universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

A capacitated and temperature-dependent pickup-and-delivery problem for heavy-duty vehicles

verfasst von | submitted by

Yuan Chen BSc BSc

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

Master of Science (MSc)

Wien | Vienna, 2024

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:	Univ.-Prof. Mag. Dr. Karl Franz Dörner Privatdoz.

Acknowledgements

The financial support by the Austrian Federal Ministry of Labour and Economy, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

I would like to express my sincere gratitude to the following people and institutions who have contributed significantly to the successful completion of my Master's thesis:

I would like to sincerely thank Univ.-Prof. Mag. Dr. Karl Franz Dörner, Privatdoz from the University of Vienna, for his professional supervision and support of my Master's thesis. His expertise, patience and words of encouragement helped me navigate through the challenges of my research and realise my full potential.

Special thanks also go to Mag. Biljana Roljić from the University of Vienna, for her tireless support during my Master's thesis. Her expert advice, constructive input, and words of encouragement helped me to always feel supported and do my very best.

Additionally, special thanks are extended to the industrial partner and the entire technical staff and engineers for their cooperation and support during the completion of my Master's thesis. Their practical insights and technical expertise helped put my research into a real-world context and allowed me to gain valuable insights.

My sincere thanks go to Dipl.-Ing. Dr.techn. David Müller, from the Technical University of Vienna, for his valuable support in topics related to physics. His technical expertise and willingness to share his knowledge have been instrumental in deepening my understanding and improving the quality of my work.

I am deeply grateful for the support I have received from these wonderful people and institutions. Their contributions have enriched my academic journey and significantly influenced the success of my Master's thesis.

Abstract

This thesis studies a temperature-dependent and multi-objective Capacitated Vehicle Routing Pickup and Delivery Problem, titled the TD-CVRPDP, that stems from a logistic steel plant application. In this routing problem, crude items need to be transported within a steel plant by heavy-duty vehicles. These vehicles are subject to the items' high temperatures and heavy weights. The routing problem at hand is the first of its kind to consider item and vehicle temperatures and their interdependence. Hence, this thesis conducts fundamental research on both the determination of the vehicle temperature during route planning, i.e., by means of the so-called *Vehicle Temperature Prediction Method* based on Newton's Law of Cooling (NLC) that was developed through inductive reasoning in the course of multiple on-site experiments at the partnered steel plant, and the design of a solution framework capable of handling the temperature aspect in a holistic manner, i.e., by means of a customised Adaptive Large Neighborhood Search (ALNS) metaheuristic that incorporates the temperature aspect as part of the objective function and neighbourhood evaluation. The consideration of the fleet's temperature is accompanied by difficulties regarding unexpected side effects in the solution structure, challenging the solution's practical applicability. Hence, the so-called *Directional-Penalty* is introduced, a penalty function applicable to a wide range of routing problems that, briefly put, aims to minimise the travel time of the items' individual journeys. Additionally, a new heuristic, titled the Dynamic-5-Request-Look-Ahead-Tree-Search (D-5R-LA-TS), is presented, designed to solve the TD-CVRPDP as a dynamic problem. Finally, all applied solution methods, conventional and new, are explained in great detail and thoroughly related to existing methods in the literature. Computational experiments demonstrate the effectiveness of all proposed solution methods and reveal valuable managerial insights by analysing the solution structure of the TD-CVRPDP as opposed to the generalised CVRPDP.

Kurzfassung

Diese Arbeit untersucht ein temperaturabhängiges und multikriterielles Capacitated Vehicle Routing Pickup and Delivery Problem, das den Titel TD-CVRPDP trägt und aus einer logistischen Stahlwerksanwendung hervorgeht. Bei diesem Problem müssen Rohmaterialien innerhalb eines Stahlwerks mit Schwerlastfahrzeugen transportiert werden. Diese Fahrzeuge sind den hohen Temperaturen und dem hohen Gewicht der Güter ausgesetzt. Das vorliegende Routenplanungsproblem ist das erste seiner Art, das die Temperaturen von Gütern und Fahrzeugen berücksichtigt. In dieser Arbeit wird unter anderem Grundlagenforschung betrieben, die sich mit der Bestimmung der Fahrzeugtemperatur im Rahmen von Routenplanungsproblemen befasst. Hierfür wurde eigens eine Methode namens *Vehicle Temperature Prediction Method* entwickelt, die auf dem Newton'schen Abkühlungsgesetz (NLC) basiert und die im Rahmen mehrerer Vor-Ort-Experimente im Partnerstahlwerk durch induktive Verfahren entwickelt und validiert wurde. Zudem wird ein metaheuristisches Lösungsverfahren präsentiert, das den Temperaturaspekt ganzheitlich behandelt. Hierbei wurde die etablierte Adaptive Large Neighbourhood Search (ALNS) Metaheuristik angewandt und problemspezifisch angepasst, sodass der Temperaturaspekt als Teil der Zielfunktion und während der Evaluierung der metaheuristischen Nachbarschaft einbezogen wird. Die Berücksichtigung der Temperatur der Flotte ist mit Schwierigkeiten hinsichtlich unerwarteter Nebeneffekte in der Lösungsstruktur verbunden, was die praktische Anwendbarkeit der Lösung in Frage stellt. Daher wird die sogenannte *Directional-Penalty* eingeführt, eine Straffunktion, die auf eine Vielzahl von Routenplanungsproblemen anwendbar ist und kurz gesagt darauf abzielt, die Reisezeit der einzelnen Strecken, die die Güter zurücklegen, zu minimieren. Schließlich wird eine neue Heuristik, die Dynamic-5-Request-Look-Ahead-Tree-Search (D-5R-LA-TS) Heuristik, vorgestellt, die zur Lösung der dynamischen Version des TD-CVRPDP entwickelt wurde. Schließlich werden alle angewandten Lösungsansätze in ihrer herkömmlichen als auch adaptierten Form ausführlich dargelegt und mit bestehenden Methoden in der Literatur in Zusammenhang gebracht. Computergestützte Berechnungsergebnisse demonstrieren die Effektivität der vorgeschlagenen Lösungsmethoden und offenbaren wertvolle betriebswirtschaftliche Erkenntnisse durch die Analyse der ermittelten Lösungen und den Vergleich der Lösungs- strukturen, die sich beim TD-CVRPDP versus dem generalisierten CVRPDP ergeben.

Contents

List of Tables

List of Figures

List of Algorithms

1. Introduction

The transformation of the steel industry towards sustainable production processes has garnered significant interest and investment, and optimised transport logistics in steel production plants is a critical area of focus in this transformation. Integrating resource-efficient transportation methods based on performance-oriented approaches is essential to achieve this goal (World Steel Association, 2023). Research in this area aims to develop optimisation algorithms that meet the steel industry's complex transport requirements while considering environmental and efficiency aspects. The focus is not only on ecological objectives but also on economic aspects, such as reducing operating costs, enhancing efficiency, and increasing competitiveness. Studies by McKinsey & Company (2021) also underline the importance of a holistic sustainability strategy for companies in the metals and mining industry. This strategy includes measures to reduce emissions and resource consumption throughout the entire production and supply process. Similarly, research by Zhang et al. (2022) also highlights the challenges of designing logistics networks for the steel industry, taking into account environmental aspects such as carbon emissions.

Steel production plants, such as the partnered plant of this research, are so-called fully integrated metallurgical plants. These plants perform all steps of steelmaking, from processing pig iron and scrap metal, e.g., smelting raw material, to finishing sheet metal, e.g., steel coils. Within their extensive network are numerous facilities that support a variety of processes in steel production. These include a steel mill facility that produces crude steel, multiple strands in the continuous caster for the continuous shaping of the steel, two warm holding boxes where the steel is kept at the required temperature after casting, and numerous outdoor storage areas for the storage of steel products of various sizes and shapes. There is also a scarfing facility, which is used to remove impurities from the surface of the steel and improve its quality. These diverse facilities form the backbone of the steel plant and ensure that production runs smoothly and the demand for steel products is met effectively. Hence, the routing model at hand considers a network of different vertices representing the aforementioned production facilities and storage areas. Each vertex can represent the sources for a *Pickup* and the destination for a *Delivery* of item flows, e.g. steel slabs, in the logistics network.

Within the realm of this research, special emphasis is put on the transportation of semi-finished casting products, so-called steel slabs (Figure 1.1a), using heavy-duty vehicles, so-called straddle carriers (Figure 1.1b), within the steel plant.

Steel slabs are stackable heterogeneous items that can vary in thickness, weight and temperature. They can weigh up to $36,000$ kg and reach a temperature of 575°C. These steel slabs are produced in a continuous caster, whereby they are potentially processed in between and finally rolled into steel coils.

One critical area of focus within the transport logistics of steel production plants is

(a) Steel slabs

(b) Straddle carrier

Figure 1.1.: Example of a stack of steel slabs and a heavy-duty straddle carrier

the fleet of heavy-duty vehicles known as straddle carriers. These vehicles are specially designed for the transportation of steel slabs, with a total load capacity of up to 105 tons and a maximum steel slab transportation temperature of 450°C. A straddle carrier is equipped with an integrated crane for the transportation of steel slabs, enabling steel slabs to be stacked, loaded and unloaded autonomously. This feature is necessary as the steel slabs are transported together as a stack. These straddle carriers have a kerb weight of around 100 tons and are therefore not only costly in acquisition but also during operation, mainly due to their significant fuel consumption. There are also expensive and frequent maintenance costs, which makes resource-efficient use particularly important.

To ensure efficient and sustainable transportation of steel slabs, a challenging variant of the Capacitated Vehicle Routing Pickup and Delivery Problem (CVRPDP) is being considered in this research, inspired by the requirements of the partner steel plant in Austria. Straddle carriers stack steel slabs on top of each other in an arbitrary order, forming a transportation stack, and move them along within the steel plant. During transportation, practical constraints are imposed. Alongside the height and weight constraints, incorporating temperature constraints into the CVRPDP poses a significant challenge, as it requires the development of precise temperature measurement methods that account for the heat exchange between the straddle carrier and the transported steel slabs. However, incorporating this complexity into route planning not only enables efficient use of resources but also ensures the quality and integrity of the transported steel slabs. By integrating temperature constraints into the CVRPDP, significant progress can be made that not only improves vehicle preservation but also contributes to the safety of the driver.

Therefore, this research introduces a novel Vehicle Routing Problem (VRP) specifically tailored to the requirements of heavy industries at large: the Temperature Dependent Capacitated Vehicle Routing Pickup and Delivery Problem (TD-CVRPDP). It addresses the complex logistical challenges of transportation in heavy industries, especially considering the impact of heavy transportation loads and temperature dependencies between vehicles and items.

In heavy industry transport logistics, similar to the traditional CVRPDP, minimising the total *travel time* of the fleet is usually defined as an objective function in order to manage a high throughput volume. However, heavy industries are also characterised by continuous operation, which means the fleet is used around the clock. In order to operate the fleet in a sustainable and resource-efficient manner, the effects of temperature and weight of the transported item stacks must be taken into account. Therefore, three objective components are investigated, including *travel time*, *ton travel time*, and *vehicle temperature*. *Travel time* is a conventional factor that represents operational efficiency, whereas *ton travel time* is a weighted factor of *travel time* that takes into account the weight of the transported load measured in ton-seconds. This factor is particularly important for long and heavy transports since it affects fuel consumption and vehicle preservation. Similar to the *ton travel time* objective component, the *vehicle temperature* is another essential objective component that affects mechanical vehicle preservation, but it mainly improves the driver's safety. A temperature curve of the vehicle, expressed in degrees Celsius, is analysed to ensure that the operating temperature remains within safe limits during transport. Accurate control of *vehicle temperature* is critical to ensure optimal vehicle performance and durability and to minimise potential safety risks. Taking these three objective components into account enables a holistic optimisation of transport logistics in heavy industry by not only maximising operational efficiency but also reducing environmental impact and meeting safety standards.

This research contributes to the scientific field of real-world vehicle routing problems in the following ways.

First, a novel CVRPDP specifically designed to integrate temperature dependence is presented. This novel routing problem discloses challenges of so-called *strolling trips* that need to be addressed. In this regard, a novel constraint, named *Directional-Penalty*, which is used to control the amount of *strolling trips*, is implemented. Within this research, insights into these challenges are provided, and innovative solutions to overcome those challenges are offered.

Second, a method that estimates the *vehicle temperature* throughout the route planning called *Vehicle Temperature Prediction Method* is developed. This method utilises Newton's Law of Cooling (NLC) and heating to predict the temperature of the vehicle, taking into account various influencing factors such as the starting temperature of the vehicle, the temperature of the item stacks, the environmental temperature, the *travel time*, properties of the vehicle's load, and other parameters. This development was driven forward by inductive reasoning within the framework of extensive on-site experiments in the partnered steel plant, a leading steel plant in Austria. Real-time measurements were carried out and validated by experts to ensure the precision and reliability of the method.

Third, the effects of individual objective components such as the *travel time*, *ton travel time*, and *vehicle temperature*, and their impact on the overall efficiency of the transport system are analysed. For this purpose, test instance sets are introduced for the TD-CVRPDP based on real data sets from the partnered steel plant. By analysing the Pareto-front in detail, we show the individual effects of these objective components and provide valuable insights for management.

Fourth, we have implemented a novel heuristic, named the Dynamic-5-Request-Look-Ahead-Tree-Search (D-5R-LA-TS), as part of an excursion to explore the TD-CVRPDP as a dynamic problem. This heuristic supports practical applicability, allowing a steel plant with continuous production to manage logistical uncertainties seamlessly.

Subsequent sections of this research are structured as follows: Chapter 3 provides a comprehensive review of the relevant literature, which not only covers and supports previously mentioned approaches and solutions in the field of the CVRPDP but also considers relevant findings from related fields such as transport logistics, operations research, applied mathematics, and the steel industry. This literature review provides an in-depth context for our contributions. It lays the foundation for the development of new methods and approaches to address the temperature-dependent challenges in CVRPDP.

This is followed by a formal description of the temperature-dependent CVRPDP in Chapter 2. The problem's theoretical foundations are explained, including definitions, assumptions and restrictions required for the correct modelling of temperature dependencies in transport logistics processes.

Chapter 4 describes the methodology of the *Vehicle Temperature Prediction Method* for measuring *vehicle temperature* in detail. Data-based and mathematical analysis methods used to predict temperature changes are discussed. In addition, the challenges and limitations of temperature measurement are explored, and possible solutions for improving measurement accuracy and reliability are presented.

A detailed presentation of the method used to optimise multiple objectives, i.e. the Weighted Sum Approach (WSA), and the difficulties that arise is given in Section 5.1, where not only the Solution methods concerning these challenges are discussed, but also practical application examples are presented to demonstrate the effectiveness of the used tools.

This is followed by a specification of the ALNS solution framework in Section 5.3.2, which serves as an efficient optimisation tool for solving complex routing problems. The basic principles and algorithms of the ALNS are explained in Section 5.3.1, and specific adaptations and extensions required for a customised application to the TD-CVRPDP are presented in Section 5.3.2. Given some challenges that arise during the modelling process, a novel penalty called *Directional Penalty* is also introduced in Section 5.3.3 and its effectiveness is discussed in Section 6.3.2.

In the following, the methods proposed by us are subjected to a detailed mathematical evaluation in Section 6.5. Various performance indicators and evaluation criteria are used to assess the efficiency and practicability of the developed solutions and to identify potential areas for improvement.

In addition, the importance and interaction of the individual objective components are explained in Section 6.3.1, whereby not only the direct effects on the efficiency and performance of the transport system are considered, but also possible synergy effects and interactions between the various objective components are analysed.

Finally, Chapter 7 presents a summary of the research. In this section, managerial insights, methodological challenges, and potential application areas for further research are identified and discussed to spur future development in the research field.

2. Problem Description

The routing problem at hand, the Temperature Dependent Capacitated Vehicle Routing Pickup and Delivery Problem (TD-CVRPDP), is a special variant of the multi-objective Vehicle Routing Problem (VRP) (Jozefowiez et al., 2008). It is a challenging logistical problem as it takes temperature-dependent factors during the planning of transport routes into account. The goal is to transport steel slabs, which we refer to as items in the following, within the steel plant from and to various production and storage facilities, which we refer to as vertices in the following, while optimising several objectives such as minimising *travel time*, *ton travel time* and *vehicle temperature*, which will be discussed in the following.

The graph The steel plant's site is modelled on a directed graph $G = (V, A)$, where V represents the set of vertices and A is the set of arcs. The vertex set V includes a depot vertex, which is denoted by v_0, and n other vertices, denoted by $v_1, ..., v_n$, each of which can hold multiple items to be processed, which we refer to as requests R. Request pickups and deliveries can occur on any given vertex $V \backslash \{v_0\}$, and one or several requests can be picked up from one and the same pickup vertex and delivered to its/their corresponding delivery vertex/vertices. The set of arcs A is defined by $A = (i, j) \mid i, j \in V, i \neq j$, where each arc (i, j) has a specific *travel time* t_{ij}.

It is assumed that *travel times* are non-negative, i.e. $t_{ij} >= 0 \in A = (i, j) \mid i, j \in V, i \neq j$, and that the triangle inequality applies. The triangle inequality in the VRP states that for three vertices i, j and k in the graph, the *travel time* between i and j is always shorter than or equal to the sum of the *travel times* over a third vertex:

$$t_{ij} \leq t_{ik} + t_{kj} \tag{2.1}$$

Here t_{ij} stands for the *travel time* between vertex i and vertex j, t_{ik} for the *travel time* between vertex i and vertex k, and t_{kj} for the *travel time* between vertex k and vertex j. Compliance with this inequality is essential to ensure that the calculated route lengths in the VRP are correct and that no unnecessary detours are made that could impair the efficiency of the solution by misguiding us during our fundamental research on the impact of the temperature in VRPs.

The requests In this research, we are faced with the challenge of handling a large number of pickup and delivery processes for a wide range of items R. Each request $r \in R$ refers to a specific item and defines both the pickup location p_r and the delivery location d_r. In addition, the items vary in weight q_r (measured in kilograms), height u_r (measured in centimetres) and temperature t_r (measured in degrees Celsius).

The fleet The fleet, i.e. the set of vehicles, is denoted by K, which consists of homogeneous vehicles in terms of transport-relevant properties. Each vehicle has the capacity to hold multiple items by stacking them on top of each other. The loading and unloading of items during the stacking process leads to complex stacking operations. Although these operations are of great importance, we neglect the loading and unloading operations and the associated stacking constraints in order to not further obstruct our fundamental research on the TD-CVRPDP.

Each vehicle has maximum weight and height limitation for stacking items in the load compartment, referred to as $U_{\max}$ and $H_{\max}$, respectively. In addition, there is also a maximum temperature limitation $T_{\max}$ for the vehicles, which primarily ensures the preservation of vehicle-critical mechanical components and the associated safety of the driver. The *vehicle temperature* is influenced by several factors, which are discussed in Chapter 4. Calculating and predicting the *vehicle temperature* during its operation is a mathematically complex task, which poses a significant challenge, particularly in the context of finding appropriate methods. In addition, for each vehicle $k \in K$, we define the kerb weight of the vehicle by d_k and the initial temperature by T_0.

The solution and constraints A viable solution to the TD-CVRPDP fulfils all requests R using $|K|$ vehicle routes. The vehicle weight $U_{\max}$, height $U_{\max}$, and temperature $T_{\max}$ limitations of the fleet are not to be exceeded. These limitations are hard constraints, meaning that not adhering to them would lead to an infeasible solution, as either the vehicles would not be able to physically handle the weight and height of the transportation stack or exceeding them could lead to mechanical damage in terms of temperature.

The depot, denoted by vertex v_0, serves as the central hub where each vehicle begins and finalises its route. While vehicles K are allowed to visit the same vertex numerous times, requests R can visit each vertex only once. This difference in access rights reflects the operational requirements of our partnered steel plant, where cycles from an item's trip from its pickup vertex to its delivery vertex are prohibited, which enables a transparent flow of items. The strong interest in eliminating unnecessary cycles in an item's journey is due to minimising emissions and vehicle wear and tear, both of which are increased by the higher weight of the load. These strategic measures also help to optimise the utilisation of the vehicle fleet, which is discussed in Chapter 6.

Formally, a solution is described by the fleet's operation, which means that the successive arcs travelled during each vehicle route are determined. This movement of the fleet is represented by the binary variable $x_{ij}^{k\mathcal{R}}$, where $k \in K$ is the vehicle and $\mathcal{R} \subset R$ is the subset of the transported requests, i.e. the stack of items, on arc $(i, j) \in A$. This is called a binary variable because it can either take the value 0 or 1, which means that it is used to determine whether the travel time t from vertex i to vertex j of vehicle k with the requests $\mathcal{R}$ is included in the calculation. This can be seen in the second line of the objective function in Equation 2.2.

On multi-objective optimisation Multi-objective optimisation is a multifaceted field of optimisation theory that deals with the simultaneous optimisation of multiple, often

conflicting objectives. In contrast to traditional single objective optimisation, which aims to maximise or minimise a single objective, multi-objective optimisation focuses on finding solutions that represent a compromise on the Pareto-front, thereby achieving an optimal balance between the different objectives (Deb, 2001; Coello Coello, 2006; Marler and Arora, 2004).

The Pareto-front, named after the Italian mathematician Vilfredo Pareto, represents a collection of non-dominated solutions where no solution dominates the outcome of another in all objectives (Deb, 2001; Marler and Arora, 2004). The main objective of multi-objective optimisation is to identify these Pareto-efficient solutions that represent the best possible trade-offs between the competing objectives (Coello Coello, 2006).

The development of solution approaches for the multi-objective Temperature Dependent Capacitated Vehicle Routing Pickup and Delivery Problem (TD-CVRPDP) is challenging, as it is a new type of problem, and little experience is available. Especially when it comes to *vehicle temperature* measurement and quantification, innovative approaches are needed to find efficient and practical solutions. Therefore, a method named *Vehicle Temperature Prediction Method*, based on Newton's law of cooling, is developed, presented in detail in Chapter 4.

Objective components of the TD-CVRPDP The first objective component, which can be seen in the second line of the objective function in Equation 2.2 denoting the *travel time*, strives to minimise the maximum total *travel time* of the fleet in order to optimise the profitability of the transport system. It is a common objective in VRPs, as shorter *travel times* not only reduce operating costs but also enable faster processing of all requests.

$$\begin{aligned} min \rightarrow Z = \\ \alpha \cdot \max_{k \in K}(\sum_{(i,j) \in A} (x_{ij}^{k\mathcal{R}} \cdot t_{ij})) \\ + \beta \cdot \delta' \cdot \max_{k \in K,(i,j) \in A} (\sum_{r \in \mathcal{R}} (x_{ij}^{k\mathcal{R}} \cdot t_{ij} \cdot q_r) + x_{ij}^{k\mathcal{R}} \cdot t_{ij} \cdot d_k) \\ + \gamma \cdot \delta'' \cdot \max_{k \in K,(i,j) \in A} (T(x_{ij}^{k\mathcal{R}})) \\ + p \end{aligned} \tag{2.2}$$

The second objective component, which can be seen in the third line of the objective function in Equation 2.2 denoting the *ton travel time*, focuses on minimising the maximum *ton travel time* of the fleet. This is achieved by weighing the *travel time* of each arc of each vehicle's route by the total request load of the vehicle. This total load comprises the weight of the loaded stack of items and the vehicle's kerb weight. The aim is to reduce long and heavy trips, leading to more efficient fuel consumption and supporting the fleet's longevity. Reducing long and heavy transports also helps to minimise the wear on the fleet and thus reduce their maintenance frequency.

The third objective component, which can be seen in the fourth line of the objective

function in Equation 2.2 denoting the *vehicle temperature*, aims to minimise the maximum *vehicle temperature* of all arcs of the fleet's routes. The *vehicle temperature* is determined using four main contributing factors. First, T_e is the environmental temperature, which specifies the cooling rate of the vehicle and item stack. Second, T_r is the temperature of a request, which contributes with its weight, its specific material properties and the duration of exposure. Finally, the temperature of the vehicle is determined by the function $T(x_{ij}^{k\mathcal{R}})$, which is explained in Chapter 4 for a more comprehensive understanding. It is worth mentioning that excessively high temperatures in the vehicle can not only impair the vehicle's integrity but also lead to additional safety risks. Especially when transporting items that can reach temperatures of up to 450°C, careful control of the *vehicle temperature* is essential. Too high *vehicle temperatures* can increase the risk of hydraulic oil fires, as the hydraulic oil can become flammable at temperatures above its flash point, which is usually below 300°C (Knecht, 2022). Therefore, controlling and minimising *vehicle temperature* is critical. Hence, the intended outcome of this objective component is to ensure both the safety of transportation and the integrity of the vehicle.

The fifth line in the objective function in Equation 2.2 represents the applied penalty, which is made up of several components (Chapter 5). The most important of these is the newly developed *Directional-Penalty*, explained in Section 5.3.3. Similar to the second objective function component, i.e. *ton travel time*, it also aims to improve vehicle utilisation in terms of transport weight.

Each objective component strives to ensure uniform, objective achievement across all vehicles in the system. This is achieved by using $\max_{k \in K}$ and $\max_{k \in K,(i,j) \in A}$ as a means of maximising objective achievement. This approach ensures that no vehicle achieves excessive objective fulfilment at the expense of other vehicles.

In addition, the individual objective components are weighted by the coefficients α, β and γ to reflect their relative importance in the overall objective function. This weighting allows for an appropriate consideration of each aspect of the TD-CVRPDP and ensures a balanced optimisation. This method is also known as the Weighted Sum Approach (WSA), which is discussed in more detail in Section 5.1.

Moreover, the relativisation values δ' and δ'' play a crucial role as they relativise the *ton travel time* and the *vehicle temperature* objective values to the *travel time* objective value. This relativisation is needed because the objective values of the three objective components are measured with different units of measure and are on a different scale. It ensures that all objective components can be optimised on a fair basis. The relativisation is discussed in Section 5.1.2.

All objective function components combined, with their weight coefficients used for the Weighted Sum Approach (WSA), the relativisation of these and the penalty result in the so-called *quality value*, which is, ultimately, the objective value of the entire objective function.

On the interplay of objective components We need to consider different approaches to address the different objective components effectively. Based on the potential impact of

these objective components on the overall solution, we formulate the following hypotheses:

Travel time as one of the objective components aims for short vehicle routes. This objective component usually leads to a higher utilisation of the vehicle in terms of the load. However, a higher load also results in greater wear and tear and it could expose the fleet to higher temperatures. The latter is rather the result of the negligence of the *vehicle temperature*, which means that the interrelationship between *travel time* and *vehicle temperature* is possibly not causal and compelling.

In contrast, the *ton travel time* objective component integrates the weight of the item stack and the kerb weight of the vehicle into the calculation. It strives to minimise the total load of all vehicles by accommodating short trips for individual requests and by prioritising empty vehicle trips. However, this strategy could increase the overall *travel time* due to the empty trips as they travel without any requests being transported.

It is important to note that while short journeys of individual requests and empty trips can help reduce vehicle temperatures, both *ton travel time* and *travel time* do not consider which items are transported together in a stack. This disregard could lead to inefficient loading in terms of *vehicle temperature* optimisation and thus often contribute to higher vehicle temperatures, predominantly if the item stacks are not optimally composed and hinder heat balancing (see Chapter 4).

The *vehicle temperature* is influenced by several factors, first and foremost the individual temperatures of the loaded items t_r and their weight q_r. For items with a high temperature and a relatively high weight, keeping their trip from their pickup vertex to their delivery vertex as brief as possible is favourable. This reduces the time the items are gripped by the vehicle and thus also its temperature convection onto the vehicle. Conversely, for colder and heavier items, extending their journey by combining them with warmer items during transportation is beneficial. This measure balances the overall temperature of the item stack, which helps to reduce the amount of convection from the stack onto the vehicle, thus reducing the *vehicle temperature* compared to the primary transportation of hot items. The schematic extension of an item's trip from its pickup vertex to its delivery vertex is henceforth referred to as the *item strolling effect*. Similarly to the other two objective components, the *vehicle temperature* does not consider *travel time* and the load weight. Therefore, it conflicts with the other two objective components: the minimisation of *travel time* and the minimisation of the *ton travel time*. In Chapter 6, computational experiments include a detailed analysis of the complex interactions between all three objective components. This analysis will allow informed decisions to optimise the TD-CVRPDP and maximise the system's performance.

Additionally, we propose the following hypotheses with regard to the interplay of objective components. The *travel time* and *ton travel time* are concurrent objectives, whereas the *travel time* and *vehicle temperature* as well as the *ton travel time* and *vehicle temperature* represents conflicting objectives. These statements are investigated in detail in Chapter 6.

Although the CVRPDP is a novel routing problem tailored to the problem case of the partnered steel plant, it can be useful in routing and scheduling problems in heavy industries, such as slag transportation or steel coil and galvanising line scheduling.

3. Literature Review

Heavy-duty vehicles These vehicles are responsible for more than 6% of the total European Union's greenhouse gas emissions (European Commission, 2024). Various scientific disciplines address this issue.

Within the area of routing, Luo et al. (2022) study an exposure-based vehicle routing strategy in road freight traffic. The goal is to find a low-exposure route that would reduce community-wide exposure to heavy-duty diesel trucks' tailpipe emissions. This is done by navigating a heavy-duty diesel truck away from disadvantaged communities that are overburdened by truck traffic at the cost of increasing *travel time* by as little as possible. The results show that community-wide exposure to a truck's tailpipe emissions can be reduced by up to 50% on average, with an average *travel time* increase of 5%. Amiri et al. (2023) investigate a multi-objective routing problem for heavy-duty battery electric trucks with uncertain energy consumption in the short-haul delivery problems. The aim is to minimise transportation costs and maximise customer satisfaction. The authors test two metaheuristic solution approaches, e.g., a non-dominated sorting Genetic Algorithm (GA) and an Adaptive Large Neighborhood Search (ALNS), in combination with three multi-objective approaches, e.g., ϵ-constraint, WSA, and hybrid methods. They show that the ALNS combined with the WSA outperforms all other approaches. (Wang and Rakha, 2017) study eco-routing and eco-driving systems, e.g., recommendations on how drivers shall accelerate and brake. They provide a fuel consumption model for heavy-duty diesel trucks that generates CO_2 emission estimates and predicts fuel consumption levels.

The environmental and economic impact of heavy-duty trucks is commonly addressed in engineering literature. Cunanan et al. (2021) review heavy-duty vehicle powertrain technologies. They investigate the advantages, disadvantages and future perspectives of all Diesel engine heavy-duty vehicles, battery electric heavy-duty vehicles, and hydrogen fuel cell electric heavy-duty vehicles. They conclude that diesel engines in heavy-duty vehicles will remain an essential technology in the future due to the existing infrastructure and lower costs despite their high emissions. Similarly, Iyer et al. (2023) study diesel, electric, fuel-cell, and hybrid powertrains by providing a vehicle-cycle and life-cycle analysis. The authors find that switching from diesel powertrains to alternative electric and fuel-cell powertrains causes an increase in vehicle-cycle greenhouse gas emissions but leads to substantial reductions when considering the combined vehicle- and fuel-cycles. Nevertheless, they suggest that diesel powertrain technology will remain highly relevant in the short-term future. Sen et al. (2019) investigate an environmentally, economically, and socially optimised fleet mix given battery-electric, hybrid, and diesel heavy-duty trucks. They consider multiple objectives, such as life-cycle costs, life-cycle greenhouse gases, and life-cycle air pollution externality costs. They conclude that both techno-economic circumstances and forms of electricity generation need to be improved in order

for heavy-duty fleet mixes to mitigate greenhouse gas emissions.

All of the aforementioned literature study resource efficiency from a routing and technological perspective. However, to the best of our knowledge, there is no literature that addresses straddle carriers in particular and the reduction of fuel consumption by distributing that vehicle's load efficiently along a vehicle's route.

Transport logistic problems that stem from steel plant applications These kind of real-wold applications are commonly studied in the literature.

The in-depth analysis of continuous caster scheduling operations has become crucial as it is often perceived as a critical bottleneck in the steel industry that directly impacts production capacity and efficiency. In particular, the work of Myungho Lee and Pinedo (2023) provides a detailed insight into planning and time coordination in steel production, especially in the context of continuous casting. They not only present a mere observation but also introduce two basic models specifically geared towards overcoming planning and scheduling problems in continuous casting plants and covering all essential aspects.

In addition, the work of Özgür et al. (2021) is characterised by its comprehensive presentation of planning and scheduling methods for hot rolling mills at steel plants. A wide range of optimisation methods for the scheduling of hot rolling mills is highlighted. It is important to note that various factors are of significant importance in this area, including the level of abstraction of the models and the availability of data. This diversity leads to different problems and constraints that need to be considered during the optimisation of planning and scheduling processes to ensure efficient and smooth production in steel plants.

The routing of steel products, predominately so-called coils, between distribution centres located in different cities or countries and transported by various means of transport, e.g., ships, trucks, and locomotives, constitute a routing problem mainly studied within steel logistics. Wolfinger (2021) studies a pickup-and-delivery problem with time windows, split loads and transshipments that stems from a steel industry application. The authors use a Large Neighborhood Search (LNS) solution approach and show their method's effectiveness by using benchmark instances from the literature. They solve real-world instances and provide managerial insights. Lee and Kim (2015) propose a maritime pickup and delivery problem of a steel manufacturing company. The authors title their work the Industrial Ship Routing Problem (ISRP). They consider time windows, split deliveries, and a capacitated and heterogeneous fleet. A mixed integer programming model is provided, and an ALNS solution approach is used to solve real-world-sized test instances. Huang et al. (2023) address the locomotive routing problem that stems from a steel industry use case. It is a pickup and delivery problem with time windows, where request incompatibilities, variable locomotive capacities, and last-in-first-out constraints are considered. The authors provide a mixed integer programming model and an ALNS solution approach. They compare their computational results to the results provided by their partnered steel plant.

Intra-facility routing problems within a fully integrated metallurgical plant, such as the problem of this thesis at hand, are studied in the literature, however, only to a limited

extent. Roljic et al. (2024) studies a Capacitated Vehicle Routing Problem (CVRP) with item deliveries and active and passive vehicle types, i.e., a crewed truck with no capacity to stow items and capacitated trailers and item transshipments without detours. The problem stems from a steel plant application titled the Active-Passive Vehicle Routing Problem with Item Transshipments (APVRPIT). The authors propose a nested LNS and discuss managerial insights. Additionally, they show the effectiveness of their solution approach by solving a generalised version of their problem, namely the APVRP (Meisel and Kopfer, 2014), and providing improved results on existing benchmark test instances. (Cheng et al., 2010) extended the well-known APVRP that takes into account various features and constraints unique to a steel plant in China.

Another logistical problem commonly studied in steel-producing plants is the Block Relocation Problem (BRP) and Pre-Marshalling Problem (PMP). Since many steel products, mainly steel slabs, are stacked on top of each other, retrieving or reorganising single or multiple steel slabs can result in complex stacking operations. Tricoire et al. (2018) studies a BRP from a steel plant. In this problem, the crane that stacks the blocks, i.e., steel slabs, can only hold a single block. The authors contribute fast heuristics and metaheuristics. They additionally embed their solution approaches in a branch-and-bound algorithm in order to study the efficiency of the proposed solution approaches in further detail. The authors are able to improve upon benchmark instances from the literature. Roljic et al. (2021) study an extended BRP which, similar to us, uses straddle carriers. The peculiarity of this type of vehicle is that it has an integrated crane that can hold multiple steel slabs at once. The authors study a combined approach where they solve the intra-facility routing and stacking of steel slabs as part of a straddle carriers retrieval during the pickup operation. They use a LNS to solve the routing problem, a heuristic solution approach for the stacking problem, and show the effectiveness of their approach by solving real-world test instances. Pfrommer et al. (2024) study a PMP that can be found, among other use cases, in the steel industry. Similar to the BRP, only one load can be moved at a time. During pre-marshalling, single loads are sorted in a block-stacking storage system during off-peak hours to prepare for future orders. The authors combine a network flow model and an A* algorithm with an adapted lower bound. The solution approach is tested on randomly generated instances.

To the best of our knowledge, the only contributions to the literature that study temperature prediction models in the steel industry within a logistic and operations research context are (Duarte et al., 2022) and (Gupta and Chandra, 2004). However, both works address liquid steel and do not involve any routing aspects.

Tri-objective routing problems These are mainly found in logistics and operations research literature that addresses real-life logistics planning and decision-making. Additionally, the WSA is commonly used in papers that study applied cases (Matl et al., 2018). An et al. (2024) propose a linear programming-based matheuristic for tri-objective binary integer linear programming. The objectives model the total servicing cost, total fixed opening cost, and total customer demand coverage. The proposed method outperforms the established matheuristic on tri-objective benchmark instances. The problem is inspired

by supply chain network design applications. However, this method cannot solve large-scale instances in a reasonable amount of computational time. Another tri-objective supply chain network application can be found in Abbasi et al. (2023). The authors study sustainable closed-loop supply chain networks and investigate the environmental, economic, and social trade-offs in case of emergency situations or viral outbreaks. They provide a mixed-integer programming model. Ning et al. (2018) study an industrial tri-objective use case. It concerns the planning of safe construction site layouts. An ant colony optimisation-based model is applied. The three objectives involve two safety objective functions and one cost objective function. A case study is used to verify the proposed method. Wang et al. (2023) study disaster and emergency logistics and introduce a problem called the two-echelon emergency vehicle routing problem with time window assignments. It addresses, in particular, travel restrictions in closed areas. The objectives are to minimise the total operating cost, total delivery time, and number of vehicles. An extended ALNS is used. The performance of the proposed model and algorithm are tested during a real-world case study.

To conclude, this work at hand provides an innovative perspective on the novel TD-CVRPDP. In particular, it is characterised by a problem definition incorporating a temperature aspect where the development of new solution methods such as the *Vehicle Temperature Prediction Method* is needed. This specific aspect is crucial as it has received little and only tangential attention in the literature on Multi-Objective Routing Problems. The integration of a *Vehicle Temperature Prediction Method* not only extends the requirements for logistics planning but also opens up new approaches for overcoming complex challenges in the logistics industry. Thus, this work contributes significantly to the further development of the research field and offers potential for future applications and extensions in the field of logistics optimisation.

4. On Measuring Temperature

The consideration of the steel slabs, which we will refer to as items in the following, and *vehicle temperature* constitute a non-trivial extension to the TD-CVRPDP. Determining the exact temperature fluctuation of an item and vehicle over time is practically infeasible, as it is influenced by an almost infinite number of external factors, e.g., different environmental temperatures along the route, wind, driving speed, solar radiation, etc. To obtain temperature measurements that are accurate and practically viable, we apply an approximative calculation method that we title the *Vehicle Temperature Prediction Method*. As temperature does not behave in a linear manner, methods that anticipate linearity, e.g., the moving average method, cannot be used for this purpose. We refer to basic laws of physics and assess that within the realm of our research, the transfer of heat between different locations is derived from three distinct mechanisms: Convective Heat Transfer (CHT), Radiative Heat Transfer (RHT) and Conductive Heat Transfer (COHT).

4.1. Introduction to Heat Transfer

Convective Heat Transfer The laws of fluid dynamics apply to CHT, which is the movement of fluids (liquids or gases). Hot vehicles and items that are exposed to cool air warm up the surrounding air and lead to a decrease in its density. A convective flow is produced as this warmer and lighter air rises. The rising warm air is then replaced by denser and cooler air from the surroundings, and so on. By redistributing thermal energy within the fluid medium, i.e., air, the convective flow removes heat from the item's or vehicle's surface. The CHT is mathematically formulated by the equation of Newton's Law of Cooling (NLC):

$$Q_{\text{conv}} = h \cdot A \cdot (T_{\text{obj}} - T_{\text{sur}}) \tag{4.1}$$

The expression Q_{conv} denotes the CHT rate, measured in Watts or Joules per second. The term h represents the CHT coefficient, expressed in $\text{W}/(\text{m}^2 \cdot \text{K})$, which indicates how many watts of heat energy pass through per second and square meter of surface area when the temperature difference between two media is 1 Kelvin. The CHT coefficient depends on various factors, e.g., the nature of the fluid, the velocity of the fluid, the temperature difference between the solid surface and the fluid, and the characteristics of the surface itself. A denotes the surface area through which heat is being transferred, measured in square meters. T_{obj} stands for the temperature of the object's surface, measured in Kelvin, while T_{sur} represents the temperature of the surrounding fluid, also measured in Kelvin (Çengel and Ghajar, 2015). In practice, however, Celsius is preferred because it is more tangible and more common, which is why we choose Celsius as the unit of measurement in the following.

Radiative Heat Transfer In contrast to CHT, RHT is based on the transfer of electromagnetic waves—infrared radiation in particular—between objects. Infrared radiation is released by the hot items in the form of thermal waves. As radiation moves through the atmosphere, they are either absorbed by or deflected off of the surrounding surfaces that are colder or hotter, respectively, than the item itself, e.g., the vehicle. The Stefan-Boltzmann Law states that the temperature of the emitting and receiving surfaces is one of the factors that affect the rate of RHT. The item releases infrared radiation into its surroundings during the cooling process, which results in energy loss.

$$Q_{\text{rad}} = \varepsilon \cdot \sigma \cdot A \cdot (T_{obj}^4 - T_{sur}^4) \tag{4.2}$$

The term Q_{rad} describes the RHT rate, which is measured in units of Watts or Joules per second. The variable ε represents the emissivity of the surface, a dimensionless quantity that ranges between 0 and 1, indicating how effectively a surface emits thermal radiation. The constant σ is known as the Stefan-Boltzmann constant ($5.67 \times 10^{-8}/$, $W/(m^2 \cdot K^4)$), providing a fundamental value in RHT calculations.
In practice, CHT and RHT mechanisms operate simultaneously, e.g., when a hot item cools down or when a hot item heats up the vehicle. Comparing both, RHT exchanges thermal energy as electromagnetic waves, while CHT removes heat from the steel surface by allowing air to move physically (Çengel and Ghajar, 2015).

Conductive Heat Transfer Conductive Heat Transfer (COHT) is also a mechanism by which heat energy is transferred from one object to another when they are at different temperatures. Comparing it to the other two mechanisms, i.e. CHT and RHT, direct contact between the objects is needed. In the case of COHT, heat exchange occurs through the direct contact of molecules within a material or between different materials. The formula used for COHT is also known as Fourier's law and is as follows:

$$Q_{\text{cond}} = -k \cdot A \cdot \frac{\Delta T}{d} \tag{4.3}$$

The amount of heat transferred Q_{cond} by COHT is equal to the negative thermal conductivity k of the material multiplied by the contact area A and the temperature difference ΔT, divided by the thickness d of the material through which the heat is transferred (Çengel and Ghajar, 2015).

4.1.1. Comparing Temperature Measurement and Prediction

Real-world measurements (Table 4.1 & 4.2) suggest that omitting the RHT, which is compensated for by the CHT, by a slight adjustment of the calculation-related properties, results in a precise outcome regardless. This simplification allows for a faster computational time, which is beneficial in practical use.

The fact that the measured temperatures still deviate from the theoretical ones is due to environmental influences that cannot be taken into account. The bottom item, i.e. steel slab, is also cooled by the gravel (up to 10 times better thermal conductivity than

Time (in min)	Top (in °C)	Middle (in °C)	Bottom (in °C)
0	38	500	51
3	73	488	78
6	75	486,4	83
9	76	480,8	90,5
12	83	475	89,9
15	86	466	87,7
18	88,1	463,2	88,6
21	97,5	455,9	94,2

Table 4.1.: Example of an on-site experiment documenting the observed temperature change of three items, i.e. steel slabs, of approximately the same size that were stacked on top of each other. The temperature of all three items was measured at 3-minute intervals. The top item has a starting temperature of 38°C, the middle item 500°C and the bottom item 51°C.

air) on which it lies. This means that, in reality, it heats up slower than the top item. The temperature fluctuations in the measurements themselves, which are most noticeable in the lowest item, are due to the fact that only the surface temperature can be measured. If there is a stronger wind at the time of the temperature measurement, the measured temperature is cooler than the actual temperature.

4.2. The Vehicle Temperature Prediction Method

Customised approaches have been developed for this specific problem regarding the quantification and calculation of temperatures. These take into account the temperature calculation for a single item, a stack of items, and the *vehicle temperature*. The analysis of real-world measurements showed that the exact heat transfer between different items transported together in a stack can be neglected. This is due to the rapid adaptation of the surface temperature of these items to each other, which leads to an even release of heat from the entire stack after a short time. Precise on-site measurements suggest that calculating the stack temperature based on mass is an accurate approximation of the actual heat emitted by the whole stack. An additional benefit is that the mass-based calculation approach, which anticipates a uniform heat emission, enables an identical calculation of temperature variation for both a single and a stack of items. The temperature change is determined using the well-known NLC, which is typically expressed mathematically as follows:

$$T_s(t) = T_e + (T_s(0) - T_e) \cdot e^{-\frac{h}{c_v \cdot d \cdot \rho} \cdot t} \tag{4.4}$$

It is used to analyse the temperature dynamics of a stack of items in the context of CHT to the surrounding environment.

The temperature variable $T_s(t)$ represents the temperature of a stack of items under consideration at a given time t, whereas T_e represents the environmental temperature.

Time (in min)	Top (in °C)	Middle (in °C)	Bottom (in °C)
0	38	500	51
3	72,4	488,5	77,3
6	75,3	485,9	84,5
9	77,8	479,8	91,8
12	84,1	475,1	94,7
15	86,6	466,9	96,3
18	89,3	463,1	98,9
21	97,2	455,3	100,1

Table 4.2.: Predicted temperature changes provided by the adjusted convective heat transfer formula, i.e., *Vehicle Temperature Prediction Method*, discussed in Section 4.2.

The initial temperature of the stack is characterised by $T_s(0)$, while h represents the heat transfer coefficient, which determines the efficiency of heat transfer between the stack of items and its surroundings.

The specific heat capacity c_v quantifies the amount of heat required to raise the temperature of one kilogram of the material under consideration by one degree Celsius and thus characterises the thermodynamic properties of the material. The thickness of the stack is represented by the parameter d.

The density of the material, represented by ρ, affects the mass of the material, which influences its ability to absorb or release heat. Higher-density materials typically require more time for temperature changes as a greater mass needs to be heated or cooled.

This generic formulation of NLC is used to calculate the temperature change of individual items or a stack of items. A modified formula is used for the heating of the vehicle during the transportation of hot items. This is because of two reasons. First, it is impossible to take into account all the material-specific properties of all parts of the vehicle that affect heating. Secondly, the transported items cool down simultaneously during transportation. About the former, the cooling factor k, which was calculated from precise temperature measurements through experimental tests on-site at the partnered steel plant, is used instead of the cooling factors of the different vehicle parts and their corresponding material-specific properties.

Due to the limited contact surfaces during transportation of the vehicle, achieved by the special grippers on the straddle carrier with only four tongs of 3 cm x 30 cm each, it is assumed that cooling is purely due to the environmental temperature.

The underlying formula for the temperature change of the vehicle is as follows:

$$T_v(t) = T_s(t) + (T_v(0) - T_s(t)) \cdot e^{-k \cdot t} \tag{4.5}$$

Here, $T_v(t)$ denotes the temperature change of the vehicle over time. $T_s(t)$ represents the temperature of the transported stack of items at time t, and $T_v(0)$ represents the initial temperature of the vehicle. The constant k is the cooling factor of the vehicle

which has been determined through on-site experiments. The adapted formula enables the calculation of the vehicle's temperature T_v at time t, considering the simultaneous cooling of the transported item stack during transportation. This formula is expressed as follows:

$$T_v(t) = T_e + (T_s(0) - T_e)e^{-\frac{h}{c_v \cdot d \cdot \rho} \cdot t} + \left(T_v(0) - \left(T_e + (T_s(0) - T_e)e^{-\frac{h}{c_v \cdot d \cdot \rho} \cdot t}\right)\right) e^{-k \cdot t} \quad (4.6)$$

Normally, the formula requires an infinite recursion to compute the result. This is because the temperature of the transported items is, in turn, influenced by the changing vehicle temperature. However, experimental results have shown that a double recursive application of the formula only leads to an improvement in accuracy of just 0.4% (see Table 4.3). However, this is accompanied by a more than twofold increase in the calculation time of the formula. Due to the small difference in resulting temperature and because more than two recursive calls are not viable in practice due to the computational complexity, we call the recursive function once.

t	**no recursive call in °C**	**single recursive call in °C**	**double recursive call in °C**	**Δ no vs. single in °C**	**Δ single vs. double in °C**	**Δ no vs. single in %**	**Δ single vs. double in %**
100	118.49	104.77	105.19	13.72	0.42	11.58	0.40
200	191.45	147.01	147.64	44.44	0.63	23.21	0.43
300	245.50	163.79	164.48	81.71	0.69	33.28	0.42
400	285.55	165.74	166.42	119.81	0.68	41.96	0.41
500	315.21	159.45	160.08	155.76	0.63	49.41	0.40

Table 4.3.: Analysis of the calculation differences of temperature in a heating vehicle at 20°C environment temperature during the transportation of a 400°C hot stack of items, i.e., steel slabs. The application of NLC is compared between a non-recursive, single-recursive and double-recursive formulation and documented for different points in time.

The calculation shown in Table 4.3 is based on the assumption that the vehicle has an initial temperature of 20°C, the transported stack of items has an initial temperature of 400°C, and the environmental temperature measures 20°C.

Figure 4.1 illustrates that even the one-time recursive formula results in a significant improvement in the accuracy of predicting the temperature. The brown data series illustrates a vehicle that transports a stack of items that measures 600°C, whereby the cooling of the transported item stack is not taken into account. This leads to a perpetual heating of the vehicle, which in theory approximates 600°C itself. However, the assumption of a continuous rise in temperature, as shown in the brown data series, is unrealistic, so the cooling of the item stack has to be considered, as shown in the blue data series. The combination of both, the simultaneous heating of the vehicle and cooling of the item stack, is shown in the violet data series.

According to NLC, the rate of temperature change depends on the temperature difference between the objects, which is illustrated by the violet data series. In our case, it specifies that the vehicle heats up very quickly, especially at the beginning when the temperature differences between the cold vehicle and the transported hot item stack are very high. As soon as the temperature of the items decreases, the temperature of the vehicle for this transport reaches a local maximum, whereupon it cools down further together with the transported item stack.

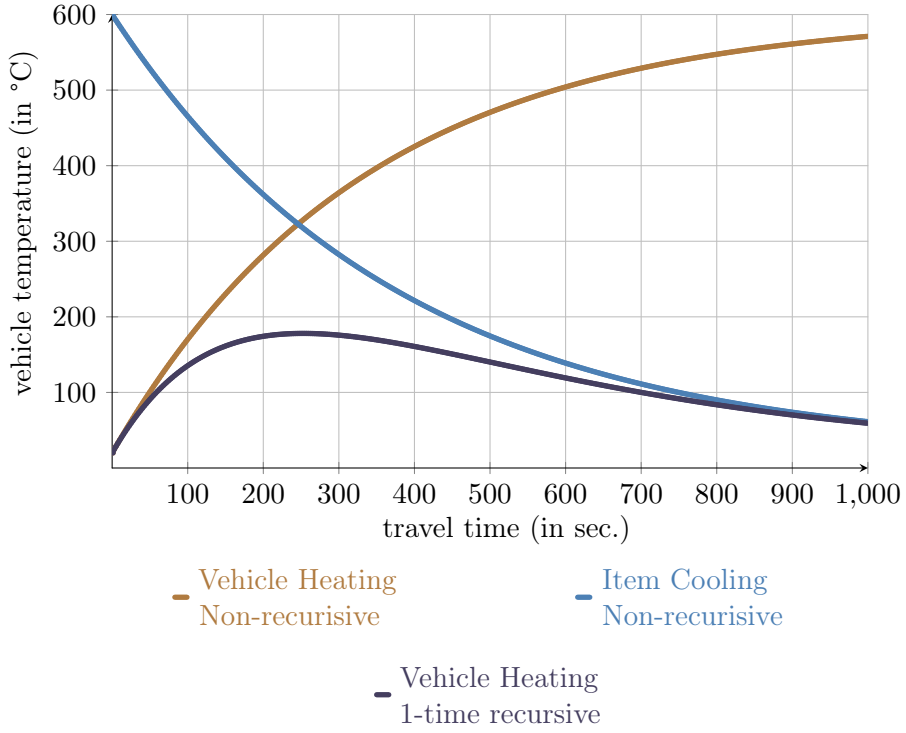


Figure 4.1.: The blue data series illustrates the cooling of the transported items according to Equation 4.4. The brown data series illustrates the continuous heating of the vehicle, which is impacted by the transported items and calculated by means of the non-recursive formulation of NLC (Equation 4.5). The violet data series presents the formula used in the *Vehicle Temperature Prediction Method* (Equation 4.6), which shows the temperature development of a vehicle over time.

In order to address the trade-off between practicality and accuracy, an endeavour was undertaken to incorporate the CHT or COHT among transported items. However, this was not further pursued due to the resource-intensive nature of the calculations involved and various other physical effects such as the Leidenfrost Effect caused by the accumulation of condensation on cold items or the Oxidation Protection Effect, which describes the formation of an oxide layer on the surface of a material that protects it from

heat (Ashby et al., 2019; Lide, 2003; Tuoliken et al., 2021). The motivation behind this exploration was centred on a pragmatic approach, emphasizing computational efficiency. As a result, the different thermal conductivity between the items themselves and the environment and their environmental temperature was also considered.

For items of different sizes, an exact calculation of the contact areas between the items would be required, for which different thermal material conductivity properties are used. This thermal interplay is shown in Figure 4.2. Equations 4.7 (Top Item) & 4.8 (Bottom Item) illustrate that even two items lying on top of each other considerably lengthen the formula and, therefore, the calculation time, which ultimately leads to the decision not to implement this in practice. Nevertheless, a brief introduction to the notations and how the formula for the CHT between two items works will be given in the following.

4.2.1. Heat Transfer between Items

In Equations 4.7 & 4.8, the temperatures of two items, which are stacked on top of each other, are calculated based on their contact surfaces to each other and the cooling by the environmental air temperature. Equation 4.7 describes the top item, while Equation 4.8 refers to the bottom item. In contrast to Equation 4.5, the temperature is calculated using the respective surfaces. A_t^i, A_b^i, A_r^i describe the area of the top, bottom and remaining sides of item i respectively, whereas A_a^i describes the total area of item i. In this equation, the values 1 and 2 were chosen for i, with the former representing the top item and the latter the bottom item. All other variables are identical to Equation 4.5. Figure 4.2 shows these equations graphically, with the green and violet lines representing the temperature of the upper and lower items, respectively, if they were cooling alone without any dependence on each other. The pink and blue lines, in turn, show the temperature curve when these two items are on top of each other, which shows the dependency quite well. As these deviations are only minor, mainly because items are transported for longer periods on average, we have decided to disregard the calculation using this extended formula.

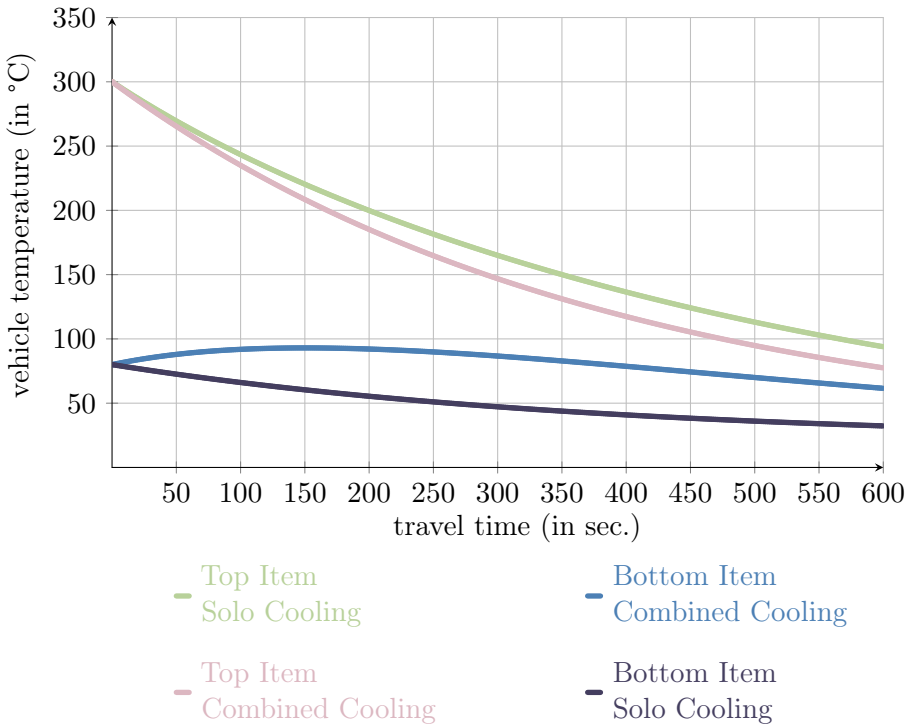


Figure 4.2.: The green and violet data series represent two items that cool down independently of one another. In contrast, the pink and blue data series show the temperature curve of two items that lie on top of each other, whereby the cooler item briefly heats up and the hotter item cools down more quickly due to the stronger heat conduction (steel, compared to gravel or air).

$$
\begin{aligned}
T_{\mathrm{Top}}(t) &= \frac{T_e(A_b^2+A_r^2)A_b^1(1-e^{-\frac{h}{\rho H_1 c_v}t})}{A_a^1A_a^2-A_t^2A_b^1(1-e^{-\frac{h}{\rho H_1 c_v}t}-e^{-\frac{h}{\rho H_2 c_v}t}(1-e^{-\frac{h}{\rho H_1 c_v}t}))} \\
&+ \frac{A_b^1e^{-\frac{h}{\rho H_2 c_v}t}(T_s^2(0)(A_t^2+A_b^2+A_r^2)-T_e(A_b^2+A_r^2))}{A_t^1A_t^2-A_t^2A_b^1(1-e^{-\frac{h}{\rho H_1 c_v}t}-e^{-\frac{h}{\rho H_2 c_v}t}(1-e^{-\frac{h}{\rho H_1 c_v}t}))} \\
&+ \frac{T_e(A_t^1+A_r^1)A_a^2+e^{-\frac{h}{\rho H_1 c_v}t}(T_s^1(0)(A_t^1+A_b^1+A_r^1)-T_e(A_t^1+A_r^1))A_t^2}{(A_t^2-A_t^2e^{-\frac{h}{\rho H_2 c_v}t})+T_e(A_b^2+A_r^2)+e^{-\frac{h}{\rho H_2 c_v}t}(T_s^2(0)(A_t^2+A_b^2+A_r^2)-T_e(A_b^2+A_r^2))}
\end{aligned}
\tag{4.7}
$$

$$
\begin{aligned}
T_{\mathrm{Bottom}}(t) &= \frac{T_e(A_b^2+A_r^2)(A_b^1-A_b^1e^{-\frac{h}{\rho H_1 c_v}t})+A_b^1e^{-\frac{h}{\rho H_2 c_v}t}(T_s^2(0)(A_t^2+A_b^2+A_r^2)-T_e(A_b^2+A_r^2))}{A_a^1A_a^2-A_t^2(A_b^1-A_b^1e^{-\frac{h}{\rho H_1 c_v}t}-e^{-\frac{h}{\rho H_2 c_v}t}(A_b^1-A_b^1e^{-\frac{h}{\rho H_1 c_v}t}))(A_t^2-A_t^2e^{-\frac{h}{\rho H_2 c_v}t})} \\
&+ \frac{T_e(A_b^2+A_r^2)+e^{-\frac{h}{\rho H_2 c_v}t}(T_s^2(0)(A_t^2+A_b^2+A_r^2)-T_e(A_b^2+A_r^2))}{A_a^2} \\
&+ \frac{-A_b^1e^{-\frac{h}{\rho H_1 c_v}t}e^{-\frac{h}{\rho H_2 c_v}t}(T_s^2(0)(A_t^2+A_b^2+A_r^2)-T_e(A_b^2+A_r^2))}{A_a^2} \\
&+ \frac{T_e(A_t^1+A_r^1)A_a^2+e^{-\frac{h}{\rho H_1 c_v}t}(T_s^1(0)(A_t^1+A_b^1+A_r^1)-T_e(A_t^1+A_r^1))A_a^2}{A_a^2}
\end{aligned}
\tag{4.8}
$$

5. Solution Methods

This chapter discusses the solution methods, in particular the ALNS, which is described in Section 5.3.1. An in-depth discussion of the construction heuristic used, i.e. Cheapest Insertion Heuristic (CI), is provided whereby new approaches are explored, which are explained in more detail in Section 5.2.1. Furthermore, the Weighted Sum Approach (WSA) and the relativisation of the objective function component values are discussed, as described in Sections 5.1 & 5.1.2, respectively. In the context of the TD-CVRPDP, we will focus on its problem-specific solution method while also providing general information on heuristics and metaheuristics. Finally, Section 6.6 introduces a new heuristic, the so-called D-5R-LA-TS, which aims to tackle the dynamic variation of this problem.

5.1. Introduction to the Weighted Sum Approach

In the context of multi-objective VRPs, weighting the objective components is a widely used method. This method, also known as the Weighted Sum Approach (WSA), offers a flexible and powerful tool during multi-objective optimisations such as the TD-CVRPDP optimisation in this research. The WSA allows multiple objective criteria to be considered by assigning weights to these criteria that reflect their relative importance in the solution process.

It works by first defining the objective function components in accordance with which the problem is to be solved. These can be, for example, minimising total costs, minimising the number of vehicles or maximising customer satisfaction. Each objective function is then assigned a weighting that indicates its importance in relation to the other objectives.

To illustrate this with an example: Suppose a logistics company has to supply several customers while minimising transport costs and ensuring that all customers are supplied on time. The objective functions could be the minimisation of total costs (e.g. fuel costs, vehicle costs) and the minimisation of delays in delivery.

A weighting of 0.7 could be set for the minimisation of total costs and a weighting of 0.3 for the minimisation of delays. This means that costs contribute 70% and delays 30% to the overall solution. By adjusting these weightings, the priorities can be changed according to the needs and requirements of the company.

Once the weightings have been determined, the objective function components are weighted and totalled to obtain an overall solution. This overall solution is then found using optimisation algorithms or heuristics such as the *Greedy Algorithm*, *Genetic Algorithm* or ALNS.

An important application of the WSA was presented by Oyola and Løkketangen (2014), who found that this method can be effectively used to solve the Capacitated Vehicle

Routing Problem (CVRP) with route balancing. Not only is the overall cost minimised, but the utilisation of the vehicles is also evenly distributed, which leads to improved efficiency of the route plan. The WSA can offer an effective solution, particularly for companies with complex supply chains and variable delivery requirements.

Matl et al. (2018) emphasise in their study the importance of a balanced workload in VRPs, especially when workers or resources have to be distributed across several vehicles. They also use the WSA to analyse and show how different weightings can affect workload distribution.

Petropoulos et al. (2023) mention the WSA as a common method for applied problems and refer to earlier surveys by Marler and Arora (2004) and Wiecek et al. (2016). These works provide a comprehensive overview of different methods of multi-objective optimisation and show how the WSA can be used in practice.

One of the strengths of the WSA is its adaptability to different problems and objectives. By adjusting the weights of the individual objective criteria accordingly, the WSA can be adapted to the specific requirements and priorities of the user. This flexibility makes the WSA an attractive option for modelling and solving different types of multi-objective problems like the TD-CVRPDP in this research.

Another advantage of the WSA is its efficiency in solving complex problems. By transforming the multi-objective problem into a single objective problem, a single optimal solution can be calculated. This facilitates the implementation of algorithms and enables high-quality solutions to be found quickly.

Although the WSA offers many advantages, it is important to note that the WSA has its limitations. Visée et al. (1998) and Oyola and Løkketangen (2014) point out that this method does not find solutions in concavities of the Pareto-front, which means that not all efficient solutions can be found and WSA may not be able to identify all potentially optimal solutions, especially those that lie in the unsupported regions of the solution space.

Osman and Laporte (1996) have compiled a comprehensive bibliography on metaheuristics, which also contains relevant literature on the WSA. This source provides a historical overview of the development of metaheuristics in the field of optimisation and illustrates the importance of the WSA as one of the first and most frequently used methods for solving multi-objective optimisation problems in the VRP context.

The WSA enables multi-objective optimisation, whereby various aspects, such as transport and logistics processes, can be flexibly taken into account and respond to changing requirements and conditions, which offers an effective way of solving complex problems.

A comparable approach that could also be used for this type of multi-objective optimisation problem is the Epsilon Constraint Method (ECM). The ECM is based on optimising a main objective, while the other objectives are treated as constraints that are not to exceed certain limit values (epsilons "ϵ") (Deb et al., 2002; Laumanns et al., 2002). This enables precise control of the compromise solutions by adjusting the epsilons. Although this method allows a higher transparency between the objectives' trade-offs, it can be challenging due to its complexity and dependence on the chosen optimisation

technique.

In contrast, the WSA uses a simpler and more flexible strategy in which the objective functions are combined into a single function by assigning appropriate weights to them (Deb et al., 2002). This allows efficient optimisation of the aggregated objective function, especially if the weights are chosen carefully.

Studies and comparisons between these two approaches have shown that the WSA is often less computationally expensive and provides a good quality solution, especially if the weightings are well chosen (Deb et al., 2002). Moreover, Matl et al. (2018) reveals that the WSA is preferably used for real-world applications.

The choice between these two approaches often depends on various factors, including the complexity of the problem, the available resources and the preferences of the decision-makers.

5.1.1. Application of the Weighted Sum Approach in the TD-CVRPDP

For multi-objective optimisation, the WSA is often used, which makes it possible to combine several objective functions by assigning a weight to each objective component. These weights can be dynamically adjusted to optimise the balance between the objectives during the search process (Matl et al., 2018).

This research aims to simultaneously optimise the *travel time*, the *ton travel time* and the *vehicle temperature*. The aggregate objective function combines these objectives, as explained in Chapter 2. We have chosen the WSA as it has proven to be extremely flexible and efficient. This approach makes it possible to combine the three objectives into a single objective using weights. As shown in the Equation 2.2, the objective function components *travel time*, *ton travel time* and *vehicle temperature*, are multiplied with their respective weight coefficients α, β and γ. The following rule applies:

$$\alpha + \beta + \gamma = 1 \tag{5.1}$$

In contrast to bi-objective optimisation, in which only two objectives are optimised and the weightings can already be determined by specifying a single weight, tri-objective optimisation requires two weights to be specified, while the third must be calculated.

In bi-objective optimisation, the weight for α is determined, whereby the weight for β is determined by $1 - \alpha$. In a tri-objective optimisation, both α and β must be determined, while γ can be calculated. Therefore, the following rules apply:

$$\alpha + \beta \leq 1 \tag{5.2}$$

$$\gamma = 1 - \alpha - \beta \tag{5.3}$$

After all weightings have been determined, all three objective components are combined into a single objective which is expressed in a simplified version in Equation 5.4.

$$Z = \alpha \times \textit{travel time} + \beta \times \textit{ton travel time} + \gamma \times \textit{vehicle temperature} \tag{5.4}$$

Despite the weighting of these objectives, the optimisation method is not necessarily fair because of the imbalance, which is due to the different value scales of the individual objective function components. As a result, objective component values on a large scale dominate those on a smaller scale. To overcome this inherent imbalance, the objective function component values need to be relativised. This approach is explained in more detail in the following section.

5.1.2. Relativisation of the Objective Component Values

When using the WSA for multi-objective optimisation, it is often necessary to relativise the objective function values. This is necessary to ensure that the different objective function values, which may have different units of measurement or be on different scales, can be compared appropriately. Without relativisation, objectives with larger scales could dominate and neglect other objectives, which could lead to unequal priorities.

The relativisation of the objective function component values involves converting these values into a common range of values that allows them to be compared with each other. This is often achieved by scaling one of the objective function values to the other one. Thereby, the median of the first objective solution range is divided by the median of the second objective solution range, as can be seen in Equations 5.5 & 5.6. This process ensures that all objectives are treated equally and none of them are overly weighted (Roy and Mousseau, 1996; Zitzler and Thiele, 2003).

$$\textbf{Relativisation Value} = \frac{\textit{Solution median of first objective component}}{\textit{Solution median of second objective component}} \tag{5.5}$$

$$\textbf{846.15} = \frac{550}{0.65} \tag{5.6}$$

To illustrate the relativisation of the objective function values, we consider a simple example of a multi-objective optimisation problem with two objective function components: Minimising cost (C) and loss in efficiency (E). Assume that the cost is around the median 550, while the loss in efficiency is around the median 0.65. Relativisation enables a fair comparison between the costs (C) and the loss in efficiency (E) by creating a uniform basis.

Once the relativisation value has been calculated, it only needs to be multiplied by the objective function component that is to be relativised. In this example, the loss in efficiency E is relativised to the costs C, which means that the relativisation value 846.15 is multiplied by the objective function value of the loss in efficiency 0.6 (for the first solution in Table 5.1) to obtain the relativised value E', which is 507.69. As only one objective function value is relativised to the other, the other objective function value remains unchanged, which in this case means $C = C'$.

Alternatively, there is also the possibility of relativising it the other way around, which means that the costs are relativised to the loss in efficiency. This is shown in

of Solution	**Cost** C	**Loss in efficiency** E	**Relativisation Value**	**Relativised Costs** (C')	**Relativised Loss in efficiency** (E')
1	200	0.6	846.15	200	507.69
2	800	0.5	846.15	800	423.08
3	100	0.8	846.15	100	676.92

Table 5.1.: Example of three different solutions, where two objective function component values are relativised. The loss in efficiency E is relativised to the costs C, resulting in the relativised loss in efficiency value E' and the costs C'.

Equation 5.7 & 5.8 and Table 5.2. In general, it does not matter which values are relativised and which values are relativised to, but it is often useful to relativise values to a value which has an understandable unit of measurement, as this provides a much better understanding of the results.

$$\textbf{Relativisation Value} = \frac{\textit{Solution median of second objective component}}{\textit{Solution median of first objective component}} \tag{5.7}$$

$$\mathbf{0.001\bar{1}8} = \frac{0.65}{550} \tag{5.8}$$

# of Solution	**Cost** C	**Loss in efficiency** E	**Relativisation Value**	**Relativised Costs** (C')	**Relativised Loss in efficiency** (E')
1	200	0.6	$0.001\bar{1}8$	$0.23\bar{6}3$	0.6
2	800	0.5	$0.001\bar{1}8$	$0.9\bar{4}5$	0.5
3	100	0.8	$0.001\bar{1}8$	$0.1\bar{1}8$	0.8

Table 5.2.: Example of three different solutions, where two objective function component values are relativised. The cost C is relativised to the loss in efficiency E resulting in the relativised cost value C' and the loss in efficiency E'.

Relativisation enables the possibility to compare two different objective function values and to make a decision that meets both objectives without one dominating the other.

Relativising the objective function values is an important step in applying the WSA to create a fair basis of comparison for the different objectives. Once the objective function values have been relativised, the WSA can be applied to calculate a combined objective function that allows a solution to be evaluated based on the chosen weights. In the following, we will explain and illustrate how relativisation is applied using a specific problem.

5.1.2.1. Application of the Relativisation in the Context of Weighted Sum Approach

In this research, *travel time*, *ton travel time* and *vehicle temperature*, which are the three objective components, are optimised simultaneously. As *travel time* is measured in seconds, *ton travel time* in ton-seconds and *vehicle temperature* in °C, all three objective function components have differently sized values.

# of Solution	Travel Time (in sec.)	Vehicle Temperature (in °C)	Ton Travel Time (in to-sec.)	Relativisation Value Ton Travel Time	Relativisation Value Vehicle Temperature	Relativised Travel Time	Relativised Ton Travel Time	Relativised Vehicle Temperature
1	1,532	243.15	341,798.32	0.0045	19.23	1,532	1,553.63	4,675.96
2	3,468	97.31	876,213.10	0.0045	19.23	3,468	3,982.79	1,871.35
3	2,231	122.40	534,785.65	0.0045	19.23	2,231	2,430.84	2,353.85

Table 5.3.: Example of three different solutions, where three objective function component values are relativised. The *ton travel time* and *vehicle temperature* are relativised to the *travel time*, resulting in the relativised *ton travel time* and *vehicle temperature* values.

In this case, these values must be relativised in order to create a comparable basis for the WSA. In the following example, we assume that the median of the *travel time* is determined at 2500 seconds, the *vehicle temperature* at 130°C and the *ton travel time* at 550,000. Two of these three objective function values must be relativised to the third. In this example, the best option would be to relativise the *ton travel time* and the *vehicle temperature* to the *travel time* to achieve values for these two objective function components that are approximately equal to the *travel time*, as the unit of measure *travel time* is easy to grasp. In Equation 5.9, the *ton travel time* is relativised to the *travel time*, whereby the median of the *travel time*, which is 2500 seconds, is divided by the median of the *ton travel time*, which is 550,000 ton-seconds, resulting in a relativisation value for the *ton travel time* of 0.0045. The medians are obtained from the repeated calculation of an instance, i.e., through CI, and evaluated from these objective values. In a similar way, the relativisation value for the *vehicle temperature* is calculated by dividing the median of *travel time* by the median of *vehicle temperature*, resulting in 19.23 as the relativisation value for the *vehicle temperature*. This is shown in Equation 5.10.

$$0.00\dot{4}\dot{5} = \frac{2500 \text{ sec.}}{550,000 \text{ to-sec.}} \tag{5.9}$$

$$19.2307 = \frac{2500 \text{ sec.}}{130 \text{ °C}} \tag{5.10}$$

In Table 5.3, three different solutions are relativised, whereby the *ton travel time* and the *vehicle temperature* are always relativised to the *travel time*. This relativisation of the first solution reveals that the *ton travel time* of 341,798.32 results in 1,553.63 in the eighth column by multiplying it by its corresponding relativisation value $0.00\bar{45}$.

Accordingly, the *ton travel time* value could now be interpreted as having an equivalent *travel time* of $1,553.63$, which in this solution means that it is approximately about the same, as the actual *travel time*. The result of this interpretation is not surprising since, as already discussed in previous sections, minimising the *travel time* leads to a minimised *ton travel time*. The same type of interpretation can also be used for the *vehicle temperature*, whereby a relativised *vehicle temperature* of 243.15 ton-seconds results in a value of 4.675.96 seconds, which would suggest that the vehicle temperature is three times as bad as the travel time, which is again not surprising, as the *vehicle temperature* objective component is opposed to the *travel time* and *ton travel time* objective components.

Based on these relativised values, the *quality value* used in this research can now be calculated. As discussed in Sections 5.1.1 & 2, this is calculated using the WSA. This is shown in Equation 5.11, where the values α, β and γ each represent the weight coefficients of the individual objective function components. If we now assume that the weightings are equally distributed (even if it cannot be true for the first solution in Table 5.3, since the *travel time* and the relativised *ton travel time* are significantly better than the relativised *vehicle temperature*), i.e., α, β and γ each have a weighting of 33.33%, a *quality value* of $2,587.20$ would result, which is represented in Equation 5.12.

$$\textbf{Quality value} = \alpha \times 1,532 + \beta \times 1,553.63 + \gamma \times 4,675.96 \tag{5.11}$$

$$\textbf{2,587.20} = 0.3\bar{3}\% \times 1,532 + 0.3\bar{3}\% \times 1,553.63 + 0.3\bar{3}\% \times 4,675.96 \tag{5.12}$$

As far as the relativisation during the CI is concerned, separate relativisation values must be calculated for each number of inserted requests. This is because, for example, the *travel time* for a single inserted request can be between 1 and 100 seconds, whereby the *vehicle temperature* can already reach a value of 10 to 200°C. If, for example, 20 requests are inserted, the *travel time* could be between 1200 and 3000 seconds, whereby the *vehicle temperature* can still be between 10 to 200°C. The same also applies to the *ton travel time* objective component. In order to guarantee a fair basis for the selected weights in each iteration, i.e. the number of requests inserted, within the framework of the WSA, a new relativisation value is calculated for each iteration.

The quality value calculated with relativised values and the WSA can be used for a fair optimisation, which is a minimisation in this research. The examples demonstrate that the relativisation of objective function component values is an important factor in multi-objective optimisation, which is indispensable without it.

5.2. Introduction to Heuristics

Heuristics are proven strategies or rules of thumb that are used in decision-making processes to quickly achieve an acceptable solution without carrying out a comprehensive analysis. They are often based on experience, intuition and simplified assumptions about the problem. Heuristics are particularly important in areas such as transport, logistics and production, where complex optimisation problems such as the Travelling Salesman

Problem (TSP), the VRP and similar problems need to be overcome (Petropoulos et al., 2023).

In the transport and logistics sector, heuristics are often used to solve route planning, cargo space optimisation, vehicle allocation and similar problems. A well-known example is the VRP, a simplified version of this research, in which a fleet of vehicles is to be used to deliver to a number of customers at minimum cost or vehicle kilometres.

A simple heuristic to solve the VRP is, e.g., the *Nearest Neighbour Heuristic*, where each vehicle drives from its current location to the nearest unvisited customer destination. This heuristic is quick and easy to apply but does not always lead to an optimal solution. In some cases, it can lead to inefficient routes or uneven vehicle utilisation (Drozdowski, 2009; Golden et al., 2008).

In production planning, heuristics are used to solve complex problems such as machine scheduling, job sequence optimisation and inventory control. A common problem is *Job Shop Scheduling*, where a set of jobs must be scheduled to a set of machines, where each machine can only process one job at a time.

A simple heuristic for job shop scheduling is the *Earliest Due Date Heuristic*, where jobs are prioritised based on their earliest completion time. Similar to transportation problems, heuristics are quick and easy to apply but do not always lead to optimal use of machine capacity or minimisation of lead times (Baker and Trietsch, 2019; Pinedo, 2016).

Heuristics have both advantages and disadvantages. They are fast, easy to use and often require less computing power than exact optimisation methods. However, they can lead to sub-optimal solutions and offer no guarantee for the quality of the results. Nevertheless, heuristics are indispensable in many practical applications, as they help to solve complex problems in a reasonable amount of time.

More advanced approaches, such as metaheuristics, also exist alongside simple heuristics, which use iterative methods to strive for an optimal solution without searching the entire solution space. An example of a metaheuristic is the *Genetic Algorithm*, which uses principles of natural selection and genetic recombination to improve solutions and search for global optima. The application of metaheuristics makes it possible to find good solutions even in complex optimisation problems without having to perform a complete search (Glover and Kochenberger, 2003).

Section 5.3 provides a comprehensive explanation of the use of metaheuristics in this research and an introduction to metaheuristics in the fields of transport, logistics, and production. The section elaborates on how metaheuristics are employed to optimise complex problems.

5.2.1. Introduction to the Cheapest Insertion Heuristic

The Cheapest Insertion Heuristic (CI), which is described in the literature as a fundamental method for solving route optimisation problems, is based on the concept of gradually inserting vertices into a route, choosing the cheapest insertion position in each case (Clarke and Wright, 1964). This greedy heuristic has proven to be a popular choice as it is comparatively easy to implement and often leads to satisfactory results (Toth and Vigo, 2014).

The CI typically starts with an empty route and successively adds vertices to form a feasible solution (Laporte, 1992b). The selection process for the next vertex to be inserted is based on minimising the cost change caused by inserting this vertex into the current route (Ropke and Pisinger, 2006). Various criteria, such as the distance between the vertices, the time or other cost measures, are taken into account.

The vertex that causes the lowest cost change is selected and inserted into the route. This process is repeated iteratively until all nodes are inserted into the route or a termination criterion is met. An TSP and a VRP example can be found in Appendix A.1 and Appendix A.2 respectively. The CI is a simple yet effective method for achieving a satisfactory solution for many problem scenarios (Golden et al., 2008). It provides a good starting point for optimising routes in various application areas, including logistics, transportation and supply chain management.

The Cheapest Insertion Sequential Insertion Heuristic (CI-SIH) is a further development of the original heuristic, which is also known as Cheapest Insertion Parallel Insertion Heuristic (CI-PIH). The CI-SIH starts similarly to the CI-PIH by creating an empty route. The difference to the CI-PIH is that a list of requests or customers is initialised, which is sorted according to various criteria. This could, for example, be random, with increasing or decreasing distance from the depot or similar criteria. According to the order of requests in this sorted list, vertices are then successively added to form a feasible solution (Ropke and Pisinger, 2006). The selection of the next vertex to be inserted into the route is based on minimising the cost change caused by inserting this vertex into the current route.

To exemplify the CI-SIH, let's look at an example from the parcel delivery d_r sector. Suppose a courier service has to deliver 5 parcels to different customers, where the transport costs between the customer locations are known. These customers are now sorted within a sequential list according to a certain criterion. This could be, for example, the distance from the courier's depot to the customers in ascending or descending order, according to a priority criterion or similar criteria. The courier service starts with an empty route and then gradually selects the customer, starting with the first item of the sequential list and choosing a position in the route that causes the least additional cost change to the current route. This process continues until all customers are inserted into the route.

The CI-SIH thus offers an efficient and simple approach to optimising routes in various application areas, including logistics, transport and supply chain management (Golden et al., 2008). How the CI-SIH is applied in this research is explained in detail in the next section (Section 5.2.1.1).

5.2.1.1. Application of the Cheapest Insertion Heuristic in the TD-CVRPDP

The CI-SIH, used within a sequential framework, is a state-of-the-art algorithmic approach that aims to gradually integrate candidates, i.e., requests, into the route planning by selecting the cheapest insertion position for each candidate that is determined by a metric measure, which in this research is the increase in *quality value*, discussed in Sections 5.1 & 2. Two basic variants for the set-up of the sequential insertion framework

are considered, which are discussed in more detail in Sections 5.2.1.1 (*Single Insertion*) & 5.2.1.1 (*Pairwise Insertion*). In the first case, a request is divided into its pickup p_r and delivery d_r, whereby both parts can be inserted individually. In the second case, the established pairwise insertion of requests is set, i.e., an insertion step features the collective insertion of the pickup p_r visit and delivery d_r visit of a request. The aim of using both variants is the exploration of a larger solution space. One insertion step constitutes that, in case of the *Single Insertion*, one pickup or delivery is inserted, and, in case of *Pairwise Insertion*, one pickup-and-delivery-pair is inserted. As with conventional pairwise insertion, a sequence of pickup-and-delivery-pairs is first created, where the creation is discussed in Section 5.2.1.1 (*Insertion Sequence*), whereby pickups and deliveries are considered separately.

Insertion Sequence The insertion sequence is a central concept in the CI-SIH, where an empty sequence list is first initialised, which is then filled with yet unfulfilled requests, more precisely, the requests' pickups and deliveries. In the specific case of the Cheapest Insertion Single Insertion Heuristic (CI-SiIH), both pickup p_r and delivery d_r of unserved requests $R^{unserved}$, are inserted into this list as separate elements, meaning that per insertion step, i.e., iteration, only a pickup p_r or a delivery d_r is inserted. In contrast, in the case of Cheapest Insertion Pairwise Insertion Heuristic (CI-PaIH), the pickup p_r and the delivery d_r are inserted together during one insertion step. These elements are then sorted according to a chosen criteria, with the first element in the list being selected for further processing. Depending on the sequence criteria, different solution spaces can be discovered. Table 5.4 provides an outline of the criteria used in this research for sorting during the construction of the initial ALNS solution, which is further discussed in Section 5.3.2.

As part of this research, the sequence list in the CI-SIH is sorted according to 5 different criteria. These include a random sorting supported by using cryptographically strong random numbers, which ensures better randomness, sorting according to pickup p_r or delivery d_r, the weight q_r, the temperature t_r or the height u_r of the requests. The latter three criteria can be sorted into increasing, decreasing, or mixed order. Mixed means the following: the first element in this sequence list is always the maximum value of a criterion, meaning the highest weight, temperature or height of all requests. The second element of the list would be the minimum value of a criterion, meaning the lowest weight, temperature or height of all requests. The third element would be the second-highest value of a criterion, followed by the fourth element, which represents the second-lowest value of a criterion, and so on. As far as pickups or deliveries are concerned, these can be sorted according to whether all pickups (1^{st} Pick.) or deliveries (1^{st} Deli.) are inserted first. In the case of mixed, it would mean that a pickup p_r and then a delivery d_r are added alternately. However, as there are no maximum or minimum values within this criterion, the sequence is randomly shuffled first before sorting, providing different starting points. All combinations together result in 26 different sequence types, which can be seen in Table 5.4. The best-solution-probabilities, in this case, are experimental values that have been gathered from the use of ALNS. With the ALNS algorithm, an

Sorting Criterion	Insertion Method	Sorting	Best Solution Prob. (in %)	Sorting Criterion	Insertion Method	Sorting	Best Solution Prob. (in %)
Random	CI-SiIH		1.85	Temp.	CI-SiIH	Incr.	2.30
Random	CI-PaIH		5.72	Temp.	CI-SiIH	Decr.	2.53
Pick. | Deli.	CI-SiIH	1st Pick.	2.20	Temp.	CI-SiIH	Mixed	2.43
Pick. | Deli.	CI-SiIH	1st Deli.	1.95	Temp.	CI-PaIH	Incr.	4.89
Pick. | Deli.	CI-SiIH	Mixed	1.81	Temp.	CI-PaIH	Decr.	5.10
Pick. | Deli.	CI-PaIH	1st Pick.	5.84	Temp.	CI-PaIH	Mixed	5.05
Pick. | Deli.	CI-PaIH	1st Deli.	5.68	Height	CI-SiIH	Incr.	2.35
Pick. | Deli.	CI-PaIH	Mixed	5.68	Height	CI-SiIH	Decr.	2.33
Weight	CI-SiIH	Incr.	2.56	Height	CI-SiIH	Mixed	2.74
Weight	CI-SiIH	Decr.	2.70	Height	CI-PaIH	Incr.	5.39
Weight	CI-SiIH	Mixed	2.81	Height	CI-PaIH	Decr.	5.48
Weight	CI-PaIH	Incr.	5.08	Height	CI-PaIH	Mixed	5.58
Weight	CI-PaIH	Decr.	4.97				
Weight	CI-PaIH	Mixed	4.97				

Table 5.4.: List of 26 sequence types used to sort a sequence list during construction of the initial ALNS. Each sequence type differs with regard to a sorting criterion, insertion method, and sorting order. The best-solution-probabilities are constant values that were obtained through computational experiments. The best-solution-probabilities in this table are only statistical values with which probability a sequence type leads to the best solution among all sequence types, which is then used as the initial solution of the ALNS. These probabilities are obtained by running each sequence type 10000 times and evaluating their performance.

initial solution, i.e., the first incumbent solution, is required first, which is generated by a so-called construction heuristic. In this research, the CI heuristic is utilised for this purpose. However, instead of generating a single solution using only one of the 26 insertion sequences of Table 5.4, all sequence types are applied, generating 26 different solutions, whereas the best solution of those is selected as the initial solution for the ALNS algorithm. The best-solution-probabilities were determined from this, indicating which sequence types lead to the best solution among all other sequence types with which probability.

Interestingly, CI-PaIH performs better on average than CI-SiIH. The reason for this is that with CI-SiIH, capacities have to be artificially kept free during the insertion of the requests, which means that the solution space is more restricted. This is explained in more detail in Section 5.2.1.1 (*Single Insertion*). In this research, however, only the best solution from all solutions calculated with the 26 different sequence types is chosen for the initial solution of the ALNS, which means that solutions calculated by CI-SiIH are

selected rarely. Since these 26 different solutions do not take a lot of calculation time, as they are calculated once per instance only, it was decided to use all 26 sequence types for the construction of the initial solution. This approach leads to more diversity and better starting points for ALNS. For the ALNS *Repair* operator, however, it was decided to reduce these sequence types to the 13 most promising sequence types, as this significantly speeds up the calculation process without noticeably reducing the quality of the solution. As these 13 sequence types are calculated up to $10,000$ times per instance permutation[1] in the ALNS *Repair* step, a saving of 10 seconds per iteration already means an acceleration of up to 27 hours. Further details on ALNS and its application are discussed in Section 5.3.2.

Sorting Criterion	**Insertion Method**	**Sorting**	**Select. Prob. (in %)**	**Sorting Criterion**	**Insertion Method**	**Sorting**	**Select. Prob. (in %)**
Random	CI-SiIH		0.33	Temp.	CI-SiIH	Incr.	0.39
Random	CI-PaIH		6.30	Temp.	CI-SiIH	Decr.	0.36
Pick. | Deli	CI-SiIH	1st Pick.	0.31	Temp.	CI-SiIH	Mixed	0.36
Pick. | Deli	CI-SiIH	1st Deli.	0.50	Temp.	CI-PaIH	Incr.	5.41
Pick. | Deli	CI-SiIH	Mixed	0.28	Temp.	CI-PaIH	Decr.	6.14
Pick. | Deli	CI-PaIH	1st Pick.	8.00	Temp.	CI-PaIH	Mixed	8.25
Pick. | Deli	CI-PaIH	1st Deli.	5.36	Height	CI-SiIH	Incr.	0.44
Pick. | Deli	CI-PaIH	Mixed	10.52	Height	CI-SiIH	Decr.	0.42
Weight	CI-SiIH	Incr.	0.39	Height	CI-SiIH	Mixed	0.58
Weight	CI-SiIH	Decr.	0.44	Height	CI-PaIH	Incr.	6.25
Weight	CI-SiIH	Mixed	0.39	Height	CI-PaIH	Decr.	3.61
Weight	CI-PaIH	Incr.	4.78	Height	CI-PaIH	Mixed	5.66
Weight	CI-PaIH	Decr.	17.96				
Weight	CI-PaIH	Mixed	6.58				

Table 5.5.: List of the most promising 13 out of 26 sequence types used during the *Repair* of the ALNS. Each sequence type differs with regard to a sorting criterion, insertion method, and sorting order. The selection probabilities are constant values that were obtained through computational experiments.

Single Insertion The CI-SiIH is one of two methods of the CI-SIH. A pickup p_r and a delivery d_r of an unserved request $R^{unserved}$ are inserted separately, meaning that either a pickup p_r or a delivery d_r is inserted in each insertion iteration. Algorithm 1 provides a detailed pseudo code of the CI-SiIH process. It starts with the input of all unserved requests $R^{unserved}$, the chosen sequence type st (Section 5.2.1.1) and the initialisation of an empty solution S, which can hold a set of routes (strictly speaking that is one route for each used vehicle K), and an empty sequence list Φ. All used vehicles K are then

[1] One permutation delivers one solution given a particular objective component weighting during the WSA.

iterated through in line 3. In line 4, for each vehicle $|K|$ an empty route is initialised, where each of them is assigned with 2 vertices, namely, depot vertex 0, determining the start and end of a route (v_0, v_0). In lines 5, 6 and 7, all unserved requests $R^{unserved}$

Algorithm 1 Cheapest Insertion (Single Insertion)

1: **Input** $R^{unserved}$, st ▷ unserved request set and sequence type
2: $S \leftarrow \{\ \}$; $\Phi \leftarrow \{\ \}$ ▷ initialize solution and sequence list
3: **for each** $k \in K$ **do**
4: initialise $route^k$ starting and ending at vertex 0 (v_0) ▷ v_0 represents the depot
5: **for each** $r \in R^{unserved}$ **do**
6: Φ.insert(p_r)
7: Φ.insert(d_r)
8: Φ.sort(st)
9: **while** $\Phi \neq \varnothing$ **do**
10: $pos_{best} \leftarrow \varnothing$
11: $q_{best} \leftarrow$ *big-M*
12: **for each** $k \in K$ **do**
13: **for each** insertion pos in $route^k$ **do**
14: **if** pos is feasible **then** ▷ pos refers to either pos^{pr} or pos^{dr}
15: compute quality value q_{pos}
16: **if** $q_{pos} < q_{best}$
17: $q_{best} \leftarrow q_{pos}$ and $pos_{best} \leftarrow pos$
18: **else** Φ.pushback(p_r or d_r)
19: **if** $pos_{best} \neq \varnothing$ **then**
20: S.insert(p_r or d_r).at(pos_{best})
21: Φ.remove(p_r or d_r)
22: **Output** S

are iterated through, and the respective pickup and deliveries are inserted as separate elements in the initially empty sequence list Φ. After the sequence list Φ has been filled, the chosen sequence type st is used in line 8 to sort this list according to this. In the following, all steps are carried out until the sequence list Φ is empty (line 9). In lines 10 and 11, a variable "pos_{best}" is initialised with `null` ($\varnothing$) and the best quality value q_{best} is initialised with *big-M*. In lines 12 to 18, it is tested for each vehicle K which insertion position of the first element from the sequence list Φ is the best. This is determined by the quality value q at the best insertion position pos_{best} of a pickup p_r or delivery d_r. Line 14 also checks whether or not the insertion of the first element is feasible at the time of insertion. Feasibility is determined by transportation capacities, the order of pickup and delivery (first pickup, then delivery) and the fact that a pickup can only be inserted into a vehicle's route where the delivery has already been inserted or vice versa. The latter only applies if one of the two has already been inserted. If it is not feasible, the first element of the sequence list Φ is moved from the current position to the last position in the list, where its insertion will be reevaluated at a later time. If it is feasible and the

best position for insertion is found, the vertex of this pickup p_r or delivery d_r is inserted into the solution S at the best position pos_{best}. The reason why a pickup or delivery may not be able to be inserted immediately is due to the inherent characteristics of CI-SiIH.

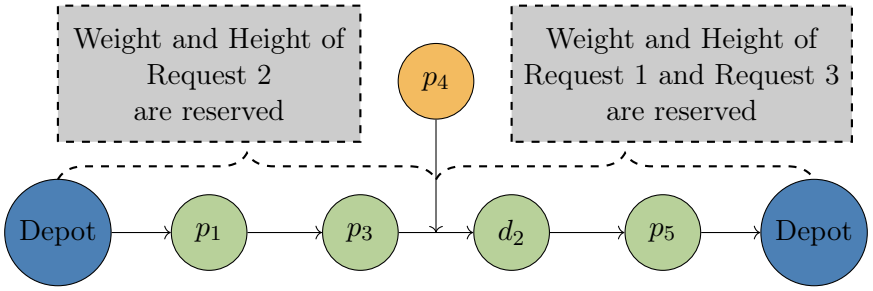


Figure 5.1.: This example shows an incomplete solution that already contains two pickups (p_1, p_3 and p_5) and one delivery (d_5), where the pickup of request 4 is to be inserted using Cheapest Insertion Single Insertion Heuristic (CI-SiIH). When planning the insertion of the pickup p_r of request 4, one must take into account that request 4 will block the vehicle's capacity starting from its insertion position until the end of the vehicle's route, wrt. weight, temperature, and height in order to facilitate a feasible insertion of the delivery d_r of request 4 in another insertion later on. For example, the pickup p_r of request 4 should be inserted between the pickup p_r of request 3 and the delivery d_r of request 2. As the insertion of the pickup p_r of request 2 has not yet taken place, its item weight must be reserved from the beginning of the vehicle's route until the delivery of request 2 is reached. The item weights of requests 1 and 3 must also be reserved for the subsequent request segments. The item weight requirements of request 5 do not have to be considered, as the delivery d_r cannot be inserted before the pickup p_r operation.

Suppose an individual pickup or delivery is inserted, meaning a request has not yet been fully inserted. In that case, transportation capacities, i.e., the request's item weight q_r, must be kept free for these so that the insertion of the missing part is not restricted. For example, suppose only the pickup of one request has been inserted. In that case, transportation capacities must be kept free for all subsequent vertices so that the delivery of this request can be inserted at any subsequent position later on. If a delivery of a request is inserted first, the transportation capacities must be kept free for the preceding vertices. Figure 5.1 illustrates this in detail.

After each successful insertion, the corresponding pickup p_r or delivery d_r is removed from the sequence list Φ in Line 21. Now that all requests have been processed, the solution S is output in line 22 as the solution of CI-SiIH.

Pairwise Insertion The CI-PaIH procedure is very similar to the CI-SiIH. The essential differences are that, firstly, as the name indicates, the insertion is done pairwise, i.e., pickup p_r and delivery d_r of a request R are inserted together in one insertion iteration.

Secondly, certain steps are modified, which is described in Algorithm 2.

In Algorithm 2 lines 6 and 7 from Algorithm 1 are combined, as the pickup p_r and delivery d_r of requests are no longer processed separately. As a result, in line 19 (Algorithm 2), no pickup p_r or delivery d_r is removed individually from the sequence list

Algorithm 2 Cheapest Insertion (Pairwise Insertion)

1: **Input** $R^{unserved}$, st ▷ unserved request set and sequence type
2: $S \leftarrow \{\ \}$; $\Phi \leftarrow \{\ \}$ ▷ initialize solution and sequence list
3: **for each** $k \in K$ **do**
4: initialize $route^k$ starting and ending at vertex 0 (v_0) ▷ v_0 represents the depot
5: **for each** $r \in R^{unserved}$ **do**
6: Φ.insert(r)
7: Φ.sort(st)
8: **while** $\Phi \neq \varnothing$ **do**
9: $pos^{p_r}_{best} \leftarrow \varnothing$
10: $pos^{d_r}_{best} \leftarrow \varnothing$
11: $q_{best} \leftarrow big\text{-}M$
12: **for each** $k \in K$ **do**
13: **for each** insertion pos^{p_r} and pos^{d_r} in $route^k$ **do**
14: **if** pos^{p_r} and pos^{d_r} are feasible **then**
15: compute quality value q_{pos}
16: **if** $q_{pos} < q_{best}$
17: $q_{best} \leftarrow q_{pos}$ and $pos^{p_r}_{best} \leftarrow pos^{p_r}$ and $pos^{d_r}_{best} \leftarrow pos^{d_r}$
18: S.insert(p_r, d_r).at($pos^{p_r}_{best}$, $pos^{d_r}_{best}$)
19: Φ.remove(r)
20: **Output** S

but rather removed collectively as a request $r \in R$. Also in the case of CI-PaIH, pickup p_r and delivery d_r are inserted at the same time, meaning that both $pos^{p_r}_{best}$ in Line 9 and $pos^{d_r}_{best}$ in line 10 are initialised with `null`, as both insertion positions have to be determined in one insertion iteration. Followed by the modification of Line 14, where the feasibility of both insertion positions are checked. Only when both positions are feasible, in terms of transportation capacity, the order of pickup p_r and delivery d_r (first pickup, then delivery) and that a pickup p_r can only be inserted into the route of a vehicle where the delivery has already been inserted or vice versa, and if the current best quality value q_{best} is improved in Line 16, the positions for pickup p_r and delivery d_r are stored in $pos^{p_r}_{best}$ and $pos^{d_r}_{best}$. In Line 18, instead of just inserting the vertex of a pickup p_r or delivery d_r, both vertices of pickup p_r and delivery d_r are inserted at the best positions $pos^{p_r}_{best}$ and $pos^{d_r}_{best}$ respectively in the solution S. The `else` condition presented in Algorithm 1 (Line 18) has been removed in Algorithm 2. This modification has been made since it is guaranteed that a feasible insertion possibility will be found for each request. This is due to the fact that transport capacities no longer need to be artificially kept free as in CI-SiIH. All the other steps remain identical to CI-SiIH.

Therefore, reference is made here to Section 5.2.1.1 (*Single Insertion*).

5.3. Introduction to Metaheuristics

Metaheuristics are powerful tools in optimisation and operations research that aim to find efficient solutions to complex problems for which no exact solution methods are available (Glover, 1986). The term "metaheuristics" refers to heuristics of heuristics, meaning that they are higher than conventional heuristics and operate at a higher level of abstraction (Blum and Roli, 2003). They are characterised by their flexibility, adaptability and efficiency and are particularly suitable when exact solutions are impractical due to the size or complexity of the problem (Talbi, 2009). In contrast to exact methods, metaheuristics offer a heuristic approach to finding good solutions without addressing specific problem properties.

The basic approach of metaheuristics is to iteratively improve a set of solutions by applying a variety of search strategies. Different techniques such as local search (Section 5.3.0.1), *Randomisation* (Section 5.3.0.2), *Diversification* (Section 5.3.0.3) and *Intensification* (Section 5.3.0.4) are combined to explore a broad solution landscape and avoid local minima.

An important aspect of metaheuristics is their ability to adapt to the problem structure. This is often achieved through adaptive mechanisms that allow the algorithm to learn during the search process and adapt its strategies accordingly. As a result, metaheuristics can be effectively applied to different types of optimisation problems without the need for specific knowledge of the problem.

Another feature of metaheuristics is their ability to find good solutions in a short time. This is particularly important for real-time or resource-constrained applications where a fast response is required.

There are many different types of metaheuristics, including Genetic Algorithm (GA), Swarm Intelligence (SI), Simulated Annealing (SA), Tabu Search (TS), Genetic Programming (GP), Adaptive Large Neighborhood Search (ALNS) and many other, which are briefly introduced in the following paragraphs. Each of these methods has its own strengths and weaknesses and is suitable for different types of problems.

Genetic Algorithm One of the best-known metaheuristics is the Genetic Algorithm (GA), which is inspired by biological evolution (Bäck, 1996). GAs simulate the process of natural selection by generating a population of candidate solutions that interact with each other and evolve over time through mutation, crossover and selection (Holland, 1992). Through these evolutionary processes, the population gradually converges to better solutions to the problem.

Swarm Intelligence Another prominent metaheuristic is Swarm Intelligence (SI), which is inspired by collective behaviour in nature (Eberhart and Shi, 2001). SI algorithms simulate the behaviour of swarming animals such as bees, ants or flocks of birds by creating a group of agents that search for solutions together (Dorigo and Stutzle, 2004).

Each agent communicates and interacts with its neighbours to exchange information and improve the solution (Dorigo and Stützle, 2003). Through the cooperation and coordination of the swarm, SI algorithms can efficiently explore complex search spaces and find good solutions.

Simulated Annealing Simulated Annealing (SA) is another well-known metaheuristic inspired by the thermal annealing technique in metallurgy (Kirkpatrick et al., 1983). SA algorithms simulate the process of slowly cooling metal, causing it to transition to a stable state (Metropolis et al., 1953). Similar to cooling metal, the SA algorithm starts with a high "temperature", which means that it takes large steps in the search space and accepts locally poor solutions (Van Laarhoven and Aarts, 1987). Over time, the temperature is gradually reduced, decreasing the probability of accepting bad solutions and leading to convergence to an optimal state.

Tabu Search Another important metaheuristic is Tabu Search (TS), which is based on the idea of "banning" solutions that have already been visited and thus avoiding the risk of getting stuck in local minima (Glover, 1989). TS algorithms use a list of forbidden moves to guide the search and ensure that new solutions are explored (Glover and Laguna, 1998). By specifically avoiding bad regions in the search space, TS algorithms can find global optima, even in difficult problem instances.

Genetic Programming Genetic Programming (GP) is another important metaheuristic that uses the process of natural selection and genetic variation to generate computer programs that represent solutions to a given problem (Koza, 1996). GP algorithms use a population of computer programs, represented as chromosomes or genetic structures, and apply genetic operators such as mutation and crossover to generate new programmes (Poli et al., 2007). By iterating and evolving these programs, GP algorithms can model complex problems and find effective solutions.

Adaptive Large Neighborhood Search Adaptive Large Neighborhood Search (ALNS) was chosen as the metaheuristic for this research as it has proven to be extremely flexible and powerful. Pisinger and Ropke (2019) and Petropoulos et al. (2023) have shown the effectiveness of ALNS in different application domains, thus justifying its suitability for use in different contexts.

It is a metaheuristic algorithm based on an adaptive framework. In contrast to rigid algorithms, ALNS dynamically adapts to the problem structure by using a variety of search operators to explore a broad solution landscape. In addition to the standard operators taken from the literature, specially developed operators can also be implemented to improve the efficiency of the algorithm.

The algorithm consists of two key components: a *Destroy* neighborood (Sections 5.3.1 & 5.3.2.3) and a *Repair* neighborhood (Sections 5.3.1 & 5.3.2.4). The *Destroy* operator selects parts of the incumbent solution (Sections 5.3.1 & 5.3.2.1) to be destroyed to make

room for potential improvements. The *Repair* operator is then responsible for restoring the solution while ensuring that the solution remains feasible.

During the search process, the current best solution, the incumbent solution, is continuously exchanged with new solutions if the new ones are better or fulfil specific criteria (Section 5.3.2.1). The adaptive element of ALNS lies in the adaptation of the *Destroy* and *Repair* strategies as well as the weighting of the operators during the search process. This allows the algorithm to react more efficiently to the problem structure and generate better solutions.

Another important component of ALNS is the *Stopping Criterion* (Sections 5.3.1 & 5.3.2.1), which defines when the search process ends. This can be based on a maximum number of iterations, a specific duration or a predefined threshold criterion.

In addition to the metaheuristics mentioned above, there are many others, including particle swarm optimisation (PSO) (Kennedy and Eberhart, 1995), ant colony optimisation (ACO) (Dorigo et al., 2006), harmony search (HS) (Geem et al., 2001) and many others. Each metaheuristic has its own advantages and disadvantages, as well as specific areas of application in which it works particularly well. The choice of the appropriate metaheuristic depends on the nature of the problem, the available resources and the goals of the optimisation process.

Metaheuristics offer a robust and flexible approach to solving complex optimisation problems in various application areas such as engineering, logistics, finance, robotics and many others. By iterating and improving candidate solutions over a series of steps, they enable the systematic search for high-quality solutions. This general explanation lays the foundation for more specific techniques that aim to further improve the efficiency and effectiveness of the search. Three such concepts are *Local Search*, *Diversification* and *Intensification*, which pursue different strategies to optimise the search. These three key concepts play a crucial role in the design of effective metaheuristics and are discussed in more detail below.

5.3.0.1. Introduction to Local Search in Metaheuristics

Local Search is a fundamental concept in metaheuristics that aims to find an improved solution by making incremental changes to a current solution and moving only in the local neighbourhood of that solution. In contrast to global search methods that explore the entire search space, *Local Search* focuses on improving a single solution by making incremental local improvements. The algorithm starts with an initial solution and continuously improves it until no further improvement is possible or a predefined *Stopping Criterion* is met.

The functionality of *Local Search* can be summarised in several steps. First, an initial solution to the optimisation problem is generated. This solution can be generated in different ways, depending on the type of problem and the available heuristics. A neighbourhood structure is then defined, which enables possible changes to the current solution. These changes can include, for example, removing, adding or replacing elements of the solution.

Once a neighbourhood structure has been defined, a local search is performed by making incremental changes to the current solution. Each change is evaluated, and if it leads to an improved solution, it is accepted and defined as the new current solution. Otherwise, the change is discarded, and a search is made for another change that could lead to an improvement. This process is repeated iteratively until no further improvement is possible or a predefined *Stopping Criterion* is met.

Local Search can be an attractive option due to its efficient and simple implementation, especially for problems with large search spaces or those where the structure of the problem makes it possible to quickly find local improvements. This search procedure has proven effective in a variety of application domains, including routing, scheduling, assignment and more (Aarts and Lenstra, 2003).

Examples of the use of *Local Search* in Metaheuristics are as discussed above Genetic Algorithm (GA), Swarm Intelligence (SI), Simulated Annealing (SA), Tabu Search (TS), Genetic Programming (GP) and Adaptive Large Neighborhood Search (ALNS).

5.3.0.2. Introduction to Randomisation in Metaheurisitcs

Randomisation is an essential concept in metaheuristics that aims to diversify the search for solutions and break out of local minima by introducing random elements into the optimisation algorithm. The use of randomness ensures that the search is broad and explores different parts of the search space rather than focusing on specific areas early on. This helps to increase the probability of finding high-quality solutions, especially in situations where the search space is very large or poorly structured.

The way *Randomisation* works is by introducing random elements into the search process to diversify the search and ensure that different solution domains are explored. This can be done, for example, by using random initial solutions, introducing perturbations into the search or using random operators to generate new solutions. The integration of random elements ensures that the optimisation algorithm does not get stuck in a local optima early on and has the opportunity to explore different parts of the solution space (Holland, 1992).

Randomisation can be integrated into metaheuristics in different ways, depending on the nature of the problem and the heuristics available. Random perturbations can be introduced into the search to promote *Diversification* and ensure that the search does not get stuck in a local optima early on. Through the targeted integration of random elements, metaheuristics can be used effectively.

5.3.0.3. Introduction to Diversification in Metaheurisitcs

Diversification is another key concept in metaheuristics that aims to expand the search to different regions of the search space to ensure that potentially good solutions are not overlooked. This strategy is crucial to ensure that the optimisation algorithm can explore a wide range of solutions instead of getting stuck in a local optima. As part of *Diversification*, various mechanisms are used to enable a diverse exploration of the

search space. These include random exploration, the introduction of perturbations into the search and the use of diversity-promoting operators.

One of the basic methods for promoting *Diversification* is random exploration of the search space. Here, candidate solutions are randomly generated and evaluated to explore a wide range of potential solutions. Although this method alone may not be efficient, it helps to ensure that the algorithm does not get stuck in a local minimum early on and has the opportunity to explore different parts of the search space.

In addition, specialised operators can be used to promote *Diversification*. These include, for example, perturbation operators that specifically change the current candidate solution in order to generate new and potentially promising solutions. These perturbations can take different forms, such as destroying and repairing solutions or inserting random elements. These targeted perturbations steer the search in different directions and enable a broader exploration of the solution space.

Another approach to promote *Diversification* is the use of mechanisms that maintain or increase the diversity in the population of candidate solutions. This includes, for example, diversity-based selection procedures that ensure that different candidate solutions are selected and developed further instead of focussing only on the most promising ones. Maintaining a diverse population ensures that the algorithm explores different parts of the search space and does not get stuck in a local optima in an early stage of exploration (Talbi, 2009). Examples of *Diversification* and its application in this TD-CVRPDP are further discussed in Section 5.3.2.1.

5.3.0.4. Introduction to Intensification in Metaheurisitcs

Intensification holds the same importance as *Diversification* in metaheuristics, which aims to focus the search on promising areas of the search space in order to improve the quality of the solutions found. In contrast to *Diversification*, which focuses on expanding the search over the entire search space, *Intensification* focuses on refining the search and concentrating on the most promising regions. This strategy helps to improve the efficiency of the algorithm by directing its resources to the most relevant areas of the search space and avoiding unnecessary exploration in less promising areas.

One of the basic methods for promoting *Intensification* is adjusting the *Local Search*. This involves identifying promising solution candidates and taking targeted steps to improve them. These steps can include, for example, local improvements to the solution, the application of optimisation techniques to sub-problems or the refinement of the solution through neighbourhood searches. The *Local Search* further improves the already promising solution candidates and increases their quality.

In addition, information about good solution candidates can be used to promote *Intensification*. This can be done by using heuristics or knowledge about the problem to direct the search to promising areas of the search space. For example, good solution candidates from previous iterations can be used to initialise the search or to develop targeted search strategies based on known patterns or properties of the problem.

Another approach to promote *Intensification* is the use of specialised mechanisms that aim to accelerate convergence to an optimal solution. These include, for example, adaptive

parameter tuning methods that dynamically adjust the parameters of the algorithm to improve performance (the application of which is discussed in Section 5.3.2.1) or efficient convergence criteria that terminate the search when a sufficiently good solution has been found. By controlling the search in a targeted manner, *Intensification* can help to improve the quality of the solutions found and increase the convergence speed of the algorithm (Talbi, 2009).

5.3.1. Introduction to the Adaptive Large Neighborhood Search

The Adaptive Large Neighborhood Search (ALNS) is a metaheuristic that was developed for solving combinatorial optimisation problems. This heuristic combines elements from different heuristics and iteratively searches for improved solutions by exploring a large neighbourhood. The main advantage of ALNS is that it adapts to the structure of the problem during the solution process and enables an adaptive selection of neighbourhood structures.

The functionality of ALNS can be divided into several steps. First, an initial solution of the optimisation problem is generated, which is initialised as the incumbent solution. This solution can be generated in different ways, depending on the type of problem and the available heuristics. Next, a neighbourhood structure is selected that makes a change to the current solution. This change can include, for example, removing and reinserting, adding or replacing elements of the solution, which is known as the *Destroy* and *Repair* procedure.

Once a neighbourhood structure has been selected, changes are made to the current solution. The quality of the modified solution is then evaluated. If the modified solution is better than the current best solution, it is accepted and defined as the new best solution. Otherwise, the modified solution can still be accepted under certain conditions so as not to block the search at an early stage, replacing the incumbent solution.

A key feature of ALNS is the adaptive selection of neighbourhood structures. This means that different neighbourhood structures are used during the solution process to diversify the search and ensure that different parts of the search space are explored. The selection of neighbourhood structures can be made by different criteria, such as the performance of each structure in previous iterations or the current characteristics of the problem.

ALNS also utilises a strategy of intensifying the search in promising areas of the search space. This means that the search focuses on promising solution areas and uses specific neighbourhood structures to make targeted improvements in these areas. By combining *Diversification*, discussed and applied in Sections 5.3.0.3 & 5.3.2.1, and *Intensification*, discussed and applied in Sections 5.3.0.4 & 5.3.2.1, ALNS enables an efficient and robust search for high-quality solutions to a variety of combinatorial optimisation problems (Ropke and Pisinger, 2006).

5.3.2. Application of the Adaptive Large Neighborhood Search in the TD-CVRPDP

ALNS is a well-established metaheuristic commonly used for solving combinatorial optimisation problems. In this section, we will examine the application of ALNS in the context of TD-CVRPDP and discuss the reasons for its effectiveness. This section is divided into many subsections, as most aspects are quite complex and will be discussed in detail. For this purpose, pseudocodes are used to guide through ALNS, and reference is made to the respective subsections (Table 5.6).

This metaheuristic offers high flexibility and efficiency in finding good solutions for the TD-CVRPDP. By adaptively selecting *Destroy* and *Repair* operators, it can explore larger solution spaces in less time and thus generate a variety of solutions that optimise the problem better than conventional heuristics. The robustness of ALNS makes it a suitable choice for the problem in this research as the algorithm is able to handle large and complex instances of the TD-CVRPDP without compromising the solution quality. This robustness is due to the adaptive nature of the algorithm, which allows it to adapt to different problem configurations (Ropke and Pisinger, 2006). It has proven to be a powerful method for solving the CVRPDP in numerous studies. Also, empirical studies have shown that ALNS achieves better results compared to other metaheuristics such as *Tabu Search* and *Simulated Annealing*. These results emphasise the effectiveness of ALNS for the CVRPDP (Subramanian et al., 2017).

To illustrate how ALNS works in the TD-CVRPDP, the pseudocode of the ALNS algorithm is shown in Algorithm 3.

Algorithm 3 Adaptive Large Neighborhood Search

Input a feasible solution x ▷ Generated with CI
$x^b \leftarrow x$; $\rho^- \leftarrow (1, \ldots, 1)$; $\rho^+ \leftarrow (1, \ldots, 1)$;
repeat
    Use RW to select *Destroy* and *Repair* procedures $d \in \Omega^-$ & $r \in \Omega^+$ using ρ^- & ρ^+
    $x' \leftarrow r(d(x))$
    if $x' < x^b$ **then**
        $x^b \leftarrow x'$
    if accept(x', x^b) **then**
        $x \leftarrow x'$
    if iteration is a multiple of 200 **then**
        update(ρ^+)
    if iteration is a multiple of 50 **then**
        update(ρ^-)
until stop criterion is met
Output x^b

The ALNS algorithm for the TD-CVRPDP begins with the input solution x, which is calculated with the CI, a detailed description of which can be found in Section 5.2.1, and the initialisation of the best solution x^b, which is initialised with the best solution of

the D-5R-LA-TS generated with the CI, as well as the probabilities for the *Destroy* ρ^- and *Repair* ρ^+ operators in lines 1 & 2. This initial solution x represents a set of routes, whereas each route is designated to a vehicle k from the set of vehicles K. In addition, the probability vectors ρ^- and ρ^+ are initialised, which represent the selection probabilities of the individual *Destroy* and *Repair* operators. The selection probabilities, evaluated using the so-called *Fitness*, are discussed in more detail in Section 5.3.2.1 (*Fitness and Selection Probabilities*).

In the main iterations of the algorithm, between Line 3 and Line 14, *Destroy* and *Repair* operations are selected to generate a new solution x'. As part of the definition, a new solution is one that has been generated by a particular search operation or strategy and has not yet been compared with the incumbent or best solution. In comparison, an incumbent solution is one that represents the current best or accepted solution found at a particular point in the search process. The best solution is used as a reference point for comparison with the other two solutions, which is the best solution found.

The fundamental idea of this metaheuristic is to iteratively explore the neighbourhood of the initial solution, also known as the *Incumbent Solution* by destroying parts of the existing solution and repairing these destroyed parts to possibly find better solutions. This is done in Lines 4 & 5 using the probability vectors ρ^+ and ρ^-, which influence the selection of the *Destroy* $d \in \Omega^-$ and *Repair* $d \in \Omega^+$ operators. A detailed overview of the operators used and possible operators is presented in Sections 5.3.2.3 (*Destroy Neighbourhoods*) & 5.3.2.4 (*Repair Neighbourhoods*). After selecting the operators, the solution x is destroyed d and repaired r, generating the new solution x'. This new solution x' is then evaluated, and if it is better than the previous best solution, it is updated as the new best solution (Lines 6 & 7). As we optimise three objectives simultaneously, the *Quality Value* achieved is used to evaluate a solution's performance. This *Quality Value* and the associated WSA are described in Sections 2, 5.1 & 5.1.2.1.

In the case of a worse solution, the *Acceptance Criterion* of the algorithm, in Line 8, makes it still possible to accept these solutions under certain conditions. Instead of updating the best solution, the *Incumbent Solution* x is replaced by the new solution x'. In this research, the *Acceptance Criterion* was set at 5% deterioration, which means that the new solution may be a maximum of 5% worse than the best solution to be accepted. More about incumbent solution *Acceptance Criteria* is provided in Section 5.3.2.1 (*Incumbent Acceptance Criterion*). This is an important mechanism that enables the algorithm to escape from local optima and increase the possibility of finding better solutions, which is also a part of the known *Diversification*, discussed in Section 5.3.2.1 (*Diversification*).

The adaptation parameters ρ^+ and ρ^- are regularly updated (Lines 11 & 13) to ensure that the search is guided and the selection of *Destroy* and *Repair* operators is adaptive. This allows the algorithm to effectively explore different solution spaces and intensify the search as it progresses (Section 5.3.2.1 (*Intensification*)). How exactly all parameters in this research were set for this metaheuristic is explained in Section 5.3.2.1.

All the steps between Line 3 and Line 14 are repeated until a *Stopping Criterion* is met. Three different *Stopping Criteria* were implemented as part of this research, whereas the first criterion, which is met, stops the algorithm. These *Stopping Criteria* used and other

possible criteria are explained in Section 5.3.2.1 (*Stopping Criterion*).

At the end of this algorithm, after running through the outlined algorithmic steps, the best solution found x^b is returned in Line 15.

Subsection	Digital Reference
Construction Heuristic	5.3.2.2
Destroy Neighbourhoods	5.3.2.3
Finding Destruction Rate Adjustment	5.3.2.3
Repair Neighbourhoods	5.3.2.4
Incumbent Acceptance Criterion	5.3.2.1
Fitness and Selection Probabilities	5.3.2.1
Stopping Criterion	5.3.2.1
Diversification	5.3.2.1
Intensification	5.3.2.1

Table 5.6.: Digital Reference of the ALNS Subsections

5.3.2.1. Adaptive Large Neighborhood Search Settings

In the context of the application of ALNS in the TD-CVRPDP, the fine-tuning of ALNS settings is crucial for the performance of the algorithm. This section is dedicated to the detailed study and definition of these settings, which play an essential role in the efficiency and effectiveness of the ALNS algorithm. The ALNS settings encompass several aspects, including the incumbent solution *Acceptance Criterion*, the selection of *Destroy* and *Repair* operators, and the control of the update frequency of their selection probabilities, and the *Stopping Criterion* of the ALNS. Each of these elements directly influences the search itself or the calculation time to find the best solution for the TD-CVRPDP. In this section, the theoretical foundations of each setting are explained, and their role in the context of and TD-CVRPDP is discussed. Possible approaches to fine-tune and optimise these settings based on current research and best practices are addressed. Through in-depth analysis and optimisation of the ALNS settings, we aim to improve the performance of the ALNS algorithm in TD-CVRPDP and thus contribute to more efficient and accurate solution finding.

Incumbent Acceptance Criterion A key component of ALNS is the Incumbent Acceptance Criterion (IAC), which regulates the acceptance or rejection of newly generated solutions based on various criteria.

An example of the IAC is the temperature-controlled criterion, which is based on a concept similar to simulated annealing (Section 5.3), where the acceptance of a new solution is controlled by a temperature function that decreases as the search progresses. This enables an increasing convergence of the search for better solutions and has been

successfully applied in the solution of vehicle routing problems and other combinatorial optimisation problems (Pisinger and Ropke, 2019).

Also, an IAC can be defined based on a fitness function that combines various criteria, including the objective function value and constraints. This IAC is used to evaluate the quality of a solution, which can include how well the objective function has been optimised, but also compliance with constraints such as time windows or capacity constraints in logistical problems. By incorporating various criteria into the fitness function, the IAC can determine the acceptability of solutions based on a comprehensive evaluation of their performance. This technique allows flexible adaptation to the specific requirements of a problem and has been successfully used in various applications such as the vehicle routing problem with time windows (Álvarez and Munari, 2016).

Another common IAC is the use of an acceptance threshold, where a new solution is accepted if the difference in the objective function values between the new solution and the current best solution is below a specified threshold. This technique has been studied in the literature for solving knapsack problems and other discrete optimisation problems (Pisinger, 2004). This IAC is also used in this research, whereby the threshold was set to allow a 5% deterioration of the best solution. For example, a new solution has been calculated that has a *quality value* which is more than 5% worse than the best solution, which means that it will not be accepted. If the new solution is only less than or 5% worse, the same or even better than the best solution, this new solution will replace the incumbent solution. If the new solution is as good or even better than the best solution, it is accepted as the new best solution.

The selection of the appropriate IAC is critical to the success of ALNS in solving a given problem. Experimental evaluations and adaptations are often required to determine the optimal IAC for a given application.

Fitness and Selection Probabilities Fitness and selection probabilities are crucial in many evolutionary algorithms and metaheuristics, including *Genetic Algorithm*, *Evolutionary Strategies*, *Evolutionary Programming* and the ALNS. These concepts are closely related and significantly influence the evolutionary process and the search for optimal solutions in a given problem space (Goldberg, 1989; Eiben and Smith, 2003; Luke, 2013).

Fitness usually refers to the quality of a potential solution in relation to the problem to be optimised. It is often measured as a numerical value or score that indicates how well a solution fulfils the objective function or evaluation criterion. Determining a solution's fitness can be problem-dependent and often requires calculating or evaluating the objective function as well as possible constraints.

The selection probabilities determine which solutions from the current population are selected for the creation of the next generation. They are often directly related to the fitness of the solutions and determine how likely it is that a solution will be selected based on its fitness compared to other solutions in the population.

A frequently used method for calculating selection probabilities is the fitness-proportional selection, also known as Roulette Wheel (RW) selection, which is used in this research for the selection of the *Repair* and *Destroy* operators during ALNS. In the RW method,

the fitness of each operator is converted into an interval. This interval corresponds to a section of a "Roulette Wheel", where the size of the section is proportional to the operator's fitness. A random value between 0 and the sum of all fitness values is then generated, and the operator whose interval contains this value is selected. Individuals with higher fitness have larger segments on the RW and are, therefore, more likely to be selected.

The fitness, which influences the segment size of the RW, has to be tracked so that better operators are selected with a higher probability. As part of this research, *Repair* and *Destroy* operator's fitness are tracked independently, which means that after each iteration of the ALNS, the fitness of each criterion is tracked and every 50 iterations, the RW, i.e. the segment size, is updated for the *Destroy* operators, which are evaluated based on the tracked fitness. The RW for the *Repair* operators is updated every 200 iterations, which are also evaluated based on the respective tracked fitness. Experimental results have shown that it is most effective to update the *Repair* and *Destory* operator's fitness independently with an interval of 50 and 200 iterations respectively, which leads to the fastest finding of the best solution.

In terms of how fitness is tracked in the first place, it has been shown that an exclusively positive evaluation is superior to an evaluation of positive and negative results, meaning that the fitness score of an operator is increased by 1 as soon as its application leads to an improved solution. If the application does not lead to an improved solution, the fitness remains unchanged, with no increase or decrease.

The application of this approach in this optimisation problem has proven to be an effective means of systematically selecting operators.

Stopping Criterion Defining suitable *Stopping Criteria* is a crucial aspect of the application of metaheuristics and evolutionary algorithms. *Stopping Criteria* define the conditions under which the search process is terminated and play an essential role in controlling the execution time and the quality of the solutions found (Holland, 1992; Talbi, 2009; Glover and Kochenberger, 2003).

A common strategy for defining *Stopping Criteria* is to set a maximum number of iterations. Once this limit is reached, the algorithm is terminated. This simple strategy limits the execution time and ensures that the algorithm does not run for an unnecessarily long time.

Another option is to terminate the algorithm when a predefined objective function value or quality threshold is reached. This can be, for example, a certain number of consecutive iterations without improving the best solution found or reaching a predefined objective function value. By using such criteria, the algorithm can be stopped as soon as an acceptable solution is found, and unnecessary calculations can be avoided.

In addition, time-based *Stopping Criteria* can also be defined to limit the execution time of the algorithm. This can be done, for example, by specifying a maximum CPU time or a maximum runtime. Such criteria are particularly useful when computing resources are limited, or the algorithm is used in real-time environments.

Another option is to terminate the algorithm when a certain convergence condition is

met. This can be, for example, the achievement of a stable state or the convergence of the fitness values of the solutions in the population. By using such criteria, the algorithm can be stopped as soon as the solutions no longer improve significantly or the search has converged.

In this research project, a combination of three *Stopping Criteria* is chosen. The first *Stopping Criterion* to be met terminates the search process. The first criterion is based on the number of iterations without improvement, which is set to 3000 iterations. As soon as an improvement of the incumbent solution is achieved, this counter is reset. The second criterion defines the maximum number of iterations that the ALNS can run, which is set to $10,000$ iterations. The third criterion relates to Vienna Scientific Cluster 4 (VSC4), which is explained in more detail in section 6, as calculation nodes can be occupied for a maximum of 3 days, which compulsorily determines the last *Stopping Criterion*.

The choice of suitable *Stopping Criteria* often depends on the specific application and the requirements of the problem. Careful selection of the criteria is crucial in order to limit the execution time on one hand and to ensure that sufficient time is available to find a high-quality solution on the other.

Diversification *Diversification* holds major importance in metaheuristics and refers to the strategy of distributing the search for solutions across different parts of the search space in order to explore a wide range of potential solutions (Talbi, 2009; Gendreau, 2003). The goal of *Diversification* is to ensure that the algorithm does not get stuck in local optima and instead finds a variety of solutions that can potentially provide better results.

To increase the efficiency of the search for the best solution in this research, *Noise* is integrated into the solution approach. The application of *Noise* is a technique used within the metaheuristic ALNS (Blum and Roli, 2003). This method aims to generate an appropriate diversity and minimise the risk of local optima by introducing random perturbations into the solution space. This ensures that the algorithm does not remain in a specific region of the solution space. Therefore, in a random interval of 10 to 20 iterations, either the second worst element is removed during the *Destroy* operation or the second best option is selected during the *Repair* operation (Deb, 1999).

There are various techniques and strategies for implementing *Diversification* in metaheuristics. One commonly used method is the use of diversified search operators that perform the search in different ways or emphasise different aspects of the search space (Talbi, 2009). For example, different variants of mutation or *Destroy* operators can be used to increase the variety of solutions generated and ensure that not all solutions are manipulated in a similar way.

Another approach is to diversify the search by using different starting solutions or initialisation strategies (Gendreau, 2003). By choosing different starting points in the search space, the probability that the algorithm will explore different parts of the space and thus find a wider range of potential solutions can be increased.

In addition to using diversified operators and starting solutions, adaptive *Diversification* strategies can also be implemented that dynamically adjust the *Diversification* during the search process. For example, these strategies can be controlled based on the current

solution quality, convergence speed or other problem characteristics.

The effective implementation of *Diversification* requires a balance between exploration and exploitation of the search space. While *Diversification* helps to comprehensively explore the search space and find potentially better solutions, it is also important to intensify the most promising regions of the space to make the search efficient and achieve faster convergence.

Intensification *Intensification* is closely related to *Diversification* and describes the strategy of focussing the search on promising regions of the search space in order to improve the quality of the solutions found. While *Diversification* aims to explore the search space broadly, *Intensification* focuses on directing the search to promising areas that have already been identified (Talbi, 2009; Gendreau, 2003).

There are various approaches to implementing *Intensification* in metaheuristics. One commonly used method is to apply *Local Search* procedures or refinement steps to promising solutions to further improve their quality. These methods may include gradient descent methods, local search or other heuristic improvement techniques. For example, a *Local Search* method such as the 2-opt algorithm could be applied to promising routes to make local improvements and reduce the overall travel time Aarts and Lenstra (2003). The 2-opt algorithm is a heuristic that aims to minimize the length of a route by exchanging pairs of edges to achieve improvements. This type of procedure can be particularly effective when applied to promising solutions, as small changes near good solutions often lead to further improvements.

Another option is to guide the search through the use of *Intensification* strategies or heuristics. For example, a heuristic could be used to identify promising routes by analysing certain characteristics of the routes and prioritising those with short travel times (Gendreau and Potvin, 2010). By focusing the search on these promising routes, *Intensification* can improve solution quality and reduce the number of iterations required.

The choice and adaptation of *Intensification* strategies depend on the specific application and the requirements of the problem. For example, the effectiveness of different *Intensification* strategies may vary depending on the structure of the problem. In some cases, it may make sense to perform a *Local Search* on promising solutions, while in other cases, a heuristic to identify promising solutions is preferred. It is important to test different approaches and identify those that provide the best results (Talbi, 2009; Gendreau, 2003).

In this research, the control of the destruction rate is implemented to ensure that promising parts of the solution are destroyed during the ALNS *Destroy*. This control is continuous and adapts to the current requirements of the search space. If no improvement can be achieved over a longer period of time, the destruction rate is increased accordingly to enable a broader exploration of the search space. A detailed description of this control strategy is provided in Section 5.3.2.3.

As already discussed in Section 5.3.2.1, the selection strategy of *Repair* and *Destroy* operators tracked by their fitness and the use of the RW to select these operators is part of the *Intensification* strategy in this research. Operators that are more likely to lead to a better solution are given a higher selection probability. However, this is already part

of the fundamental idea of ALNS, but should still be mentioned as it contributes to the overall strategy of *Intensification*.

The effective combination of *Diversification* and *Intensification* is crucial for the success of metaheuristics. By combining these two strategies, metaheuristics can find robust and efficient solutions that meet the requirements of different optimisation problems. For example, a metaheuristic could first use a *Diversification* strategy to broadly explore the search space and find promising solutions. Subsequently, an *Intensification* strategy could be used to focus the search on promising areas and further improve the quality of the solutions found (Talbi, 2009; Gendreau, 2003). Through this intelligent combination, metaheuristics can find high-quality solutions that meet the requirements of various optimisation problems.

5.3.2.2. Construction Heuristic

Construction Heuristic (CH) for metaheuristics are algorithmic approaches that aim to create initial solutions for metaheuristics. They form the starting point for the iterative optimisation process in metaheuristics. In contrast to metaheuristics, which are often based on iterative improvements, construction heuristics focus on generating a feasible initial solution that serves as a starting point for the further metaheuristic optimisation process. Essentially, all known heuristics already discussed in Section 5.2 can be used as CH.

Greedy Algorithms, for example, could be used to create the initial solution. In this approach, the best local decision is made in each step in order to build up the solution step by step. *Nearest Neighbour Heurstic* is probably one of the best-known greedy algorithms, in which the nearest available option to the current position is selected and added to the solution. This heuristic is particularly efficient for problems with a spatial structure, such as the TSP.

Another *Greedy Algorithm*, which is used in this research as the construction heuristic, is the CI, which is explained in more detail in Sections 5.2.1 & 5.3.1. This heuristic gradually selects the cheapest vertex and inserts it into the route at the best position to minimise the total cost. This process is repeated until all vertices are inserted, providing an obvious but not necessarily optimal solution.

These types of heuristics are easy to implement and fast to execute, which makes them an attractive option for building initial solutions. However, greedy heuristics tend to get stuck in local optima and may not escape a globally optimal solution.

Another approach is the randomised heuristic approach, which makes random decisions to generate an initial solution. This heuristic can be particularly useful when there are no clear clues for an optimal solution or the search space is very large. By introducing randomness into the design process, the randomised approach can generate a wide variety of solutions and thus explore potentially promising regions of the search space.

The selection of the appropriate CH depends on several factors, including the structure of the problem, the size of the search space and the available resources. It is important to test different heuristics and select the one that provides the best results for the specific optimisation problem.

Algorithm 4 Cheapest Insertion (Construction)

1: **Input** S; $R^{unserved}$, ST ▷ solution, unserved request set and list of sequence types
2: $S_{best} \leftarrow \varnothing$ ▷ initialise best solution
3: **for each** $st \in ST$ **do** ▷ for each sequence type in the list of sequences
4: $\Phi \leftarrow \{\ \}$ ▷ sequence list
5: **if** st refers to Single Insertion **then**
6: Algorithm 1 from Line 3 to 21
7: **else if** st refers to Pairwise Insertion **then**
8: Algorithm 2 from Line 3 to 19
9: **if** $S < S_{best}$
10: $S_{best} \leftarrow S$
11: **Output** S_{best}

The effectiveness of CHs for metaheuristics has been demonstrated in numerous studies and applications. A comprehensive overview of different CHs and their applications can be found in Blum and Roli (2003). Further insights into the development and application of these heuristics are provided by works such as Lawler (1995) and Papadimitriou and Steiglitz (1982).

5.3.2.3. Destroy Neighbourhoods

Within the ALNS metaheuristic, *Destroy* Neighborhoods hold a decisive role at the core of the search for high-quality solutions in complex combinatorial optimisation problems. These strategic mechanisms provide a powerful way to guide and shape the search in the solution space in a variety of ways. By deliberately removing parts of the current solution, *Destroy* neighbourhoods open up new possibilities for systematic exploration of the solution space and help to identify high-quality solutions.

The concept of *Destroy* involves the temporary removal of parts of the current solution using various *Destroy* operators. These operators are designed to remove specific structures or components of the solution in order to generate new neighbourhoods of destroyed solutions. For example, an operator may aim to disrupt specific routes in a VRP or remove customers in a TSP.

The selection and definition of the *Destroy* neighbourhoods strongly depend on the characteristics of the optimisation problem. In practice, various strategies are used to explore a wide range of potential solutions. Here are some examples of *Destroy* neighbourhoods:

- **Sequential Removal**
  - **Description**: This operator removes elements from the solution one after the other in a predefined order.
  - **Example**: In a backpack problem, the sequential removal operator could remove items from the backpack one at a time, starting with the least valuable item.

- **Cluster Removal**
  - **Description**: This operator removes clusters of elements from the solution, where a cluster consists of neighbouring or related elements.
  - **Example**: In a graph colouring problem, the *Cluster Removal* operator could identify and remove a group of adjacent vertices with the same colour to explore different colouring possibilities.
- **Pattern-based Removal**
  - **Description**: This operator removes elements from the solution based on identified patterns or structures within the solution.
  - **Example**: In a scheduling problem with recurring patterns, such as shifts in a labour scheduling problem, the *Pattern-based Removal* operator could identify and remove instances of certain patterns that do not contribute positively to solution quality.
- **Greedy Removal**
  - **Description**: This operator greedily removes elements from the solution based on certain criteria, with the aim of improving the solution quality in the subsequent *Repair* phase.
  - **Example**: In a bin packing problem, the *Greedy Removal* operator could remove items from bins based on their contribution to the overall packing efficiency to free up space for better placements during the *Repair* phase.
- **Worst Removal**[2]
  - **Description**: This operator identifies the least promising elements in the solution according to certain criteria and removes them.
  - **Example**: In a scheduling problem, the operator could identify the task with the highest delay at the worst distance and remove it from the schedule.
- **Random Removal**
  - **Description**: This operator randomly selects a subset of elements from the solution and removes them.
  - **Example**: In a vehicle routing problem, the *Random Removal* operator could randomly remove some customer visits from a route.

The effective application of *Destroy* Neighborhoods requires a careful selection and configuration of *Destroy* operators and a strategic definition of *Destroy* neighbourhoods.

[2] *Greedy Removal* removes elements from the solution based on an immediate benefit to solution quality, focussing on short-term improvements. In contrast, *Worst Removal* identifies the least promising elements according to certain criteria and removes them in order to achieve better solutions in the long term. While *Greedy Removal* aims at local improvements, *Worst Removal* aims at removing long-term barriers to improved solution quality. (Petropoulos et al., 2023)

Through targeted exploration of destroyed solutions, ALNS metaheuristics can search a wide range of solution domains, leading to improved results.

The choice of *Destroy* operators and the *Repair* method is based on established literature, including the work of (Pisinger and Ropke, 2007), (Shaw, 1998), (Cordeau et al., 2001) and (Cordeau et al., 2001). The operators (Pisinger and Ropke, 2007; Shaw, 1998) *Random Destroy* and *Worst Removal* are used during the ALNS *Destroy* operation. The operator *Random Destroy* randomly selects requests to be removed from the incumbent solution, allowing a broad and diversified exploration of the neighbourhood. The *Worst Removal* operator, on the other hand, focuses on systematically removing the most unfavourable requests to promote objective improvements. In the *Worst Removal* operator, the decision is based on four different criteria, namely the *travel time* of a requests' journey, its weight q_r, its temperature t_r, and its *quality value*. The latter represents a combined and weighted value made up of the three objective components discussed in Chapter 2.

The use of *Destroy* neighbourhoods in metaheuristics has been investigated in various studies and applications. A comprehensive overview of ALNS metaheuristics and their applications can be found in Ropke and Pisinger (2006). Further insights into the development and application of *Destroy* neighbourhoods in optimisation research are provided by works such as Shaw (1998).

Finding Destruction Rate Adjustment The dynamic adaptation of the destruction rate is a critical aspect in the effective application of destruction neighbourhoods within the ALNS metaheuristic. It plays a crucial role in controlling the degree to which *Destroy* is applied to solutions (Section 5.3.2.3), which has a direct impact on the efficiency and convergence of the optimisation process. In this context, it has been found that a varying configuration of the destruction rate, in the range of 5% to 60% of the solution, is particularly effective. This range enables flexible adaptation to different problem structures and requirements (Shaw, 1998).

Rapid and adaptive adjustment of the destruction rate is crucial in order to react appropriately to the progress of the algorithm. In concrete terms, this means that the destruction rate is adjusted quickly and, depending on progress, usually every 1 to 20 iterations. If the algorithm finds an improved incumbent solution, the destruction rate is immediately reduced by 5% to reduce the intensity of the destruction and increase the exploration of promising solution areas. However, if no significant improvement is achieved within a defined time frame, i.e., 20 iterations in this research, the destruction rate is increased accordingly to enlarge the search for alternative solutions. Once the destruction rate reaches the defined upper bound, in our case 60%, and there is no solution improvement after 20 iterations, the rate is reduced by 5% instead of continuing to test the 60%. The combination of a fast reaction with a reversal of the rate of destruction allows a higher destruction rate to be applied and speeds up the solution improvement process. This adaptive approach ensures a dynamic and responsive control of the destruction rate, which leads to an efficient and target-oriented optimisation (Pisinger and Ropke, 2007).

In addition, it is important to ensure the feasibility of the solutions during the *Destroy* and *Repair* process. This means that despite the destruction process, care is taken to

ensure that the resulting solutions remain valid and practicable. One approach to ensure this is to consider requests in pairs and remove both the pickup p_r and delivery d_r of a request, which makes it possible to preserve the feasibility of the solutions (Cordeau et al., 2001). This is because we found that relaxing the restrictions during *Destroy* and *Repair* rarely leads to an improved solution on the one hand, and on the other, it is associated with a significant increase in calculation time.

The effective adaptation of the destruction rate, coupled with the guarantee of the feasibility of the solutions, forms the basis for the successful use of *Destroy* neighbourhoods in ALNS metaheuristics. It enables efficient and dynamic exploration of the solution space, which ultimately leads to improved convergence and high-quality solutions.

5.3.2.4. Repair Neighbourhoods

Repair neighbourhoods are an essential part of the ALNS metaheuristic, which aims to improve solutions in combinatorial optimisation problems. In contrast to *Destroy* neighbourhoods, which remove parts from the existing solutions, *Repair* neighbourhoods focus on repairing and reinserting the elements removed by the *Destroy* operation to generate a new solution. Moreover, they are used to correct solutions that violate certain constraints or lie outside the feasible solution space. In practical applications, *Repair* neighbourhoods are used in a variety of metaheuristics, including *Simulated Annealing*, *Tabu Search*, *Genetic Algorithm*, ALNS and more. Different *Repair* strategies can be used depending on the nature of the optimisation problem and the specific requirements of the application. Some of the best-known operators are outlined in the following.

- **Swap-Operator**:
  - **Description**: This metaheuristic exchanges two elements of a solution in order to generate a new solution.
  - **Example**: For example, in a route optimisation problem, it could swap two customer locations in a route to improve the overall route (Glover and Laguna, 1998).
- **Insertion-Operator**:
  - **Description**: An element is inserted at a different position in the solution in order to obtain a new solution.
  - **Example**: This can be useful for optimising the arrangement of elements in a sequence, e.g. the order of tasks in a schedule (Raidl et al., 2019; Aarts and Lenstra, 1997).
- **Reverse-Operator**:
  - **Description**: This metaheuristic reverses the order of a subsequence in the solution in order to obtain a new solution.
  - **Example**: For example, in a production planning problem, it might reverse a series of orders to change the sequence of processing and improve overall performance (Raidl et al., 2019; Glover, 1989).

- **Cross-Over-Operator**:
  - **Description**: This metaheuristic combines parts of two different solutions to create a new solution.
  - **Example**: This is similar to the biological concept of recombination and can be applied in many optimisation problems represented by a sequence of decisions (Raidl et al., 2019; Aarts and Lenstra, 1997).

The selection and adaptation of *Repair* strategies strongly depend on the nature of the optimisation problem and the goals of the application. Careful evaluation and implementation of different *Repair* methods are crucial to ensure the effectiveness of the ALNS metaheuristic and to generate high-quality solutions.

A commonly used *Repair* method is the reinsertion of removed elements based on a specific rule or heuristic, i.e. CI. For example, the reinsertion of customers or vertices in a TSP can be based on their geographical proximity to other customers or vertices. As in the case of this research, the reinsertion of the requests is based on the CI-PaIH, discussed in Section 5.2.1.1.

For the reinsertion of the removed solution parts, i.e. removed requests, by the *Destroy* operator, which was introduced in Section 5.3.2.3, the CI *Repair* operator is chosen in its sequential form (CI-SIH), which is also used for the construction of the initial solution of the ALNS (Pisinger and Ropke, 2007). As discussed in section 5.2.1.1, both and are used to construct the initial solution of the ALNS. However, it turned out that this approach only works well for the construction, but not as a *Repair* operator, since CI-PaIH clearly dominates CI-SiIH during the *Repair* operation, which is shown in Table 5.5. Therefore, we opt to use only the CI-PaIH in the *Repair* procedure. The sorting criteria of the individual sequences remain the same, which are based on five different criteria: Random, pickup p_r and delivery d_r, weight q_r, temperature t_r and height u_r of requests R to be transported. With regard to the latter four sorting criteria, these can be either sorted in an increasing, decreasing or mixed order, where the largest or highest request of the corresponding sorting criterion is selected alternately with the smallest or lowest, explained in more detail in Section 5.2.1.1.

Since for the *Repair* operation, only the pairwise insertion method is used, the algorithm (Algorithm 2) has to be changed slightly for the *Repair* operator. Instead of rebuilding the solution S from scratch, only the part destroyed by the *Destroy* operator needs to be repaired. Therefore, S in Line 2, the solution is not initialised empty but input as a partial solution in Line 1.

5.3.3. The Directional-Penalty Method

One of the constraints integrated into the optimisation of this tri-dimensional problem is based on a novel approach, which was developed in the course of this thesis, called the *Directional-Penalty*. This approach proves to be relevant in practical applications such as the TD-CVRPDP at hand that applies an objective where maximum objective values are to be minimised, as it helps to reduce *strolling trips*, similar to the *ton travel time*

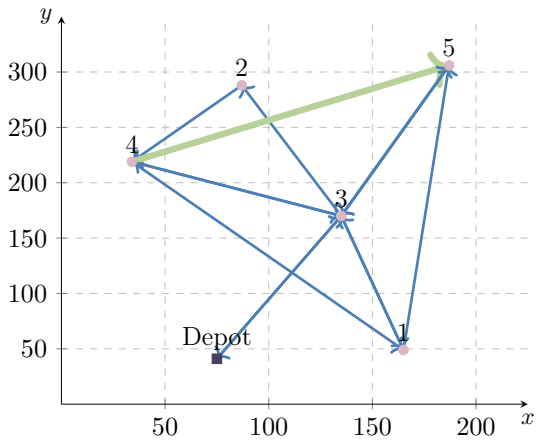
objective. We refer to *strolling trips* as unnecessary loaded trips from an economic point of view, although as explained in Section 5.3.3, they are beneficial during the optimisation regarding the *vehicle temperature* objective component. However, unlike the *ton travel time* objective, the *Directional-Penalty* takes a unique approach. While the maximum value on the route is minimised for the *ton travel time*, the *Directional-Penalty* applies a variable penalty evenly to all sections of the route.

This restriction investigates an item's journey from its pickup p_r vertex to its delivery d_r vertex and checks whether it takes paths on the transport route that lead in the direction of its pickup p_r vertex. An approval threshold can be used to determine which routes can be travelled without penalty. An example of this would be a threshold of 100%, which means that an item can travel the furthest distance within the map's layout twice in vertical and horizontal directions without being penalised. The functionality of this Penalty is described in Algorithm 5. I_h describes the maximum interval in which an item can move horizontally, whereas I_v describes the maximum interval in which an item can move vertically, and ϵ_h and ϵ_v describe the previously mentioned threshold value.

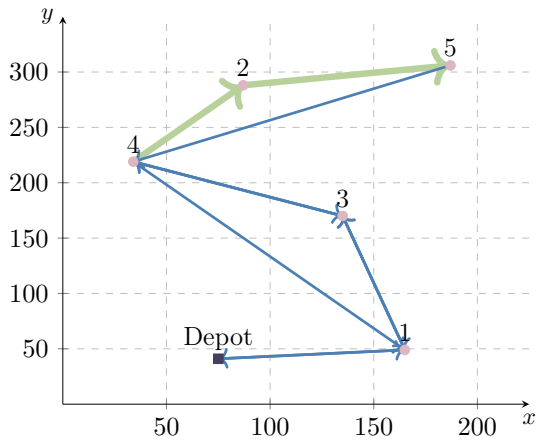
Algorithm 5 Directional-Penalty

Input $I_h = [h_1, h_2]$, $I_v = [v_1, v_2]$, ϵ_h, ϵ_v, $R^{unserved}$
for each $r \in R^{unserved}$ **do**
    H_{mov}.sum(h_{mov}) and $V_{mov}.sum(v_{mov})$
    if $H_{mov} > (h_2 - h_1) \times \epsilon_h$ **then**
        $p_h \leftarrow H_{mov}$
    if $V_{mov} > (v_2 - v_1) \times \epsilon_h$ **then**
        $p_v \leftarrow V_{mov}$
$Penalty \leftarrow p_h + p_v$
Output $Penalty$

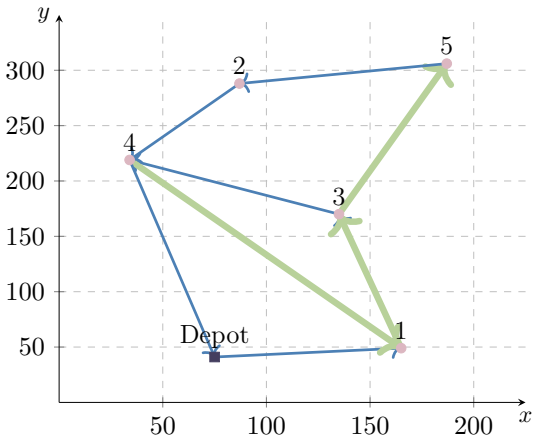
The *Directional-Penalty* restricts the solution space only so slightly, compared to when it is not applied. This is illustrated in Figure 5.2c, in which the heaviest item of this instance is transported from vertex 4 via 1 and 3 to vertex 5. If the threshold value of the *Directional-Penalty* is set to 50%, for example, an item may only travel half the distance in order for it to not be penalised. This is illustrated in Figure 5.2b, in which the same item is transported from vertex 4 via 2 to vertex 5. In the extreme case, if the threshold value is set to 0%, items are always penalised as soon as they make a return trip in both the horizontal and vertical directions, as shown in Figure 5.2a. In this case, the item is transported directly from vertex 4 to vertex 5, without detours, which naturally leads to a longer total *travel time* of the entire route.



(a) *Directional-Penalty*'s threshold: 0%
Travel time of the vehicle route: 2089 sec.
Travel time of the request: 176 sec.



(b) *Directional-Penalty*'s threshold: 50%
Travel time of the vehicle route: 1471 sec.
Travel time of the request: 189 sec.



(c) *Directional-Penalty*'s threshold: 100%
Travel time of the vehicle route: 1185 sec.
Travel time of the request: 486 sec.

Figure 5.2.: Each subfigure illustrates a solution to the same example instance (a small random example), where different threshold values for the *Directional-Penalty* are used. The impact of the *Directional-Penalty* on the total travel time of the vehicle route and the travel time of a request's individual journey is illustrated. The blue arcs outline the route of the vehicle, while the green arcs highlight a request's journey. It is shown that the smaller the threshold value, i.e., the more the *Directional-Penalty* is applied, the longer the vehicle route and the shorter the journeys of all requests.

In practice, the application of the *Directional Penalty* is particularly relevant, as it is unusual to transport items over long distances, even if this would be preferable in theory. This also contributes to the consideration of the *vehicle temperature* issue, as the CHT of the transported requests are ignored in this problem setup. However, this only makes sense and is error-free if items are not transported over a very long distance, as the heat transfer would naturally increase in this case. Based on measurements, transporting an item for up to 40 minutes is non-affecting, as not enough heat can be transferred during this period to falsify the *vehicle temperature* calculation.

With regard to the application of the penalty, the amount by which an item has exceeded the defined threshold is calculated. This excess is then multiplied by an exponential factor to ensure that even a minor excess has only a minimal effect. This ensures that solutions can continue to exist in the solution space despite minor exceedances. In this solution, the following exponential formula was used for both the horizontal and vertical directions:

$$p_h = \left(\frac{H_{mov}}{2}\right) + 1.5 \cdot 1.015 \cdot e^{H_{mov}-200} \tag{5.13}$$

$$p_v = \left(\frac{V_{mov}}{2}\right) + 1.5 \cdot 1.015 \cdot e^{V_{mov}-200} \tag{5.14}$$

These formulas capture the absolute excess of the penalty in the horizontal (5.13) or vertical (5.14) direction and then apply an exponential factor, using 200 as the threshold for the exponential decrease. By applying these formulas, an adequate weighting of the penalties is achieved, whereby even small deviations from the threshold value specification are appropriately taken into account. This helps ensure that the solutions in the optimisation process remain stable and practically applicable despite small deviations.

5.4. Excursion: Dynamic-5-Request-Look-Ahead-Tree-Search

The Dynamic-5-Request-Look-Ahead-Tree-Search (D-5R-LA-TS) is specially tailored for the multi-objective TD-CVRPDP in order to meet the requirements and special features of the transportation problem. Inspired by the experience and workflow of logistic planners at the partnered steel plant, this approach enables continuous and seamless route planning without requiring full knowledge of all requests' properties at the beginning of the calculation. This heuristic was created by inductive reasoning, whereby experimental tests were also carried out on-site. Essentially, this is an improvement in planning that logistic planners would carry out based on their experience, which means that this approach could be applied to many planning problems. Typically, if no heuristics, metaheuristics or even exact solution methods can be used, logistics planners would do the planning from their minds, based on their subjective judgement, so to speak. This heuristic is also based on a similar approach, although it takes into account all possibilities of the next 5 best requests $|R|$. In other words, the logistics planner checks for each vehicle in use which requests are best to handle next, both in terms of the pickup and the delivery actions. Experienced planners can think up to $4-6$ actions ahead, which means that if

2 vehicles are used, 2-3 requests can be planned in advance per vehicle, whereas these planning decisions are made in a subjective way and can be of poor quality. In contrast, the D-5R-LA-TS heuristic calculates the near-optimal or even optimal solution for the next 5 requests, i.e., 10 pickup or delivery actions, regardless of how many vehicles are in use. According to the available computing power, more look-ahead-tree-search operations could be done, but we opt to leave it at 5 requests, i.e., 10 pickup or delivery action, as it can be calculated with an average computer within a reasonable amount of time. This is discussed in more detail in Section 6.6.2. Referring to near-optimal, these result from using the *Tree Search* for the next 5 requests considering all requests $|R|$, shown in Algorithm 6.

Tree Search is essentially a method of restricted enumeration in which, unlike full enumeration, not all possible combinations or paths in the solution space are tested. Instead, only the most promising paths are followed through the search tree, based on certain criteria or heuristics (Russell and Norvig, 2010).

The idea behind the *Tree Search* is to search the solution space efficiently by following only those paths that are thought to lead to the best solution. This can be done by using heuristics that prioritise the most promising options or by using techniques such as alpha-beta pruning when searching in game trees to discard irrelevant paths (Pearl, 1984).

Through this guided selection of paths to explore, the *Tree Search* can significantly improve the efficiency of the search, especially in cases where the search space is very large. It allows to focus on those areas of the solution space that are most likely to lead to the desired solution, thus reducing the overall search effort (Korf, 1985).

Algorithm 6 Dynamic-5-Request-Look-Ahead-Tree-Search

1: **Input** S; $R^{unserved}$ ▷ unserved requests, current (partial) solution
2: **while** $R^{unserved} \neq \varnothing$ **do**
3: $S_{best} \leftarrow \{\ \}$
4: P.detertminedSinglePickups(S) ▷ all pickups contained in solution S without their corresponding deliveries
5: **for** R^{permut} in permutations($R^{unserved}, 5 - |P|$) **do**
6: $S' \leftarrow$ bestInsertionSequenceSearch(R^{permut}, P)
7: **if** $S' < S_{best}$ **then**
8: $S_{best} \leftarrow S'$
9: $S.insert(S_{best}, n)$ ▷ insert first n elements until first delivery action is reached
10: $R^{served} \leftarrow$ determineCompletedPickupAndDeliveryPairs(S_{best})
11: $R^{unserved}$.remove(R^{served})
12: $R^{unserved}$.update()
13: **Output** S

To be able to start with D-5R-LA-TS, the current solution S and the unserved requests $R_{unserved}$ are required as input, as shown in Line 1. As this is a heuristic that can solve dynamic problems, S represents the solution planned so far. If no planning has yet taken

place, all vehicles used start from vertex 0 (v_0), which represents the depot. If planning is in progress, the current positions of the vehicles can be retrieved from S. A `while` loop, defined in Line 2, runs the following algorithm until there are no more unserved request $R^{unserved}$ to process. The number of these requests varies during optimisation as further requests may be added. To ensure the current status of the number of orders, set $R_{unserved}$ is updated in Line 12 so that the requests currently to be processed can be seen in the next iteration. Within the `while` loop, the best solution set S_{best} is initialised as an empty set in Line 3, which is filled during the course of the algorithm. S_{best} consists of a list of routes for each vehicle used.

Another important factor is list P, which represents all single pickups in S. If no requests have been processed so far, i.e., S is empty, P is also empty. However, if some requests have been partially processed, i.e., some orders have already been picked up but not yet delivered, these are inserted in P. If both pickup and delivery of all requests in S have been processed, P also remains empty. This P list is required to lock requests that have already been picked but not delivered, as during the evaluation of the various request permutations, the requests that have been picked up can no longer be stored temporarily somewhere else. As an example, if 2 requests have already been picked up but not delivered yet, it would mean that the *Tree Search* can only evaluate 3 more new requests, although 8 pickup or delivery actions still need to be planned. This is because only the pickup of 2 requests has to be locked in this example, and another 3 pickups and 5 deliveries need to be planned. This procedure is shown in Line5, where for all R^{permut}, which is one of the permutations consisting of 5 requests minus the already loaded pickups P from all requests R.

The best solution sequence with R^{permut} and P is then calculated for S' using the `bestInsertionSequenceSearch`, shown in Algorithm 7. If S' is better than S_{best}, S_{best} is updated with the best solution found so far S'. After all request combinations have been tested, the first actions of the best solution S_{best} are inserted in S up to and including the index n. In this case, n represents the index of the first delivery, as this is the point at which capacity becomes available again, and consequently, the next combination of 5 requests can be calculated. It was also tested to insert all 10 pickup and delivery actions immediately, i.e., to accept the whole solution and then calculate the next 10 actions for the next 5 requests. However, it turned out that this method produces less efficient solutions. The reason for this is that the more up-to-date the planning is, the better the solutions are, which means that new solution spaces should be investigated as soon as capacities become free again. With regard to freed-up capacities, it was also analysed to look for new solutions after each action, as it would be assumed that another new action could be planned in the next iteration. Unfortunately, this is not possible for completed pickups without deliveries, as the requests that have already been picked up must be completed, and their delivery must also be incorporated into the next iteration.

In Lines 10 & 11, the requests inserted in S are determined and removed from $R^{unserved}$, as these have already been partially or even completely processed. In Line 12, the set of unserved requests is updated so that new requests that have been added can be taken into account in the next iteration.

Within the function BestInsertionSeqeunceSearch, the *Tree Search* is used to find the best sequence out of the 10 actions in R^{permut}. To do this, R^{permut} and P are required as input in Line 1, whereby the latter represents the single pickups without deliveries, which are fixed during the *Tree Search* and must be taken into account in future search. This fixation is shown in Line 2, where A denotes the actions. The remaining pickup and delivery actions are added to A in Lines 4 & 5. For example, if P already contains 2 pickups, these are fixed in the first two positions in A, as those requests are already picked up. The position of its delivery is determined subsequently. With 2 fixed pickups, i.e., also pre-reserved but not fixed delivery positions, further 3 requests (6 actions) can now be considered, which is represented in R^{permut}. In Line 6, *Tree Search* is used to find the best solution S within R^{permut}, i.e., the best sequence of the actions. This solution S is then returned to the Algorithm 6.

Algorithm 7 BestInsertionSequenceSearch

1: **Input** R^{permut}, P ▷ unserved requersts, served pickup actions
2: $A \leftarrow P$ ▷ A is the set of pickup and delivery actions
3: **for** $r \in R^{permut}$ **do**
4: A.insert(p_r)
5: A.insert(d_r)
6: $S \leftarrow$ findCheapestSolution(A) ▷ find best solution by means of a *Tree Search* algorithm
7: **Output** S

The distinctive feature of the D-5R-LA-TS heuristic is that it enables continuous planning. The calculation can be continued seamlessly at any point in time. Compared to the other static heuristics and metaheuristics used, this heuristic does not need all data to be available at the beginning for the optimisation, which can commonly lead to a worse performance on average, however, it is continuous and faster. The solution and detailed performance are discussed in more detail in Section 6.6.1.

6. Results

All computed results, performance metrics, and their interpretations are presented and analysed in detail, along with their significance for the research field. Key findings, such as the effect of the *Vehicle Temperature Prediction Method* and the *item strolling effect*, are discussed. In Addition, the results calculated by the Dynamic-5-Request-Look-Ahead-Tree-Search (D-5R-LA-TS) are shown in an excursion, and their strengths, weaknesses and possible extensions for future research are examined.

6.1. Introduction to the Pareto-front in the Context of the Multi-Objective TD-CVRPDP

This section introduces a common concept for analysing solutions to multidimensional problems, i.e., multi-objective optimisation. This analytical concept, the so-called Pareto-front, and its application, as well as the test instance sets used, are discussed.

6.1.1. Pareto-Front: Conceptual Principles

The Pareto-front is a concept in multi-objective optimisation that refers back to Vilfredo Pareto's work (1964). It describes the set of all non-dominating solutions in a multi-objective optimisation problem. A solution is considered non-dominated if it is better than another solution in at least one objective without being worse in any other objective (Deb, 2001). The Pareto-front thus forms the edge of the solution space on which no improvement is possible in one objective criterion without causing a deterioration in at least one other objective criterion.

A central concept in multi-objective optimisation is Pareto-dominance. A solution A dominates a solution B if A is better than B in at least one objective and is not worse in any other objective. A solution not dominated by any other solution belongs to the Pareto-front and is described as Pareto-optimal. In the example illustrated in Figure 6.1, Solution A is directly on the Pareto-front line. This means that solution A is not dominated by any other solution. Also, it can be seen, that solution B is dominated by solution A, as in this example, Solution A is better in both objectives compared to solution B. It can also be observed in this figure that no solution exists beyond the Pareto-front. This is because solutions below the Pareto-front would violate at least one constraint. This is exemplified in the infeasible area. However, there may also be constraints in the optimisation, resulting in a "non-smooth" Pareto-front. This would imply that the Pareto-front is not a straight line or exponential curve, as shown in Figure 6.1, but rather an irregular line.

Pareto-optimality is a decisive criterion in the search for optimal solutions in multi-objective optimisation problems (Zitzler et al., 2001). By identifying Pareto-optimal solutions, informed decisions can be made that ensure an optimal balance between different objectives.

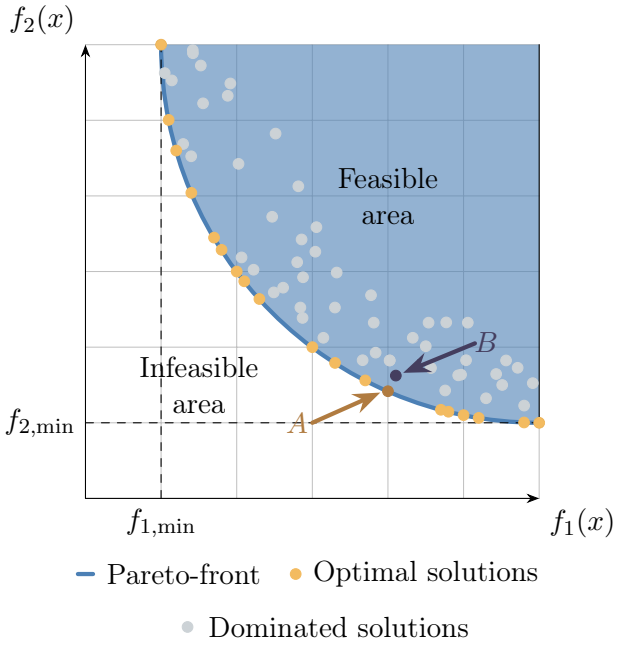


Figure 6.1.: 2-dimensional Pareto-front created during the optimisation of a minimisation problem consisting of two objectives $f_1(x)$ and $f_2(x)$

A typical area of application for multi-objective optimisation in operations research is route planning (Laporte, 1992a). Different objectives such as minimum *travel time*, minimum cost and vehicle utilisation can compete in a transport network. In this context, the Pareto-front represents all non-dominated routes that represent the best trade-offs between these objectives. For example, a Pareto-optimal route could have a longer *travel time* but lower costs and more efficient vehicle utilisation compared to other routes.

The difference between a 2-dimensional and a 3-dimensional Pareto-front lies in the number of objectives considered and, thus, in the dimension of the solution space. In a 2-dimensional Pareto-problem, as can be seen in Figure 6.1, only two competing objectives ($f_1(x)$ & $f_2(x)$) are considered, whereas in a 3-dimensional problem, three competing objectives are considered. The Pareto-front in a 2-dimensional problem consists of a line in the solution space, whereas in a 3-dimensional problem, it forms a plane. Each point on this line or plane represents a compromise between the corresponding objectives.

6.1.1.1. Application to the TD-CVRPDP

In the context of the TD-CVRPDP, i.e., a tri-objective routing problem, the optimisation is between the competing objectives such as the *travel time*, *ton travel time* and *vehicle temperature*. When trying to integrate these three objectives of the TD-CVRPDP, which essentially address operational efficiency, fuel efficiency and resource preservation, it's crucial to carry out a precise analysis of the Pareto-front. The Pareto-front allows us to identify solutions where no improvement in *travel time*, *ton travel time* or *vehicle temperature* is possible without worsening at least one of the other objective components. A Pareto-optimal solution could have a shorter *travel time* but at the cost of a higher *vehicle temperature* and/or *ton travel time*. This is particularly relevant as these three objectives are interrelated in a complex way. Thus, the Pareto-front offers a way to indicate the trade-offs between these objectives and make informed decisions concerning optimal route planning in the TD-CVRPDP. The exact behaviour of the solution in a 3-dimensional space is explained in more detail in Section 6.3.1.

6.2. Test Instance Sets

We introduce our own test instance sets consisting of a total of 200 different instances inspired by real-life scenarios. These instances were divided into four sets of 50 instances each, named A to D. All instances include 9 different vertices $|V|$, denoted as vertices $1-9$, where multiple pickups p^r and/or deliveries d^r of requests R can take place. Each instance also contains a separate vertex (vertex v_0) that acts as a depot from which all vehicles K must start and end. The generated locations of the vertices are randomly placed, according to a uniform distribution, in a 500×500 area. The Euclidean Distance is then used to calculate the *travel time* t_{ij} between these individual vertices. In reality, 3 vertices are approached particularly often by vehicles, as these are major facilities, e.g. mill, continuous caster, and scarfing facility. In order to replicate these in our test instances as well, each of these 3 randomly selected vertices holds 20% of all requests. The remaining requests are distributed according to a uniform distribution, but with the prerequisite that every vertex holds at least one request, overruling the 20% traffic rule. With regard to the requests, their height u_r and weight q_r range from $20 - 22$ centimetres and $15,000 - 36,000$ kilograms, with a standard deviation of 2.04 centimetres and 4767.52 kilograms and a mean of 21.84 centimetres and $25,808.94$ kilograms, respectively. The height and weight of the items are normally distributed. The temperature t_r ranges from $0 - 575$°C, which is exponentially distributed with a mean of 95 °C. The heights, weights, and temperatures are drawn from the aforementioned distribution functions based on data from our partnered steel plant. Since all these instances are inspired by the real world and were not directly imitated, statistical data on distances between the vertices to be served and properties of the requests were evaluated and applied. This is because this master thesis conducts fundamental research on VRPs with temperature dependence, and test instances that meet specific parameters (number of vehicles, temperature distribution, etc.) are needed. The environmental temperature of the respective instances is always

the equivalent of the coldest item temperature, as usually, an item can never be colder or at least not by a considerable amount than the environmental air. The scenario where items are colder than the environmental air appears if the temperature rises during the day and heats up the items that have been completely cooled during the night, causing them to have a slightly cooler temperature than the environmental air. The vehicles also have a fixed capacity, which is limited by the maximum height $H_{max} = 110$ centimetres and weight $U_{max} = 105000$ kilograms. The maximum temperature T_{max} that a vehicle may reach is limited to 400°C.

Set	# **Requests** $\lvert R\rvert$	# **Vertices** $\lvert V\rvert$	# **Vehicels** $\lvert K\rvert$
A	10	10	1
B	30	10	2
C	50	10	3
D	70	10	4

(a) Set-specific parameters of our test instance sets

	Height (in cm)	**Weight (in kg)**	**Temperature (in °C)**
$Min.$	20	15,000	0
$Max.$	22	36,000	575
$\bar{x}$	21.42	25,808.94	95
σ	2.04	4,767.52	116.35
DT	normal	normal	exponential

(b) Request parameters. The values were obtained from an analysis of the provided data by the partnered steel plant.

Table 6.1.: Set-specific parameters of the test instance sets

6.3. Pareto-front Results

This section discusses the test instance sets solutions and the resulting Pareto-front, as well as the solution behaviour in relation to the three objective function components. The impact of the *Directional-Penalty* used during the calculation of each solution, which is explained in the theory in Section 6.3.2, is demonstrated.

6.3.1. Solution Behaviour

In this research, we focus on using the Pareto-front concept to identify non-dominant solutions in the TD-CVRPDP. This and Section 6.6 contain a detailed analysis of the Pareto-front in the context of our applied methods, i.e., ALNS and Dynamic-5-Request-Look-Ahead-Tree-Search (D-5R-LA-TS). By looking at non-dominant solutions on the Pareto-front, we will gain insights into the trade-offs between operational efficiency, expressed as *travel time*, fuel efficiency, expressed as *ton travel time*, as well as resource preservation, expressed as the reached *vehicle temperature*. This analysis will allow us to evaluate the performance of our proposed solution method and contribute to the advancement of optimisation methods in the TD-CVRPDP context.

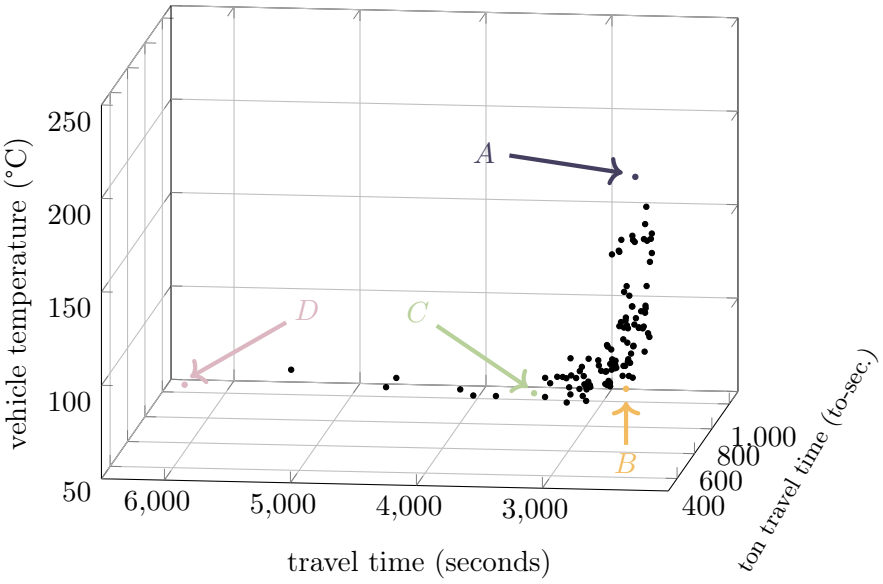


Figure 6.2.: 3D illustration of the Pareto-front plane of an instance of instance set C, consisting of 50 requests $|R|$, 3 vehicles $|K|$ and 10 vertices $|V|$. Each dot represents one solution. In total, 111 solutions are visible, making up the Pareto-front plane. The x-, y-, and z-axes represent each objective component.

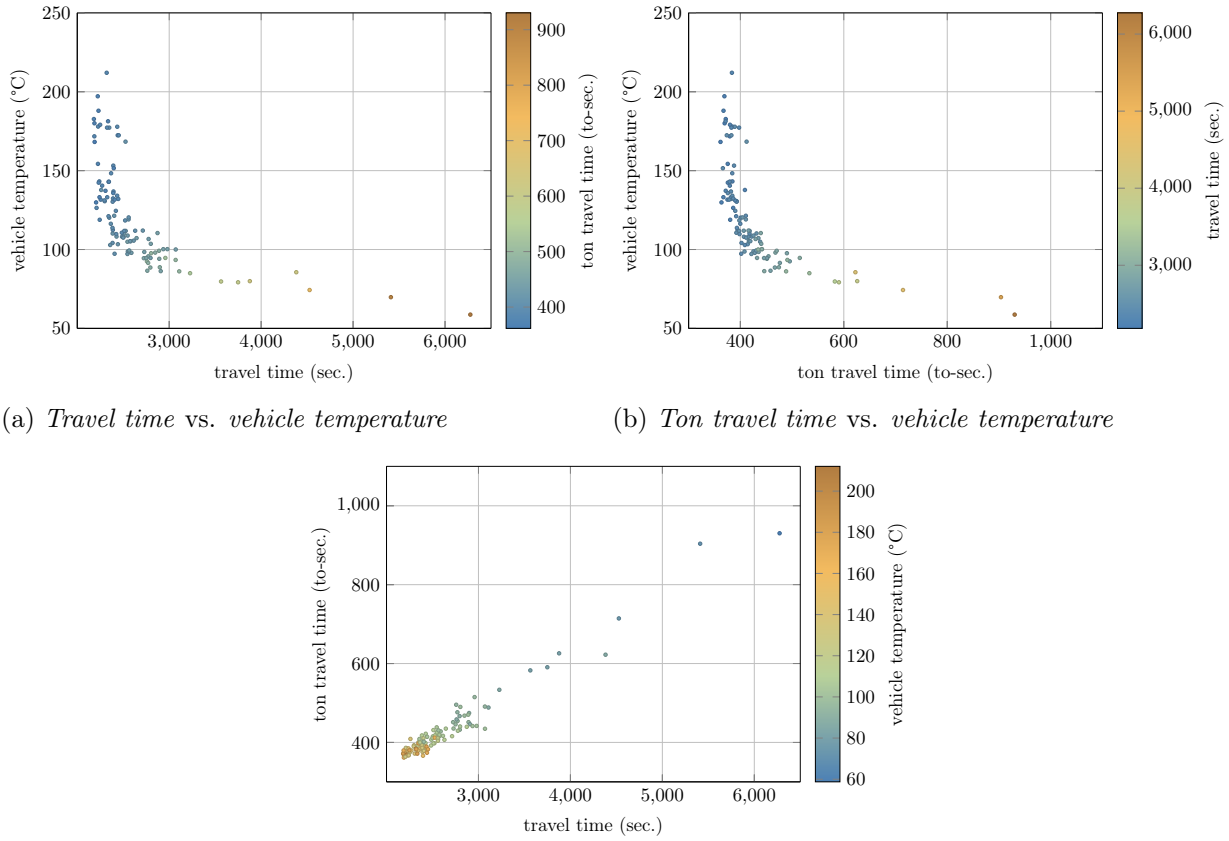
Due to the concurrent optimisation of three different objective components, the resulting solution manifests itself in a three-dimensional space. The results depicted in Figure 6.2 show that the proposed solutions, tested on our test instances discussed in Section 6.2, converge towards the Pareto-front plane. This specific solution in Figure 6.2 shows the three objective component values of a test instance where 50 requests $|R|$ were processed by 3 vehicles $|K|$ on 10 vertices $|V|$.

The analysis reveals that a pure minimisation of *travel time* ($\alpha = 100\%$) and *ton travel time* ($\beta = 100\%$) both lead to a strong increase in *vehicle temperature*, which is shown as solution A in Figure 6.2. However, this correlation is not necessarily causal as the rise in *vehicle temperature* results from disregarding the *vehicle temperature* objective

component. In other words, adding weight to the *vehicle temperature* objective component would reduce it by a tremendous amount, increasing the other two objective components just marginally. As an example, solution B is optimised at 35%, 35% and 30% for *travel time*, *ton travel time* and *vehicle temperature*, respectively, resulting in a reduction in *vehicle temperature* by −117.89% (−114.72°C). The *travel time* and *ton travel time* only increase by +3.57% (+86 seconds) and +4.49% (+18.06 to-sec.), respectively.

At the same time, minimising *vehicle temperature* has a direct causal link with increased *travel time* and ton *travel time*. This is due to the *item strolling effect*, where colder requests are exploited to reduce the average request stack temperature, thus also the *vehicle temperature*, which are being transported. This increases the number of unnecessary trips, which reduces overall efficiency and is referred to as *strolling trips* (Section 6.4.2.1). As solution D illustrates, a pure minimisation of *vehicle temperature* ($\gamma = 100\%$), i.e. the complete negligence of the objective components *travel time* and *ton travel time*, leads to the lowest *vehicle temperature* solution of all those found. Resulting in a *vehicle temperature* of 58.73°C, a *travel time* of 6275 seconds and a *ton travel time* of 930.32 to-sec. Similar to the aforementioned case, by adding weight to the objective components *travel time* and/or *ton travel time*, a strong reduction of both can be achieved by only slightly increasing *vehicle temperature*. As an example, solution C illustrates a solution, where *vehicle temperature* is weighted by 70% and *travel time* and *ton travel time* by 15% each, resulting in a reduction in *travel time* by −94.45% (−3,048 seconds) and *ton travel time* of −74.41% (−396.92 to-sec.). The *vehicle temperature* increased only by 30.89% (+26.25°C). Using this example, it is also interesting to see why an extreme optimisation for one of the objective components is never common in practice. It is also interesting to note that the well-known Pareto principle, also known as the 80/20 principle, holds true. This is an empirical rule that states that around 80% of the results are generated by around 20% of the causes (Pareto, 1964).

Upon close examination of the solutions, it becomes evident that while the objective components *travel time* and *ton travel time* are concurrent, they offer various optimisation options. As elaborated in Chapter 2, minimising *travel time* results in a simultaneous reduction in *ton travel time*. However, there are alternative solutions, where *ton travel time* ($\beta = 100\%$) is purely minimised, resulting in an even lower *ton travel time*. Similarly, purely minimising *ton travel time* ($\beta = 100\%$) leads to a reduction in *travel time*, however, there are still solutions with even shorter *travel times*, by purely minimising *travel time* ($\alpha = 100\%$), given a slightly higher *ton travel time*. This relationship is visually depicted in Figure 6.2 and Figure 6.3c, where the distance between a *travel time*-optimised solution and a *ton travel time*-optimised solution becomes more pronounced as the weight coefficient of each of these objective components increases.



(a) *Travel time* vs. *vehicle temperature*

(b) *Ton travel time* vs. *vehicle temperature*

(c) *Travel time* vs. *ton travel time*

Figure 6.3.: The 3D Pareto-front plane of Figure 6.2 split into three 2D illustrations of the Pareto-front. For each subfigure, two objective components are represented on each the x- and y-axis, and one objective component is represented by the colour of the dot that represents a solution (the colour-coded bar indicates the objective component's value).
Subfigure 6.3a illustrates a comparison between *travel time* (x-axis), *vehicle temperature* (y-axis), and *ton travel time* (colour-coded bar).
Subfigure 6.3b illustrates a comparison between *ton travel time* (x-axis), *vehicle temperature* (y-axis), and *travel time* (colour-coded bar).
Subfigure 6.3c illustrates a comparison between *travel time* (x-axis), *ton travel time* (y-axis), and *vehicle temperature* (colour-coded bar).

The interaction between *travel time* and *ton travel time* is illustrated in Figure 6.3c. The two objective components are highly synchronised, exhibiting a clear linear regression between them. An increase in *travel time* results in an increase in *ton travel time*, and vice versa. The distribution of solutions narrows down as we move towards the lower range of *travel time* and *ton travel time*. This indicates that even small reductions in these two objective components in the critical range can lead to a significant increase in

vehicle temperature. Therefore, minimising *vehicle temperature* usually comes at the cost of a significant increase in *travel time* and *ton travel time*, as represented by the solutions in the upper right quadrant in Figure 6.3c.

A discrepancy between the presented solutions and the exact theoretical Pareto-front could exist. On the one hand, this would be due to the selection of a non-exact solution method, i.e., the ALNS, and on the other, it indicates that due to certain problem restrictions, feasible solutions do not exist at certain spots. A clear indicator of the latter case is illustrated by the use of the *Directional-Penalty*, which is discussed in the following Section 6.3.2, where an extremely restrictive, i.e. low, threshold value leads to a more scattered solution image. This means that there are even less feasible solutions along the seemingly smooth Pareto-front. In general, the more constraints are applied, the more irregular the Pareto-front becomes.

6.3.2. Directional-Penalty Affecting Solution and Solution Space

The implementation of the *Directional-Penalty*, which is discussed in detail in Section 5.3.3, when strictly enforced, significantly alters the solution in the three-dimensional space. Compared to the solutions shown in Figure 6.2, alternative solutions are presented in Figure 6.4, in which the threshold value for the exceedances is not set to 100%, but to the

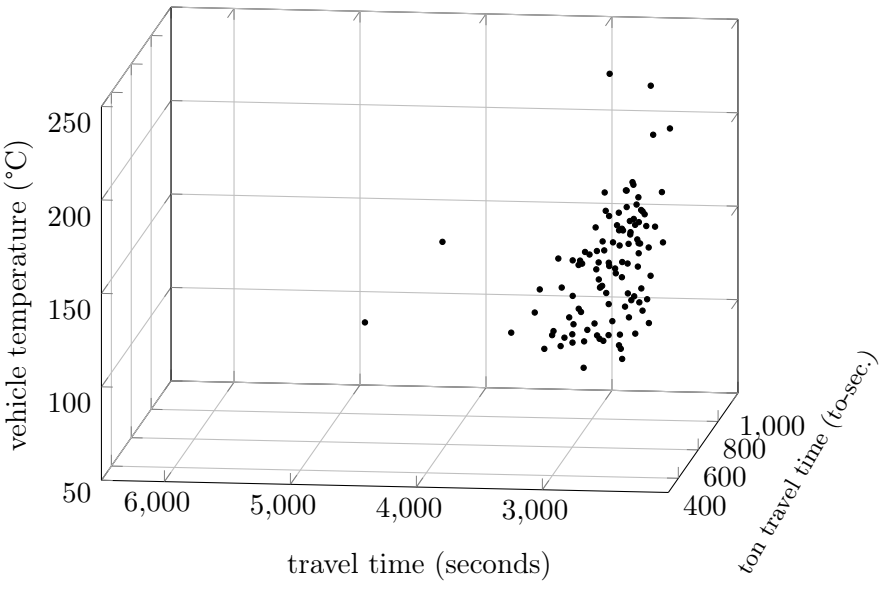


Figure 6.4.: 3D illustration of the Pareto-front plane of an instance of instance set C, consisting of 50 requests $|R|$, 3 vehicles $|K|$ and 10 vertices $|V|$. The Pareto-front plane solutions are obtained under the consideration of the *Directional-Penalty*. The threshold is set to 100% which is considered to be a low penalty. Each dot represents one solution. In total, 111 solutions are visible, making up the Pareto-front plane. The x-, y-, and z-axes represent each objective component.

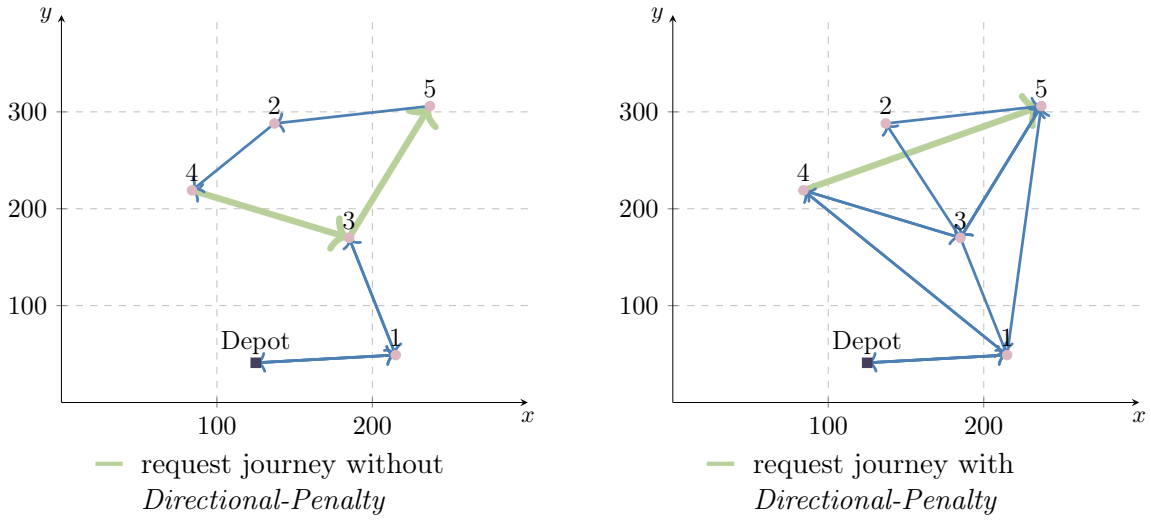
highest penalty threshold value, which is 0%, meaning that any movement that indicates a return of a request is penalised. This results in the solutions being much more spread out in the space. However, it is important to emphasise that this dispersion does not necessarily mean a deterioration in solution quality. As mentioned in Section 6.1.1, the more scattered solutions – compared to the ones, which are calculated with a 100% threshold value, presented in Figure 6.2 – that seem distant are not necessarily suboptimal solutions. This is especially true due to the *Directional-Penalty*'s strict constraint. Particularly interesting are the solutions that have a significantly hotter *vehicle temperature*, which are shown in the upper right quadrant in Figure 6.4. Since colder requests are normally exploited to lower the average temperature of a transportation stack, these can be heavily penalised with a very restrictive use of the *Directional-Penalty* - i.e. a very low threshold value, resulting in an overall hotter solution. One disadvantage of an incorrectly set threshold value is that it can also impair other objective function components. Since the *Directional-Penalty* always checks each request's journey – and possibly penalises them –, it can lead to partial routes being penalised in the case of pure *travel time* ($\alpha = 100\%$) optimisation, where requests are transported over longer distances, since they would be "on the way" anyway. Also, return movements in horizontal and/or vertical directions would be then inevitable, which would result in unnecessary penalisation of a specific request. As an example, the green marked route in Subfigure 6.5a presents a solution[1] where a request is transported from vertex 4 via vertex 3 to vertex 5. This solution is a pure *travel time* ($\alpha = 100\%$) optimisation, where no *Directional-Penalty* is used. It results in an overall short *travel time* of the route, which is 1101 seconds, but a longer *travel time* for this specific request's journey, which is 258 seconds. Compared to the solution in Subfigure 6.5b, which is also a pure *travel time* ($\alpha = 100\%$) optimisation but with the use of the *Directional-Penalty*. In this solution, the threshold value, which is explained in detail in Section 5.3.3, is set to 0%, meaning that every return movement of a request is penalised. This results in an overall longer *travel time* of the route, which is nearly double (2090 seconds) compared to the solution, where no *Directional-Penalty* is used. Nevertheless, the *travel time* of the specific request decreased by 82 seconds. This example also demonstrates that penalising this specific request would not have been quite necessary, as the arc between vertices 4 and 3 and the arc between vertices 3 and 5 are already being used by other requests anyway.

In contrast, Figure 6.6 exemplifies a request within the same solution as in Figure 6.5, which has a very long *strolling trip* without the use of the *Directional-Penalty* (Subfigure 6.6a). It is transported from vertex 1 via vertices 3 and 4, bypasses vertex 3 again and finally reaches vertex 5, resulting in a *travel time* of 495 seconds for this request. Compared to the solution, where the *Directional-Penalty* is used, the same request is transported directly from vertex 1 to vertex 5. Even though all arcs used for this request of the one solution (where the *Directional-Penalty* is not used) are still used in the other solution

[1]This specific solution is not represented in our test instances. It is used to demonstrate the effect of using the *Directional-Penalty*. This particular instance has 6 vertices $|V|$, 10 requests $|R|$ and 1 vehicle $|K|$. All other settings, regarding the distribution of vertices and dimensions of the requests, match the test instance settings.

(where the *Directional Penalty* is used), a practitioner would rather choose the second option in this specific case because it preserves the vehicle in the long run and reduces fuel consumption.

In general, the integration of the *Directional-Penalty* can be viewed as incorporating a fourth objective into the objective function[2], with the purpose of minimising *strolling trips*, discussed in Section 6.4.2.1. The resulting *strolling trips* provide a good guideline for setting the threshold value, which is explained in Section 6.4.2.

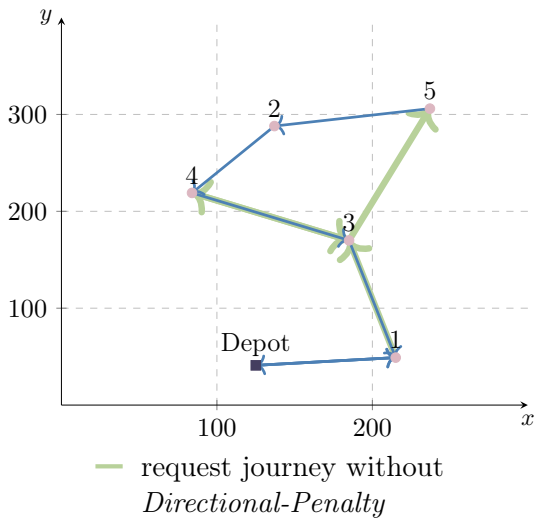


(a) *Travel time* of the vehicle route: 1101 sec.
Travel time of the request: 258 sec.

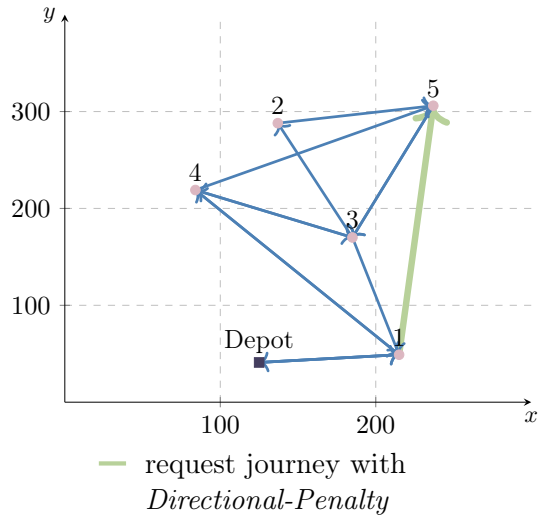
(b) *Travel time* of the vehicle route: 2090 sec.
Travel time of the request: 176 sec.

Figure 6.5.: Each subfigure illustrates a solution to the same example instance (a small random example), wherein the solution of Subfigure 6.5a no *Directional-Penalty* is used and in the solution of Subfigure 6.5b a strongly weighted *Directional-Penalty* is used. The blue arrows indicate the vehicle route. The green arrows emphasise the journey of an individual request as part of the vehicle's route. The comparison of both solutions exemplifies the impact of the *Directional-Penalty* particularly when it is as strongly weighted as possible, i.e., the threshold is set to 0%. Even though both solutions are pure *travel time* ($\alpha = 100\%$) optimisations, i.e., *ton travel time* and *vehicle temperature* are neglected, the use of the *Directional-Penalty* allows for a short journey of the emphasised request (see green arcs) while it results in a nearly double increase in total *travel time* for the rest of the route (see blue arcs), which indicates excessive use of the *Directional-Penalty*

[2]This objective, although secondary in our research, is treated as a regular constraint.



(a) The green marked solution represents a request journey calculated without any *Directional-Penalty*, resulting in a shorter *travel time* of the whole solution, but with more *strolling trips*.
Travel time of vehicle route: 1101 sec.
Travel time of request: 495 sec.



(b) The green marked solution represents a request journey calculated using *Directional-Penalty* with a 0% threshold value, resulting in a longer *travel time* of the whole solution, but with less *strolling trips*.
Travel time of vehicle route: 2090 sec.
Travel time of request: 258 sec.

Figure 6.6.: Each subfigure illustrates a solution to the same example instance (a small random example), wherein the solution of Subfigure 6.6a no *Directional-Penalty* is used and in the solution of Subfigure 6.6b a strongly weighted *Directional-Penalty* is used. The blue arrows indicate the vehicle route. The green arrows emphasise the journey of an individual request as part of the vehicle's route. The comparison of both solutions exemplifies the impact of the *Directional-Penalty* particularly when it is as strongly weighted as possible, i.e., the threshold is set to 0%. Even though both solutions are pure *travel time* ($\alpha = 100\%$) optimisations, i.e., *ton travel time* and *vehicle temperature* are neglected, the use of the *Directional-Penalty* allows for a short journey of the emphasised request (see green arcs) while it results in a nearly double increase in total *travel time* for the rest of the route (see blue arcs). Although these subfigures provide an example of a case of an important penalisation of a request, as this solution, which does not use the *Directional-Penalty*, has a long, and in practice not executable, *strolling trip*.

6.4. Solution Analysis

This section investigates the three-dimensional solution space (see Section 6.3) and its individual solutions. In particular, the behaviour and effects of the different objective

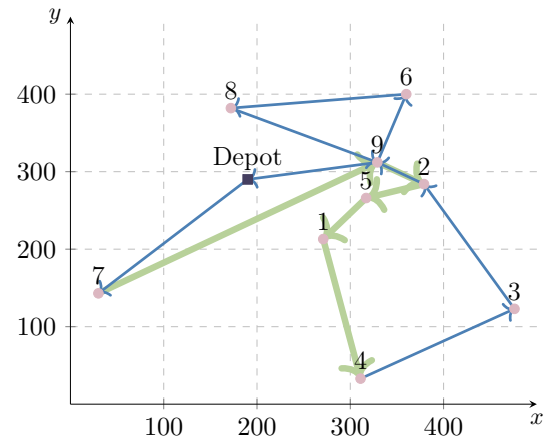
function component weightings, which are calculated using the WSA, are discussed. This approach has been elaborated in detail in Section 5.1. We also examine side effects of the *Vehicle Temperature Prediction method* and the resulting *item strolling effect*.

6.4.1. Detailed Analysis of Single Solutions

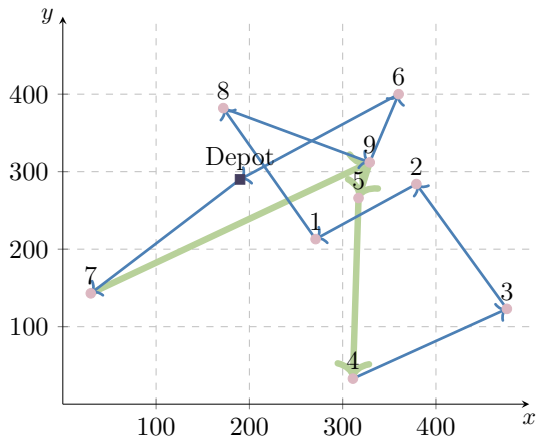
Figure 6.7 showcases four distinct solutions of an instance of the smaller test instance set A (Section 6.2). This specific instance was selected in order to showcase the solution structure when applying particular weights to the three objective components, expressed as the weighting coefficients α, β and γ, discussed in Chapter 2. Its objective component values can be found in Table 6.2. This instance consists of 10 requests $|R|$, 1 vehicle $|K|$, and 10 vertices $|V|$. The variation in the importance of each objective component weight demonstrates how shifting focus from these individual objective components can affect the outcome. Figure 6.7a optimises *travel time* by weighting it with 100% ($\alpha = 100\%$), disregarding both *ton travel time* ($\beta = 0\%$) and *vehicle temperature* ($\gamma = 0\%$), which represents the generalised CVRPDP. This results in a solution with minimum *travel time* (1964 seconds) where *ton travel time* is also reduced, as previously addressed in Section 6.3.1 using a larger example of an instance. Compared to the optimisation in Figure 6.7b (*ton travel time* optimisation), Figure 6.7c (*vehicle temperature* optimisation), and Figure 6.7d (mixed optimisation), *travel time* is reduced by 2.3% (2009 seconds), 46.3% (2873 seconds), and 5.9% (2080 seconds), respectively.

Compared to the pure *vehicle temperature* ($\gamma = 100\%$) optimisation in Figure 6.7c and Table Row 3, the *vehicle temperature* is 129.71% (77.1° C) higher if pure *travel time* ($\alpha = 100\%$) is optimised (Figure 6.7a, Table Row 1) at 177.12° C. As opposed to the pure *travel time* ($\alpha = 100\%$), pure *vehicle temperature* ($\gamma = 100\%$), and mixed optimisation, the pure *ton travel time* ($\beta = 100\%$) optimisation in Figure 6.7b and Table Row 2 results in a 0.6% (310.28 to-sec.), 40.4% (432.97 to-sec.), and 5.9% (329.62 to-sec.) lower *ton travel time*, respectively.

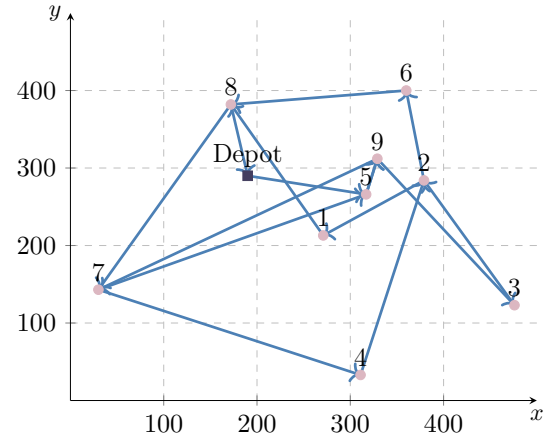
Figure 6.7d presents a mixed optimisation variant where the *travel time* and *vehicle temperature* objective components are each optimised by 35%, and the *ton travel time* objective component is optimised by 30%. As explained in detail in Section 6.3.1, this kind of solution is particularly relevant for practitioners as the reduction in *vehicle temperature* simultaneously optimises vehicle wear by reducing *ton travel time* and *travel time* as a good compromise. In a numerical comparison, the *travel time* is only extended by 5.6% (2080 seconds) compared to the pure *travel time* ($\alpha = 100\%$) objective component optimisation, the *ton travel time* is only extended by 6.5% (329.63) compared to the pure *ton travel time* ($\beta = 100\%$) objective component optimisation, and the *vehicle temperature* is extended by 34.5% (117.68° C) compared to the pure *vehicle temperature* ($\gamma = 100\%$) objective component optimisation.



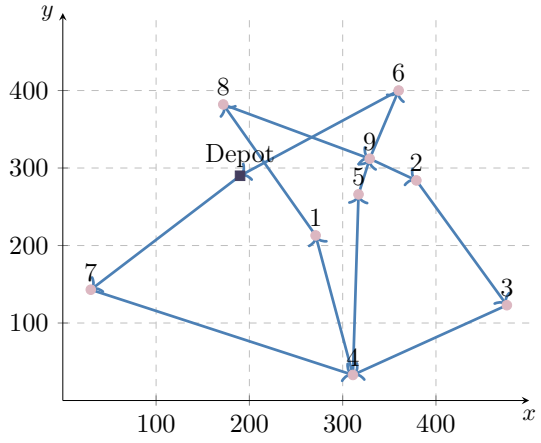
(a) $\alpha = 100\%$ (*travel time*)
$\beta = 0\%$ (*ton travel time*)
$\gamma = 0\%$ (*vehicle temperature*)



(b) $\alpha = 0\%$ (*travel time*)
$\beta = 100\%$ (*ton travel time*)
$\gamma = 0\%$ (*vehicle temperature*)



(c) $\alpha = 0\%$ (*travel time*)
$\beta = 0\%$ (*ton travel time*)
$\gamma = 100\%$ (*vehicle temperature*)



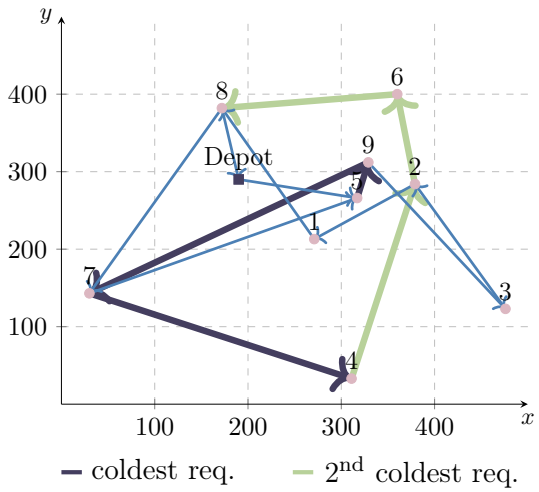
(d) $\alpha = 35\%$ (*travel time*)
$\beta = 30\%$ (*ton travel time*)
$\gamma = 35\%$ (*vehicle temperature*)

Figure 6.7.: Each subfigure illustrates different solutions for the same example instance of instance set A, where for each solution, different objective weight coefficients (α, β and γ) were applied as part of the WSA (see Chapter 2 and Section 5.1). The blue arrows indicate the vehicle route. The green arrows emphasise the journey of an individual request as part of the vehicle's route used to exemplify the *ton travel time* objective component.

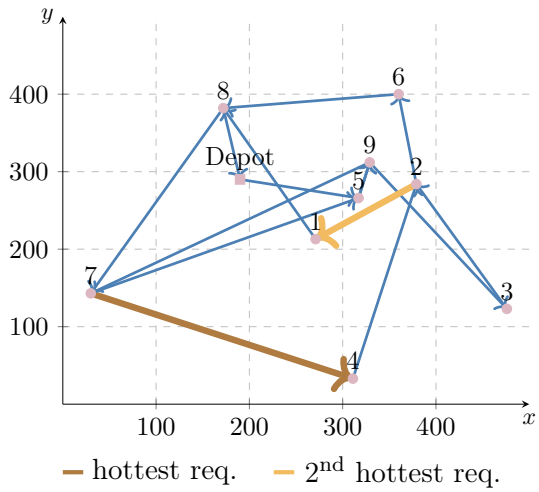
Objective Component Weight Coefficients				Objective Component Values		
Figure	$\boldsymbol{\alpha}$ **Travel Time (in %)**	$\boldsymbol{\beta}$ **Ton Travel Time (in %)**	$\boldsymbol{\gamma}$ **Vehicle Temperature (in %)**	**Travel Time (in sec.)**	**Ton Travel Time (in to-sec.)**	**Vehicle Temperature (in °C)**
6.7a	100	0	0	1964	310.28	177.12
6.7b	0	100	0	2009	308.32	163.75
6.7c	0	0	100	2873	432.97	77.1
6.7d	35	30	35	2080	329.66	117.68

Table 6.2.: Selected objective weight coefficients (left) and resulting objective component values (right) for the example instance of test instance set A. This Instance consists of 10 requests $|R|$, 1 vehicle $|K|$ and 10 vertices $|V|$. The solutions are illustrated in Figure 6.7.

With regard to the *ton travel time*, it has already been pointed out in previous sections that it is concurrent to *travel time* but still pursues a distinctively different goal. The minimisation of *ton travel time* aims to minimise heavy and long transports to improve fuel efficiency and reduce the wear and tear on vehicles in the long term. This proves to be relevant in practice to reduce the phenomenon of so-called *strolling trips*, which is discussed in more detail in Section 6.4.2.1, without increasing the total *travel time* to an excessive extent. Figures 6.7a and 6.7b illustrate the journey of the heaviest requests from its pickup vertex to its delivery vertex using the bold blue marked route section. In contrast to the pure *travel time* ($\alpha = 100\%$) objective component optimisation, in which the heaviest item is transported from vertex 7 via vertices 9, 2, 5 and 1 to vertex 4, in the pure *ton travel time* ($\beta = 100\%$) objective component optimisation the transport takes place exclusively from vertex 7 via vertex 9 and 5 to vertex 4. This adjustment results in a *travel time* reduction of the request by 19.8% (−154 seconds). The total *travel time* of the solution in which only the *ton travel time* is optimised is solely 2.3% (+45 seconds) longer than the solution in which only the *travel time* objective component is optimised.



(a) *Travel time* of the coldest request: 693 sec.
Travel time of the 2nd coldest request: 567 sec.



(b) *Travel time* of the hottest request: 302 sec.
Travel time of the 2nd hottest request: 129 sec.

Figure 6.8.: The solution of 6.7c, where α=β=0% and γ=100%, is illustrated again, with the difference that the bold lines emphasise the journey of individual requests, i.e., the two coldest and two hottest requests, as part of the vehicle's route. Subfigure 6.8a exemplifies the side effect of the *Vehicle Temperature Prediction Method*, where colder requests perform *strolling trips*, meaning that they detour during the journey from their respective pickup vertex to their respective delivery vertex (see Section 6.4.2.1). Subfigure 6.8b exemplifies a desired result of the *Vehicle Temperature Prediction Method*, where hotter requests are directly transported from their respective pickup vertex to their respective delivery vertex.

6.4.2. Side Effects of the Vehicle Temperature Prediction Method

Chapter 4 has already provided an introduction to how the *Vehicle Temperature Prediction Method* works, and side effects were also briefly addressed. In this section, these effects, including the *item strolling effect*, are explained in more detail.

The *Vehicle Temperature Prediction Method* is based on Newton's Law of Cooling (NLC), which takes into account not only the cooling of the requests but also the simultaneous heating and cooling of the vehicle. Since this is a pure convective heat transfer (Section 4.2) from request to vehicle and environmental air as well as from vehicle to environmental air, not taking into account the heat transfer between the requests leads to a certain inaccuracy. Although this inaccuracy is only marginal and not impactful to this research, it is nevertheless present. This means that if requests in a stack have very different temperatures, the actual temperature of the heated/cooled requests is not transferred over several distances on the route. With this approach, e.g., if a very cold request is

transported together with many hot requests, it is expressed in the calculation as an unchanged cold temperature, i.e., its temperature remains constant in the course of the transportation, and it lowers the overall temperature of the stack infinitely long. However, this does not hold true in reality. As already mentioned, this - in reality prevalent - heat transfer effect is negligible in our case[3], but it means that colder requests are exploited to lower the overall temperature of the transportation stack, which in turn leads to a lower *vehicle temperature*. As a result, especially colder requests are carried over longer distances. We refer to this effect as the *item strolling effect*.

6.4.2.1. Item Strolling Effect

The *item strolling effect* is, among other reasons, also a consequence of the *Vehicle Temperature Prediction Method*. We refer to this effect specifically to trips where requests are transported over long distances in order to achieve a certain secondary goal.

This effect occurs most frequently when purely minimising the *vehicle temperature* ($\gamma = 100\%$) objective component. By calculating the vehicle's heating during transportation based on the mass-average temperature of the requests, colder requests can lead to a reduction in this. As a result, cooler requests are preferred in this type of optimisation. Following the solution structure, it can be observed that cooler requests stroll around, hence resulting in long request trips from the pickup vertex to the delivery vertex. In comparison, hotter requests are quickly transported from their pickup to their delivery vertex, hence resulting in short request trips. This is illustrated in Figure 6.8 and discussed in Section 6.4.1.

Another case that can lead to more extended item trips is the pure optimisation of the *travel time* ($\alpha = 100\%$) objective component. As the sole transportation weight and *vehicle temperature* are irrelevant in this type of optimisation, requests are often carried on longer trips because other requests are picked-up or delivered along the way. Although this helps to minimise the overall *travel time*, it also increases vehicle wear and tear and fuel consumption. As already mentioned, neglecting the *vehicle temperature* objective component also leads to an increase in it.

As all three—the increase in vehicle temperature, the increase in wear and tear, and the increase in fuel consumption—are not desired, the *item strolling effect* there are also time windows that must be obeyed. However, the time window constraint is neglected in this research, as it would interfere with the fundamental research in temperature-dependent multi-objective optimisation.

Two different methods are used in this research to control the right amount of *strolling trips*. Firstly, the third objective function component, i.e. *ton travel time*, inherently minimises this effect. This is achieved by reducing the total transport weight along the transport routes. Secondly, the *Directional-Penalty* constraint, discussed in Section 5.3.3, is applied, which specifically takes into account individual request movements and penalises, depending on the threshold value setting, those that indicate a return trip to the proximity

[3]However, this approach only works if the heat transfer between the requests, which are transported together, does not exceed the permitted deviation from the actual temperature within the usual delivery time of a request.

of previously visited vertices. The number of *strolling trips* can be regulated by adjusting the threshold value used in the *Directional-Penalty*. As discussed in Section 5.3.3, a lower threshold value indicates a stricter constraint. As mentioned, the latter method could also be introduced as a fourth objective function component instead of a constraint. However, the disadvantage of using it as the fourth objective function component would be that weighting the individual objective function components would be more difficult when using the WSA.

6.5. On the Performance of the Solution Method

All 200 instances of our test instance sets are solved. The performance of our solution method is shown in the Tables 6.4a, 6.4b, 6.4c and 6.4d. A summary of the key results is presented in Table 6.3.

Since we weigh and calculate each objective component of these instances in 10% steps, there are a total of 111 different permutations per instance after subtracting the permutations, which lead to identical solutions. In practice, 111 different permutations would not be calculated for each problem scenario regarding the different weighting coefficients of the objective function components because it cannot be done in a reasonable amount of computational time. When calculating a single permutation, i.e. a single specific weighting, $10,000$ iterations – which is the second *Stopping Criterion*, discussed in Section 5.3.2.1 – would only take 45 minutes on average, which is reasonable in practice for the instance size of instance Set D. With regard to the calculation time, 111 different permutations were calculated in parallel for each instance as part of these test instance sets. This results in an average calculation time of 12.74 seconds per permutation across all instance sets. The calculation would be significantly faster if a calculation node does not calculate 111 permutations in parallel but rather $40 - 50$ permutations.

The calculation of all 200 Instances was carried out on the Vienna Scientific Cluster 4 (VSC4), using a total of 200 calculation nodes. Each of these calculation nodes has 2 Intel® Xeon® Platinum 8174 processors, each with 24 cores and 48 threads, totalling 96 threads per node. This means that 0.86 threads are available for each permutation. Even if only 0.86 threads per permutation could be used, the total calculation time is only marginally longer, as some permutations are stopped by the *Stopping Criterion* at a very early stage and further permutations can be calculated. Due to the high computing intensity, twice as many calculation nodes could reduce the calculation time by around $30 - 40\%$. This is due to the fact that Central Processing Unit (CPU) performance decreases exponentially with increasing load per CPU-core. As the calculation of all instances is very computationally expensive, it was not possible to access more than 200 nodes at the time of calculation. Only a doubling of the number of calculation nodes would lead to comparable results, but this amount is not possible. In other words, if all CPUs of a node are used instead of half, the calculation is not accelerated by double, but only marginally. Considering the 2 Intel® Xeon® Platinum 8174 processors, the following reasons could explain why full utilisation of these CPUs does not necessarily lead to the best performance (Hennessy and Patterson, 2011; Intel Corporation, 2017):

- **NUMA architecture:** The Intel® Xeon® Platinum 8174 processors use a NUMA architecture in which each CPU accesses its own memory area. If a task is distributed across multiple CPUs and the memory access is not local, additional overhead is created by the exchange of data between the memory areas. This can affect overall performance.

- **Turbo Boost and Thermal Throttling:** The Intel® Xeon® Platinum 8174 processors feature Turbo Boost technology, which increases the clock frequency when only a few cores are active. When all cores are fully utilised, the clock frequency can be reduced in order to control heat development. This can reduce the performance per core when all cores are fully utilised.

- **L3 Cache and interconnect:** The processors share a common L3 cache and use an interconnect (such as Intel® Ultra Path Interconnect) to exchange data between the CPUs. Full utilisation of all cores can lead to latency and bottlenecks when accessing the L3 cache or exchanging data, which can affect overall performance.

- **Memory bandwidth and latency:** Full utilisation of all cores can place a heavy load on the memory bandwidth and lead to increased latency when accessing memory. This can be particularly problematic if the workload requires many memory accesses or the memory bandwidth is limited.

- **Scheduler-Overhead and Thread-Scheduling:** The operating system and the scheduler must distribute the threads efficiently and allocate them to the CPUs. A high number of threads can cause additional overhead, especially if the thread scheduling decisions are not optimal.

For the smallest instance with 10 requests $|R|$, an average improvement of 2.78% was achieved when comparing the solution of the initial solution using CI and the ALNS solution. The largest improvement was 21.36%. As the instance size increases, the possible solution space also increases, whereby the larger instances achieve a higher average improvement of 18.27%, 23.49% and 22.66%, respectively.

A comparison of the results for different instance sets shows that the requests in sets $B - D$ traverse approximately the same number of arcs (Table 6.3, Column 7 - "Average # Arcs per Request"), which indicates that the proportions between the number of requests and vehicles per set, as well as the distances of the locations of the vertices are evenly distributed. Only set A has a lower average number of arcs traversed by a request, shown in Table 6.3, Column 7 - "Average # Arcs per Request". This is due to the fact that the distribution of requests for the small set is designed differently, as already explained in Section 6.2. Since in instance set A, every vertex must hold at least one request and there are only 10 requests and 9 vertices (besides the depot vertex (v_0), which does not hold any requests), almost every vertex holds only a single request. Compared to instance sets $B - D$, this leads to a lower possibility of transporting multiple requests on one route, which in turn leads to a decrease in the number of arcs that a request traverses.

The same applies to the average *travel time* per request, shown in Table 6.3 Column 8. As there are fewer requests to be served in instance set A, these requests generally have to travel longer distances. However, instance sets $B - D$ again show a very similar average *travel time* that a request has to cover, which indicates an even distribution of the locations of the vertices.

The fact that the average improvement of the *quality value* between CI and ALNS, which is expressed as "Average objective Gap (CH to)" in Table 6.3 Column 10), is lower for the largest instance (Instance Set D) than for the second largest instance set (Instance Set C) is due to the fact that the total computing time was limited to 3 days for the calculation of an instance and its Pareto-front, which is the third *Stopping Criterion* discussed in Section 5.3.2.1. As a result, the full desired $10,000$ iterations (second *Stopping Criterion*) cannot be reached by the instances, which turned out to be an appropriate number for the desired solution quality. As a result, the solution quality, in terms of the quality value, for the largest instance is slightly lower than for the other instances. These quality losses can basically manifest themselves in three ways. Firstly, it is the GAP between the CH and ALNS, shown in Table 6.3 Column 10. Secondly, it is the three-dimensional representation of the solution as a Pareto-front, where a strongly scattered solution could indicate a loss in quality. Thirdly, it is possible to determine the solution quality by checking at which iteration the last improvement was found, which can be seen in Tables 6.4 Column 12. If the ALNS algorithm is not stopped by the first *Stopping Criterion*, which sets the limit at 3000 iterations without improvements, but rather by the second or third, it is highly likely that improvements can still be achieved in the overall solution.

Name	**# Requests**	**# Vehicles**	**Total # Iter.**	**Average Total CPU Time per Inst. (in sec.)**	**Best Sol found after (in sec.)**	**Average # Arcs per Request**	**Average travel time per Request (in sec.)**	**Best objective Gap (CH to ALNS) (in %)**	**Average objective Gap (CH to ALNS) (in %)**	**Average CPU time per iteration (in sec.)**
A	10	1	4145.39	913.07	101.59	2.02	90.22	-21.36	-2.78	0.22
B	30	2	8570.47	36101.42	25232.99	2.44	74.29	-43.08	-18.27	4.23
C	50	3	9879.73	173338.98	161277.03	2.42	74.92	-183.94	-23.49	16.42
D	70	4	8304.30	250203.27	225101.01	2.36	76.34	-229.63	-22.66	30.1
Average		**7,724.97**	**115,139.19**	**102.928,16**	**2.31**	**78.94**	**-119.50**	**-16.80**	**12.74**	

Table 6.3.: Highlighted key results on 10-, 30-, 50-, 70-request instance sets using ALNS with CI

	Travel Time (in sec.)	**Vehicle Temp. (in °C)**	**Ton Travel Time (in to-sec.)**	**Quality Value**	**Total # Iter.**	**Average CPU Time per Iter. (in sec.)**	**CH CPU Time (in sec.)**	**ALNS CPU Time (in sec.)**	**Total CPU Time (in sec.)**	**Best Sol. found after (in sec.)**	**Last Improv. of Inc. Sol. at Iter.**	**Iter. Without Improv.**	**Average # Arcs per Request**	**Average Travel Time per Request (in sec.)**	**GAP between CH and ALNS (in %)**
Min.	1429,00	13,35	194,10	577,41	3000,00	0,09	6,00	285,00	292,00	0,00	846,00	835,00	1,30	45,14	0,00
Max.	5592,00	272,31	837,92	3012,63	10000,00	0,76	56,00	4085,00	4115,00	2660,00	9812,00	3000,00	3,08	199,56	21,36
$\bar{x}$	2332,11	111,21	347,66	2129,46	4145,39	0,22	13,47	899,60	913,07	101,59	3694,93	2998,62	2,02	90,22	2,78
$\bar{\sigma}$	270,95	26,53	42,12	255,15	997,56	0,03	3,12	181,56	182,20	191,50	589,48	20,10	0,19	11,46	2,72
$\tilde{x}$	2244,02	111,88	332,69	2219,10	3885,12	0,22	12,90	907,10	920,88	29,60	3587,94	3000,00	2,01	88,57	2,13
MAD	152,87	21,45	26,14	173,42	672,97	0,02	2,43	135,31	135,82	115,66	403,24	2,73	0,14	7,77	1,99
Q_1	2213,34	93,58	325,58	2065,88	3526,59	0,20	11,20	783,78	797,29	6,39	3305,27	3000,00	1,90	83,65	0,86
Q_3	2364,24	127,54	353,49	2288,78	4481,76	0,24	15,34	1005,19	1019,17	104,82	3965,46	3000,00	2,13	95,00	4,01
IQR	150,90	33,96	27,91	222,90	955,17	0,04	4,14	221,41	221,88	98,43	660,19	0,00	0,23	11,35	3,15

(a) Averages of all 50 solutions from test instance set A. Per instance 10 requests $|R|$ are processed by 1 vehicle $|K|$ on 10 vertices $|V|$.

	Travel Time (in sec.)	**Vehicle Temp. (in °C)**	**Ton Travel Time (in to-sec.)**	**Quality Value**	**Total # Iter.**	**Average CPU Time per Iter. (in sec.)**	**CH CPU Time (in sec.)**	**ALNS CPU Time (in sec.)**	**Total CPU Time (in sec.)**	**Best Sol. found after (in sec.)**	**Last Improv. of Inc. Sol. at Iter.**	**Iter. Without Improv.**	**Average # Arcs per Request (in sec.)**	**Average Travel Time per Request (in sec.)**	**GAP between CH and ALNS (in %)**
Min.	1467,00	43,21	247,16	782,89	3045,00	2,24	123,00	8805,00	8995,00	18,00	1,00	0,00	1,53	42,02	0,27
Max.	8714,00	341,23	1092,78	3020,57	10000,00	14,28	924,00	133800,00	134478,00	126394,00	9652,00	3000,00	3,18	207,48	43,08
$\bar{x}$	2378,53	128,11	380,21	2249,97	8570,47	4,23	220,47	35880,94	36101,42	25232,99	2911,86	2201,69	2,44	74,29	18,27
$\bar{\sigma}$	567,00	29,75	68,84	212,38	1650,26	0,45	34,37	6624,18	6628,57	11130,25	1443,41	972,49	0,17	9,58	5,66
$\tilde{x}$	2281,34	124,66	363,90	2306,02	9137,60	4,22	217,16	37419,58	37630,68	26157,52	3187,06	2591,54	2,43	73,45	18,21
MAD	190,37	21,65	28,78	136,03	1337,17	0,33	26,70	4995,87	4999,02	9226,62	1090,31	818,98	0,12	5,43	4,24
Q_1	2221,60	109,14	356,12	2192,60	7617,43	3,96	195,72	32519,32	32735,35	17088,13	2022,89	1588,59	2,35	69,40	14,85
Q_3	2381,11	142,68	381,89	2376,53	9883,49	4,49	241,86	40423,32	40641,48	34292,02	3786,80	2956,42	2,54	77,52	21,82
IQR	159,51	33,54	25,76	183,93	2266,06	0,52	46,14	7904,00	7906,13	17203,89	1763,91	1367,83	0,20	8,12	6,97

(b) Averages of all 50 solutions from test instance set B. Per instance 30 requests $|R|$ are processed by 2 vehicle $|K|$ on 10 vertices $|V|$.

	Travel Time (in sec.)	Vehicle Temp. (in °C)	Ton Travel Time (in to-sec.)	Quality Value	Total # Iter.	Average CPU Time per Iter. (in sec.)	CH CPU Time (in sec.)	ALNS CPU Time (in sec.)	Total CPU Time (in sec.)	Best Sol. found after (in sec.)	Last Improv. of Inc. Sol. at Iter.	Iter. Without Improv.	Average # Arcs per Request (in sec.)	Average Travel Time per Request (in sec.)	GAP between CH and ALNS (in %)
Min.	1664,00	41,33	272,36	1016,19	3002,00	7,32	362,00	26820,00	28212,00	9,00	1,00	0,00	1,56	45,29	0,02
Max.	10444,00	355,79	1233,00	3899,87	10000,00	24,32	8970,00	243135,00	248511,00	233880,00	8459,00	3000,00	3,07	183,30	183,94
$\bar{x}$	2522,07	134,22	406,73	2670,36	9879,73	17,38	1356,21	171982,77	173338,98	161277,03	898,20	706,22	2,42	74,92	25,51
$\bar{\sigma}$	726,24	31,92	80,31	192,51	662,31	1,57	528,04	20793,41	20762,49	26993,34	876,34	606,36	0,16	10,11	13,28
$\tilde{x}$	2399,70	127,49	388,48	2693,48	9975,04	17,64	1309,14	175794,32	177170,86	165973,20	717,24	608,92	2,43	73,83	23,19
MAD	256,88	22,14	35,42	131,24	179,59	1,14	382,40	12739,25	12699,82	16541,65	566,86	418,18	0,11	5,28	7,75
$\bar{Q}_1$	2298,70	114,61	375,44	2583,66	9960,84	16,59	988,63	165272,69	166569,50	152822,10	347,00	300,52	2,34	70,34	18,91
$\bar{Q}_3$	2567,71	145,25	413,85	2791,29	9984,97	18,40	1649,41	183241,14	184491,94	176405,97	1136,56	947,80	2,51	77,67	28,86
IQR	269,01	30,65	38,41	207,63	24,13	1,81	660,78	17968,45	17922,44	23583,87	789,56	647,28	0,17	7,34	9,94

(c) Averages of all 50 solutions from test instance set C. Per instance 50 requests $|R|$ are processed by 3 vehicle $|K|$ on 10 vertices $|V|$.

	Travel Time (in sec.)	Vehicle Temp. (in °C)	Ton Travel Time (in to-sec.)	Quality Value	Total # Iter.	Average CPU Time per Iter. (in sec.)	CH CPU Time (in sec.)	ALNS CPU Time (in sec.)	Total CPU Time (in sec.)	Best Sol. found after (in sec.)	Last Improv. of Inc. Sol. at Iter.	# Iter. Without Improv.	Average # Arcs per Request (in sec.)	Average Travel Time per Request (in sec.)	GAP between CH and ALNS (in %)
Min.	1613,00	63,53	279,31	1537,66	3001,00	14,42	919,00	43417,00	44617,00	14,00	1,00	0,00	1,43	48,88	0,00
Max.	10242,00	346,48	997,85	4149,90	10000,00	45,06	16386,00	256572,00	257830,00	256194,00	5918,00	3000,00	3,03	162,00	229,63
$\bar{x}$	2750,83	141,61	436,55	2899,85	8304,30	30,10	2388,08	247815,18	250203,27	225101,01	836,24	715,38	2,36	76,34	22,66
$\bar{\sigma}$	802,30	33,25	83,74	222,73	1139,77	3,72	912,66	27158,50	26903,23	39142,64	758,18	617,70	0,15	9,85	14,43
$\tilde{x}$	2602,40	134,83	415,09	2922,80	8361,86	30,25	2419,96	254762,56	257222,66	232829,54	709,36	626,38	2,36	75,12	20,37
MAD	308,34	23,19	41,80	151,95	777,10	2,81	657,95	11749,50	11496,15	20924,78	510,84	428,85	0,10	5,34	7,99
$\bar{Q}_1$	2487,06	121,12	400,36	2799,28	7776,63	27,85	1722,99	252154,94	255114,79	220430,68	313,83	277,98	2,28	71,55	15,87
$\bar{Q}_3$	2799,55	154,49	443,40	3037,68	9027,18	32,50	2923,64	255653,34	257421,88	244284,98	1107,96	964,32	2,45	79,12	26,22
IQR	312,49	33,36	43,05	238,39	1250,55	4,65	1200,65	3498,40	2307,09	23854,30	794,13	686,34	0,16	7,57	10,35

(d) Averages of all 50 solutions from test instance set D. Per instance 70 requests $|R|$ are processed by 4 vehicle $|K|$ on 10 vertices $|V|$.

Table 6.4.: Detailed results on 10-, 30-, 50-, 70-request test instance sets using ALNS with CI as the CH and *Repair* operator. Minimum values ($Min.$), maximum values ($Max.$), avg. mean values ($\bar{x}$), avg. standard deviation ($\bar{\sigma}$), avg. median ($\tilde{x}$), avg. mean absolute deviation ($\bar{MAD}$), avg. first and third quartiles ($\bar{Q_1}$, $\bar{Q_3}$), and the avg. interquartile range ($\bar{IQR}$) are used to analyse the solution in detail. In the first three columns, the individual objective component values can be evaluated. *Quality Value*, which is discussed in Chapter 2, is the objective value resulting from the combination of various weightings of the objective component values, its relativisation and penalties. The column "Last Improv. of Inc. Sol. at Iter." displays at which iteration the last improvement of the incumbent solution occured. "# Iter. Without Improv" counts the number of iterations after the last improvement of the best-found solution until the *Stopping Criterion* is met (see Section 5.3.2.1.

6.6. Excursion: Results on the Dynamic-5-Request-Look-Ahead-Tree-Search

As part of an excursion, the results of the Dynamic-5-Request-Look-Ahead-Tree-Search (D-5R-LA-TS) and its strengths and weaknesses are presented. For better comparability, the same example instance from Section 6.3 is solved. The solution is also visualised in a three-dimensional space, as shown in Figure 6.9, and the performance of the D-5R-LA-TS is discussed.

6.6.1. Pareto-Front Results on the Dynamic-5-Request-Look-Ahead-Tree-Search

The application of the D-5R-LA-TS on the same test instances (Instance set C) as in Section 6.3, in which the solution generated with the ALNS was already analysed, shows a quite similar solution image. This suggests that the D-5R-LA-TS works effectively. However, a detailed analysis reveals that the solutions are on average 25% worse in terms of the achieved quality value, also shown as GAP in Table 6.5 Column 6. This discrepancy is explained in more detail in Section 6.6.2. Despite these differences, the D-5R-LA-TS offers a decisive advantage over ALNS or similar (meta)heuristics, as already explained in Section 6.6. By applying the D-5R-LA-TS, it is possible to solve dynamic problems, which means that continuous planning is seamlessly possible. This is particularly relevant for companies that do not carry out Full Truck Load (FTL) but rather Less Than Truck Load (LTL) transports and have continuous pick-up and delivery orders along the transport route. Nevertheless, it works with FTLs, but the results are expected to be significantly worse, in terms of the achieved quality value, than with ALNS.

As the interpretation is identical to the solution solved with ALNS, we refer to Section 6.3. Differences in solution structure compared to the ALNS solution are only highly visible when analysing the performance, which is discussed in more detail in the next section.

6.6.2. Dynamic-5-Request-Look-Ahead-Tree-Search Performance

The instance from the test instance set C is solved using three different settings. Instead of solving all 50 instances of this set, a single instance was chosen, which is due to the fact that calculating a single weight with 5 request look-ahead takes $4,854.78$ seconds on average. The calculation of all 111 permutations takes about 6 days.

As already explained in Section 6.6, the number of predictive requests can be defined. Based on this number, all possible combinations are determined, and the best one is selected. Computational experiments have shown that looking ahead for 5 requests leads to better planning and, therefore, a better solution than looking ahead for only 3 requests. This is clearly reflected in the performance as shown in Table 6.5, where the average gap between D-5R-LA-TS and ALNS, shown in Table 6.5 Column 6, when using 5 request look-ahead is only 24.67%. In some cases, it is even possible to achieve an identical solution to ALNS with 5 request look-ahead, which is denoted by 0%. However,

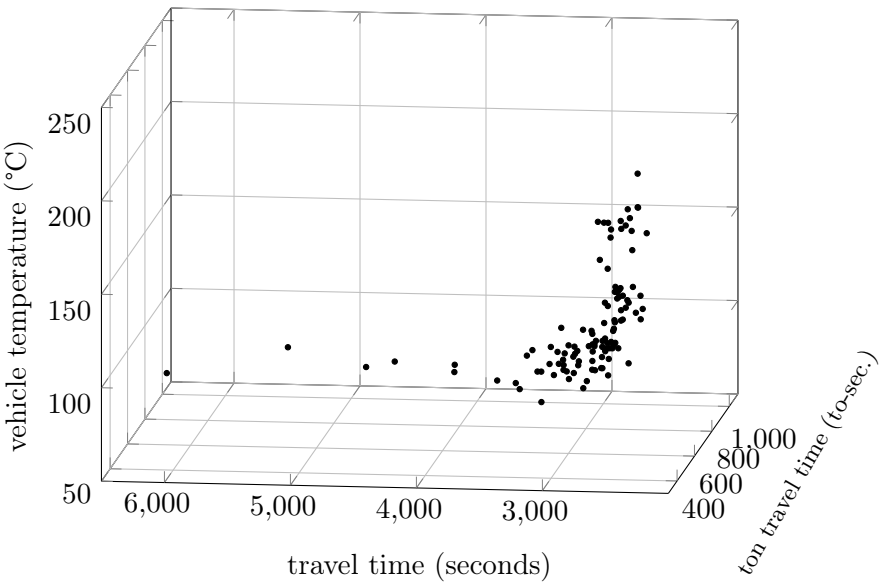


Figure 6.9.: 3D illustration of the Pareto-front plane of an instance of instance set C, consisting of 50 requests $|R|$, 3 vehicles $|K|$ and 10 vertices $|V|$. The Pareto-front plane solutions are generated with the D-5R-LA-TS. Each dot represents one solution. In total, 111 solutions are visible, making up the Pareto-front plane. The x-, y-, and z-axes represent each objective component.

it was not possible to find a better solution than the ALNS solution, which is expected, as ALNS is a metaheuristic, which is used to solve static problems. Unlike the ALNS, the D-5R-LA-TS is able to solve dynamic problems, which usually leads to worse solutions when solving static problem sets.

The interplay between calculation time and solution quality is clearly recognisable. The fewer requests are looked ahead, the faster each calculation is, but the poorer the solution quality. For example, if only 4 requests are looked ahead, the calculation is accelerated by 245.37%, but the solution quality deteriorates by 30.32%. If only 3 requests are looked ahead, the calculation speeds up again by 295.77%, but the solution quality deteriorates by a further 40.88%.

Another interesting finding is that as the number of request look-aheads increases, more arcs are traversed by requests on average. This is due to the fact that the route plans change more frequently during transport when more requests are looked ahead.

LA # of Requests	**Average Total CPU Time (in sec.)**	**Average # Arcs per Request**	**Average travel time per Request (in sec.)**	**Best objective Gap (CH to ALNS) (in %)**	**Average objective Gap (CH to ALNS) (in %)**	**Average CPU time per iteration (in sec.)**
5	4,854.78	2.41	79.21	0.00	24.67	93.63
4	1,457.50	2.39	75.33	3.25	32.15	27.11
3	212.13	2.27	74.45	12.37	45.29	6.85

Table 6.5.: Performance of the D-5R-LA-TS, introduced in Section 6.6 sorted by different numbers of Look Ahead (LA) of requests and comparison between the D-5R-LA-TS and ALNS. An instance of the test instance set C is solved that contains 50 requests $|R|$, 10 vertices $|V|$ and 3-vehicles $|K|$.

7. Conclusion

This paper presents a transportation logistics-specific problem identified in collaboration with a partner steel plant in Austria, titled the TD-CVRPDP, which represents a novel problem in the scientific field of CVRPDPs. A multi-objective approach for the TD-CVRPDP is presented, which aims to increase operational efficiency by minimising the *travel time* of transports, minimise fuel consumption and vehicle wear by reducing the *ton travel time*, and ensure operational safety by minimising the *vehicle temperature*. Hence, not only economic but also ecological and safety-related goals are being pursued.

New methods are presented to tackle the novel aspect of the TD-CVRPDP, including the *Vehicle Temperature Prediction Method*, which is based on NLC. Its aim is to predict the *vehicle temperature* as accurately as possible during optimisation. As the problem at hand is characterised by a continuous planning process in practice and efficient routes are calculated continuously, minor calculation deviations can occur, similar to the bullwhip effect known from supply chain management (Lee et al., 1997), which can lead to large temperature deviations over time. To counteract this, this method was developed, which guarantees an accurate prediction of the vehicle temperature without requiring excessive computing power.

Moreover, a new penalty function, the *Directional Penalty*, is introduced as part of this multi-objective optimisation. Its aim is to efficiently counteract the discovered *item strolling effect* by monitoring individual item journeys and thus penalising movements that indicate a backward directional movement. This effect is contradictory to usual planning habits and is of particular relevance as managerial insight. Empty trips and *strolling trips* are intuitively bad for planning, but in fact, they are revealed to be desirable in the context of the *vehicle temperature* objective. While empty trips aim to cool down the vehicles on routes where nothing is loaded, *strolling trips* use colder items as "cooling elements" by reducing the average temperature of the item transport batch. However, both behaviours should not occur to an excessive extent as they can affect *travel time* and *ton travel time*. It is recommended that both empty trips and *strolling trips* should be allowed, but as they have contradicting advantages and disadvantages, they should be carefully evaluated and serve as key performance indicators.

In formulating our objective function, we chose the WSA, in which the various objective function components are assigned with a specific weight to enable an unbiased comparison. Those objective components are also relativised to guarantee a fair basis. This, in turn, helps balance the individual objective function components during optimisation and the trade-offs become more apparent. It was shown that even minor changes to the weightings of the WSA could significantly impact the solutions generated, enabling more precise tuning of the objective priorities. This flexibility also allows slight changes in the optimisation goals, such as a slightly increased prioritisation of the *vehicle temperature*,

to be taken into account appropriately.

However, the application of the WSA also depends on the size of the instances treated, as explained in more detail in Chapter 6. For smaller instances, the solution space is generally more limited, which leads to a smaller variety of generated solutions. This emphasises the importance of a differentiated evaluation of the objective function components depending on the instance size to ensure the method's effectiveness.

As far as the instances are concerned, it was found that the choice of size is not of critical importance as long as the processing speed requirements of the requests do not overtake the optimisation speed. Currently, the calculation time is within acceptable limits, which indicates that the method can be applied in practice without limitations. This finding is particularly important as managerial insight, although a drastic increase in optimisation speed requirements due to increased processing speed opens up opportunities to improve the methods used or even to explore them anew.

Regarding the relativisation of the individual objective function components, we found that the choice of the relativisation method as well as the adaptive adjustment, in terms of the number of requests inserted used during CI and ALNS *Repair*, provides highly satisfying solutions. Since individual relativisation values are calculated for both a few inserted requests or complete solutions, a fair basis for the WSA can be guaranteed.

We have presented two solution approaches for the underlying problem: First, the ALNS using CI and its modified variant as CH and *Repair* operator. Second, the D-5R-LA-TS heuristic as an approach to solve the problem under a dynamic consideration. When selecting the CH, we deliberately chose the CI, which is easy to implement, as we consider the novel temperature aspect as a priority in the course of our fundamental research. Despite using a simple heuristic, it turned out that it had to be significantly adapted, which proved to be a challenging task. The modified version of the CI, CI-SiIH, showed on average a slightly worse performance compared to CI-PaIH. However, it explores a more extensive solution space and leads to more diversification for the initial solution during ALNS's construction phase. This could provide incentives for further research into the modified version of CI, particularly with regard to a less restrictive design by adjusting the capacity reservation.

For the ALNS, we relied exclusively on CI and its various sequence types to serve as a CH. *Random Removal* and *Worst Removal* were used as *Destroy* operators, with the latter removing solution elements according to four different criteria. The adjustment of the adaptive *Destruction Rate* was also modified, such that it is updated more frequently. Experiencing that a variable *Destruction Rate* often leads to better solutions is not a new finding, but we were able to discover that in this problem in particular, faster destruction rate adjustment leads to better solutions in less time.

The use of the reversed destruction rate adjustment, where the destruction rate is reduced again once the maximum destruction rate is reached even though the newly generated solutions during the ALNS *Repair* remained poor, proved to improve the overall solution faster. We also found that solutions using only the fast destruction rate adjustment without implementing a reversal often get stuck in local optima. It is, therefore, generally advised to extend the faster adjustment of the destruction rate by a

reversed destruction rate adjustment.

We were also able to implement a higher destruction rate than what is common in literature, which speaks in favour of diversification. In view of the solutions generated, it can be concluded that the decision on these methods has led to a highly pleasing solution quality. This is due to the fact that the generated *Pareto-front* is undoubtedly visible, under which meaningful solutions are found.

A new heuristic, i.e., the D-5R-LA-TS heuristic, which aims to solve the problem considered in this research as a dynamic variant, was also presented in the context of an excursion. This is an insight into a solution method new to the field of CVRPDPs in general. However, it has been investigated only in its basic form within this thesis as it represents a preview for future research endeavours. Nevertheless, even at its basic stage, solutions have already been generated that either do not differ from the ALNS solution at all or only slightly, which encourages further development of this heuristic.

When analysing the solutions, we were able to identify various effects that are particularly relevant for practical application and managerial insights. First, we confirmed the initial hypothesis that *travel time* and *vehicle temperature* as well as *ton travel time* and *vehicle temperature* represent conflicting objectives. Although *travel time* and *ton travel time* pursue different objectives, they nevertheless lead to similar solutions, which indicates that they are concurrent. Both a pure *travel time* optimisation ($\alpha = 100\%$) and a pure *ton travel time* optimisation ($\beta = 100\%$) lead to better solutions in their respective objectives.

Furthermore, a complete calculation of all weightings, i.e. 111 different permutations in relation to the individual objective components, is not only not practically feasible but also not relevant in practice. A comparison between a pure *vehicle temperature* optimisation ($\gamma = 100\%$) and a mixed *vehicle temperature* and *travel time* optimisation where the prior is predominant (e.g. $\gamma = 90\%$, $\alpha = 10\%$) shows that the pure *vehicle temperature* optimisation leads to a significantly longer *travel time*, while the *vehicle temperature* is only marginally lower. This suggests that not only with regard to the optimisation of *vehicle temperature* objective function components but also with regard to the other two objective function components, an extremely strong weighting of the individual objective function components comes with a significantly stronger trade-off. Therefore, it is advisable to choose the weightings carefully and prefer predominantly mixed optimisations.

Confidential interviews with representatives of the partnered steel plant revealed the expectations for solutions that would be practically applicable. Hence, computational experiments allow us to conclude and propose the following recommendation to practitioners with regard to selecting concrete weighting coefficients for solving the TD-CVRPDP. We advise setting up the optimisation such that the *travel time* objective component is prioritised while taking into account a *vehicle temperature* limit and a desired vehicle usage in terms of *ton travel time*. Based on the analysis of the solutions, we recommend weighting the *travel time* (α) with about 45%, the *ton travel time* (β) with 20% and the *vehicle temperature* (γ) with about 35%. The slightly lower weighting of the *ton travel time* is explained by the fact that the *Directional Penalty* was also used, which pursues a similar goal as the *ton travel time*. These weightings strike a sensible balance

between all three objective components, providing results that meet the expectations of the representatives of the partnered steel plant.

To summarise, this thesis provides a comprehensive insight into the challenges and solutions of a novel variation of a CVRPDP in the context of a steel plant. By introducing the multi-objective TD-CVRPDP, studying the problem's search behaviour, and presenting new methods such as the *Vehicle Temperature Prediction Method* and *Directional Penalty*, innovative approaches were presented to equally consider operational efficiency, environmental aspects, and safety relevance. The investigation of different weightings of the objective function components showed the importance of balanced prioritisation and underlined the relevance of mixed optimisation approaches in practice. The analysis of the solutions suggests that a careful balancing of different objectives is required. More precisely, concrete recommendations for the weighting of the objective function components are identified. These results offer a valuable contribution to the further development of optimisation methods in the field of transport logistics and open up the potential for future research projects on the optimisation of complex logistical real-world challenges.

Bibliography

Aarts, E. and Lenstra, J. K. (1997). *Local search in combinatorial optimization; ed. by Emile Aarts, Jan Karel Lenstra*. Wiley.

Aarts, E. and Lenstra, J. K. (2003). *Local Search in Combinatorial Optimization*. Princeton University Press.

Abbasi, S., Daneshmand-Mehr, M., and Kanafi, A. G. (2023). Designing a tri-objective, sustainable, closed-loop, and multi-echelon supply chain during the covid-19 and lockdowns. *Foundations of Computing and Decision Sciences*, 48(3):269–312.

Amiri, A., Zolfagharinia, H., and Amin, S. H. (2023). A robust multi-objective routing problem for heavy-duty electric trucks with uncertain energy consumption. *Computers and Industrial Engineering*, 178.

An, D., Parragh, S. N., Sinnl, M., and Tricoire, F. (2024). A matheuristic for tri-objective binary integer linear programming. *Computers and Operations Research*, 161. All Open Access, Hybrid Gold Open Access.

Ashby, M. F., Shercliff, H., and Cebon, D. (2019). *Materials: Engineering, Science, Processing and Design*. Butterworth-Heinemann an imprint of Elsevier.

Baker, K. R. and Trietsch, D. (2019). *Principles of Sequencing and Scheduling*. Wiley.

Bishop, C. M., editor (2006). *Graphical Models*, pages 359–422. Springer New York, New York, NY.

Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*, pages 466–474. Cambridge University Press.

Bäck, T. (1996). *Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press on Demand.

Casella, G. and Berger, R. L. (2002). *Statistical Inference*, page 102. Duxbury Thomson Learning.

Cheng, Y.-r., Liang, B., and Zhou, M.-h. (2010). Optimization for vehicle scheduling in iron and steel works based on semi-trailer swap transport. *Journal of Central South University of Technology*, 17(4):873–879.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581.

Coello Coello, C. (2006). Evolutionary multi-objective optimization: a historical view of the field. *IEEE Computational Intelligence Magazine*, 1(1):28–36.

Cordeau, J.-F., Laporte, G., and Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52(8):928–936.

Cox, M. A. A. and Cox, T. F. (2008). *Multidimensional Scaling*, pages 315–347. Springer Berlin Heidelberg, Berlin, Heidelberg.

Cunanan, C., Tran, M.-K., Lee, Y., Kwok, S., Leung, V., and Fowler, M. (2021). A review of heavy-duty vehicle powertrain technologies: Diesel engine vehicles, battery electric vehicles, and hydrogen fuel cell electric vehicles. *Clean Technologies*, 3(2):474 – 489. All Open Access, Gold Open Access.

Deb, K. (1999). Genetic algorithms. *Wiley Encyclopedia of Electrical and Electronics Engineering*.

Deb, K. (2001). *Elitist Multi-Objective Evolutionary Algorithms*, page 239–288. John Wiley Sons.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Devore, J. L. (2015). *Probability and statistics for engineering and the Sciences*. Cengage Learning.

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

Dorigo, M. and Stützle, T. (2003). *The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances*, pages 250–285. Springer US, Boston, MA.

Dorigo, M. and Stutzle, T. (2004). The Ant Colony Optimization Metaheuristic. In *Ant Colony Optimization*. The MIT Press.

Drozdowski, M. (2009). *Classic Scheduling Theory*, pages 55–86. Springer London, London.

Duarte, I. C. D., de Almeida, G. M., and Cardoso, M. (2022). Heat-loss cycle prediction in steelmaking plants through artificial neural network. *Journal of the Operational Research Society*, 73(2):326–337.

Eberhart and Shi, Y. (2001). Particle swarm optimization: developments, applications and resources. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, volume 1, pages 81–86 vol. 1.

Eiben, A. E. and Smith, J. E. (2003). *What is an Evolutionary Algorithm?*, pages 15–35. Springer Berlin Heidelberg, Berlin, Heidelberg.

European Commission (2024). Reducing co_2 emissions from heavy-duty vehicles. `https://climate.ec.europa.eu/eu-action/transport/road-transport-reducing-co2-emissions-vehicles/reducing-co2-emissions-heavy-duty-vehicles_en`. Accessed: 2024-02-12.

Geem, Z. W., Kim, J. H., and Loganathan, G. V. (2001). A new heuristic optimization algorithm: Harmony search. *Simulation*, 76(2):60–68.

Gendreau, M. (2003). An introduction to tabu search. In Glover, F. and Kochenberger, G. A., editors, *Handbook of Metaheuristics*, pages 37–54. Springer US, Boston, MA.

Gendreau, M. and Potvin, J.-Y. (2010). Tabu search. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 41–59. Springer US, Boston, MA.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549. Applications of Integer Programming.

Glover, F. (1989). Tabu search—part i. *ORSA Journal on Computing*, 1(3):190–206.

Glover, F. and Kochenberger, G. A. (2003). *Handbook of metaheuristics*. Springer New York, NY.

Glover, F. and Laguna, M. (1998). Tabu search. In Du, D.-Z. and Pardalos, P. M., editors, *Handbook of Combinatorial Optimization*, pages 2093–2229. Springer US, Boston, MA.

Goldberg, D. E. (1989). A simple genetic algorithm. In *Genetic algorithms in search, optimization, and machine learning*, page 10. Addison-Wesley.

Golden, B. L., Raghavan, S., and Wasil, E. A. (2008). *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer.

Gupta, N. and Chandra, S. (2004). Temperature prediction model for controlling casting superheat temperature. *ISIJ International*, 44(9):1517–1526.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). Prototype methods and nearest-neighbors. In *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, pages 459–483. Springer New York, New York, NY.

Hennessy, J. L. and Patterson, D. A. (2011). Thread-level parallelism. In *Computer Architecture: A Quantitative Approach*, page 348. Morgan Kaufmann, 5th edition.

Hogg, R. V., McKean, J. W., and Craig, A. T. (2013). Some special distributions. In *Introduction to Mathematical Statistics*, page 155. Pearson Education India.

Holland, J. H. (1992). *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Huang, B., Tang, L., Baldacci, R., Wang, G., and Sun, D. (2023). A metaheuristic algorithm for a locomotive routing problem arising in the steel industry. *European Journal of Operational Research*, 308(1):385–399.

Intel Corporation (2017). Intel® xeon® platinum 8174 prozessor. `https://www.intel.de/content/www/de/de/products/sku/136874/intel-xeon-platinum-8174-processor-33m-cache-3-10-ghz/specifications.html`. Accessed: 2024-03-10.

Iyer, R. K., Kelly, J. C., and Elgowainy, A. (2023). Vehicle-cycle and life-cycle analysis of medium-duty and heavy-duty trucks in the united states. *Science of The Total Environment*, 891:164093.

Jozefowiez, N., Semet, F., and Talbi, E.-G. (2008). Multi-objective vehicle routing problems. *European Journal of Operational Research*, 189(2):293–309.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948.

Kirkpatrick, S., Gelatt Jr, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Knecht, R. (2022). Improvement of work safety and reduction of environmental issues with fire resistant lubricants in steel plants. *BHM Berg- und Hüttenmännische Monatshefte*, 167(5):224–228.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109.

Koza, J. R. (1996). *On the programming of computers by means of natural selection*. MIT Press.

Laporte, G. (1992a). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247.

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

Larsen, R. J. and Marx, M. L. (2018). Hypothesis testing. In *An Introduction to Mathematical Statistics and Its Applications*, page 380. Pearson.

Laumanns, M., Thiele, L., Deb, K., and Zitzler, E. (2002). Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary computation*, 10(3):263–282.

Lawler, E. (1995). *The traveling salesman problem: A guided tour of combinatorial optimization*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, Chichester [u.a.], reprint. edition.

Lee, H. L., Padmanabhan, V., and Whang, S. (1997). The bullwhip effect in supply chains. *Sloan Management Review*.

Lee, J. and Kim, B.-I. (2015). Industrial ship routing problem with split delivery and two types of vessels. *Expert Systems with Applications*, 42(22):9012–9023.

Lide, D. R. (2003). Section 5: Thermochemistry, electrochemistry, and solution chemistry. In *CRC Handbook of Chemistry and Physics*, pages 1–210. CRC Press.

Luke, S. (2013). *Essentials of metaheuristics*. Lulu, second edition. Aufrufbar unter: https://cs.gmu.edu/~sean/book/metaheuristics/Essentials.pdf.

Luo, J., Wang, C., Wallerstein, B., Barth, M., and Boriboonsomsin, K. (2022). Heavy-duty truck routing strategy for reducing community-wide exposure to associated tailpipe emissions. *Transportation Research Part D: Transport and Environment*, 107(103289).

Marler, R. T. and Arora, J. S. (2004). Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26:369–395.

Matl, P., Hartl, R., and Vidal, T. (2018). Workload equity in vehicle routing problems: A survey and analysis. *Transportation Science*, 52(2):239 – 260. All Open Access, Green Open Access.

McKinsey & Company (2021). The future of the european steel industry. `https://www.mckinsey.com/industries/metals-and-mining/our-insights/the-future-of-the-european-steel-industry`. Accessed: 2024-04-12.

Meisel, F. and Kopfer, H. (2014). Synchronized routing of active and passive means of transport. *OR Spectrum*, 36(2):297 – 322.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092.

Microsoft Documentation (aktualisiert 2024). Rngcryptoserviceprovider class. Online. Abrufbar unter: `https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rngcryptoserviceprovider`.

Myungho Lee, Kyungduk Moon, K. L. J. H. and Pinedo, M. (2023). A critical review of planning and scheduling in steel-making and continuous casting in the steel industry. *Journal of the Operational Research Society*, 0(0):1–35.

Ning, X., Qi, J., Wu, C., and Wang, W. (2018). A tri-objective ant colony optimization based model for planning safe construction site layout. *Automation in Construction*, 89:1 – 12.

Osman, I. H. and Laporte, G. (1996). Metaheuristics: A bibliography. *Annals of Operations Research*, 63(5):511–623.

OWASP (2017). Insecure randomness. Online. Abrufbar unter: `https://owasp.org/www-community/vulnerabilities/Insecure_Randomness#:~:text=Insecure%20randomness%20errors%20occur%20when,unable%20to%20produce%20true%20randomness.`

Oyola, J. and Løkketangen, A. (2014). Grasp-asp: An algorithm for the cvrp with route balancing. *Journal of Heuristics*, 20(4):361 – 382.

Papadimitriou, C. and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*, volume 32.

Papoulis, A. and Pillai, S. (2002). *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill series in electrical and computer engineering. McGraw-Hill.

Pareto, V. (1964). *Cours d'économie politique*. "Travaux de Sciences Sociales". Librairie Droz.

Pearl, J. (1984). Strategies and models for game-playing programs. In *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, page 222. Addison-Wesley.

Petropoulos, F., Laporte, G., Aktas, E., Alumur, S. A., Archetti, C., Ayhan, H., Battarra, M., Bennell, J. A., Bourjolly, J.-M., Boylan, J. E., Breton, M., Canca, D., Charlin, L., Chen, B., Cicek, C. T., Cox, L. A., Currie, C. S., Demeulemeester, E., Ding, L., Disney, S. M., Ehrgott, M., Eppler, M. J., Erdosğan, G., Fortz, B., Franco, L. A., Frische, J., Greco, S., Gregory, A. J., Hämäläinen, R. P., Herroelen, W., Hewitt, M., Holmström, J., Hooker, J. N., Işık, T., Johnes, J., Kara, B. Y., Karsu, , Kent, K., Köhler, C., Kunc, M., Kuo, Y.-H., Letchford, A. N., Leung, J., Li, D., Li, H., Lienert, J., Ljubić, I., Lodi, A., Lozano, S., Lurkin, V., Martello, S., McHale, I. G., Midgley, G., Morecroft, J. D., Mutha, A., Oğuz, C., Petrovic, S., Pferschy, U., Psaraftis, H. N., Rose, S., Saarinen, L., Salhi, S., Song, J.-S., Sotiros, D., Stecke, K. E., Strauss, A. K., Tarhan, , Thielen, C., Toth, P., Van Woensel, T., Berghe, G. V., Vasilakis, C., Vaze, V., Vigo, D., Virtanen, K., Wang, X., Weron, R., White, L., Yearworth, M., Yıldırım, E. A., Zaccour, G., and Zhao, X. (2023). Operational research: methods and applications. *Journal of the Operational Research Society*. All Open Access, Green Open Access, Hybrid Gold Open Access.

Pfrommer, J., Meyer, A., and Tierney, K. (2024). Solving the unit-load pre-marshalling problem in block stacking storage systems with multiple access directions. *European Journal of Operational Research*, 313(3):1054–1071.

Pinedo, M. (2016). Single machine models (deterministic). In *Scheduling: Theory, Algorithms, and Systems*, page 44. Springer, Cham.

Pisinger, D. (2004). Where are the hard knapsack problems? *Computers & Operations Research*, 32(2005):2271–2284.

Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers Operations Research*, 34(8):2403–2435.

Pisinger, D. and Ropke, S. (2019). Large neighborhood search. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 99–127. Springer International Publishing, Cham.

Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization: An overview. *Swarm Intelligence*, 1.

Raidl, G. R., Puchinger, J., and Blum, C. (2019). Metaheuristic hybrids. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 385–417. Springer International Publishing, Cham.

Rixner, S. (2019). NUMA and Cache Coherency. `https://www.cs.rice.edu/~rixner/comp322/lectures/NUMA-CacheCoherence.pdf`. Accessed: 2024-03-21.

Roljic, B., Leitner, S., and Doerner, K. F. (2021). Stacking and transporting steel slabs using high-capacity vehicles. *Procedia Computer Science*, 180:843–851. Proceedings of the 2nd International Conference on Industry 4.0 and Smart Manufacturing (ISM 2020).

Roljic, B., Tricoire, F., and Doerner, K. F. (2024). The active-passive vehicle routing problem with item transshipments. Technical report, Department of Business Decisions and Analytics, University of Vienna.

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

Ross, S. M. (2010). The exponential distribution and the poisson process. In *Introduction to Probability Models*, page 291–370. Elsevier, 10 edition.

Roy, B. and Mousseau, V. (1996). A theoretical framework for analysing the notion of relative importance of criteria. *Journal of Multi-Criteria Decision Analysis*, 5(2):145–159.

Russell, S. J. and Norvig, P. (2010). Solving problems by searching. In *Artificial Intelligence: A Modern Approach*, page 119. Prentice Hall.

Sen, B., Ercan, T., Tatari, O., and Zheng, Q. P. (2019). Robust pareto optimal approach to sustainable heavy-duty truck fleet composition. *Resources, Conservation and Recycling*, 146:502 – 513.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In Maher, M. and Puget, J.-F., editors, *Principles and Practice of Constraint Programming — CP98*, pages 417–431, Berlin, Heidelberg. Springer Berlin Heidelberg.

Subramanian, A., Deep, K., and Pant, M. (2017). A metaheuristic based approach for solving capacitated vehicle routing problem with pickup and delivery. *Engineering Applications of Artificial Intelligence*, 66:23–35.

Talbi, E.-G. (2009). *Metaheuristics : from design to implementation.* Wiley, Hoboken, NJ.

Toth, P. and Vigo, D. (2014). *Vehicle Routing: Problems, Methods, and Applications.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 2 edition.

Tricoire, F., Scagnetti, J., and Beham, A. (2018). New insights on the block relocation problem. *Computers Operations Research*, 89:127–139.

Tuoliken, A., Zhou, L., Bai, P., and Du, X. (2021). On the leidenfrost effect of water droplet impacting on superalloy plate surface. *International Journal of Heat and Mass Transfer*, 172:121218.

Van Laarhoven, P. J. M. and Aarts, E. H. L. (1987). Simulated annealing. *Simulated annealing: Theory and applications*, 3:7–15.

Visée, M., Teghem, J., Pirlot, M., and Ulungu, E. (1998). Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem. *Journal of Global Optimization*, 12(2):139–155.

Wackerly, D., Mendenhall, W., and Scheaffer, R. (2014a). The gamma probabilty distribution. In *Mathematical Statistics with Applications*, pages 185–194. Cengage Learning.

Wackerly, D., Mendenhall, W., and Scheaffer, R. (2014b). The uniform probability distribution. In *Mathematical Statistics with Applications*, pages 174–178. Cengage Learning.

Wang, J. and Rakha, H. A. (2017). Fuel consumption model for heavy duty diesel trucks: Model development and testing. *Transportation Research Part D: Transport and Environment*, 55:127 – 141.

Wang, Y., Wang, X., Fan, J., Wang, Z., and Zhen, L. (2023). Emergency logistics network optimization with time window assignment. *Expert Systems with Applications*, 214. All Open Access, Bronze Open Access, Green Open Access.

Wasserman, L. (2004). Random variables. In *All of Statistics: A Concise Course in Statistical Inference*, pages 19–46. Springer New York, New York, NY.

Wiecek, M. M., Ehrgott, M., and Engau, A. (2016). Continuous multiobjective programming. In Greco, S., Ehrgott, M., and Figueira, J. R., editors, *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 739–815. Springer New York, New York, NY.

Wolfinger, D. (2021). A large neighborhood search for the pickup and delivery problem with time windows, split loads and transshipments. *Computers Operations Research*, 126:105110.

World Steel Association (2023). Sustainability indicators 2023 report. `https://worldsteel.org/steel-topics/sustainability/sustainability-indicators-2023-report/`.

Zhang, Z., Zhao, Z., Zhang, Y., and Liu, S. (2022). Multi-process logistics planning for cost minimization and workload balance in steel production systems. In *2022 IEEE International Conference on Networking, Sensing and Control (ICNSC)*, pages 1–6.

Zitzler, E., Laumanns, M., and Thiele, L. (2001). Spea2: Improving the strength pareto evolutionary algorithm. Report, Zurich.

Zitzler, E. and Thiele, L. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE transactions on evolutionary computation*, 7(2):117–132.

Álvarez, A. and Munari, P. (2016). Abordagens metaheurísticas para o problema de roteamento de veículos com janelas de tempo e múltiplos entregadores. *Gestão & Produção*, 23(2):279–293. Epub 14 June 2016.

Çengel, Y. A. and Ghajar, A. J. (2015). *Heat and Mass Transfer: Fundamentals and Applications*. McGraw-Hill Education.

Özgür, A., Uygun, Y., and Hütt, M.-T. (2021). A review of planning and scheduling methods for hot rolling mills in steel production. *Computers Industrial Engineering*, 151:106606.

Acronyms

Δ Deviation. 18

$I\bar{Q}R$ Average Interquartile Range. 82, 83

$M\bar{A}D$ Average Mean Absolut Deviation. 82, 83

$\bar{Q_1}$ Average First Quartiles. 82, 83

$\bar{Q_3}$ Average Third Quartiles. 82, 83

$\bar{\sigma}$ Average Standard Deviation. 82, 83

$\bar{x}$ Mean Value. 66, 82, 83

σ Standard Deviation. 66

$\tilde{x}$ Average Median. 83

ALNS Adaptive Large Neighborhood Search. v, vi, 4, 10, 11, 13, 23, 32–34, 38–41, 43, 44, 46–52, 54–56, 67, 70, 80–86, 88, 89

APVRP Active-Passive Vehicle Routing Problem. 12

APVRPIT Active-Passive Vehicle Routing Problem with Item Transshipments. 12

BRP Block Relocation Problem. 12

CH Construction Heuristic. v, vi, 51, 52, 81–83, 86, 88

CHT Convective Heat Transfer. 14–16, 19, 20, 59

CI Cheapest Insertion Heuristic. iv, 23, 28–31, 33, 44, 45, 51, 56, 80, 81, 83, 88

CI-PaIH Cheapest Insertion Pairwise Insertion Heuristic. 32–34, 36, 37, 56, 88

CI-PIH Cheapest Insertion Parallel Insertion Heuristic. 31

CI-SIH Cheapest Insertion Sequential Insertion Heuristic. 31, 32, 34, 56

CI-SiIH Cheapest Insertion Single Insertion Heuristic. vii, 32–34, 36, 37, 56, 88

COHT Conductive Heat Transfer. 14, 15, 19

CPU Central Processing Unit. 48, 79–83, 86, 104, 105

CVRP Capacitated Vehicle Routing Problem. 12, 23

CVRPDP Capacitated Vehicle Routing Pickup and Delivery Problem. 2–4, 9, 44, 74, 87, 89, 90

D-5R-LA-TS Dynamic-5-Request-Look-Ahead-Tree-Search. v, vi, 4, 23, 45, 59, 60, 62, 63, 67, 84–86, 88, 89

DT Distribution Type. 66

ECM Epsilon Constraint Method. 24

FTL Full Truck Load. 84

GA Genetic Algorithm. 10, 38, 41

GP Genetic Programming. 38, 39, 41

IAC Incumbent Acceptance Criterion. 45–47

ISRP Industrial Ship Routing Problem. 11

LA Look Ahead. 86

LNS Large Neighborhood Search. 11, 12

LTL Less Than Truck Load. 84

NLC Newton's Law of Cooling. 3, 14, 16–19, 77, 87

NUMA Non-Uniform Memory Access. 80, 105

PMP Pre-Marshalling Problem. 12

RHT Radiative Heat Transfer. 14, 15

RW Roulette Wheel. 44, 47, 48, 50

SA Simulated Annealing. 38, 39, 41

SI Swarm Intelligence. 38, 39, 41

TD-CVRPDP Temperature Dependent Capacitated Vehicle Routing Pickup and Delivery Problem. iv, v, 2–9, 13, 14, 23–25, 31, 42, 44, 46, 56, 59, 63, 65, 67, 87, 89, 90

TS Tabu Search. 38, 39, 41

TSP Travelling Salesman Problem. v, viii, 29, 31, 51, 52, 56, 107

VRP Vehicle Routing Problem. 2, 5, 7, 23, 24, 30, 31, 52, 65

VSC4 Vienna Scientific Cluster 4. 49, 79

WSA Weighted Sum Approach. iv, 4, 8, 10, 12, 23–29, 34, 45, 74, 75, 79, 87, 88

Glossary

Cryptographically strong random numbers In `C#`, the `RNGCryptoServiceProvider` class provides a way to generate cryptographically secure random numbers. Unlike the conventional `Random` class, which is based on a pseudorandom algorithm and may not be suitable for cryptographic applications, `RNGCryptoServiceProvider` uses cryptographically secure methods to generate random numbers suitable for security-critical applications such as encryption, digital signatures and authentication (Microsoft Documentation, 2024). While the `Random` class may be sufficient for many use cases, it should not be used for security-critical purposes as its random numbers could be predictable, especially when used to generate cryptographic keys or protect sensitive data (OWASP, 2017). 32

Euclidean Distance The Euclidean distance is a concept from linear algebra and geometry that is used to measure the distance between two points in an n-dimensional space. It is named after the Greek mathematician Euclid. This distance metric is often used in various scientific disciplines such as data analysis, pattern recognition and machine learning.

Formally, the Euclidean distance between two points $\mathbf{p}$ and $\mathbf{q}$ in an n-dimensional space can be calculated using the following formula:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Where n is the dimension of the space, p_i and q_i are the ith components of the points $\mathbf{p}$ and $\mathbf{q}$ (Cox and Cox, 2008; Hastie et al., 2009; Bishop, 2006). 65

Exponential Distribution The exponential distribution is a continuous probability distribution that is often used to model waiting times or the duration between events. It is particularly useful for processes with a continuous flow rate or decay rate.

The probability density function (PDF) of an exponential distribution with the parameter $\lambda > 0$ is given by

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{for } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Here e is the Euler number and x is the time variable. The parameter λ is the rate or inverse scale of the distribution and determines how quickly the exponential distribution falls off.

The exponential distribution has the so-called memoryless property, which means that the probability of an event occurring within a certain time interval is independent of the elapsed waiting time. This property makes the exponential distribution particularly useful for modelling purposes in various areas such as queueing theory, reliability analysis and lifetime analysis (Ross, 2010; Wackerly et al., 2014a; Papoulis and Pillai, 2002). 65

Gradient Descent Method The gradient descent method is an optimisation algorithm that is frequently used in mathematical optimisation and especially in machine learning. The aim of gradient descent is to minimise a function by iteratively taking steps towards the negative gradient of the function.
In the context of optimising functions, the gradient descent algorithm searches for a local minimum of the function. The gradient of a function is a vector that indicates the direction of the steepest slope at a particular point. The negative gradient indicates the direction of the steepest descent and instructs the algorithm to move in the direction of the minimum.
The gradient descent procedure is quite simple: Firstly, a starting position is defined in the search space. The gradient of the target function at this position is then calculated. Based on the gradient and a learning rate, the current position is updated in an attempt to minimise the value of the target function. This process is repeated iteratively, with the position of the point in the search space gradually approaching the local minimum.
It is important to note that the gradient descent method is a local optimisation algorithm and therefore may only find a local minimum of the objective function. The effectiveness of the gradient descent method strongly depends on the choice of the learning rate. A learning rate that is too large can cause the algorithm to become unstable or even diverge, while a learning rate that is too small can lead to slow convergence speeds.
The gradient descent method is often used in various machine learning applications, such as neural network training or linear regression. It is a fundamental algorithm that forms an important building block for many more advanced optimisation methods (Boyd and Vandenberghe, 2004). 50

L3 Cache The L3 Cache (Level 3 Cache) is a shared cache that is used jointly by several CPU cores of a processor. It is normally located between the L1 cache (level 1 cache) and the L2 cache (level 2 cache) in the CPU cache hierarchy. The purpose of the L3 cache is to store frequently used data and instructions in order to minimise access to the main memory and thus improve performance. 80

Leidenfrost Effect The Leidenfrost effect describes the phenomenon in which a liquid meets a surface whose temperature is well above its boiling point. Instead of

evaporating immediately, the liquid forms an insulating layer of vapour between itself and the surface. This layer reduces the heat transfer and causes the liquid to appear to float and bubble. The effect was first described in the 18th century by Johann Gottlob Leidenfrost and has been the subject of scientific research ever since. Literature references include the work of Tuoliken et al. (2021). 19

Neighbourhood A neighbourhood is a set of potential solutions to a given optimisation problem that is closely related to a given initial solution. These solutions are often generated by a limited number of changes or transformations of the initial solution. The neighbourhood thus defines the local search space around a given solution and enables optimisation algorithms to search efficiently in this solution space (Talbi, 2009). 45

Normal Distribution The normal distribution, also known as the Gaussian distribution, is one of the most important probability distributions in statistics. It is characterised by its characteristic bell curve, which is symmetrical around the mean. The normal distribution occurs frequently in nature and is used in many scientific disciplines to model phenomena that scatter around a mean value.

The probability density function (PDF) of a normally distributed random variable X is given by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where μ is the mean and σ is the standard deviation of the distribution. The normal distribution is fully characterised by these two parameters. The mean value determines the peak of the bell curve, while the standard deviation controls the width of the curve. The larger the standard deviation, the broader the distribution.

The normal distribution plays a central role in many areas, including inferential statistics, hypothesis testing, linear regression and many others (Wasserman, 2004; Casella and Berger, 2002; Devore, 2015). 65

NUMA The Non-Uniform Memory Access is a memory architecture used in modern multiprocessor systems to organise memory access. In NUMA systems, each CPU has its own local memory (often referred to as node memory), which it can access quickly. In addition, there is a shared global memory that all CPUs can access, but with different latency times. This means that access to the local memory is faster than to remote memory areas.

NUMA architectures are designed to improve the scalability of multiprocessor systems by increasing memory bandwidth and reducing memory access bottlenecks. By allocating work to CPUs that access local memory, performance can be improved as latency is lower and memory bandwidth is higher (Hennessy and Patterson, 2011; Rixner, 2019). 80

Oxidation Protection Effect The oxidation protection effect is a significant phenomenon in materials science, especially when exposed to high temperatures. It refers to the formation of a protective oxide layer on the surface of a material when it is exposed to extreme temperatures. This layer acts as a barrier against further oxidation and corrosion of the underlying material, helping to maintain its structural integrity. The mechanisms underlying this effect include the formation of dense oxide layers, the adhesion and diffusion of oxygen molecules and the passivation of surface defects through oxidation (Ashby et al., 2019; Lide, 2003). 19

Uniform Distribution The uniform distribution is a basic probability distribution in statistics and probability theory. It is used to model events in which all possible outcomes occur with equal probability. A uniform distribution can occur in both discrete and continuous forms.

In the discrete uniform distribution, the probability of each possible outcome is the same. Assuming there are n possible outcomes, the probability of each individual outcome is $\frac{1}{n}$.

In the continuous uniform distribution, the probability density is constant over a fixed interval. Within this interval, the probability of observing a particular event is evenly distributed.

Mathematically, the probability density $f(x)$ for a continuous uniform distribution on the interval $[a, b]$ can be defined as follows

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

This distribution is often used when there is no information about the probability distribution or when all possible outcomes can be considered equally likely (Wackerly et al., 2014b; Larsen and Marx, 2018; Hogg et al., 2013). 65

A. Appendix

A.1. Cheapest Insertion Travelling Salesman Problem Example

In mathematical terms, the Cheapest Insertion Heuristic selects in each step the location j that causes the smallest additional cost change ΔC to the existing route. This means that for each location j in the set of remaining locations S, the cost change is calculated when j is inserted into the route. The location with the minimum cost change is then added to the route. This process is repeated until all locations have been added to the route.

Algorithm A.1 Cheapest Insertion Heuristic (TSP)

Input: Set of locations S, distances between locations d_{ij}
Initialize an empty route R with a starting location
$S' \leftarrow S$ ▷ Set of unvisited locations
Choose a starting location i from S'
Add i to R
Remove i from S'
while S' is not empty
    $j^* \leftarrow$ Location in S' that minimizes ΔC_{j^*}
    Insert j^* into R at the position that minimizes ΔC_{j^*}
    Remove j^* from S'
end
return Route R

Let's assume we start with customer A as the initial customer. The algorithm would run as follows:

1. Start with route R: [A]
2. Calculate the cost change for adding each remaining customer:

	A	B	C	D	E
A	0	10	15	20	25
B	10	0	10	12	18
C	15	10	0	8	16
D	20	12	8	0	10
E	25	18	16	10	0

Cheapest Insertion Heuristic example distance matrix

- For B: $\Delta C = 10$
- For C: $\Delta C = 15$
- For D: $\Delta C = 20$
- For E: $\Delta C = 25$

3. Add customer B, as it has the lowest cost change: Route R: [A, B]
4. Calculate the cost change for adding the remaining customers:
   - Für C: $\Delta C = 5$
   - Für D: $\Delta C = 10$
   - Für E: $\Delta C = 15$
5. Add customer C, as it has the lowest cost change: Route R: [A, C, B]
6. Add the remaining customers in a similar way until all customers have been added.

The final route in this example would be: [A, C, D, B, E].

A.2. Cheapest Insertion Vehicle Routing Problem Example

Assume a courier service needs to transport packages from various pickup locations to various delivery locations in a city. Each pickup location has a corresponding delivery location where the package needs to be delivered. The goal is to plan the route(s) for the vehicles to pick up and deliver all packages while minimizing the total cost.

Consider a specific scenario:

- There are a total of 5 pickup locations and 5 delivery locations.
- The costs for transportation between pickup and delivery locations are as follows:
  - Pickup 1 (A1) to Delivery 1 (L1): 10
  - Pickup 2 (A2) to Delivery 2 (L2): 12
  - Pickup 3 (A3) to Delivery 3 (L3): 15
  - Pickup 4 (A4) to Delivery 4 (L4): 8
  - Pickup 5 (A5) to Delivery 5 (L5): 20
- Each vehicle can carry a maximum of 3 packages.
- The capacity of each vehicle is 25 units.

Assume we start with an empty route plan and one vehicle at the starting depot (D).

1. Choose the pickup location nearest to the starting depot. Let's assume this is A1.

2. Calculate the cost for adding A1 and L1 to the route.
3. Repeat this process until the vehicle's capacity is reached or all pickup and delivery locations have been visited.
4. Once the vehicle's capacity is reached or no pickup and delivery locations are left, return to the starting depot.
5. Repeat this process for each vehicle until all packages have been picked up and delivered.

The final routes might look like this:

- Vehicle 1: D $\rightarrow$ A1 $\rightarrow$ L1 $\rightarrow$ D
- Vehicle 2: D $\rightarrow$ A2 $\rightarrow$ L2 $\rightarrow$ D
- Vehicle 3: D $\rightarrow$ A3 $\rightarrow$ L3 $\rightarrow$ D
- Vehicle 4: D $\rightarrow$ A4 $\rightarrow$ L4 $\rightarrow$ D
- Vehicle 5: D $\rightarrow$ A5 $\rightarrow$ L5 $\rightarrow$ D

This is a simplified representation of a VRP with Pickup and Delivery using the Cheapest Insertion Heuristic. In practice, additional factors such as vehicle capacities, delivery time windows, and road traffic conditions would be taken into account.