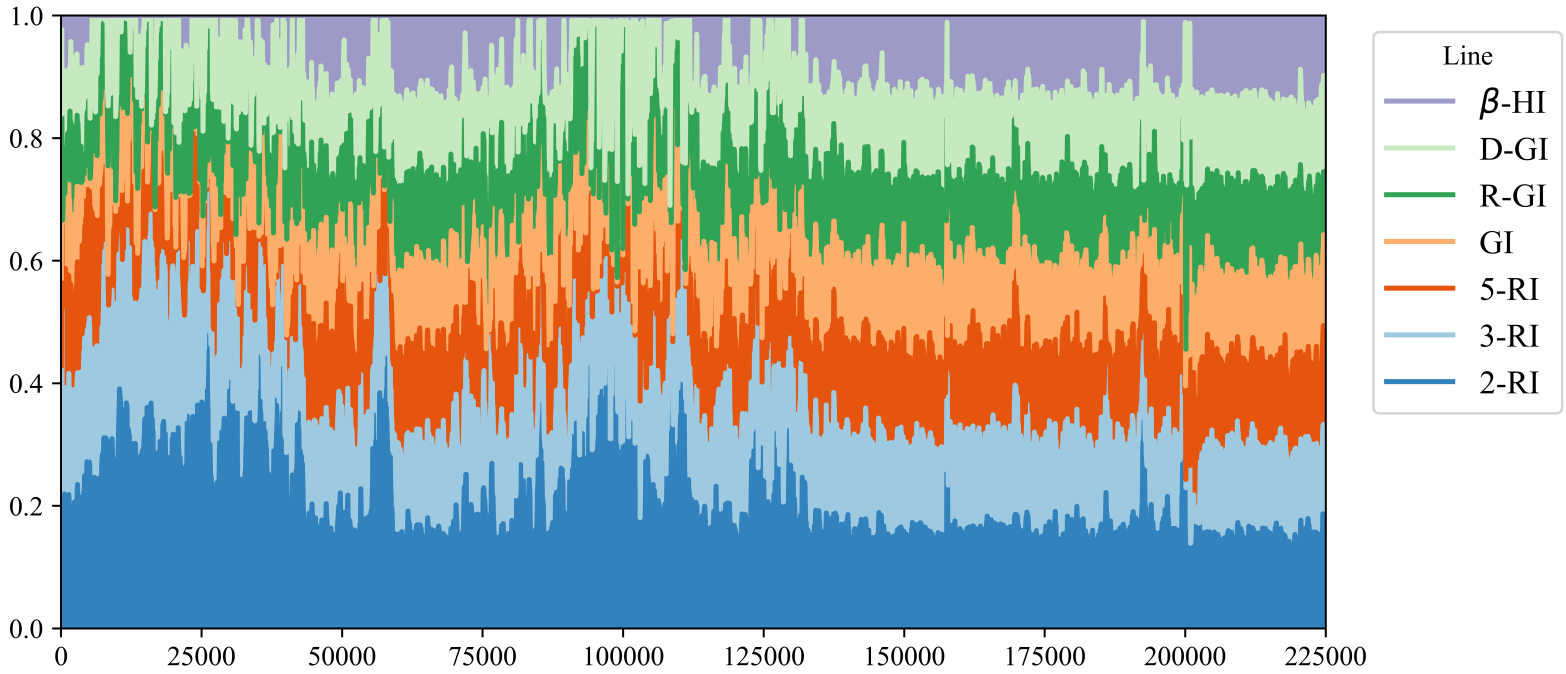


Figure 7.11: Insertion operator probability over time given iteration dependent weighting.

O Appendix: Removal operator probability given time normalized weighting

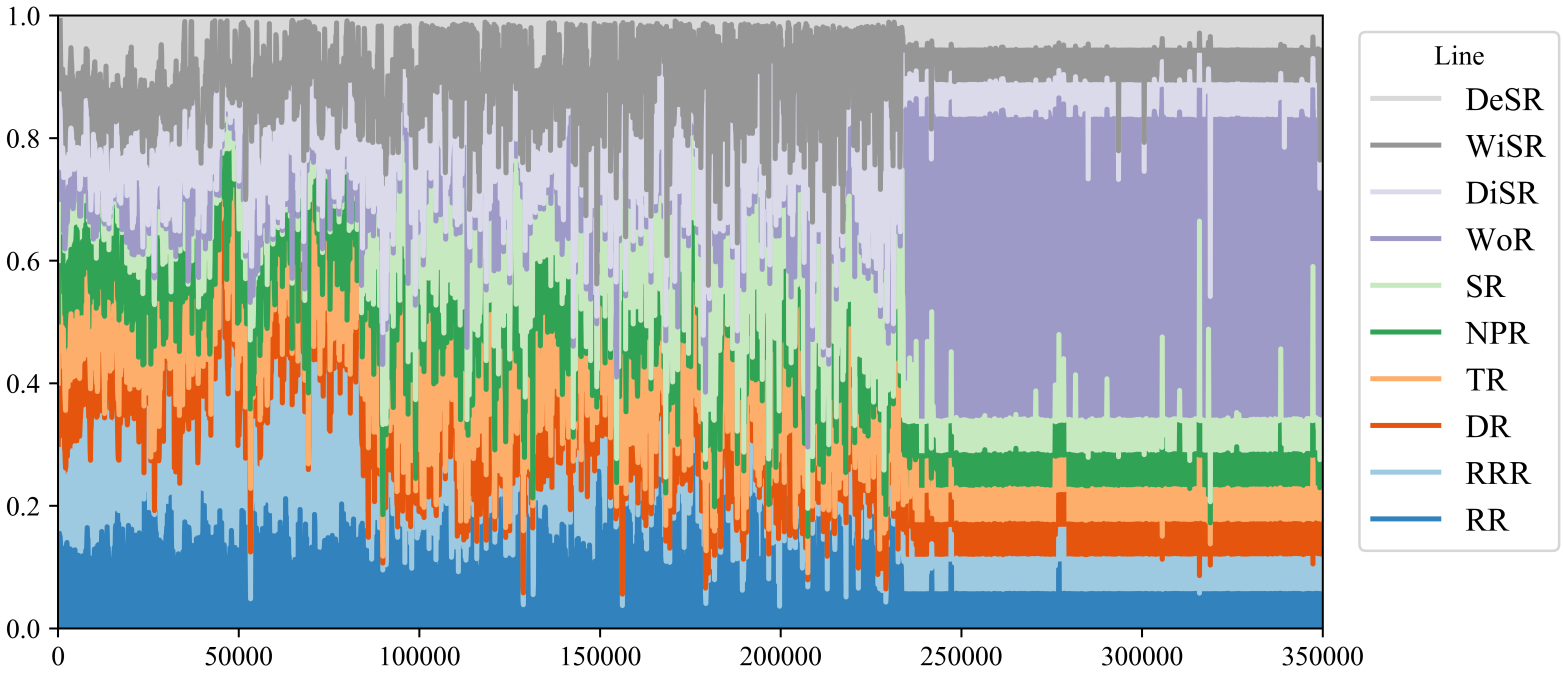


Figure 7.12: Removal operator probability over time given time dependent weighting.

P Appendix: Insertion operator probability given time normalized weighting

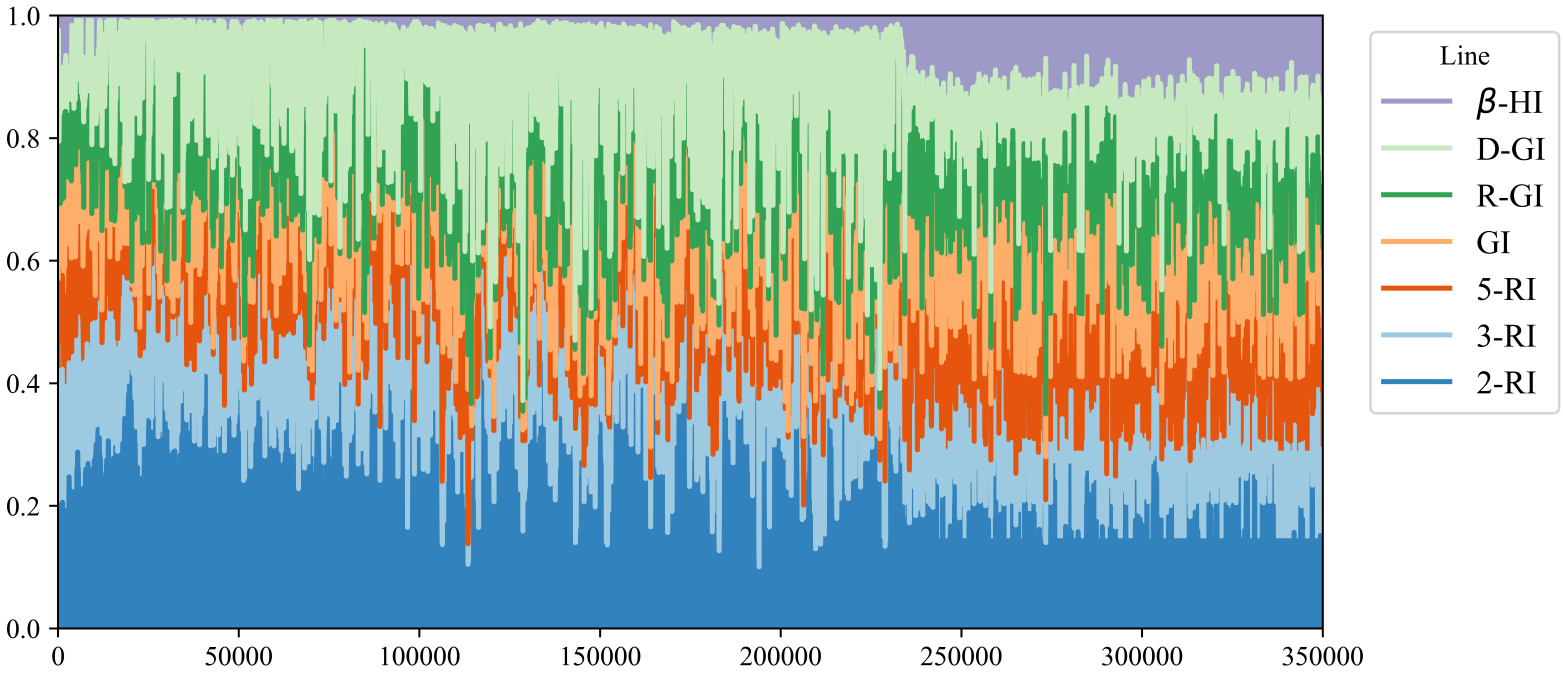


Figure 7.13: Insertion operator probability over time given time dependent weighting.

Q Appendix: Benchmarking Fontaine, selected operators (all inst.)

Table 7.4: Average performance of 10 runs with all operators on Fontaines instances (n=20)

Inst	**BKS** Opt.	**Avg**	**Gap%**	**Best**	**Gap%**	**Iters** Avg.	**Time (s)** Avg.	**Nr. Veh.** Avg.
Fu1	26.37	26.37	0.01	26.37	0.01	14277.1	14.28	3.0
Fu2	33.75	33.75	0.02	33.75	0.02	26958.3	26.96	6.0
Fu3	39.87	39.9	0.05	39.9	0.05	11264.3	11.26	7.0
Fu4	25.85	25.86	0.03	25.86	0.03	20899.6	20.9	3.0
Fu5	32.85	32.87	0.07	32.87	0.07	12522.5	12.52	6.0
Fu6	38.01	38.01	0.02	38.01	0.02	12720.6	12.72	7.0
Ma1	26.72	26.72	0.0	26.72	0.0	23873.4	23.87	3.0
Ma2	36.18	36.19	0.05	36.19	0.05	38346.7	38.35	5.0
Ma3	45.01	45.01	0.0	45.01	0.0	13935.9	13.94	7.0
Ma4	26.07	26.07	0.01	26.07	0.01	28863.2	28.86	3.0
Ma5	36.49	36.49	0.0	36.49	0.0	14174.7	14.17	5.0
Ma6	43.98	43.98	0.0	43.98	0.0	12213.7	12.21	7.0
Pi1	28.91	28.91	0.0	28.91	0.0	13729.2	13.73	3.0
Pi2	44.09	44.09	0.0	44.09	0.0	15609.5	15.61	5.0
Pi3	51.99	51.99	0.0	51.99	0.0	17267.7	17.27	7.0
Pi4	27.54	27.55	0.03	27.55	0.03	16590.6	16.59	3.0
Pi5	40.59	40.59	0.0	40.59	0.0	13940.1	13.94	5.0
Pi6	48.17	48.17	0.0	48.17	0.0	16750.5	16.75	7.0
Se1	24.13	24.14	0.0	24.13	0.0	16076.3	16.08	3.0
Se2	33.52	33.52	0.0	33.52	0.0	14586.3	14.59	5.0
Se3	40.12	40.13	0.02	40.12	0.0	14519.7	14.52	7.0
Se4	24.59	24.59	0.0	24.59	0.0	15526.7	15.53	3.0
Se5	35.73	35.73	0.0	35.73	0.0	26270.1	26.27	5.0
Se6	43.64	43.64	0.0	43.64	0.0	16189.8	16.19	6.0
Sy1	17.8	17.8	0.0	17.8	0.0	14240.7	14.24	4.0
Sy2	26.62	26.62	0.0	26.62	0.0	15666.9	15.67	5.0
Sy3	31.36	31.36	0.0	31.36	0.0	17548.8	17.55	7.0
Sy4	16.93	16.94	0.05	16.93	0.0	14417.9	14.42	3.0
Sy5	22.64	22.64	0.0	22.64	0.0	15992.1	15.99	5.0
Sy6	27.42	27.42	0.0	27.42	0.0	32778.8	32.78	7.0

R Appendix: Benchmarking Fontaine, selected operators (all inst.)

Table 7.5: Average performance of 10 runs with selected operators on Fontaines instances (n=20)

Inst	**BKS** Opt.	**Avg**	**Gap%**	**Best**	**Gap%**	**Iters** Avg.	**Time (s)** Avg.	**Nr. Veh.** Avg.
Fu1	26.37	26.37	0.01	26.37	0.01	14277.1	14.28	3.0
Fu2	33.75	33.75	0.02	33.75	0.02	26958.3	26.96	6.0
Fu3	39.87	39.9	0.05	39.9	0.05	11264.3	11.26	7.0
Fu4	25.85	25.86	0.03	25.86	0.03	20899.6	20.9	3.0
Fu5	32.85	32.87	0.07	32.87	0.07	12522.5	12.52	6.0
Fu6	38.01	38.01	0.02	38.01	0.02	12720.6	12.72	7.0
Ma1	26.72	26.72	0.0	26.72	0.0	23873.4	23.87	3.0
Ma2	36.18	36.19	0.05	36.19	0.05	38346.7	38.35	5.0
Ma3	45.01	45.01	0.0	45.01	0.0	13935.9	13.94	7.0
Ma4	26.07	26.07	0.01	26.07	0.01	28863.2	28.86	3.0
Ma5	36.49	36.49	0.0	36.49	0.0	14174.7	14.17	5.0
Ma6	43.98	43.98	0.0	43.98	0.0	12213.7	12.21	7.0
Pi1	28.91	28.91	0.0	28.91	0.0	13729.2	13.73	3.0
Pi2	44.09	44.09	0.0	44.09	0.0	15609.5	15.61	5.0
Pi3	51.99	51.99	0.0	51.99	0.0	17267.7	17.27	7.0
Pi4	27.54	27.55	0.03	27.55	0.03	16590.6	16.59	3.0
Pi5	40.59	40.59	0.0	40.59	0.0	13940.1	13.94	5.0
Pi6	48.17	48.17	0.0	48.17	0.0	16750.5	16.75	7.0
Se1	24.13	24.14	0.0	24.13	0.0	16076.3	16.08	3.0
Se2	33.52	33.52	0.0	33.52	0.0	14586.3	14.59	5.0
Se3	40.12	40.13	0.02	40.12	0.0	14519.7	14.52	7.0
Se4	24.59	24.59	0.0	24.59	0.0	15526.7	15.53	3.0
Se5	35.73	35.73	0.0	35.73	0.0	26270.1	26.27	5.0
Se6	43.64	43.64	0.0	43.64	0.0	16189.8	16.19	6.0
Sy1	17.8	17.8	0.0	17.8	0.0	14240.7	14.24	4.0
Sy2	26.62	26.62	0.0	26.62	0.0	15666.9	15.67	5.0
Sy3	31.36	31.36	0.0	31.36	0.0	17548.8	17.55	7.0
Sy4	16.93	16.94	0.05	16.93	0.0	14417.9	14.42	3.0
Sy5	22.64	22.64	0.0	22.64	0.0	15992.1	15.99	5.0
Sy6	27.42	27.42	0.0	27.42	0.0	32778.8	32.78	7.0

S Appendix: Benchmarking Solomon, all operators (all inst.)

Table 7.6: Average performance of 10 runs with all operators on Solomon instances (n=100)

Inst	**BKS** Opt.	**Avg**	**Gap**	**Best**	**Gap**	**Iters** Avg.	**Time (s)** Avg.	**Veh.** Avg.
C101	827.3	828.94	0.2	828.94	0.2	23618.3	23.62	10.0
C102	827.3	828.94	0.2	828.94	0.2	25114.6	25.11	10.0
C103	826.3	828.06	0.21	828.06	0.21	29822.6	29.82	10.0
C104	822.9	824.78	0.23	824.78	0.23	23062.2	23.06	10.0
C105	827.3	828.94	0.2	828.94	0.2	24515.6	24.52	10.0
C106	827.3	828.94	0.2	828.94	0.2	22483.4	22.48	10.0
C107	827.3	828.94	0.2	828.94	0.2	25115.8	25.12	10.0
C108	827.3	828.94	0.2	828.94	0.2	20391.0	20.39	10.0
C109	827.3	828.94	0.2	828.94	0.2	30964.1	30.96	10.0
C201	589.1	591.56	0.42	591.56	0.42	20833.5	20.83	3.0
C202	589.1	591.56	0.42	591.56	0.42	24302.0	24.3	3.0
C203	588.7	591.17	0.42	591.17	0.42	20833.6	20.83	3.0
C204	588.1	590.6	0.42	590.6	0.42	25676.1	25.68	3.0
C205	586.4	588.88	0.42	588.88	0.42	25627.5	25.63	3.0
C206	586.0	588.49	0.43	588.49	0.43	19297.4	19.3	3.0
C207	585.8	588.29	0.42	588.29	0.42	21439.0	21.44	3.0
C208	585.8	588.32	0.43	588.32	0.43	24467.2	24.47	3.0
R101	1637.7	1648.16	0.64	1643.09	0.33	31223.6	31.22	20.0
R102	1466.6	1475.83	0.63	1472.92	0.43	46159.5	46.16	18.0
R103	1208.7	1221.3	1.04	1213.62	0.41	41206.9	41.21	14.3
R104	971.5	985.2	1.41	976.61	0.53	38943.4	38.94	10.8
R105	1355.3	1367.22	0.88	1360.78	0.4	40972.0	40.97	15.2
R106	1234.6	1240.54	0.48	1239.37	0.39	43859.9	43.86	13.0
R107	1064.6	1076.12	1.08	1073.77	0.86	37321.5	37.32	11.0
R108	-	947.0	-	943.71	-	36470.1	36.47	10.0
R109	1146.9	1155.56	0.76	1151.84	0.43	28724.9	28.72	13.0
R110	1068.0	1083.76	1.48	1072.41	0.41	35182.7	35.18	12.0
R111	1048.7	1055.22	0.62	1053.5	0.46	32948.2	32.95	11.9
R112	-	963.58	-	958.7	-	28320.1	28.32	10.1
R201	1143.2	1175.24	2.8	1157.98	1.29	27028.8	27.03	6.7
R202	-	1043.31	-	1040.96	-	28355.0	28.36	6.0
R203	-	896.15	-	881.77	-	26483.4	26.48	5.2

Inst	**BKS** Opt.	**Avg**	**Gap**	**Best**	**Gap**	**Iters** Avg.	**Time (s)** Avg.	**Veh.** Avg.
R204	-	749.73	-	735.86	-	26419.1	26.42	4.2
R205	-	963.57	-	954.16	-	30580.2	30.58	4.8
R206	-	900.19	-	885.44	-	29936.4	29.94	4.2
R207	-	832.53	-	813.65	-	24527.7	24.53	3.7
R208	-	714.49	-	706.46	-	23400.5	23.4	3.5
R209	-	866.39	-	859.39	-	32612.7	32.61	4.6
R210	-	924.2	-	912.48	-	28557.2	28.56	4.8
R211	-	772.54	-	761.2	-	32092.9	32.09	3.7
RC101	1619.8	1653.95	2.11	1623.58	0.23	32453.7	32.45	16.0
RC102	1457.4	1483.96	1.82	1464.73	0.5	34770.7	34.77	14.9
RC103	1258.0	1280.88	1.82	1274.74	1.33	29386.5	29.39	11.9
RC104	-	1138.63	-	1135.8	-	35933.7	35.93	10.0
RC105	1513.7	1536.81	1.53	1523.4	0.64	30188.2	30.19	15.5
RC106	-	1383.12	-	1377.35	-	40774.1	40.77	13.0
RC107	1207.8	1227.62	1.64	1212.83	0.42	32600.2	32.6	12.0
RC108	1114.2	1126.29	1.09	1117.53	0.3	36287.8	36.29	11.0
RC201	1261.8	1285.16	1.85	1267.95	0.49	29279.5	29.28	7.5
RC202	1092.3	1111.34	1.74	1102.43	0.93	23432.0	23.43	7.0
RC203	-	945.88	-	929.63	-	26083.2	26.08	4.4
RC204	-	801.39	-	786.38	-	24115.7	24.12	4.0
RC205	1154.0	1157.62	0.31	1157.55	0.31	33873.0	33.87	7.0
RC206	-	1082.35	-	1069.1	-	29627.7	29.63	4.5
RC207	-	1000.95	-	982.06	-	19409.9	19.41	5.0
RC208	-	806.55	-	785.42	-	35221.0	35.22	3.9

T Appendix: Benchmarking Solomon, selected operators (all inst.)

Table 7.7: Average performance of 10 runs with selected operators on Solomon instances (n=100)

Inst	BKS	Avg	Gap	Best	Gap	Iters	Time (s)	Veh.
	Opt.					Avg.	Avg.	Avg.
C101	827.3	828.94	0.2	828.94	0.2	16994.7	16.99	10.0
C102	827.3	828.94	0.2	828.94	0.2	20097.8	20.1	10.0
C103	826.3	828.06	0.21	828.06	0.21	22817.3	22.82	10.0
C104	822.9	824.78	0.23	824.78	0.23	22976.3	22.98	10.0
C105	827.3	828.94	0.2	828.94	0.2	26452.5	26.45	10.0
C106	827.3	828.94	0.2	828.94	0.2	28160.2	28.16	10.0
C107	827.3	828.94	0.2	828.94	0.2	26181.6	26.18	10.0
C108	827.3	828.94	0.2	828.94	0.2	21778.1	21.78	10.0
C109	827.3	828.94	0.2	828.94	0.2	26222.5	26.22	10.0
C201	589.1	591.56	0.42	591.56	0.42	25901.4	25.9	3.0
C202	589.1	591.56	0.42	591.56	0.42	21582.0	21.58	3.0
C203	588.7	591.17	0.42	591.17	0.42	29106.2	29.11	3.0
C204	588.1	590.6	0.42	590.6	0.42	29200.4	29.2	3.0
C205	586.4	588.88	0.42	588.88	0.42	27427.7	27.43	3.0
C206	586.0	588.49	0.43	588.49	0.43	19438.6	19.44	3.0
C207	585.8	588.29	0.42	588.29	0.42	24033.0	24.03	3.0
C208	585.8	588.32	0.43	588.32	0.43	25584.2	25.58	3.0
R101	1637.7	1646.73	0.55	1643.39	0.35	28442.5	28.44	20.0
R102	1466.6	1476.15	0.65	1473.62	0.48	33922.8	33.92	18.0
R103	1208.7	1219.27	0.87	1213.62	0.41	31872.9	31.87	14.0
R104	971.5	983.83	1.27	976.61	0.53	27833.5	27.83	10.9
R105	1355.3	1366.44	0.82	1363.74	0.62	27323.3	27.32	15.1
R106	1234.6	1239.46	0.39	1239.37	0.39	38062.3	38.06	13.0
R107	1064.6	1077.93	1.25	1073.34	0.82	25804.4	25.8	11.2
R108	-	948.56	-	941.08	-	31686.9	31.69	10.0
R109	1146.9	1153.68	0.59	1151.84	0.43	36461.3	36.46	13.0
R110	1068.0	1079.41	1.07	1072.41	0.41	30753.0	30.75	12.0
R111	1048.7	1054.18	0.52	1053.5	0.46	31423.4	31.42	12.0
R112	-	960.19	-	953.63	-	28937.9	28.94	10.0
R201	1143.2	1160.93	1.55	1157.15	1.22	15055.1	15.06	7.1
R202	-	1044.03	-	1037.49	-	29021.4	29.02	6.2
R203	-	879.65	-	874.87	-	20813.7	20.81	6.0

Inst	**BKS**	**Avg**	**Gap**	**Best**	**Gap**	**Iters**	**Time (s)**	**Veh.**
	Opt.					Avg.	Avg.	Avg.
R204	-	740.62	-	735.8	-	28738.6	28.74	4.8
R205	-	961.19	-	954.16	-	33305.5	33.31	5.1
R206	-	895.83	-	884.85	-	30707.0	30.71	4.9
R207	-	808.86	-	797.99	-	28553.5	28.55	4.1
R208	-	716.71	-	706.91	-	39850.7	39.85	4.1
R209	-	865.55	-	860.11	-	22969.0	22.97	5.1
R210	-	914.61	-	909.32	-	21310.1	21.31	5.3
R211	-	764.75	-	755.78	-	39102.8	39.1	4.4
RC101	1619.8	1648.15	1.75	1635.11	0.94	32377.2	32.38	15.6
RC102	1457.4	1480.21	1.57	1462.52	0.35	30601.2	30.6	14.6
RC103	1258.0	1281.68	1.88	1276.05	1.43	28104.4	28.1	12.0
RC104	-	1137.76	-	1135.52	-	37642.7	37.64	10.0
RC105	1513.7	1538.72	1.65	1518.77	0.34	29855.3	29.86	15.7
RC106	-	1384.18	-	1377.35	-	36896.0	36.9	13.2
RC107	1207.8	1222.01	1.18	1212.83	0.42	43923.7	43.92	12.1
RC108	1114.2	1121.2	0.63	1117.53	0.3	25171.1	25.17	11.0
RC201	1261.8	1280.3	1.47	1267.88	0.48	27285.8	27.29	8.5
RC202	1092.3	1105.55	1.21	1095.64	0.31	23176.0	23.18	7.3
RC203	-	934.17	-	926.82	-	27668.7	27.67	5.0
RC204	-	792.3	-	786.38	-	31319.4	31.32	4.0
RC205	1154.0	1157.73	0.32	1157.55	0.31	27238.1	27.24	7.0
RC206	-	1071.77	-	1057.65	-	21154.2	21.15	5.6
RC207	-	981.83	-	971.21	-	18365.1	18.37	5.5
RC208	-	786.94	-	778.93	-	25593.8	25.59	4.5

U Appendix: New benchmarking, all operators (all inst.)

Table 7.8: Average performance of 10 runs with all operators on new instances (n=50-200)

Inst	n	BKS Best 10	Avg	Gap%	Iters Avg. 10	Time (s) Avg. 10	Nr. Veh. Avg. 10
Fu1	50	289.98	291.34	0.47	20949.4	20.95	6.5
Fu2	50	290.71	291.28	0.2	20992.5	20.99	6.1
Fu3	50	298.25	299.64	0.47	25429.5	25.43	6.5
Fu1	100	564.3	573.79	1.68	42943.5	42.94	13.0
Fu2	100	569.9	572.64	0.48	37571.5	37.57	13.0
Fu3	100	567.69	571.12	0.6	27409.6	27.41	13.0
Fu1	200	1035.53	1049.75	1.37	78321.5	78.32	25.6
Fu2	200	1023.13	1032.9	0.95	58799.4	58.8	24.8
Fu3	200	1014.11	1021.13	0.69	63851.6	63.85	24.1
Ma1	50	43.94	43.94	0.0	31300.0	31.3	7.0
Ma2	50	43.43	43.46	0.06	30541.0	30.54	6.0
Ma3	50	43.99	44.03	0.07	32550.1	32.55	6.9
Ma1	100	83.14	83.47	0.39	34115.0	34.12	13.0
Ma2	100	81.87	82.89	1.24	32207.5	32.21	12.9
Ma3	100	82.22	82.47	0.3	39308.1	39.31	12.0
Ma1	200	152.56	153.88	0.86	72050.7	72.05	25.6
Ma2	200	148.21	150.1	1.28	54995.7	55.0	24.8
Ma3	200	148.57	150.38	1.22	54682.1	54.68	24.7
Pi1	50	121.19	121.85	0.55	21509.5	21.51	6.1
Pi2	50	120.64	121.1	0.38	17093.7	17.09	6.0
Pi3	50	120.46	120.51	0.04	18754.7	18.75	7.0
Pi1	100	220.05	221.71	0.75	32457.2	32.46	12.1
Pi2	100	220.09	222.04	0.88	29421.8	29.42	12.6
Pi3	100	211.91	212.76	0.4	47322.6	47.32	12.0
Pi1	200	393.73	396.64	0.74	59823.6	59.82	25.0
Pi2	200	386.86	390.61	0.97	56979.0	56.98	24.7
Pi3	200	376.02	378.44	0.64	65213.1	65.21	24.2
Se1	50	117.52	117.67	0.13	18339.6	18.34	6.3
Se2	50	115.95	116.79	0.73	23733.8	23.73	6.4
Se3	50	117.51	118.55	0.88	18480.5	18.48	6.3

Inst	**n**	**BKS** Best 10	**Avg**	**Gap%**	**Iters** Avg. 10	**Time (s)** Avg. 10	**Nr. Veh.** Avg. 10
Se1	100	205.32	205.37	0.02	31079.2	31.08	13.0
Se2	100	202.29	203.41	0.56	30293.4	30.29	12.9
Se3	100	202.34	203.86	0.75	29538.9	29.54	12.3
Se1	200	375.08	376.23	0.31	56808.3	56.81	25.3
Se2	200	365.71	369.0	0.9	47658.9	47.66	25.0
Se3	200	363.15	367.12	1.09	60488.4	60.49	24.0
Sy1	50	245.06	245.2	0.06	22769.3	22.77	6.9
Sy2	50	243.63	244.01	0.16	17739.3	17.74	6.0
Sy3	50	245.93	252.34	2.61	20623.2	20.62	6.5
Sy1	100	456.85	463.77	1.51	33307.1	33.31	12.4
Sy2	100	456.61	461.72	1.12	27246.5	27.25	12.7
Sy3	100	456.26	459.98	0.82	38914.6	38.91	12.0
Sy1	200	817.79	825.2	0.91	61850.9	61.85	25.5
Sy2	200	801.14	812.68	1.44	66037.5	66.04	24.8
Sy3	200	804.8	811.59	0.84	80176.0	80.18	24.3

V Appendix: New benchmarking, selected operators (all inst.)

Table 7.9: Average performance of 10 runs with selected operators on new instances (n=50-200)

Inst	n	BKS Best 10	Avg	Gap%	Iters Avg. 10	Time (s) Avg. 10	Nr. Veh. Avg. 10
Fu1	50	290.11	291.26	0.4	16176.6	16.18	6.4
Fu2	50	290.71	291.72	0.35	19602.8	19.6	6.1
Fu3	50	298.25	299.08	0.28	28368.8	28.37	6.4
Fu1	100	569.94	572.9	0.52	25015.2	25.02	13.2
Fu2	100	569.73	572.28	0.45	39761.8	39.76	13.0
Fu3	100	565.31	571.08	1.02	31063.0	31.06	13.0
Fu1	200	1035.4	1051.82	1.58	55503.0	55.5	25.6
Fu2	200	1019.75	1025.61	0.57	75922.5	75.92	24.8
Fu3	200	1011.03	1018.74	0.76	57407.3	57.41	24.1
Ma1	50	43.94	43.94	0.0	31609.6	31.61	7.0
Ma2	50	43.43	43.48	0.11	27832.2	27.83	6.2
Ma3	50	43.99	43.99	0.0	24646.1	24.65	7.0
Ma1	100	83.2	83.36	0.2	35324.4	35.32	13.0
Ma2	100	81.82	82.22	0.49	35218.6	35.22	12.5
Ma3	100	82.22	82.32	0.13	40288.6	40.29	12.0
Ma1	200	153.19	154.46	0.83	47743.9	47.74	25.6
Ma2	200	147.5	148.21	0.48	63995.7	64.0	24.9
Ma3	200	148.25	149.49	0.83	57027.3	57.03	24.9
Pi1	50	121.19	121.52	0.27	28719.0	28.72	6.0
Pi2	50	120.59	120.81	0.18	16421.6	16.42	6.0
Pi3	50	120.46	120.46	0.0	18316.6	18.32	7.0
Pi1	100	219.78	221.17	0.63	45242.0	45.24	12.0
Pi2	100	219.99	221.99	0.91	22720.8	22.72	12.7
Pi3	100	212.33	213.02	0.33	37191.3	37.19	12.0
Pi1	200	392.12	393.54	0.36	65349.3	65.35	25.0
Pi2	200	384.05	389.78	1.49	50214.5	50.21	24.2
Pi3	200	371.39	377.74	1.71	52150.7	52.15	24.3
Se1	50	117.52	117.75	0.2	21699.7	21.7	6.4
Se2	50	116.23	116.83	0.52	21446.6	21.45	6.1
Se3	50	117.51	118.15	0.54	18267.7	18.27	6.0

Inst	**n**	**BKS** Best 10	**Avg**	**Gap%**	**Iters** Avg. 10	**Time (s)** Avg. 10	**Nr. Veh.** Avg. 10
Se1	100	205.33	205.5	0.09	30638.5	30.64	13.0
Se2	100	202.13	202.91	0.39	38530.0	38.53	12.6
Se3	100	203.42	203.86	0.22	24081.1	24.08	12.0
Se1	200	371.8	374.47	0.72	57773.3	57.77	25.1
Se2	200	363.52	365.52	0.55	53016.0	53.02	24.4
Se3	200	363.83	366.26	0.67	47777.8	47.78	24.2
Sy1	50	245.06	245.2	0.06	32502.2	32.5	6.9
Sy2	50	243.63	245.21	0.65	21661.2	21.66	6.1
Sy3	50	245.93	253.5	3.08	24247.3	24.25	6.7
Sy1	100	457.73	461.55	0.83	33587.4	33.59	12.8
Sy2	100	456.25	458.72	0.54	39207.4	39.21	12.4
Sy3	100	455.49	459.26	0.83	28195.5	28.2	12.2
Sy1	200	806.82	816.56	1.21	66836.5	66.84	25.3
Sy2	200	798.67	807.55	1.11	65055.9	65.06	24.6
Sy3	200	803.47	808.64	0.64	64030.2	64.03	24.4

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbständig angefertigt habe. Die aus fremden Quellen direkt und indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Ich weiß, dass die Arbeit in digitalisierter Form daraufhin überprüft werden kann, ob unerlaubte Hilfsmittel verwendet wurden und ob es sich - insgesamt oder in Teilen - um ein Plagiat handelt. Zum Vergleich meiner Arbeit mit existierenden Quellen darf sie in eine Datenbank eingestellt werden und nach der Überprüfung zum Vergleich mit künftig eingehenden Arbeiten dort verbleiben. Weitere Vervielfältigungs- und Verwertungsrechte werden dadurch nicht eingeräumt. Die Arbeit wurde weder einer anderen Prüfungsbehörde vorgelegt noch veröffentlicht.

Ort, Datum

Unterschrift