universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Large Neighborhood Search for a Real-World Vehicle Routing Problem with Time Windows

verfasst von | submitted by

Tianpei Feng BSc (WU)

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2026

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 915
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Betriebswirtschaft
Betreut von | Supervisor:	Ass.-Prof. Dr. Christian Tilk
Mitbetreut von | Co-Supervisor:	Dr. Ninja Scherr B.Sc. M.Sc.

Abstract

This thesis focuses on the application of a Large Neighborhood Search (LNS)-based metaheuristic to the Vehicle Routing Problem with Time Windows within real-world short-haul logistics operations. The research aims at minimizing truck routing distances using historical dispatch data provided by an Austrian transportation company over a selected period.

The results show that the developed LNS framework is capable of producing high-quality solutions within practical computation times. The findings highlight both managerial implications and limitations of the implemented methods. Furthermore, the LNS framework establishes a basis for future research on larger-scale problem instances and more complex operational constraints.

Zusammenfassung

Diese Arbeit befasst sich mit der Anwendung einer auf Large Neighborhood Search (LNS) basierenden metaheuristischen Optimierung auf das Vehicle Routing Problem mit Zeitfenstern im Kontext realer Kurzstrecken-Logistikoperationen. Ziel der Untersuchung ist die Minimierung der Fahrtstrecken von Lkw-Touren auf Basis historischer Dispositionsdaten eines österreichischen Transportunternehmens aus einem ausgewählten Zeitraum.

Die Ergebnisse zeigen, dass das entwickelte LNS-Framework hochwertige Lösungen innerhalb praktikabler Rechenzeiten liefert. Die Erkenntnisse verdeutlichen sowohl die betrieblichen Potenziale als auch die Grenzen der implementierten Methoden. Darüber hinaus bildet das LNS-Framework eine Grundlage für zukünftige Forschungsarbeiten zu umfangreicheren Probleminstanzen und komplexeren operativen Restriktionen.

Statement on the Use of Artificial Intelligence

Artificial intelligence (AI) tools, including Claude (Anthropic), Gemini (Google), and SciSpace, were used during the preparation of this thesis. These tools were primarily employed for language refinement, supporting the literature review process, and assisting in the development and debugging of code.

All AI-generated suggestions were critically reviewed and appropriately adapted. The AI tools were used solely as assistance tools and did not replace independent work, methodological design, or the development of results.

The author retains full responsibility for the content and academic integrity of this thesis.

Acknowledgment

I would like to express my sincere gratitude to my supervisors, Ass.-Prof. Dr. Christian Tilk and Dr. Ninja Scherr, for their valuable guidance, constructive feedback, and continuous support throughout the preparation of this thesis. Their insights and expertise were essential in shaping the direction and quality of this work.

I am also deeply grateful to my employer, Gebrüder Weiss GmbH, and to Mr. Jürgen Drnec, Head of Process Management Operations, for providing the opportunity, data, and practical environment that made this research possible.

Finally, I wish to express my deepest appreciation to my parents, Jianping Song and Fan Feng, for their love, support, and encouragement throughout my studies.

Vienna, May 2026

Tianpei Feng

Contents

List of Abbreviations:

ACO: Ant Colony Optimization

ADR: European Agreement concerning the International Carriage of Dangerous Goods by Road

ALNS: Adaptive Large Neighborhood Search

EDD: Earliest Due Date

GA: Genetic Algorithm

GR: Greedy Repair

HOS: Hours of Service

LNS: Large Neighborhood Search

NN: Nearest Neighbor

NP-hard: Non-deterministic Polynomial-time hard

OSRM: Open Street Routing Machine

PDPTW: Pickup and Delivery Problem with Time Windows

RaR: Random Removal

RR: Regret Repair

RoR: Route Removal

TS: Tabu Search

VND: Variable Neighborhood Descent

VNS: Variable Neighborhood Search

VRP: Vehicle Routing Problem

VRPTW: Vehicle Routing Problem with Time Windows

WR: Worst Removal

List of Tables

List of Figures

List of Algorithms

1. Introduction

Modern logistics systems require decision-making across a wide range of operational aspects, including network design, production scheduling, bin packing, and vehicle routing. Many of these decision problems can be formulated as combinatorial optimization problems, often modeled using mixed-integer linear programming (MILP), which provides a flexible mathematical framework and has been widely applied to complex logistics operations (Nemhauser & Wolsey, 1988). Recent advances in commercial solvers such as Gurobi and CPLEX have significantly increased the number of instances that can be solved to optimality within reasonable computation times (Koch et al., 2022). Nevertheless, exact solution techniques remain computationally demanding for large-scale real-world instances, where tight operational constraints further increase the complexity of the problem (Wang et al., 2025).

As a result, although classical optimization problems have been extensively studied in literature, a persistent gap remains between theoretical models and their practical implementation, as many solution approaches rely on simplifying assumptions to maintain computational tractability (Hosny, 2014). Real-world logistics environments are subject to numerous constraints that are difficult to fully capture in standard formulations. Consequently, the practical implementation of optimization approaches often requires additional modeling considerations beyond those addressed in theoretical research.

At the same time, the growing complexity of logistics operations has increased the importance of automated and decision-support systems. Companies are required to make decisions within short time frames to maintain operational efficiency and competitiveness. This development has driven increased interest in solution methods that are not only fast and robust, but also practically deployable in real operational environments.

This thesis focuses on one of the most prominent and practically relevant routing problems in logistics, the Vehicle Routing Problem with Time Windows (VRPTW). The research aims to develop a practically oriented solution approach for real-world logistics routing operations that can generate high-quality results within practical computation times.

1.1 Background and Motivation

The thesis is conducted in cooperation with Gebrüder Weiss GmbH, an international logistics and transportation company headquartered in Lauterach, Austria. One of its largest branches, located in Maria Lanzendorf, manages short-haul freight transport operations covering the metropolitan area of Vienna as well as the federal states of Lower Austria and Burgenland.

In consultation with Gebrüder Weiss, the optimization of truck routing in the 22nd district of Vienna (Donaustadt) is defined as the main objective of this thesis. This district was selected due to its high optimization potential. As the largest district in Vienna in terms of geographic area, Donaustadt is characterized by a large service region, longer travel distances, and a more spatially dispersed customer base compared to districts that are more centrally located. These factors increase routing complexity and amplify the operational impact of inefficient routing decisions.

The number of customers in Donaustadt typically ranges between 80 and 100 per day, which results in problem instances of considerable size. While such instances can, in principle, be solved using exact optimization methods, obtaining proven optimal solutions could be computationally expensive and time-consuming. This limitation is particularly relevant in practical logistics operations, where routing decisions must be generated quickly, such as prior to daily depot opening times. Accordingly, this thesis employs a metaheuristic solution approach to generate high-quality solutions within reasonable computation times, supporting timely and effective dispatching decisions.

In operational practice, short-haul routing at Gebrüder Weiss GmbH is significantly more complex than the classical vehicle routing problem. First, drivers typically operate two shifts per day, namely a morning and an afternoon shift. In this context, Hours of Service (HOS) regulations must be considered. According to EU Regulation (EC) No 561/2006, drivers are required to take an uninterrupted break of 45 minutes after a driving period of 4.5 hours. Furthermore, the regulation stipulates a maximum daily driving time of 9 hours, which may be extended to 10 hours no more than twice per week (European Commission, 2024). Consequently, route planning must ensure compliance with these legal driving time limits, which introduces additional temporal constraints.

Second, shipments may differ in their characteristics and handling requirements. For instance, certain goods are classified under ADR (European Agreement concerning the International Carriage of Dangerous Goods by Road) and therefore require transportation using specially equipped vehicles. Other shipments may necessitate trucks with specific handling equipment.

Lastly, a substantial proportion of customers have predefined delivery appointments that must be strictly respected. These time constraints naturally align with the structure of the VRPTW and further increase the complexity of feasible route construction.

The primary objective of the optimization is to minimize the total distance traveled across all routes. While many routing models additionally aim to minimize the number of vehicles, either as a primary objective or as part of a weighted objective function, this aspect is deprioritized in the present study. From a strategic perspective at Gebrüder Weiss, reducing the number of vehicles is currently of limited relevance, particularly as a substantial share of the fleet is operated by subcontractors under fixed contractual agreements. Moreover, preserving a certain degree of operational flexibility is considered desirable. Consequently, distance minimization is adopted as the principal optimization objective.

1.2. Problem Description and Mathematical Formulation

The VRPTW represents one of the most significant extensions of the classical Vehicle Routing Problem (VRP). To understand the VRPTW, it is first necessary to consider the fundamental VRP, which was originally introduced and formulated as the “Truck Dispatching Problem” by Dantzig & Ramser (1959). In its classical form, the VRP considers a single depot, a set of customers with known deterministic demands, and a homogeneous fleet of vehicles with limited capacity. The objective is typically to minimize total operational costs (commonly expressed as total travel distance), the number of vehicles deployed, or a combination of both. The optimization is subject to several core constraints: vehicle capacities must not be exceeded, each customer must be serviced exactly once, and every route must start and end at the depot.

The VRPTW was subsequently introduced and formalized by Solomon (1987). Its defining characteristic is the incorporation of temporal constraints, commonly referred to as time windows. These specify predefined intervals during which service at each customer must commence. In literature, time windows are generally classified as either hard or soft. Hard time windows constitute strict constraints: vehicles must arrive and begin service within the specified interval. Early arrival results in waiting until the opening of the time window, whereas late arrival is infeasible. In contrast, soft time windows allow service outside the prescribed interval but impose a penalty cost. Also, time windows can be one-sided, which is the case when it is stated as the latest time for delivery (Desaulniers et al., 2014).

In this thesis, the VRPTW model is adapted from the formulation of Kallehauge et al. (2005):

$$min \sum_{k \in V} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ijk}$$

subject to:

$$\sum_{k \in V} \sum_{j \in N} x_{ijk} = 1 \qquad \forall i \in C \qquad (1)$$

$$\sum_{i \in C} w_i \sum_{j \in N} x_{ijk} \leq q \qquad \forall k \in V \qquad (2)$$

$$\sum_{j \in N} x_{0jk} = 1 \qquad \forall k \in V \qquad (3)$$

$$\sum_{i \in N} x_{ihk} - \sum_{j \in N} x_{hjk} = 0 \qquad \forall h \in C, \forall k \in V \qquad (4)$$

$$\sum_{i \in N} x_{i,n+1,k} = 1 \qquad \forall k \in V \qquad (5)$$

$$A_{ik} + s_i + t_{ij} - A_{ij} \leq M(1 - x_{ijk}) \qquad \forall i, j \in N, \forall k \in V \qquad (6)$$

$$e_i \leq A_{ik} \leq l_i \qquad \forall i \in N, \forall k \in V \qquad (7)$$

$$A_{0k} = t_{m,start} \qquad \forall k \in V_m \qquad (8)$$

$$A_{0k} = t_{a,start} \qquad \forall k \in V_a \qquad (9)$$

$$A_{n+1,k} \leq t_{m,end} \qquad \forall k \in V_m \qquad (10)$$

$$A_{n+1,k} \leq t_{a,end} \qquad \forall k \in V_a \qquad (11)$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i, j \in N, \forall k \in V \qquad (12)$$

Sets and Indices

V: Set of all available homogeneous vehicles

V_m, V_a: Subsets of vehicles assigned to morning and afternoon shifts, respectively

C: Set of customer nodes $\{1, 2, \ldots, n\}$

N: Set of all nodes, including start depot (0) and end depot $(n + 1)$

i, j, h: Indices representing nodes

k: Index representing a specific vehicle

Decision Variables

x_{ijk}: Binary variable; 1 if vehicle k travels directly from node i to node j, 0 otherwise

A_{ik}: Arrival time (and start of service) for vehicle k at node i

Parameters

c_{ij}: Travel distance between node i and node j

t_{ij}: Travel time required to move from node i to node j

s_i: Service time required at customer i

w_i: Demand (shipment weight) of customer i

q: Maximum load capacity of each vehicle

e_i: Start of time window

l_i: End of time window

$t_{m,start}$: Fixed start time for morning shift (7:00)

$t_{m,end}$: Fixed end time for morning shift (11:30)

$t_{a,start}$: Fixed start time for afternoon shift (12:30)

$t_{a,end}$: Fixed end time for afternoon shift (17:00)

M: A very large number (Big-M)

The objective function minimizes the total travel costs. Constraint (1) ensures that each customer is visited exactly once. Constraint (2) ensures that a vehicle can only be loaded up to its maximum load capacity. Constraint (3) ensures that each vehicle must start its tour at the depot. Constraint (4) is the flow conservation constraint which ensures that if a vehicle arrives at a customer, it must also leave it for another destination. Constraint (5) ensures that each vehicle finishes its tour at the depot again. The inequality constraint (6) is the sub-tour elimination constraint that ensures that the start of service at node j cannot occur before the completion of service time at its predecessor i (s_i) and the corresponding travel time t_{ij}. Constraint (7) ensures that arrival time at node i must start no earlier than e_i and no later than l_i.

Constraint (8) and (9) ensure that ensures that morning routes and afternoon routes start at the exact defined start time, respectively. Constraint (10) and (11) are the HOS constraints that ensure that the morning routes and afternoon routes must not end after the designated finish time, respectively. Finally, constraint (9) ensures the binary nature of the decision variables.

Following consultation with Gebrüder Weiss, several assumptions were established to define the scope of this research. First, customer time windows are treated as hard constraints. Second, despite the presence of special shipment transport requirements in some instances, a homogeneous fleet is assumed due to the low frequency of such cases. To comply with HOS regulations, each route is limited to a maximum duration of 4.5 hours (270 minutes), with drivers performing up to two tours per day. Finally, the temporal scope is defined by the depot's operating hours (07:00 to 17:00). Consequently, the morning shift must finish at 11:30, and the afternoon shift commences at 12:30.

2. Literature Review

Given the NP-hard nature of routing problems, a wide range of approaches have been developed to address their computational complexity in literature. This chapter provides an overview of selected solution approaches. Section 2.1 reviews the principal categories of methods for the VRP, including exact algorithms, heuristics, and metaheuristics. Section 2.2 narrows the focus to metaheuristic approaches specifically applied to the VRPTW. Finally, Section 2.3 reviews the operators employed in Large Neighborhood Search (LNS), with particular emphasis on the destroy and repair operators.

2.1 Review of Solution Approaches on VRP

The classical VRP is one of the most studied problems of logistics and operations research. As an NP-hard optimization problem, its computational complexity grows exponentially with the number of nodes, making it exceptionally challenging to solve as problem instances increase. To this day, an extensive body of literature has been developed to address the VRP and its many variants. Generally, these solution approaches can be classified into three major categories: exact algorithms, heuristic algorithms, and metaheuristic algorithms (Konstantakopoulos et al., 2022).

Exact algorithms are designed to explore the solution space systematically to guarantee the identification of the global optimum (given that the problem is solvable). However, this mathematical certainty comes at a high computational cost. The required processing time often becomes prohibitive as the problem size increases, which is often the case when it comes to industrial applications (Toth & Vigo, 2014). Examples of exact algorithms include branch-and-bound (Laporte & Nobert, 1987), column generation (Desrochers et al., 1992), dynamic programming (Fisher et al., 1997), and branch-price-and-cut (Costa et al., 2019).

Heuristics are "rules of thumb" designed to find a feasible and good solution within a short timeframe. Unlike exact algorithms, heuristics do not explore the entire solution space and thus cannot guarantee global optimality. Furthermore, they are susceptible to becoming trapped in local optima, i.e., when no single small change can yield further improvement. For routing problems, heuristics can be categorized into two stages: construction heuristics, which build an initial feasible solution from scratch, and improvement heuristics, which refine existing routes. Famous examples of the former include the Savings algorithm (Clarke & Wright, 1964), which iteratively merges routes based on distance-saving calculations, and the sweep heuristic (Gillett & Miller, 1974), which clusters customers using polar coordinates.

Metaheuristics serve as high-level algorithmic frameworks that deploy lower-level, problem-specific heuristics to navigate complex search landscapes. While simple heuristics could often become trapped in local optima, metaheuristics employ stochastic elements and memory structures to explore broader regions of the solution space, balancing exploration (diversification) and exploitation (intensification) (Blum & Roli, 2003). In literature, metaheuristics are generally categorized by their search philosophy. Local search-based metaheuristics, such as Simulated Annealing (Kirkpatrick et al., 1983) and Tabu Search (Glover, 1986), move between neighboring solutions by accepting occasionally non-improving moves to escape local optima. Another local search method is the Large Neighborhood Search (Shaw,

1998), which uses a destroy and repair mechanism that allows the search to jump to distant regions of the solution space. On the other hand, population-based metaheuristics, like Genetic Algorithms (Baker & Ayechew, 2003), maintain a set of diverse solutions and recombine them to evolve higher-quality results. Others are nature-inspired, such as Ant Colony Optimization (Bullnheimer et al., 1999), which mimics the pheromone-trailing behavior of ants to find shortest paths. Finally, construction-based metaheuristics, such as the Greedy Randomized Adaptive Search Procedure (Feo & Resende, 1995), iteratively build solutions using randomized greedy logic.

When it comes to solving the VRP and its variants, academic literature consistently demonstrates that metaheuristics are the most widely adopted approaches. According to Braekers et al. (2016), an analysis of 277 VRP articles published between 2009 and mid-2015 revealed that 233 utilized metaheuristics, while 56 applied exact methods and 32 used classic heuristics. This trend is further supported by Konstantakopoulos et al. (2022) who expanded the scope to 406 VRP papers published between 2010 and 2020. Their findings indicate that 252 of these studies applied metaheuristic methods, reinforcing their status as the dominant tool for addressing modern optimization challenges.

2.2 Review of Metaheuristics on VRPTW

A wide range of metaheuristic approaches have been proposed and applied to the VRPTW in literature. To review the different metaheuristics used for the VRPTW, several papers were examined with respect to key aspects. These include the applied methodology/metaheuristics, the underlying instances (benchmark and/or real instances), the type of time windows (hard vs. soft), and the fleet composition (homogeneous vs. heterogeneous). An overview of all named literature can be found in Table 1.

Author(s)	Year	Method(s)	Instances	TW	Fleet
Pureza et al.	2012	TS, ACO	benchmark instances	hard	homogenous
Johar et al.	2015	LNS, VNS, TS	benchmark instances	soft	homogenous
Sun et al.	2019	ALNS, TS	benchmark instances	soft	homogenous
Gocken & Yaktubay	2019	GA	benchmark instances	hard	homogenous
Wu et al.	2019	ACO	benchmark instances	soft	homogenous
Cruz-Chávez et al.	2019	GA	real instances	hard	homogenous
Jiang et al.	2020	VNS	benchmark instances	hard	heterogenous
Konstantakopoulos et al.	2020	MOLNS	benchmark instances	hard	homogenous
Ali et al.	2021	ALNS	benchmark and real instances	hard	heterogenous
Pan et al.	2021	ALNS, VND	benchmark instances	hard	homogenous
He et al.	2021	VNS, ACO	benchmark instances	soft	homogenous
Gmira et al.	2021	TS	benchmark instances	hard	homogenous
Kosolsombat & Ratanavilisagul	2022	ACO	benchmark instances	hard	homogenous
Zhang et al.	2022	TS	benchmark instances	hard	homogenous
Aydınalp & Özgen	2023	SA, ALNS	real instances	hard	homogenous
Syafrizal & Sugiharti	2023	TS, GA	benchmark instances	soft	homogenous
Azmi & Hamontree	2024	GA	real instances	hard	homogenous
Ketaren et al.	2025	GA	real instances	hard	homogenous
Nguyen et al.	2026	ACO, AGES, LNS	benchmark instances	hard	homogenous
Xu et al.	2026	VNS	real instances	hard	homogenous

Table 1: Overview of metaheuristics employed for VRPTW

One of the most widely adopted metaheuristic approaches is Ant Colony Optimization (ACO). Pureza et al. (2012) addressed a real-world VRPTW with multiple deliverymen, applying ACO to Solomon's classic VRPTW datasets, with the objective to minimize a total cost function that accounts for fleet size, number of deliverymen, and total distance traveled. Wu et al. (2019) presented a novel brainstorming-based ACO to address VRPTW with soft time windows, targeting total cost minimization, including penalty costs in addition to fleet size and distance. He et al. (2021) introduced an Adaptive Variable Neighborhood Search (VNS) ACO that incorporates two variable neighborhood search operators, swap and insertion, to improve solution quality. Kosolsombat & Ratanavilisagul (2022) introduced an ACO with innovative selection, elimination, and re-initialization techniques.

Similarly, Genetic Algorithms (GA) are extensively utilized. Gocken & Yaktubay (2019) applied a GA on Solomon's C1, C2, R1, and R2 benchmark instances, aiming to minimize both the total distance and the total waiting time. Cruz-Chávez et al. (2019) introduced a Grid-Based GA that leverages grid computing infrastructure. Azmi & Hamontree (2024) implemented a GA for a real-world automotive parts company with 40 customers. Ketaren et al. (2025) employed a GA for a multi-objective VRPTW designed for waste transportation, targeting minimization of road risk along with total distance, and total travel time.

As established methods, Tabu Search (TS) and VNS are also commonly used to solve the VRPTW. Gmira et al. (2021) proposed a TS approach for a VRPTW with varying travel time, which explores neighborhood structure by swapping sequences of customers of arbitrary length between two routes. Zhang et al. (2022) proposed a hybrid algorithm which combines density peak clustering with TS to enhance the solution efficiency for large-scale VRPTW instances. Syafrizal & Sugiharti (2023) combined GA and TS with the integration of fuzzy logic to address a VRPTW with fuzzy time windows. Jiang et al. (2020) introduced a VNS-based evolutionary algorithm for a VRPTW with heterogeneous fleet for hazardous material. Xu et al. (2026) introduced a reinforcement learning-based adaptive VNS that uses a policy network based on solution context and past performance to select operators dynamically.

Furthermore, LNS and its advanced variant Adaptive Large Neighborhood Search (ALNS) are widely adopted in the literature, often hybridized with other metaheuristics. Johar et al. (2015) utilized LNS along with TS and VNS as improvement heuristics on modified Solomon instances. Sun et al. (2019) integrated TS and ALNS into a framework capable of processing real-time information for a dynamic Pickup and Delivery Problem with time windows (PDPTW). Konstantakopoulos et al. (2020) proposed a multi-objective LNS focused on simultaneously minimizing fleet size and total distance. Ali et al. (2021) developed a tailored ALNS heuristic for the VRPTW, achieving results comparable to exact approaches within competitive

computational times. Pan et al. (2021) combined ALNS for global exploration with Variable Neighborhood Descent (VND) for local exploitation. Aydınalp & Özgen (2023) implemented SA and ALNS for a real-life pharmaceutical distribution problem. Nguyen et al. (2026) introduced a hybrid framework for a PDPTW using ACO, Adaptive Guided Ejection Search (AGES), and LNS.

Given the proven effectiveness of LNS in handling the complex constraints of the VRPTW as demonstrated in the literature, it is selected as the primary metaheuristic framework for this thesis.

2.3 Review of Operators of LNS

The LNS framework is primarily defined by a "destroy and repair" philosophy. The core concept involves systematically dismantling parts of an initial solution and subsequently rebuilding the damaged structure in an attempt to achieve a better result. In the context of VRPTW, this entails removing specific customers from existing routes and reinserting them back into the solution.

Extensive literature on (A)LNS has proposed a wide array of destroy and repair operators. To maintain high levels of diversification and avoid local optima, researchers often implement multiple operators simultaneously. During each iteration, the algorithm selects different operator pairs, ensuring a varied search pattern across the solution space. This section reviews several operators that are used in the LNS-related literature.

Johar et al. (2015) implemented their LNS algorithm using two foundational operators, random removal (RaR) and greedy repair (GR). RaR provides stochastic diversification by removing customers at random, while GR employs a myopic insertion strategy, reintroducing customers into routes one at a time based on the minimum incremental increase in total distance.

Konstantakopoulos et al. (2020) also adopted RaR and GR but expanded their framework to include route removal (RoR) and worst removal (WR). RoR enhances search diversification by deleting an entire route from the solution, forcing a complete reassignment of its customers. In contrast, WR iteratively targets customers who contribute most significantly to the objective function's cost. Specifically, the authors developed three variations of the WR operator. The distance-based variation removes the customer who exhibits the greatest distance relative to their predecessor and successor. The time-based variation targets the customer with the longest waiting time, while the hybrid variation utilizes a composite metric that balances both factors.

Ali et al. (2021) implemented an ALNS framework utilizing RaR and related removal (as known as the Shaw Removal) as destroy operators. Their version, inspired by Shaw (1998), begins by removing a customer at random. Subsequent customers are then selected for removal based on their proximity to the initial customer, calculated via Euclidean distance. As for repair operators, the authors employed an operator which iteratively inserts customers into their best insertion positions based on the lowest incremental cost. They also utilized regret-k insertion, which iteratively selects the customer with the highest regret value for reinsertion. The regret value is defined as "the cost difference between the best insertion position (the cheapest route) and the $(k-1)^{th}$ best route for $k > 2$" (Ali et al., 2021). By prioritizing customers with high regret values, the algorithm ensures that those with the most limited or sensitive insertion options are placed before their best positions are taken by other nodes.

Pan et al. (2021) employed five different destroy operators for the ALNS framework. Aside from RaR and WR, they also employed related removal, a trip-effectiveness based related removal, and a mixed selection of these two operators. The latter is similar to the related removal but differs in how the initial seeds customers are selected. Rather than selecting randomly, this operator evaluates the effectiveness of each route which is calculated as the ratio of demand served to incurred cost. The most ineffective route is identified, and customers from

this route are selected as seeds. As for the repair operators, they used the regret-k heuristic with k selected from $\{1, 2, 3\}$. The authors also employed a roulette wheel selection mechanism to choose between destroy and repair operators. In this approach, each operator is assigned a dynamic weight reflecting its historical performance, and the selection probability for an operator is proportional to its weight relative to the sum of all operator weights.

Windras Mara et al. (2022) employed another destroy operator, the block destroy removal. The idea here is to remove a contiguous sequence of nodes. The operator selects two nodes at random and removes all customers located between them within the route structure. To complement this, they employed the archive repair operator. This approach maintains an archive of elite, low-cost solutions, with updates occurring whenever a new global best is found. During the repair phase, the operator selects a random elite solution from the archive and uses its corresponding route segments to populate the voids in the current destroyed solution, specifically matching the sequence indices of the previously removed components.

Considering the scope of the problem instances, the common usage and implementational complexity, it is decided that for this thesis, RaR, RoR and WR will be implemented as removal operators, while regret-2 removal and GR will be implemented as repair operators.

3. Methodology

This thesis employs a metaheuristic framework to address the VRPTW. The optimization procedure is structured into two primary phases: the construction phase and the LNS phase.

The process begins with the construction phase, which focuses on generating an initial feasible solution that satisfies all operational constraints. Once a baseline is established, the algorithm transitions into the LNS phase to iteratively refine the solution and minimize the objective value (i.e., total distance).

As previously discussed, the LNS follows a “destroy and repair” strategy. Accordingly, this phase is further subdivided into two steps: the destroy stage applies a selection of removal operators to partially deconstruct the current solution by removing specific nodes or routes, whereas the repair stage employs reinsertion operators to reconstruct the damaged solution.

Section 3.1 describes the two construction heuristics implemented, while Section 3.2 provides an overview of the various destroy and repair operators utilized within the LNS framework.

3.1 Construction Heuristics

In this session, two greedy construction heuristics used in the metaheuristic optimization are explained in detail: nearest neighbor (NN) and earliest due date (EDD). While these two approaches are often considered myopic due to their local decision-making, they provide a computationally efficient starting point for LNS optimization. Because of the similarity between the two heuristics, they utilize the same general algorithmic framework, differing only in their customer selection criteria.

Input: customers C
Output: list of routes R

$U \leftarrow C$
$R \leftarrow \emptyset$
while U is not empty **do**
    $r_{morning} \leftarrow$ new route (morning start)
    BuildRoute($r_{morning}, U$)
    add $r_{morning}$ to R
    $r_{afternoon} \leftarrow$ new route (afternoon start)
    BuildRoute($r_{afternoon}, U$)
    if $r_{afternoon}$ is not empty **then**
        add $r_{afternoon}$ to R
    end if
end while
return R

Algorithm 1: Construction

Algorithm 1 outlines the general procedure for the construction phase. The process begins by importing a list of customers C from the input data and initializing a set of unrouted customers, U, and an empty list of routes, R. The algorithm iteratively executes the following steps until all customers are routed: First, because the shift starts in the morning, an empty morning route $r_{morning}$ is created, with the starting time initialized at the beginning of the day (i.e., 7:00).

$r_{morning}$ is then constructed using the BuildRoute helper method (Algorithm 2), which invokes either the NN (Algorithm 3) or EDD (Algorithm 4) heuristic to generate a list of feasible customers F, sorted by the respective selection criteria. Once $r_{morning}$ is constructed, this route is added to the sets of routes R. Subsequently, an afternoon tour $r_{afternoon}$ is created and constructed in the same manner. If $r_{afternoon}$ is not empty, it is added to R. This process continues until all customers have been routed.

Input: route r, set of unrouted customers U
Output: updated r and U
1 ordered customer list $F \leftarrow$ GetFeasibleCustomers(r, U)
2 **while** F is not empty **do**
3 $\quad c \leftarrow F[0]$
4 $\quad$ append c to r
5 $\quad U \leftarrow U \setminus \{c\}$
6 $\quad F \leftarrow$ GetFeasibleCustomers(r, U)
7 **end while**

Algorithm 2: BuildRoute

Algorithm 2 describes how the helper method BuildRoute works in detail. Before each insertion, a method based on NN or EDD is called which returns a list of feasible customers F, sorted according to the respective criteria. As long as the list F is not empty, the first customer in F is appended to the end of the route. After each insertion, the set of unrouted customers U and the feasible customer list F are updated accordingly.

3.1.1 Nearest Neighbor Heuristic

The NN heuristic is considered as one of the oldest and fundamental construction heuristics for many routing problems. The idea is to repeatedly select the closest unrouted customer to the last visited customer on the route, while respecting all additional constraints. For the VRPTW, the NN must be adapted to consider time-related constraints, aside from the vehicle capacity constraint.

Input: route r, list of unrouted customers U
Output: ordered list of feasible customers F
1 $F \leftarrow \emptyset$
2 **for each** customer $c \in U$ **do**
3 **if** c fits vehicle capacity of r **and**
4 c can be served within its time window **and**
5 c meets HOS constraint **then**
6 $F \leftarrow F \cup \{c\}$
7 **end if**
8 **end for**
9 sort all customers in F in ascending order of their travel distances to the last customer in r
10 **return** F

Algorithm 3: GetFeasibleCustomersUsingNN

Algorithm 3 describes the procedure of the method GetFeasibleCustomersUsingNN. It returns a list of feasible customers sorted by their distance to the last customer on the route. If no customer has been visited yet (i.e., route r is empty), all feasible customers are sorted by their distance to the depot.

Feasibility here is evaluated based on three dimensions: First, the vehicle capacity constraint is checked by ensuring that adding the customer does not cause the total shipment weight of the route to exceed the vehicle's capacity. Second, the time window constraint is verified by checking whether visiting the customer would violate the customer's specified time window. Finally, the algorithm ensures compliance with the HOS constraint by verifying that adding the customer to the route would not cause the driver to return to the depot later than the predefined

end time of their shift. If a customer fits all three criteria, they are inserted into the list F. Lastly, all customers in F are sorted by their travel distance to the last customer on the route.

3.1.2 Earliest Due Date Heuristic

The EDD heuristic is similar to the NN approach. The main idea here is to sort all feasible customers by their earliest due date (i.e., their appointment time/the end of their time window) and then iteratively select the first customer in the sorted list, namely the most urgent one.

Algorithm 4 showcases how the EDD heuristic is implemented. The only difference with NN (Algorithm 3) is the way feasible customers are sorted.

Input: route r, list of unrouted customers U
Output: ordered list of feasible customers F

```
F ← ∅
for each customer c ∈ U do
    if c fits vehicle capacity of r and
    c can be served within its time window and
    c meets HOS constraint then
        F ← F ∪ {c}
    end if
end for
sort all customers in F in ascending order of their appointment time
return F
```

Algorithm 4: GetFeasibleCustomersUsingEDD

3.2 The LNS Metaheuristic

As previously mentioned, the LNS framework consists of a destroy stage and a repair stage, each utilizing a distinct set of operators. During the experimental phase, the specific operators employed per iteration vary depending on the general configuration of the algorithm. This section provides an overview of the implemented LNS framework, including the acceptance criterion. The individual operators are explained in detail in the respective subsections. The specific parameter settings and configurations for the experiments are discussed in Chapter 4.

Input: customers C
Output: best routes R^*

$R_0 \leftarrow$ Construction(C)
$d^* \leftarrow$ GetTotalDistance(R_0)
$R_{current} \leftarrow R_0$
$R^* \leftarrow R_0$
while time elapsed < time limit **do**
  $R_{work} \leftarrow$ DeepCopy($R_{current}$)
  $\text{op}_\text{D} \leftarrow$ randomly chosen operator from {RandomRemoval, RouteRemoval, WorstRemoval}
  $L \leftarrow \text{op}_\text{D}(R_{work})$
  $\text{op}_\text{R} \leftarrow$ randomly chosen operator from {GreedyRepair, RegretRepair}
  success $\leftarrow \text{op}_\text{R}(R_{work}, L)$
  if not success **then**
    continue
  end if
  $d_{new} \leftarrow$ GetTotalDistance(R_{work})
  if $d_{new} < d^*$ **then**
    $d^* \leftarrow d_{new}$
    $R_{current} \leftarrow$ DeepCopy(R_{work})
    $R^* \leftarrow$ DeepCopy(R_{work})
  end if
end while
return R^*

Algorithm 5: Large Neighborhood Search

Algorithm 5 shows the general structure of the LNS implementation, with details regarding inputs and parameter settings omitted for better clarity. The algorithm starts by using a construction heuristic to generate an initial solution, R_0. The total distance of R_0 is computed and stored initially as the global best distance, d^*, while R_0 itself is assigned to both the current solution, $R_{current}$, and the best solution, R^*.

The algorithm uses total runtime as stopping criterion and enters an iterative improvement phase. In each iteration, a deep copy R_{work} of $R_{current}$ is created as a working copy to isolate experimental changes from the established baseline. As long as the elapsed time has not reached

the given time limit, A destroy operator ($\mathrm{op_D}$) is randomly selected to remove a randomly determined number of customers from R_{work}, returning them in a list L.

Subsequently, a repair operator ($\mathrm{op_R}$) is randomly chosen to reinsert the customers in L into R_{work}. If the repair fails (i.e., there is at least one customer in L who could not be reinserted), the iteration is aborted, leaving $R_{current}$ unchanged. If the repair is successful, the new distance d_{new} is calculated. Following a hill climbing acceptance criterion, if $d_{new} < d^*$, the algorithm updates d^*, $R_{current}$, and R^* with the improved state. Upon reaching the time limit, the algorithm returns R^*.

Figure 1 illustrates the procedure of the Large Neighborhood Search (LNS) framework in the form of a flowchart. The algorithm begins by using the Open Source Routing Machine (OSRM) to compute the distance and time matrices from the customer-related data. These matrices, together with the customer data, serve as inputs to the algorithm.

During the construction phase, an initial solution is generated by selecting one of two heuristics: nearest neighbor or earliest due date. In the destroy phase, one operator is randomly selected from three implemented destroy operators (random removal, route removal, and worst removal). Similarly, in the repair phase, one operator is randomly chosen from two implemented repair operators (greedy repair and regret repair) to reconstruct the solution. The algorithm iteratively repeats these steps until the predefined time limit is reached, at which point the best solution found so far is returned.

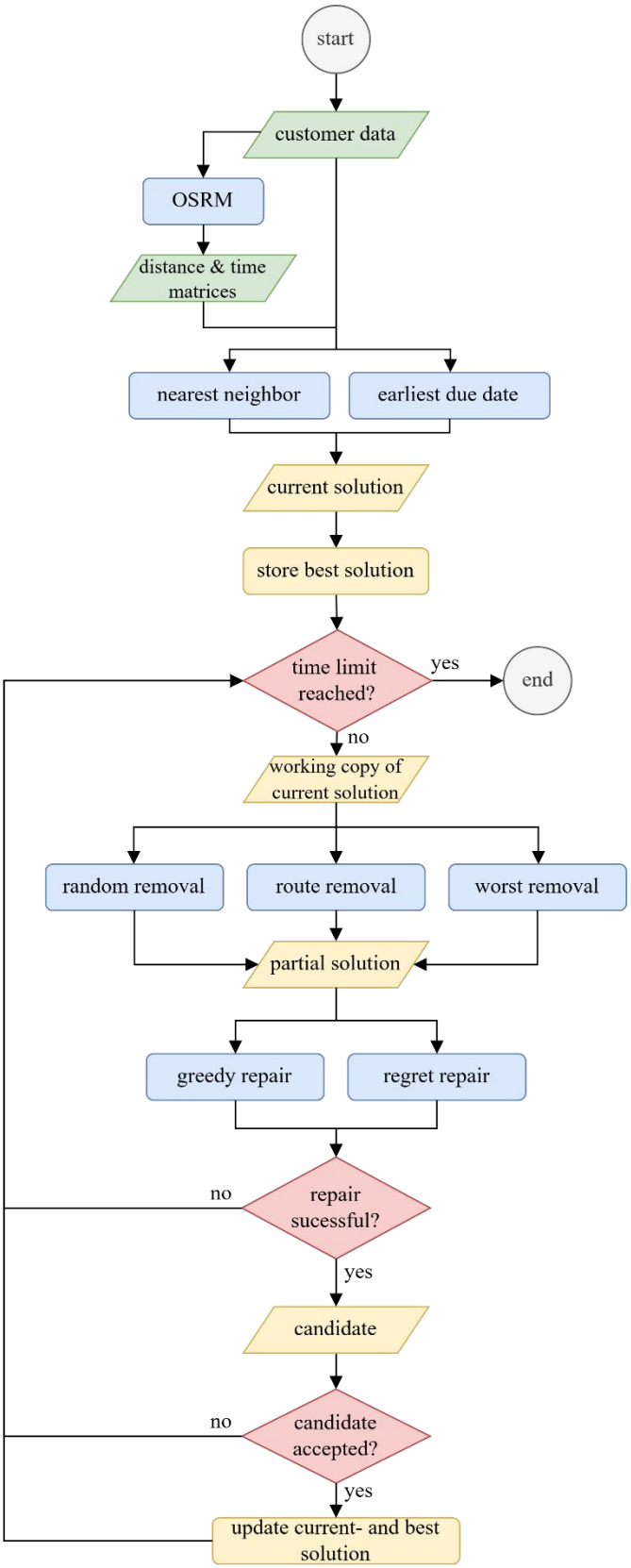

Figure 1: LNS Framework

3.2.1 Destroy Operators

Random Removal

The first destroy operator that is implemented within the LNS optimization program is the RaR heuristic. Its idea is fairly simple: it randomly selects customers from the routes, removes them from the routes, and stores them in a separate list that is later used to reinsert the removed customers.

Algorithm 6 illustrates this procedure in detail. The algorithm takes the solution generated by the construction heuristic, which is a set of routes R, and a predefined removal intensity k, and returns a list of removed customers.

Input: set of routes R, removal intensity k
Output: list of removed customers L
1 $n_{total} \leftarrow$ sum of customers in all routes
2 $n_{remove} \leftarrow$ random integer $\in [1, \lfloor k \cdot n_{notal} \rfloor]$
3 $L \leftarrow \emptyset$
4 **while** $|L| < n_{remove}$ **do**
5 $\quad r \leftarrow$ random route from R
6 $\quad c \leftarrow$ random customer from r
7 $\quad r \leftarrow r \setminus \{c\}$
8 $\quad r.\text{load} \leftarrow r.\text{load} - c.\text{weight}$
9 $\quad L \leftarrow L \cup \{c\}$
10 **end while**
11 **return** L

Algorithm 6: Random Removal

First, the algorithm sums up the number of customers in each route $r \in R$ to obtain the total number of customers n_{total}. It then draws a random integer n_{remove} between 1 and $\lfloor k \cdot n_{notal} \rfloor$. Next, it initializes an empty set L to store the removed customers.

The algorithm then repeatedly performs the following steps while $L < n_{remove}$: it selects a random non-empty route $r \in R$, chooses a random customer $c \in r$, removes c from r, and updates the total load of r accordingly. Finally, it appends c to L to record the removal.

Route Removal

The second destroy operator utilized in the program is RoR heuristic. The logic of this operator is more direct than RaR: instead of selecting individual customers, an entire route is chosen at random, and all its assigned customers are removed simultaneously. This process repeats until the required total number of removed customers is reached.

Input: set of routes R, removal intensity k
Output: list of removed customers L
$n_{total} \leftarrow$ sum of customers in all routes
$n_{remove} \leftarrow$ random integer $\in [1, \lfloor k \cdot n_{notal} \rfloor]$
$L \leftarrow \emptyset$
while $|L| < n_{remove}$ **do**
  $r \leftarrow$ random route from R
  $L \leftarrow L \cup r$
end while
return L

Algorithm 7: Route Removal

Algorithm 7 illustrates this operator in detail. While the initial steps are identical to Algorithm 6, the removal of entire routes, rather than individual customers, typically results in a larger volume of removed customers. Consequently, this operator provides a higher level of diversification within the search process. By emptying multiple routes, it allows repair operators to reinsert customers into entirely different sequences. It is also worth noting that empty routes are deliberately kept open here to ensure that repair operators could reinsert removed customers back into these original routes.

Worst Removal (Inspired by Røpke & Pisinger)

The final destroy operator implemented is the WR heuristic. The core objective is to iteratively select customers that contribute most significantly to the total route distance. To mitigate the myopic nature of a purely greedy selection, a randomized biased version of WR is implemented, as proposed by Ropke & Pisinger (2006).

Input: set of routes R, removal intensity k, randomization parameter p
Output: list of removed customers L

$n_{total} \leftarrow$ sum of customers in all routes
$n_{remove} \leftarrow$ random integer $\in [1, \lfloor k \cdot n_{notal} \rfloor]$
$L \leftarrow \emptyset$
while $|L| < n_{remove}$ **do**
  candidate list $C \leftarrow \emptyset$
  for each route $r \in R$ **do**
    for position $i = 0$ to $|r.\text{customers}| - 1$ **do**
      $c \leftarrow r.\text{customers}[i]$
      if $i = 0$ **then**
        $prev \leftarrow$ depot
      else
        $prev \leftarrow r.\text{customers}[i-1]$
      end if
      if $i = |r.\text{customers}| - 1$ **then**
        $next \leftarrow$ depot
      else
        $next \leftarrow r.\text{customers}[i+1]$
      end if
      cost $\leftarrow Distance(prev, c) + Distance(c, next) - Distance(prev, next)$
      $C \leftarrow C \cup \{(c, r, \text{cost})\}$
    end for
  end for
  sort C by cost in descending order
  $y \leftarrow$ random number $\in [0, 1)$
  $index \leftarrow \lfloor y^p \cdot |C| \rfloor$
  $(c, r) \leftarrow C[index]$
  $r \leftarrow r \setminus \{c\}$
  $r.\text{load} \leftarrow r.\text{load} - c.\text{weight}$
  $L \leftarrow L \cup \{c\}$
end while
return L

Algorithm 8: Worst Removal (randomized)

Algorithm 8 describes how this heuristic works in detail. In addition to the initial solution R and the removal intensity k, the algorithm requires a parameter p to control the degree of randomization. When $p = 1$, the selection process is uniformly random. As p increases, the

degree of randomization decreases, and the selection becomes more biased toward customers with lower indices.

The algorithm works as the following: While the target number of removal n_{remove} has not been reached, in each iteration, a candidates list C is initialized. For every customer c in every route r, the algorithm calculates the cost contribution of that customer. This is determined by identifying the predecessor ($prev$) and successor ($next$). If c is the first or last node in a route, the predecessor or successor is the depot, respectively. The insertion cost is defined as:

$$cost = Distance\ (prev, c) + Distance\ (c, next) - Distance\ (prev, next)$$

Equation 1: Insertion Cost

This value represents the additional distance required to include customer c in its current position. Once all costs are documented, the candidate list C is sorted in descending order. A random number y in $[0, 1)$ is drawn, and the index for removal is calculated as $\lfloor y^p \cdot |C| \rfloor$. For example, if $|C| = 20$, $y = 0.4$, and $p = 3$, the index would be $\lfloor (0.4^3 \cdot 20) \rfloor = 1$, resulting in the removal of the second customer in the sorted list (the first customer has the index 0). After a customer is removed, route load is updated accordingly, and the costs are recalculated in the next iteration until n_{remove} is achieved.

3.2.2 Repair Operators

Greedy Repair (Inspired by Røpke & Pisinger)

The first repair operator implemented in the program is the GR heuristic. The general idea here is to evaluate the insertion costs of all removed customers into existing routes and select those that contribute the least to the increase in total route distance. The randomization parameter p is also introduced here to ensure the algorithm does not always select the cheapest customer, which helps avoid getting stuck in local optima.

Input: set of routes R, list of removed customers L, randomization parameter p
Output: success or failure

```
1  while |L| > 0 do
2      candidate list C ← ∅
3      for each customer c ∈ L do
4          for each route r ∈ R do
5              if c does not fit capacity in r then
6                  continue
7              end if
8              if r.startTime > c.latest then
9                  continue
10             end if
11             for i = 0 to |r.customers| do
12                 if inserting c at i maintains feasibility for r then
13                     cost ← insertion cost of c at i
14                     C ← C ∪ {(c, r, i, cost)}
15                 end if
16             end for
17         end for
18     end for
19     if |C| > 0 then
20         sort C by cost on asending order
21         y ← random number ∈ [0, 1)
22         index ← ⌊y^p · |C|⌋
23         (c, r, i) ← C[index]
24         insert c into r at position i
25         r.load ← r.load + c.weight
26         L ← L \ {c}
27     else
28         return false
29     end if
30 end while
31 return true
```

Algorithm 9: Greedy Repair

Algorithm 9 illustrates this heuristic in detail. The algorithm takes as input a partially destroyed solution and a list of removed customers L, and returns a fully reinserted, i.e., repaired solution. As long as L is not empty, the algorithm initializes an empty list of insertion candidates C at the start of each iteration. It then evaluates every customer $c \in L$ for every possible insertion

position in existing routes $r \in R$. If the insertion is feasible, the option is recorded in C. At the end, if the list C is not empty, meaning that there is at least one feasible insertion, then C is sorted by insertion cost in ascending order. A customer is then selected for insertion using the same randomized process employed in WR (Algorithm 8). The algorithm then proceeds to repair the next removed customer. Once every customer is reinserted, the algorithm returns true, signaling that this iteration is successful.

If C remains empty while the list L is not empty, it indicates that there is at least one removed customer in L who could not be reinserted into any existing route due to constraint violations. In this case, the algorithm returns false, indicating that this iteration has failed and is to be rejected in the main program. The main program then proceeds to the next iteration.

Regret Repair

The second repair operator implemented in the algorithm is the regret repair (RR) heuristic. The idea of this approach is to prioritize the insertion of customers who would incur the greatest "regret" if their insertion were deferred. In this context, regret is defined as the difference in cost between inserting a customer into their best (i.e., cheapest) available position versus a suboptimal one.

Specifically, the regret heuristic implemented here is regret-2, where the regret value is computed as the absolute difference between the insertion cost of the best position and the second-best position. By selecting the customer with the highest regret value in each iteration, the algorithm proactively addresses customers who have limited efficient insertion options, thereby preventing costly insertions in the later stages of the repair process.

Algorithm 10 outlines the procedure of the RR heuristic. While the list of removed customers L is not empty, the heuristic identifies the customer with the highest regret value in each iteration and reinserts them into the solution.

Input: set of routes R, list of removed customers L
Output: success (true) or failure (false)

while $|L| > 0$ **do**
  $bestCustomer \leftarrow$ null
  $bestRoute \leftarrow$ null
  $bestPosIndex \leftarrow -1$
  $maxRegret \leftarrow -\infty$
  for each customer $c \in L$ **do**
    $firstBest, secondBest \leftarrow$ GetTwoBestInsertions(c, R)
    $regret \leftarrow secondBest.\text{cost} - firstBest.\text{cost}$
    if $regret > maxRegret$ **then**
      $maxRegret \leftarrow regret$
      $bestCustomer \leftarrow c$
      $bestRoute \leftarrow firstBest.\text{route}$
      $bestPosIndex \leftarrow firstBest.\text{positionIndex}$
    end if
  end for
  if $bestRoute \neq$ null **then**
    insert $bestCustomer$ into $bestRoute$ at $bestPosIndex$
    $bestRoute.\text{load} \leftarrow bestRoute.\text{load} + bestCustomer.\text{weight}$
    $L \leftarrow L \setminus \{bestCustomer\}$
  else
    return false
  end if
end while
return true

Algorithm 10: Regret Repair

To achieve this, the algorithm maintains four variables: $bestCustomer$ (customer to be reinserted), $bestRoute$ (route for reinsertion), $bestPosIndex$ (position on the route), and $maxRegret$ (highest regret value recorded so far).

At the start of the iteration, the algorithm uses a foreach loop to perform the following steps for each customer $c \in L$: It calls the helper method GetTwoBestInsertions (Algorithm 11) to determine the best and second-best insertion options ($firstBest$ and $secondBest$). The regret value is then computed as the difference between the insertion costs of these two options. If the

regret value exceeds the current value of $maxRegret$, all four variables are updated accordingly.

It is very important to note here that if fewer than two insertion options exist for the customer (i.e., there is either none or only one feasible insertion option), dummy insertion options are created with $route = null$, $positionIndex = -1$, and costs of 1,000,000 or 2,000,000, as the case may be. This ensures customers with limited insertion options receive higher regret values, prioritizing them for reinsertion.

After evaluating all customers (at the end of the foreach loop), the algorithm checks two cases:

In the first case, $bestRoute$ is not null, meaning there exists at least one feasible insertion. The most urgent customer $bestCustomer$ is then inserted into $bestRoute$ at $bestPosIndex$. The algorithm then proceeds to repair the next removed customer.

In the second case, $bestRoute$ is null, meaning that no feasible insertion options exist for the most urgent customer, and the repair fails. This indicates that at least one removed customer cannot be feasibly inserted into any existing route due to capacity, time window, or HOS constraints. The algorithm returns false, and the entire iteration of destroyed and repaired solution is rejected in the main program.

If all removed customers are successfully reinserted, the algorithm returns true, signaling that this iteration is successful.

Algorithm 11 describes how the $firstBest$ and $secondBest$ insertion options are determined for a customer c. First, an empty set $feasibleInsertions$ is initialized. The algorithm then uses a foreach loop to evaluate each route r in the current solution R.

Input: customer c, set of routes R
Output: best and second-best insertion options for c

```
feasibleInsertions ← ∅
for each route r ∈ R do
    if c is capacity incompatible or time incompatible with r then
        continue
    end if
    for position i = 0 to |r.customers| do
        if inserting c at i maintains feasibility for r then
            cost ← insertion cost of c at i
            feasibleInsertions ← feasibleInsertions ∪ {(r, i, cost)}
        end if
    end for
end for
sort feasibleInsertions by cost in ascending order
if |feasibleInsertions| > 0 then
    firstBest ← feasibleInsertions[0]
else
    firstBest ← {route = null, positionIndex = −1, cost = 1,000,000}
end if
if |feasibleInsertions| > 1 then
    secondBest ← feasibleInsertions[1]
else
    secondBest ← {route = null, positionIndex = −1, cost = 2,000,000}
end if
return firstBest, secondBest
```

Algorithm 11: GetTwoBestInsertions

For each route, a preliminary feasibility check is performed to exclude routes that are infeasible for customer c. Such routes include those without sufficient remaining capacity to accommodate the demand of c, as well as routes whose start time exceeds the latest allowable time window of c (for example, an afternoon route cannot serve a customer that must be delivered before noon).

If a route passes this precheck, a for loop is used to evaluate every possible insertion position within route r. For each position, it is checked whether inserting customer c would render the entire route infeasible, for instance by causing time window violations for subsequent

customers or preventing the vehicle from returning to the depot within the allowed time. Only feasible insertion positions are considered further.

For each feasible insertion position, the insertion cost is computed using Equation 1, i.e., the additional travel distance incurred by inserting customer c. The route r, insertion position i, and corresponding cost are then stored as a feasible insertion in $feasibleInsertions$.

After all routes and insertion positions have been evaluated, the set $feasibleInsertions$ is sorted in ascending order of insertion cost. The $firstBest$ and $secondBest$ insertion options are determined as the first and second entries of this sorted list, respectively, if at least two feasible options exist. As mentioned in Algorithm 10, if no feasible insertion options exist for customer c, dummy insertion options are created. $firstBest$ is assigned with dummy cost of 1,000,000 with $route = null$ and $positionIndex = -1$. $secondBest$ is assigned cost 2,000,000 with the same null values. If exactly one feasible option exists, only $secondBest$ is created as a dummy, and the regret is 2,000,000 – actual cost of $firstBest$. This ensures that regret values are artificially inflated for customers with limited insertion options.

4. Data and Experiments

The complete LNS algorithm was implemented in C# utilizing the .NET 8.0 framework. All computational experiments were conducted on a laptop equipped with an Intel Core i5-8350U CPU (1.70 GHz) and 8 GB of RAM, operating on a 64-bit Windows environment. A total of four distinct experimental series were carried out to test the solution quality of different configurations within the implemented LNS framework.

Section 4.1 provides an overview of the input data received from Gebrüder Weiss and the respective preprocessing steps. Section 4.2 explains the detailed settings and configurations of each experiment, while section 4.3 discusses the computational results.

4.1 Data Preprocessing & Assumptions

The empirical basis of this research are the historical dispatch logs provided by Gebrüder Weiss with permission for academic use. The dataset comprises 33 instances covering the period from September 1st to October 15th, 2025 (weekends excluded). The data were extracted from the company's dispatching software and included delivered customer orders from all district pools within the Vienna metropolitan area. This dataset constitutes the primary input for the optimization algorithm and contains detailed customer-related information (e.g., customer name, address, geographic coordinates, demand in kilograms, and time windows) as well as tour-related information (e.g., assigned vehicle, initially estimated time of arrival, and actual time of arrival).

The dataset was preprocessed to prepare input data for the optimization algorithm. In line with the scope of the study, the raw data was filtered to include only shipment orders that were delivered in Vienna's 22nd district (Donaustadt). Furthermore, the dataset was restricted to delivery orders, and pick-up orders were excluded.

In the next step, the filtered data were segmented on a daily basis. For each day within the selected time period, the following information was extracted: (a) customer name, (b) customer street address, (c) customer geographic coordinates, (d) shipment weight, and (e) delivery appointment.

Since the orders are recorded at the shipment level rather than the customer level, the dataset often contains multiple shipments addressed to the same customer on a given day. In addition, data quality issues, such as occasional misspellings of customer names, were observed. Consequently, a data aggregation step was required to consolidate all shipments belonging to the same customer on a given day. This aggregation ensures that customer demands are combined correctly and prevents the algorithm from incorrectly accounting for multiple service times at the same customer.

To do so, all recorded customer orders in the dispatch logs on a given day are first validated based on their geographic coordinates using conditional formatting in Microsoft Excel. Shipment orders sharing identical customer coordinates were then identified and grouped together, and then the respective customer names were compared. If the customer's name indicates that the shipments belonged to the same customer, the respective orders were merged into a single customer order with an aggregated demand, provided that the shipments did not have conflicting delivery appointments. The consolidated customer and order information is subsequently saved as a single CSV file, which serves as an input file for the metaheuristic algorithm.

In the next step, pairwise travel distances and travel times between all customers were computed based on their geographic coordinates. Instead of relying on approximation methods such as Euclidean or Haversine distances, the algorithm utilizes routing matrices derived from OpenStreetMap using the Geofabrik Austria extract. OSRM was employed to calculate realistic road network-based travel times and distances between all customers.

Since OSRM's default routing profile assumes passenger-car travel behavior, while heavy trucks are subject to lower speed limits and route restrictions, the raw OSRM travel times were adjusted using a multiplicative correction factor of 1.5. This rather conservative adjustment aims to improve the practical feasibility of the generated routes. The resulting distance and travel time matrices were then exported as CSV files for use by the algorithm.

In the final preprocessing step, time windows were defined for each customer. It was assumed that delivery operations start at 07:00 and must be completed by 17:00. For customers without a specified delivery appointment, a default time window of 07:00 to 17:00 was assigned. For customers with a specified appointment, the time window was defined as 07:00 up to the appointment time. In cases where the recorded appointment occurred after 17:00, the time window was truncated at 17:00.

4.2. Experimental Setup and Parameter Tuning

Within the scope of this thesis, four series of computational experiments were conducted. For the 33 available instances derived from historical dispatch logs, five independent runs were performed per instance, and the average value was used for evaluation. All experiments were executed under a predefined time limit. A hill-climbing acceptance criterion was applied, i.e., only solutions strictly better than the current best solution were accepted.

The operational parameters were defined as follows: vehicle capacity was set to 6,000 kg, service time to 10 minutes per customer, maximum tour duration to 270 minutes. The depot opening time was set to 7:00, and the closing time to 17:00.

The default configuration for all experiments was defined as follows: a time limit of 180 seconds, the NN heuristic as the construction method, equal selection probability for all destroy and repair operators in each iteration, a destroy size of 0.3, and a randomization parameter of 3 for both worst removal and greedy repair.

The first series of experiments investigated the influence of the construction heuristic on final solution quality. Since the majority of customers in the historical dispatch logs do not have urgent delivery dates, the EDD heuristic typically produces solutions with greater travel distance than NN. The objective was therefore to compare the final solution quality obtained using NN and EDD under otherwise identical default settings.

The second series of experiments analyzed the contribution of individual operators. In each experiment, one operator was deactivated, and the resulting performance was compared to the default configuration, which served as the baseline.

The third series examined the impact of different destroy sizes. Three parameter values were tested (0.1, 0.3, and 0.5) while maintaining the default configuration. The results were then compared across configurations.

The fourth series evaluated the influence of varying time limits. Five different limits (30, 60, 180, 300, and 600 seconds) were tested under the default configuration, and the resulting performance differences were analyzed.

4.3 Computational Results

This section presents and discusses the key findings from computational experiments. For a comprehensive overview of the results, please refer to the appendix.

4.3.1 Experiment Series 1

In the first series of experiments, the optimization program was executed under two different configurations, using NN and EDD as construction heuristics, respectively. The aggregated results across all 33 tested instances for these configurations are presented in Table 2.

Metric (Average across all instances)	EDD	NN
Construction Time (s)	0.0015	0.0041
LNS Time (s)	179.9985	179.9959
Construction Distance	620.40	434.51
Final Distance (Best Run)	355.88	359.11
Final Distance (Average)	362.81	363.35
Δ Best Run (%)	42.2%	17.2%
Δ Average (%)	41.1%	16.2%
Iterations run	17927	16592

Table 2: Aggregated results for Experiment Series 1: construction heuristics

The results indicate that NN significantly outperforms EDD in generating initial solutions. For all tested instances, the initial distance produced by NN was considerably lower than that of EDD. On average, the total distance of the EDD-based initial solutions was 42.8% longer than those generated by NN.

This disparity likely stems from the fact that the majority of customers in these instances do not have narrow or urgent time windows, which is the primary scenario for which the EDD heuristic is designed. Because EDD prioritizes delivery deadlines over spatial proximity, the resulting routes frequently exhibited “crisscrossing” or route intersections leading to substantially longer

total distances. This spatial inefficiency is visualized in Figure 2, where the EDD construction (left) shows notably more overlapping paths than the NN construction (right). Consequently, the poor spatial quality of the EDD-based starting point provided the LNS algorithm with a larger margin for improvement, explaining the higher relative improvement percentages observed in the EDD series.

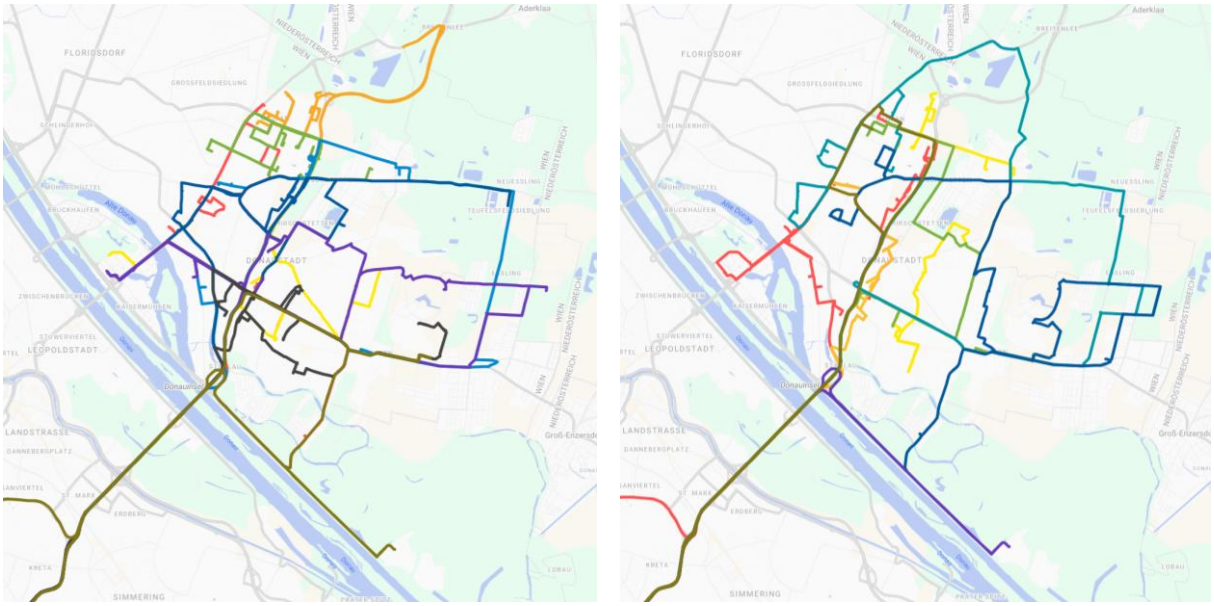

Figure 2: Initial route solutions of EDD (left) and NN (right) for September 22nd, 2025

However, regarding computational efficiency, EDD requires considerably less time for construction. Across the 33 tested instances, EDD outperformed NN in construction speed in 32 cases and was closely tied in one. On average, the EDD heuristic generated an initial solution 63.4% faster than the NN approach. This efficiency stems from the reduced algorithmic complexity of the EDD heuristic. Unlike NN, which must calculate the distance between the last visited customer and every remaining unrouted customer to find the nearest neighbor, EDD primarily relies on a pre-sorted sequence based on time windows and follows a more direct insertion logic, which significantly reduces the number of distance matrix lookups required during the construction phase. As a result, the EDD initial routes, being further from the optimum, possessed higher potential for improvement, enabling the algorithm to identify and

execute easy gains more rapidly. It hence allowed the LNS algorithm to perform a higher number of iterations within the given time limit compared to the NN configuration.

Regarding the quality of the final optimized solutions, both the EDD and NN configurations achieved good results following the LNS phase. In terms of the best individual runs, the EDD configuration produced lower total distances in 19 out of 33 instances. Specifically, EDD achieved a mean best distance of 355.88, compared to 359.11 for NN. When considering the average outcomes across five runs per instance, EDD maintained a slight advantage with an average distance of 362.81, compared to NN's 363.35. These results suggest that within this experimental framework, EDD slightly outperformed NN. This may be attributed to the larger margin for improvement provided by EDD's initial solutions. However, these findings are not conclusive evidence that initial solution quality is a primary determinant of final performance. The ultimate outcome is heavily influenced by other parameters within the LNS configuration, such as the destroy size and the specific operator selection logic, which may mitigate the impact of the starting heuristic.

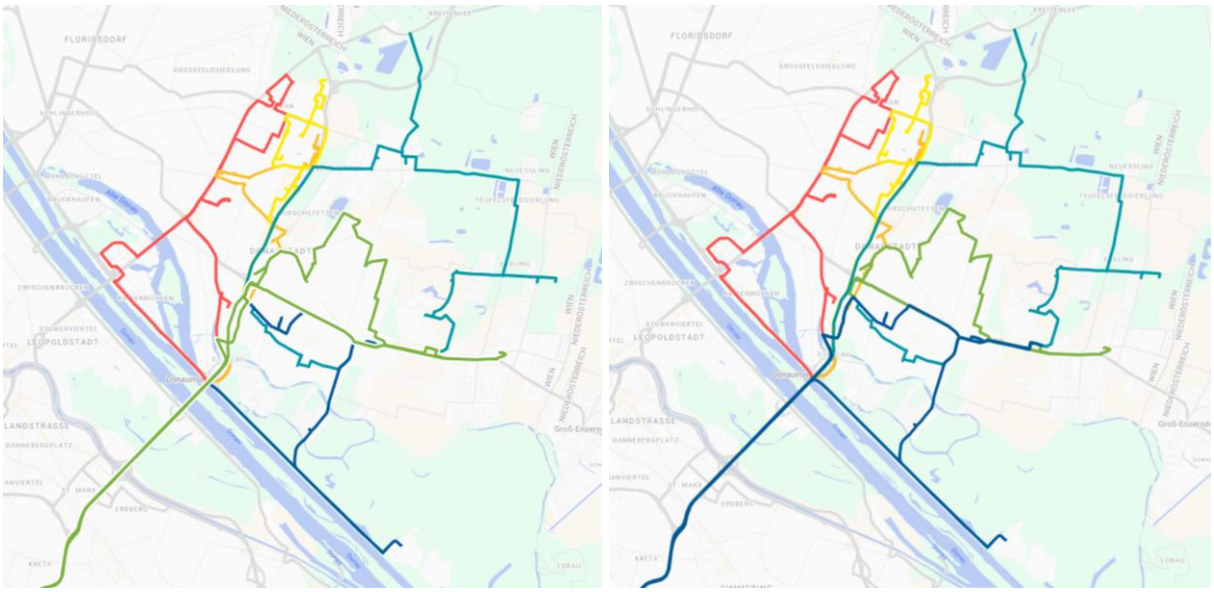

Figure 3: Final route solutions of EDD (left) and NN (right) for September 22nd, 2025

Both EDD and NN's final solutions are visualized in Figure 3**.** It can be seen that both solutions are nearly identical. This suggests that while NN provides a substantially superior starting point, the subsequent LNS optimization procedure is sufficiently robust to compensate for variations in initial solution quality.

4.3.2 Experiment Series 2

The second series of experiments evaluate the impact of deactivating individual destroy and repair operators from the available pools. In each configuration, one operator was removed while all others remained active. The aggregated results across all 33 tested instances are presented in Table 3.

Metric (Average across all instances)	Baseline	No RaR	No RoR	No WR	No GR	No RR
Final Distance (Best Run)	359.11	357.86	354.57	357.64	358.23	372.84
Final Distance (Average)	363.35	363.99	362.33	363.54	362.31	378.76
Iterations run	16592	14014	22045	16541	13516	18313

Table 3: Aggregated results for Experiment Series 2: deactivation of individual operators

The results indicate that deactivating individual operators often yielded better best run solutions than the baseline (with the exception of RR). At the same time, mean performance across five runs remains largely comparable to the baseline, again with RR as the only notable exception. This pattern suggests that under uniform operator selection, certain operators may not always contribute positively to search performance and may instead reduce overall efficiency through non-optimal usage of the operator pool. Since all operators are selected with equal probability, stronger operators are not utilized more frequently, limiting their potential contribution to improving solution quality. In addition, potential interference effects between operators are not accounted for under uniform selection, which may further reduce overall efficiency.

Deactivating RaR reduced total iteration count by approximately 15% relative to baseline, suggesting it is the least computationally expensive destroy operator. This is consistent with its

design: RaR applies simpler selection logic than WR and removes fewer customers per call than RoR.

Deactivating RoR increased total iteration count by approximately 33%, identifying it as the most computationally expensive destroy operator. This is attributable to its destruction scale: Rather than removing between 1 and $\lfloor k \cdot n_{total} \rfloor$ customers, RoR removes entire routes until the removal threshold is reached, generating solutions that demand more effort from the repair operators. Notably, this configuration produced the best overall solution quality across all experiments (354.57 best run, 362.33 average). A possible explanation could be that the substantial increase in iteration counts allowed for a more exhaustive exploration of the solution space that outweighed the benefits of RoR's large, disruptive perturbation that is hard to repair.

Deactivating WR had negligible impact on iteration count, suggesting its computational cost is close to the average of the operator pool.

Turning to the repair operators, deactivating GR reduced the total iteration count by approximately 18.5%, suggesting that it is considerably less computationally expensive than RR, which is consistent with its simpler design: It inserts customers at the cheapest available position without evaluating the cost of foregone alternatives.

As for RR, deactivating this operator produced a clear deterioration in solution quality, with both best-run and average distances increasing substantially relative to baseline (359.11 to 372.84 and 363.35 to 378.76 respectively). This is the largest negative impact observed across all deactivation experiments, identifying RR as the most critical individual operator in the entire pool. The approximately 10% increase in iteration count when RR is removed further confirms that it is more computationally demanding than GR.

Overall, the findings suggest that uniform operator selection introduces some inefficiencies. The consistent improvements observed in the best runs when selectively deactivating operators

indicate that the full operator set is not optimally utilized. Uniform selection lacks the ability to prioritize high-performing operators or suppress underperforming ones, increasing the risk of stagnation in local optima. It is also important to note that uniform selection does not account for interaction effects between destroy and repair operators. Certain combinations may partially negate each other's intended impact. For example, when WR is followed by GR, previously removed customers are likely to be reinserted into nearly identical positions, effectively undoing the perturbation. This limitation could be addressed in ALNS, where operator weights are adjusted dynamically based on observed performance. Incorporating such an adaptive mechanism would likely improve search efficiency and solution quality.

4.3.3. Experiment Series 3

The third series of experiments evaluate the impact of various destroy sizes k on the quality of the final solutions. In each configuration, k was set as 0.1, 0.3, 0.5, and 0.7, respectively. The aggregated results across all tested instances are presented in Table 4.

Metric (Average across all instances)	k = 0.1	k = 0.3	k = 0.5	k = 0.7
Final Distance (Best Run)	362.76	359.11	356.49	357.73
Final Distance (Average)	367.97	363.35	362.67	363.67
Iterations run	57295	16592	8545	4075

Table 4: Aggregated results for Experiment Series 3: varying destroy sizes

It can be seen that the number of iterations decreases significantly as k increases, which is likely due to the additional computational effort required by the repair operators when more customers are removed from the tours in each iteration. Across all four configurations, $k = 0.5$ achieved the best results on average. A small removal size such as $k = 0.1$ does not allow sufficient diversification despite having many iterations. Conversely, a high removal size such as $k = 0.7$ destroys too much of the existing solution structure, potentially discarding high-quality route segments, and leaves the repair operators with an extensive reconstruction task within the

limited time budget which results in substantially fewer iterations compared to other configurations.

4.3.4. Experiment Series 4

The fourth series of experiments evaluate the impact of having various run time on the program. In each configuration, the max. run time was set as 30 seconds, 60 seconds, 180 seconds, 300 seconds, and 600 seconds, respectively. The aggregated results across all test instances are presented in Table 5.

Metric (Average across all instances)	30s	60s	180s	300s	600s
Final Distance (Best Run)	359.61	358.39	359.11	356.39	357.86
Final Distance (Average)	366.56	364.47	363.35	362.35	362.38
Iterations run	2900	5754	16592	29038	53408

Table 5: Aggregated results for Experiment Series 4: varying run time

In general, it can be observed that the number of iterations increases with longer runtime. Nevertheless, an interesting observation can be made: the 300s runtime achieved better results on average compared to the 600s setting. While previous configurations showed improving solution quality alongside longer runtimes, the 600s runtime yielded slightly worse best-run solutions on average, while the mean across 5 runs remained at roughly the same level as with 300s. This near-identical average distance between the 300s and 600s configurations suggests that the results largely converge within 300s, with additional runtime yielding diminishing returns. The slightly better best-run result at 300s is likely a statistical artifact of the unfixed random seed rather than a meaningful quality difference. Upon closer inspection of the experiment output, there were two instances where the 300s configuration achieved significantly lower total distances than the 600s configuration: on the 7th of October (289.46 vs. 325.54) and on the 14th of October (330.94 vs. 357.81), which explains the lower average value in the best-run metric for the 300s configuration.

4.3.5. Final Experiment

After considering the best-performing configurations from the previous experiments, a final experiment is conducted using the following settings: a time limit of 300 seconds, a destroy size of 0.5, EDD as the construction heuristic, and the use of all available operators in both the destroy and repair phases. The results indicate that this configuration outperforms all previous experiments on average, achieving an average best solution value of 354.18 and an average mean solution value of 360.98 across five runs for all instances.

5. Conclusion

The VRPTW, as a key extension to the classical VRP, represents one of the most relevant optimization challenges in modern logistics, as the inclusion of time windows introduces additional operational constraints and complexities. This thesis aims to solve this problem by implementing a basic LNS framework that utilizes real-world input data to generate feasible routing solutions under predefined conditions.

The proposed framework incorporates two construction heuristics, three destroy operators, and two repair operators. The experimental results indicate that algorithm performance is highly sensitive to parameter configurations, with certain settings significantly improving solution quality while others lead to deterioration. In addition, the findings highlight a key limitation of the current approach, namely the use of a uniform operator selection mechanism, which restricts the algorithm's adaptive capabilities.

Section 5.1 discusses the managerial implications derived from the results, while Section 5.2 outlines the limitations of the proposed approach and identifies directions for future research.

5.1 Managerial Implications

The experimental results demonstrate that the LNS framework is capable of generating high-quality solutions within a limited computational time (e.g., 5 minutes). However, the practical

applicability of the approach critically depends on the quality and preparation of the input data. In particular, a well-structured and reliable dataset is essential to ensure both computational efficiency and the usability of the resulting solutions.

The initial dataset obtained from the company's dispatching software exhibits several deficiencies that must be addressed prior to optimization. First, the data contains multiple duplicate customer entries, primarily due to inconsistent manual input of customer names by different employees. This indicates a need for comprehensive master data cleansing and consolidation. Without such preprocessing, the optimization model is unable to correctly identify identical customers and instead treats them as distinct entities, which could potentially lead to multiple visits (and services) at the same customer location.

Second, inaccuracies in customer coordinates were observed in some cases, with the customers being located in districts other than the selected district (Donaustadt). Such errors can severely compromise solution quality, as the optimization algorithm relies on geographical proximity within service regions. These outliers would lead to inaccuracies in route planning and lead to substantial increases in total travel distances.

Third, ensuring solution feasibility requires consideration of both shipment weight and volume constraints. Currently, volume data is missing for a substantial proportion of shipments. As a result, drivers must attempt to load vehicles first and may need to return to dispatchers if capacity constraints are violated. This inefficiency highlights the necessity of integrating complete and accurate volume data into the system.

Overall, these data quality issues must be systematically addressed to enable the successful large-scale implementation of the optimization framework in practice.

Regarding the evaluation of the solutions generated by the implemented LNS algorithm, a direct comparison with actual traveled distances for the tested instances is not feasible. Although

historical data for the trucks on the corresponding days is available, it also includes distances associated with pick-up orders. Furthermore, in practice, trucks assigned to Donaustadt frequently perform deliveries in neighboring districts, as dispatchers may assign additional orders dynamically while the vehicles are already en route. Therefore, a direct comparison would not provide a fair assessment of potential savings. Nevertheless, the solutions generated by the LNS framework can serve as a valuable decision-support tool for dispatchers, enabling quick manual adjustments to routes on a day-to-day basis.

With respect to the current assignment of trucks to Donaustadt, the optimization results suggest that the existing fleet configuration is largely appropriate. Under the best parameter setting (as presented in Section 4.3.5), an average of 6.58 routes is generated per day, with approximately 4.21 trucks required on average. At present, four trucks with a capacity of 6,000 kg are consistently deployed in Donaustadt, complemented by additional trucks that are assigned as needed.

Given that the analyzed data (September 1st to October 15th) corresponds to the peak season according to Gebrüder Weiss, this operational approach appears reasonable. However, depending on the daily volume of pick-up orders, it may also be beneficial to consider a moderate expansion of the fixed fleet in order to maintain a greater degree of flexibility and responsiveness to demand fluctuations.

5.2 Limitation and Future Research

Despite the implemented LNS framework's quick computational time to get a solution, the solutions generated via this approach are still subject to numerous limitations and require further adjustments.

First, the absence of certain critical input data may significantly affect solution feasibility. In particular, incorporating shipment volume data would enhance the assessment of loading

constraints, as weight-based feasibility alone is insufficient to ensure that shipments physically fit within a vehicle.

Second, the experimental design relies on simplifying assumptions that may limit real-world applicability. Shipments are treated as homogeneous, without differentiation (e.g., between standard and ADR-classified goods), and are therefore assumed to be compatible with any available vehicle. In practice, however, the transport of dangerous goods is governed by ADR regulations, which introduce additional operational constraints that could influence vehicle assignment. Furthermore, the vehicle fleet is modeled as homogeneous. While this partially reflects the current situation in Donaustadt, where four trucks with a capacity of 6,000 kg are consistently deployed, other vehicle types with varying capacities, loading spaces, speeds, and route restrictions are used in other districts. These factors should be incorporated when extending the scope of the optimization model.

Third, the routing component of the optimization is formulated in a static manner and does not account for dynamic operational conditions. Fixed shifts start times (07:00 and 12:30) are assumed, although actual departure times may vary due to actual loading processes (e.g., missing or reassigned shipments could delay the start time). Similarly, service times are approximated as a constant 10 minutes, despite being highly variable in practice due to factors such as customer availability, acceptance issues, or spatial proximity between delivery locations. Moreover, the model assumes complete information availability ex ante, whereas new orders may arrive throughout the day and require real-time assignment by dispatchers in reality. Additional real-world factors, such as traffic congestion and driver familiarity with routes and customer locations, are also not captured as well.

Lastly, the optimization model focuses exclusively on minimizing total route distance, using data from all delivered shipments on given days as input. While this objective is relevant for estimating potential reductions in fuel consumption and environmental impact, it does not fully

reflect the company's operational priorities. In practice, maximizing the number of successfully delivered shipments per day is of equal, if not greater, importance. In particular, contractual arrangements with subcontractors create an incentive to assign shipments to vehicles in a way that maximizes delivery throughput. Future research could therefore extend the model by incorporating an objective function that prioritizes the number of completed deliveries, potentially with differentiated weights based on shipment characteristics (e.g., higher priority for premium shipments with delivery time windows).

Furthermore, as observed in Experiment 2, the use of a uniform operator selection mechanism limits the performance of the current approach. Future research could address this limitation by extending the implemented LNS framework to an ALNS. Such an approach would dynamically adjust operator selection probabilities based on their historical contribution to solution improvement, thereby prioritizing more effective operators over time. In addition, exploring interdependencies between operators may provide further performance gains. Finally, the implementation of additional operators (particularly destroy operators) could also enhance the diversification capabilities of the search process and improve overall solution quality.

All codes created within the scope of this thesis can be found in the following GitHub repository. The operational data used in this thesis were provided by Gebrüder Weiss and are not publicly available. Anonymized data may be made available upon request and with permission from Gebrüder Weiss.

References

Ali, O., Côté, J.-F., & Coelho, L. C. (2021). Models and algorithms for the delivery and installation routing problem. *European Journal of Operational Research*, *291*(1), 162–177. https://doi.org/10.1016/j.ejor.2020.09.011

Aydınalp, Z., & Özgen, D. (2023). Solving vehicle routing problem with time windows using metaheuristic approaches. *International Journal of Intelligent Computing and Cybernetics*, *16*(1), 121–138. https://doi.org/10.1108/IJICC-01-2022-0021

Azmi, S. F., & Hamontree, C. (2024). *Implementation of Genetic Algorithm in Real-World Capacitated Vehicle Routing Problem with Time Windows*. SSRN. https://doi.org/10.2139/ssrn.4876984

Baker, B. M., & Ayechew, M. A. (2003). A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, *30*(5), 787–800. https://doi.org/10.1016/S0305-0548(02)00051-5

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, *35*(3), 268–308. https://doi.org/10.1145/937503.937505

Braekers, K., Ramaekers, K., & Van Nieuwenhuyse, I. (2016). The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, *99*, 300–313. https://doi.org/10.1016/j.cie.2015.12.007

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1999). An improved Ant System algorithm for theVehicle Routing Problem. *Annals of Operations Research*, *89*(0), 319–328. https://doi.org/10.1023/A:1018940026670

Clarke, G., & Wright, J. W. (1964). Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, *12*(4), 568–581. https://doi.org/10.1287/opre.12.4.568

Costa, L., Contardo, C., & Desaulniers, G. (2019). Exact Branch-Price-and-Cut Algorithms for Vehicle Routing. *Transportation Science*, *53*(4), 946–985. https://doi.org/10.1287/trsc.2018.0878

Cruz-Chávez, M. A., Rodríguez-León, A., Rivera-López, R., & Cruz-Rosales, M. H. (2019). A Grid-Based Genetic Approach to Solving the Vehicle Routing Problem with Time Windows. *Applied Sciences*, *9*(18), 3656. https://doi.org/10.3390/app9183656

Dantzig, G. B., & Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science*, *6*(1), 80–91. https://doi.org/10.1287/mnsc.6.1.80

Desaulniers, G., Madsen, O. B. G., & Ropke, S. (2014). Chapter 5: The Vehicle Routing Problem with Time Windows. In P. Toth & D. Vigo (Eds.), *Vehicle Routing* (pp. 119–159). Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611973594.ch5

Desrochers, M., Desrosiers, J., & Solomon, M. (1992). A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. *Operations Research*, *40*(2), 342–354. https://doi.org/10.1287/opre.40.2.342

Feo, T. A., & Resende, M. G. C. (1995). Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, *6*(2), 109–133. https://doi.org/10.1007/BF01096763

Fisher, M. L., Jörnsten, K. O., & Madsen, O. B. G. (1997). Vehicle Routing with Time Windows: Two Optimization Algorithms. *Operations Research*, *45*(3), 488–492. https://doi.org/10.1287/opre.45.3.488

Gillett, B. E., & Miller, L. R. (1974). A Heuristic Algorithm for the Vehicle-Dispatch Problem. *Operations Research*, *22*(2), 340–349. https://doi.org/10.1287/opre.22.2.340

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*(5), 533–549. https://doi.org/10.1016/0305-0548(86)90048-1

Gmira, M., Gendreau, M., Lodi, A., & Potvin, J.-Y. (2021). Tabu search for the time-dependent vehicle routing problem with time windows on a road network. *European Journal of Operational Research*, *288*(1), 129–140. https://doi.org/10.1016/j.ejor.2020.05.041

Gocken, T., & Yaktubay, M. (2019). Comparison of Different Clustering Algorithms via Genetic Algorithm for VRPTW. *International Journal of Simulation Modelling*, *18*(4), 574–585. https://doi.org/10.2507/IJSIMM18(4)485

He, M., Wei, Z., Wu, X., & Peng, Y. (2021). An Adaptive Variable Neighborhood Search Ant Colony Algorithm for Vehicle Routing Problem With Soft Time Windows. *IEEE Access*, *9*, 21258–21266. https://doi.org/10.1109/ACCESS.2021.3056067

Hosny, M. I. (2014). Bridging the Gap Between Theory and Practice in the Vehicle Routing Research. *International Journal of Applied Mathematics, Electronics and Computers*, *2*(3), 15. https://doi.org/10.18100/ijamec.95178

Jiang, P., Men, J., Xu, H., Zheng, S., Kong, Y., & Zhang, L. (2020). A Variable Neighborhood Search-Based Hybrid Multiobjective Evolutionary Algorithm for HazMat Heterogeneous Vehicle Routing Problem With Time Windows. *IEEE Systems Journal*, *14*(3), 4344–4355. https://doi.org/10.1109/JSYST.2020.2966788

Johar, F., Potts, C., & Bennell, J. (2015). Vehicle routing problem with time constraints. *Malaysian Journal of Fundamental and Applied Sciences*, *11*(4). https://doi.org/10.11113/mjfas.v11n4.400

Kallehauge, B., Larsen, J., Madsen, O. B. G., & Solomon, M. M. (2005). Vehicle Routing Problem with Time Windows. In G. Desaulniers, J. Desrosiers, & M. M. Solomon (Eds.), *Column Generation* (pp. 67–98). Springer-Verlag. https://doi.org/10.1007/0-387-25486-2_3

Ketaren, D. R., Gultom, P., & Nasution, M. K. M. (2025). *Multi-objective vehicle routing problem with time windows via genetic algorithm*.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, *220*(4598), 671–680. https://doi.org/10.1126/science.220.4598.671

Koch, T., Berthold, T., Pedersen, J., & Vanaret, C. (2022). Progress in mathematical programming solvers from 2001 to 2020. *EURO Journal on Computational Optimization*, *10*, 100031. https://doi.org/10.1016/j.ejco.2022.100031

Konstantakopoulos, G. D., Gayialis, S. P., & Kechagias, E. P. (2022). Vehicle routing problem and related algorithms for logistics distribution: A literature review and classification. *Operational Research*, *22*(3), 2033–2062. https://doi.org/10.1007/s12351-020-00600-7

Konstantakopoulos, G. D., Gayialis, S. P., Kechagias, E. P., Papadopoulos, G. A., & Tatsiopoulos, I. P. (2020). A Multiobjective Large Neighborhood Search Metaheuristic for the Vehicle Routing Problem with Time Windows. *Algorithms*, *13*(10), 243. https://doi.org/10.3390/a13100243

Kosolsombat, S., & Ratanavilisagul, C. (2022). Modified ant colony optimization with selecting and elimination customer and re-initialization for VRPTW. *Bulletin of Electrical Engineering and Informatics*, *11*(6), 3471–3482. https://doi.org/10.11591/eei.v11i6.3943

Laporte, G., & Nobert, Y. (1987). Exact Algorithms for the Vehicle Routing Problem. In *North-Holland Mathematics Studies* (Vol. 132, pp. 147–184). Elsevier. https://doi.org/10.1016/S0304-0208(08)73235-3

Nemhauser, G., & Wolsey, L. (1988). *Integer and Combinatorial Optimization* (1st ed.). Wiley. https://doi.org/10.1002/9781118627372

Nguyen, T. K., Nguyen, N. H., & Luong, N. H. (2026). A memetic ant colony optimization algorithm for the large-scale pickup and delivery problem with time windows. *Swarm and Evolutionary Computation*, *101*, 102295. https://doi.org/10.1016/j.swevo.2026.102295

Pan, B., Zhang, Z., & Lim, A. (2021). Multi-trip time-dependent vehicle routing problem with time windows. *European Journal of Operational Research*, *291*(1), 218–231. https://doi.org/10.1016/j.ejor.2020.09.022

Pureza, V., Morabito, R., & Reimann, M. (2012). Vehicle routing with multiple deliverymen: Modeling and heuristic approaches for the VRPTW. *European Journal of Operational Research*, *218*(3), 636–647. https://doi.org/10.1016/j.ejor.2011.12.005

Ropke, S., & Pisinger, D. (2006). An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, *40*(4), 455–472. https://doi.org/10.1287/trsc.1050.0135

Shaw, P. (1998). Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In M. Maher & J.-F. Puget (Eds.), *Principles and Practice of Constraint Programming—CP98* (Vol. 1520, pp. 417–431). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-49481-2_30

Solomon, M. M. (1987). Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, *35*(2), 254–265.

Sun, B., Yang, Y., Shi, J., & Zheng, L. (2019). Dynamic Pick-Up and Delivery Optimization With Multiple Dynamic Events in Real-World Environment. *IEEE Access*, *7*, 146209–146220. https://doi.org/10.1109/ACCESS.2019.2944739

Syafrizal, W., & Sugiharti, E. (2023). Electric Vehicle Routing Problem with Fuzzy Time Windows using Genetic Algorithm and Tabu Search. *Journal of Advances in Information Systems and Technology*, *4*(2), 205–221. https://doi.org/10.15294/jaist.v4i2.62314

Toth, P., & Vigo, D. (Eds.). (2014). *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611973594

Wang, X., Altundas, B., Li, Z., Zhao, A., & Gombolay, M. (2025). *Improvement of Optimization using Learning Based Models in Mixed Integer Linear Programming Tasks* (Version 1). arXiv. https://doi.org/10.48550/ARXIV.2506.06291

Windras Mara, S. T., Rifai, A. P., & Sopha, B. M. (2022). An adaptive large neighborhood search heuristic for the flying sidekick traveling salesman problem with multiple drops. *Expert Systems with Applications*, *205*, 117647. https://doi.org/10.1016/j.eswa.2022.117647

Wu, L., He, Z., Chen, Y., Wu, D., & Cui, J. (2019). Brainstorming-Based Ant Colony Optimization for Vehicle Routing With Soft Time Windows. *IEEE Access*, *7*, 19643–19652. https://doi.org/10.1109/ACCESS.2019.2894681

Xu, K., Cao, Z., Zheng, C., & Liu, L. (2026). Learning to search for vehicle routing with multiple time windows. *Computers & Industrial Engineering*, *213*, 111760. https://doi.org/10.1016/j.cie.2025.111760

Zhang, Z., Dou, Y., Cheng, W., Xu, X., Jiang, J., & Tan, Y. (2022). A Tabu Search Algorithm Based on Density Peak Clustering to Solve VRPTW. *2022 8th International Conference on Big Data and Information Analytics (BigDIA)*, 472–478. https://doi.org/10.1109/BigDIA56350.2022.9874007

Appendix A: Results of Experiment Series 1 (Construction Heuristics)

		EDD Heuristic					NN Heuristic				
Instance	n	Const.	Const. Time	Final Best	Final Avg.	Iter.	Const.	Const. Time	Final Best	Final Avg.	Iter.
09-01	88	631.7	0.0023	369.89	374.49	13113	464.16	0.0177	370.62	374.99	12077
09-02	75	683.62	0.001	330.78	335.3	22717	440.18	0.0013	332.21	335.37	20875
09-03	90	869.22	0.0015	367.15	372.23	12306	407.33	0.0036	371.28	373.78	11210
09-04	86	768.59	0.0014	319.08	321.45	13386	406.48	0.0024	313.96	318.06	12190
09-05	61	442.25	0.0006	265.59	268.02	39575	305.67	0.0023	267.43	267.84	36116
09-08	83	621.39	0.0017	381.76	387.78	16646	498.49	0.0029	382.67	385.7	14051
09-09	94	658.8	0.0015	387.77	390.11	12784	463.47	0.0037	384.31	388.56	10943
09-10	82	599.19	0.0016	338.5	341.97	16253	407.39	0.0047	337.3	339.95	13962
09-11	93	627.36	0.0022	369.05	372.48	10575	428.14	0.0108	370.83	372.85	10452
09-12	78	494.12	0.0011	324.99	328.86	19514	408.09	0.0017	323.54	325.24	18245
09-15	84	599.38	0.0013	369.64	375.51	16696	456.1	0.0036	370.87	373.02	15587
09-16	76	526.45	0.0011	321.3	336.7	23437	414.19	0.0042	319.67	335.76	22909
09-17	92	656.63	0.0015	371.47	376.66	13191	429.05	0.0028	376.6	381.29	13635
09-18	84	554.51	0.0014	320.78	322.88	13421	418.43	0.0027	320.36	323.67	14156
09-19	82	581.86	0.0027	332.66	333.79	15943	399.62	0.0027	329.09	331.54	15119
09-22	93	626.43	0.0025	378.36	382.59	12865	510.69	0.0029	379.72	382.15	12517
09-23	90	559.61	0.0015	367.02	368.83	14718	421.82	0.003	367.87	369.77	14860
09-24	85	646.03	0.0013	370.64	377.17	15252	425.93	0.0032	382.71	384.8	14683
09-25	96	617.48	0.002	383.63	387.69	9419	424.63	0.0033	380.65	383.3	9386
09-26	70	556.55	0.0008	333.62	337.35	28198	408.43	0.0015	333.71	336.42	27773
09-29	99	724.28	0.0018	436.23	439.27	7505	533.76	0.0051	436.91	438.46	9151
09-30	106	718.34	0.0023	426.08	428.39	6780	466.78	0.0036	425.76	428.13	7124
10-01	97	697.76	0.0029	392.01	415.38	12650	492.16	0.0047	421.67	426.24	12256
10-02	97	639.82	0.002	382.54	389.29	8889	452.83	0.0041	380.78	383.4	8887
10-03	74	615.8	0.0009	366.81	369.1	35983	438.39	0.005	366.57	369.21	33826
10-06	76	565.35	0.001	327.34	331.52	27728	432.98	0.0034	326.69	331.42	24303
10-07	70	531.38	0.0009	289.72	320.82	28390	352.54	0.0037	327.41	329.72	23793
10-08	76	562.36	0.0011	315.13	316.64	18723	405.82	0.0053	316.39	317.37	16297
10-09	89	633.35	0.0015	322.53	333.3	11399	398.81	0.0047	323.31	339.13	10434
10-10	74	579.5	0.0008	378.85	386.61	40697	478.05	0.0033	377.3	382.32	35180
10-13	87	697.84	0.0013	432.05	434.3	16992	507.13	0.0051	427.6	430.72	14382
10-14	88	618.57	0.0015	327.67	346.32	12209	427.43	0.0029	332.03	355.36	10742
10-15	77	567.82	0.0011	343.41	369.82	23628	413.95	0.0045	372.97	375.03	20418

Appendix B: Results of Experiment Series 2 (Destroy Operators)

				No Random Removal			No Route Removal			No Worst Removal		
Instance	n	Baseline (Best)	Baseline (Avg.)	Final (Best)	Final (Avg.)	Iter.	Final (Best)	Final (Avg.)	Iter.	Final (Best)	Final (Avg.)	Iter.
09-01	88	370.62	374.99	365.19	373.56	6130	369.83	373.75	14865	370.67	372.93	12549
09-02	75	332.21	335.37	331.36	335.85	10381	332.68	334.82	28207	333.17	336.94	20184
09-03	90	371.28	373.78	369.96	374.09	5933	367.05	371.53	15988	370.94	375.62	11584
09-04	86	313.96	318.06	319.65	321.31	5062	319.88	321.14	17563	317.67	319.34	12677
09-05	61	267.43	267.84	264.71	267.44	33003	264.42	266.79	52988	264.96	269.83	35303
09-08	83	382.67	385.7	380.83	387.84	12994	375.67	381.39	21155	376.69	381.29	14696
09-09	94	384.31	388.56	377.9	386.57	10547	379.71	386.09	14997	383.06	386.12	11937
09-10	82	337.3	339.95	339.54	340.3	9982	334.98	339.52	20193	337.57	341.96	14586
09-11	93	370.83	372.85	371.54	374.12	7161	371.8	372.85	13551	370.73	374.63	9774
09-12	78	323.54	325.24	324.28	326.39	14499	324.31	324.59	23580	325.37	326.45	16757
09-15	84	370.87	373.02	370.86	374.32	13995	344.35	367.48	20304	370.61	375.69	14938
09-16	76	319.67	335.76	321.6	331.4	19815	320.51	341.42	29300	320.68	323.59	23184
09-17	92	376.6	381.29	378.06	381.08	11585	376.65	380.63	16159	377.41	385.59	13156
09-18	84	320.36	323.67	321.38	322.69	11689	320.33	322.2	18501	321.64	324.08	13330
09-19	82	329.09	331.54	330.01	334.22	12812	326.36	330.43	19723	329.54	334.81	14416
09-22	93	379.72	382.15	381.08	383.59	11339	377.76	381.72	15540	378.72	382.23	12203
09-23	90	367.87	369.77	367.43	371.09	13559	366.3	369.44	17820	367.34	368.2	14517
09-24	85	382.71	384.8	383.74	385.19	12226	345.99	377.53	18929	384.53	386.03	14093
09-25	96	380.65	383.3	380.47	384.29	7802	384.51	386.61	11899	379.88	383.13	8895
09-26	70	333.71	336.42	341.18	341.87	24033	332.15	338.85	37279	334.11	339.14	26735
09-29	99	436.91	438.46	431.7	436.64	7842	426.14	433.94	11729	432.1	436.68	9392
09-30	106	425.76	428.13	425.64	430.48	6121	423.19	427.72	9020	425.08	431.46	6872
10-01	97	421.67	426.24	394.08	422.52	11061	426.31	429.95	15024	423	428.81	11642
10-02	97	380.78	383.4	380.5	383.98	8025	381.95	384.91	12202	380.1	382.31	9179
10-03	74	366.57	369.21	364.62	368.54	31632	365.08	365.99	39593	363.55	367.35	34750
10-06	76	326.69	331.42	326.63	331.66	22324	332.72	333.4	32015	327.88	332.75	24737
10-07	70	327.41	329.72	329.45	333.94	21200	294.49	321.31	35599	321.29	328.86	23862
10-08	76	316.39	317.37	277.36	310.82	14683	278.7	309.11	25193	275.08	302.1	16273
10-09	89	323.31	339.13	324.35	341.37	9457	324.88	348.2	14552	323.28	346.24	10592
10-10	74	377.3	382.32	376.28	384.28	33168	377.38	381.25	40173	379.02	382.9	34347
10-13	87	427.6	430.72	430.09	433.91	13979	429.45	430.87	19880	429.55	432.43	15426
10-14	88	332.03	355.36	357	362.42	9872	331.75	346.74	15321	332.41	360.65	11247
10-15	77	372.97	375.03	370.86	373.99	18541	373.53	374.87	28641	374.55	376.71	22019

Appendix C: Results of Experiment Series 2 (Repair Operators)

				No Greedy Repair			No Regret Repair		
Instance	n	Baseline (Best)	Baseline (Avg.)	Final (Best)	Final (Avg.)	Iter.	Final (Best)	Final (Avg.)	Iter.
09-01	88	370.62	374.99	367.29	371.77	5231	380.06	386.19	12613
09-02	75	332.21	335.37	335.8	337.61	6387	348.97	351.95	20927
09-03	90	371.28	373.78	365.35	375.12	3768	380.83	385.81	12032
09-04	86	313.96	318.06	317.62	319.72	4522	326.09	330	13255
09-05	61	267.43	267.84	267.5	270.95	14049	270.12	272.81	41035
09-08	83	382.67	385.7	376.38	385.74	5373	393.62	398.08	18033
09-09	94	384.31	388.56	380.01	385.58	3843	400.71	429.23	13253
09-10	82	337.3	339.95	338.45	340.29	5015	346.95	359.08	16192
09-11	93	370.83	372.85	372.18	374.09	3385	380.06	388.08	11148
09-12	78	323.54	325.24	322.52	325.41	11412	328.5	331.62	18890
09-15	84	370.87	373.02	369.99	373.08	14614	381.21	387.15	16088
09-16	76	319.67	335.76	315.96	327.62	20267	356.73	361.79	23347
09-17	92	376.6	381.29	377.74	380.87	13236	398.98	402.45	14059
09-18	84	320.36	323.67	321.28	322.71	13367	328.24	334.37	14459
09-19	82	329.09	331.54	326.53	327.82	14804	333.27	334.34	15515
09-22	93	379.72	382.15	378.96	382.55	12330	393.01	404.72	13155
09-23	90	367.87	369.77	368.05	368.81	13876	377.49	384.24	16961
09-24	85	382.71	384.8	383.22	385.03	14235	391.07	394.55	15722
09-25	96	380.65	383.3	380.86	382.76	8908	394.44	398.53	10021
09-26	70	333.71	336.42	332.59	338.23	27666	346.15	350.88	28833
09-29	99	436.91	438.46	433.24	434.95	9291	446.61	450.04	10039
09-30	106	425.76	428.13	426.85	431.15	7067	439.54	444.15	7625
10-01	97	421.67	426.24	389.46	413.2	12398	442	445.49	13008
10-02	97	380.78	383.4	380.28	383.2	9337	389.26	396.46	9714
10-03	74	366.57	369.21	365.38	366.71	33489	374.71	379.62	37944
10-06	76	326.69	331.42	332.59	333.47	26811	339.54	350.13	28808
10-07	70	327.41	329.72	328.8	329.4	24254	333.26	336.55	27720
10-08	76	316.39	317.37	314.92	317.34	17383	329.83	331.32	19744
10-09	89	323.31	339.13	319.55	323.28	10547	369.2	373.79	12265
10-10	74	377.3	382.32	380.11	383.19	32918	389.23	394.79	38531
10-13	87	427.6	430.72	428.1	432.89	14728	438.34	441.19	16843
10-14	88	332.03	355.36	353.28	358.31	11104	375.39	386.99	13162
10-15	77	372.97	375.03	370.91	373.44	20413	380.38	382.72	23398

Appendix D: Results of Experiment Series 3 (Destroy Sizes)

		k = 0.1			k = 0.3			k = 0.5			k = 0.7		
Instance	n	Final (Best)	Final (Avg.)	Iter.	Final (Best)	Final (Avg.)	Iter.	Final (Best)	Final (Avg.)	Iter.	Final (Best)	Final (Avg.)	Iter.
09-01	88	367.66	373.59	45153	370.62	374.99	12077	366.3	374.2	5910	373.29	376.47	3790
09-02	75	337.7	340.07	69614	332.21	335.37	20875	332.29	335.52	10817	331.64	333.79	6447
09-03	90	374.63	376.96	41430	371.28	373.78	11210	377.42	379.13	6008	371.45	374.84	3475
09-04	86	324.01	325.61	42242	313.96	318.06	12190	319.12	321.1	6754	311.89	316.87	3937
09-05	61	267.66	270.04	97169	267.43	267.84	36116	265.98	267.11	20587	266.07	267.03	12332
09-08	83	379.88	383.49	52194	382.67	385.7	14051	376.67	379.55	7706	384.44	387.21	4521
09-09	94	388.76	411.69	44100	384.31	388.56	10943	381.11	386.39	5733	384.2	387.67	3266
09-10	82	340.43	342.86	49064	337.3	339.95	13962	337.11	339.37	7982	337.23	339.02	4553
09-11	93	374.32	376.15	38582	370.83	372.85	10452	368.6	371.26	4946	369.85	372.88	2889
09-12	78	324.7	327.59	58015	323.54	325.24	18245	323.35	325.49	9264	323.7	327.46	5386
09-15	84	371.96	374.3	50273	370.87	373.02	15587	368.35	374.85	7752	369.32	373.67	4208
09-16	76	350.1	353.64	63150	319.67	335.76	22909	320.58	324.19	11942	321.9	331.5	6509
09-17	92	388.64	393.12	49328	376.6	381.29	13635	372.45	378.44	6462	375.28	378.11	3694
09-18	84	322.29	327.91	41307	320.36	323.67	14156	319.76	322.56	6919	320.85	322.63	3920
09-19	82	330.32	332.41	46941	329.09	331.54	15119	326.8	329.2	7844	328.48	331.24	4478
09-22	93	378.2	384.01	47442	379.72	382.15	12517	380.76	384.19	6074	381.37	386.62	2127
09-23	90	366.85	372.02	56269	367.87	369.77	14860	365.71	369.22	7059	369.38	373.1	1160
09-24	85	385.28	387.51	53553	382.71	384.8	14683	380.84	384	7477	349.75	375.03	981
09-25	96	383.8	389.17	33937	380.65	383.3	9386	380.71	382.39	4673	385.44	388.21	779
09-26	70	340.64	344.01	85782	333.71	336.42	27773	330.7	337.24	14257	331.15	339.9	1871
09-29	99	439.06	440.73	37827	436.91	438.46	9151	435.27	436.86	4416	435.71	438.92	781
09-30	106	426.38	431.52	27267	425.76	428.13	7124	424.19	430.12	3535	430.09	433.76	658
10-01	97	426.86	429.78	50468	421.67	426.24	12256	425.34	428.39	5485	402.22	422.79	1792
10-02	97	382.95	385.47	34793	380.78	383.4	8887	381.65	383.02	4535	383.73	387.11	2198
10-03	74	365.66	370.88	128044	366.57	369.21	33826	365.22	367.55	15656	366.71	370.45	7415
10-06	76	331.63	334.46	93649	326.69	331.42	24303	331.88	333.25	13115	330.81	335.35	5475
10-07	70	333.46	335.21	64806	327.41	329.72	23793	293.66	320.83	12954	294.41	324.52	7067
10-08	76	318.31	322.77	54742	316.39	317.37	16297	279.27	308.33	9254	315.24	317	4551
10-09	89	323.81	349.54	38031	323.31	339.13	10434	325.76	346.75	5651	324.19	340.75	2627
10-10	74	386.62	389.67	127583	377.3	382.32	35180	376.64	382.79	17111	378.88	381.65	7644
10-13	87	430.6	433.19	57112	427.6	430.72	14382	427.27	431.71	7578	430.32	434.44	4127
10-14	88	332.95	356.78	40448	332.03	355.36	10742	332.12	358.92	6041	354.99	358.88	3500
10-15	77	375.12	376.95	70406	372.97	375.03	20418	371.34	374.11	10500	370.96	372.34	6302

Appendix E: Results of Experiment Series 4 (Run Time)

		30s			60s			180s			300s			600s		
Instance	n	Final (best)	Final (Avg.)	Iter.	Final (best)	Final (Avg.)	Iter.	Final (best)	Final (Avg.)	Iter.	Final (best)	Final (Avg.)	Iter.	Final (best)	Final (Avg.)	Iter.
09-01	88	370.12	375.71	2118	370.54	374.15	4257	370.62	374.99	12077	366.48	372.74	20840	365.77	369.24	40408
09-02	75	335.14	338.92	3734	333.1	337	7144	332.21	335.37	20875	331.87	334.82	36803	333.48	337.06	33514
09-03	90	368.24	377.69	1959	373.15	375.07	4163	371.28	373.78	11210	366.55	370.95	20260	368.53	372.45	41711
09-04	86	321.92	324.45	2208	320.45	321.83	4415	313.96	318.06	12190	319.62	320.29	23283	318.63	321.13	47827
09-05	61	268.09	268.56	6406	267.5	268.34	12941	267.43	267.84	36116	267.52	268.13	66319	265.98	267.4	130583
09-08	83	380.06	387.01	2652	377.86	384.33	5131	382.67	385.7	14051	378.12	380.87	26261	383.64	384.21	51935
09-09	94	384.26	389.33	1969	380.85	386.3	4124	384.31	388.56	10943	380.23	386.47	20318	381.02	384.44	41432
09-10	82	339.94	341.44	2491	338.4	339.81	5236	337.3	339.95	13962	334.64	337.98	25913	336.51	338.93	51986
09-11	93	372.99	375.28	1679	370.5	373.56	3375	370.83	372.85	10452	368.85	372.34	17361	370.52	372.16	34283
09-12	78	324.54	328.06	2975	324.38	329.16	5973	323.54	325.24	18245	323.41	325.93	30189	322.13	324.3	60182
09-15	84	373.39	377.91	2626	348.35	369.42	5194	370.87	373.02	15587	370.08	372.59	26950	371.85	373.61	52598
09-16	76	318.72	340.72	3693	319.39	335.81	7596	319.67	335.76	22909	318.29	328.92	39499	320.01	330.43	78519
09-17	92	379.66	385.46	2156	378.33	382.28	4418	376.6	381.29	13635	377.07	379	22123	373.41	379.18	44686
09-18	84	320.77	326.09	2202	323.04	326.6	4665	320.36	323.67	14156	319.78	322.05	23859	321.41	322.21	46668
09-19	82	330.57	335.22	2551	326.8	331.88	5060	329.09	331.54	15119	326.7	328.83	26128	328.5	334.16	47468
09-22	93	381.61	384.14	2135	376.19	381.13	4190	379.72	382.15	12517	378.05	381.97	21263	377.5	380.63	39670
09-23	90	370.25	371.7	2493	368.53	370.12	5039	367.87	369.77	14860	366.21	367.28	24979	366.27	369.79	27308
09-24	85	385.31	386.07	2490	375.9	383.73	4956	382.71	384.8	14683	378.31	383.8	24982	372.42	380.05	33801
09-25	96	381.64	386.32	1591	380.69	382.67	3103	380.65	383.3	9386	379.71	382.81	15772	381.4	381.92	28183
09-26	70	340.48	343.95	4913	334.03	341.34	9415	333.71	336.42	27773	334.88	340.26	46755	333.76	339.13	88032
09-29	99	438.58	440	1557	435.86	439.69	3127	436.91	438.46	9151	434.32	437.78	15512	421.55	433.84	28651
09-30	106	427.07	430.65	1188	427.37	430.65	2384	425.76	428.13	7124	424.86	428.83	11961	425.72	426.31	22065
10-01	97	424.71	428.19	2002	423.31	427.05	3837	421.67	426.24	12256	396.22	420.01	20699	392.05	419.83	33020
10-02	97	382.75	387.64	1608	382.02	385.12	3148	380.78	383.4	8887	382.58	384.94	9490	382.06	384.51	28857
10-03	74	365.61	369.64	6104	363.07	368.65	11930	366.57	369.21	33826	366.73	369.16	56791	365.7	366.8	105895
10-06	76	331.47	333.96	4733	330.87	332.52	8625	326.69	331.42	24303	329.45	332.05	49087	327.69	334.7	79713
10-07	70	294.41	323.79	4297	328.06	332.85	8466	327.41	329.72	23793	289.46	314.35	44424	325.54	329.38	80054
10-08	76	314.82	319.95	3163	315.06	319.32	6372	316.39	317.37	16297	314.5	319.89	30227	318.64	320.84	58538
10-09	89	325.15	350.11	1907	323.75	348.44	3705	323.31	339.13	10434	324.08	345.85	18925	320.73	324.6	33871
10-10	74	378.46	391.05	5713	378.05	385.29	11757	377.3	382.32	35180	379.47	385.15	63112	377.32	385.32	119030
10-13	87	431.12	434.54	2720	430.25	432.92	5107	427.6	430.72	14382	431.01	433.01	26395	430.29	431.88	50406
10-14	88	330.89	367.17	2039	328.89	354.9	3833	332.03	355.36	10742	330.94	355.28	17906	357.81	364.4	35642
10-15	77	374.34	375.8	3632	372.28	375.7	7211	372.97	375.03	20418	370.86	373.13	33858	371.52	373.57	65934

Appendix F: Results of the Final Experiment (Best Configuration)

Instance	n	Final (best)	Final (Avg.)	Iter.
09-01	88	335.45	360.96	9363
09-02	75	330.62	335.46	16563
09-03	90	365.46	369.44	8397
09-04	86	315.67	319.87	9901
09-05	61	265.88	267.07	29664
09-08	83	380.55	382.43	11907
09-09	94	376.2	383.05	8365
09-10	82	335.14	338.9	12150
09-11	93	367.84	370.03	7216
09-12	78	321.44	323.99	14503
09-15	84	372.11	373.52	11692
09-16	76	320.1	329.38	17254
09-17	92	373.7	378.61	9306
09-18	84	319.64	320.88	10295
09-19	82	328.23	329.8	11846
09-22	93	378.16	381.66	8905
09-23	90	368.45	370.6	10686
09-24	85	376.08	380.93	10790
09-25	96	381.2	385.39	6690
09-26	70	331.4	334.63	21348
09-29	99	400.07	432.26	6718
09-30	106	425.75	428.03	5047
10-01	97	395.61	411.39	8687
10-02	97	386.73	390.86	6573
10-03	74	366.36	370.12	22459
10-06	76	328.02	329.75	18792
10-07	70	287.27	312.23	20799
10-08	76	316.62	319	13784
10-09	89	321.23	339.48	8120
10-10	74	376.87	379.95	27001
10-13	87	434.16	437.27	11682
10-14	88	332.94	350.84	8788
10-15	77	372.94	374.57	15379