UNIVERSITY OF TWENTE.

Master Thesis

Improving the routing solving software of ORTEC with the help of parallel computing

by

Tom Huizingh

Industrial Engineering and Management
Specialization Production and Logistics Management
Orientation Supply Chain and Transportation Management
Faculty of Behavioural, Management and Social Sciences

Examination committee
Dr. Breno Alves Beirigo
Dr.ir. E.A. Lalla-Ruiz
University of Twente

External supervision
Cynthia Slotboom
Nienke Jansen
ORTEC

2024

Abstract

The effects of the use of parallel computing on the solution quality in relation to the additional computational costs were unknown to ORTEC. This research seeks to provide ORTEC with relevant insights into using parallel computing in the OHD software by providing a solution design that transfers a serial algorithm into a parallel algorithm. In this way, a new parallel algorithm that uses multiple construction and R&R methods is developed by modifying parameter settings from a single serial algorithm that is used to solve the VRP of Company X. The results show that our new parallel algorithm outperforms the current algorithm by 1.56% in terms of costs which shows that the use of parallel computing within OHD can significantly improve the solution quality.

Management Summary

This research was conducted at ORTEC which is a company that among others provides optimization software to solve VRPs. Company X is a large supermarket that offers a delivery service for groceries and uses ORTEC's OHD software, designed for companies offering home delivery services. The OHD software has recently been made available in the cloud, enabling the use of parallel computing. As a result, the utilization of parallel computing at ORTEC is currently in its nascent stages. The fundamental principle of parallel computing involves the concurrent utilization of multiple processors to solve a computational problem. Consequently, a greater number of potential solutions can be computed within the same computational time at the expense of higher computational costs. This approach typically yields improved solutions within the equivalent computational time. This has led to the following research question posed by ORTEC:

To what extent can parallel computing improve the performance of the OHD software?

To address this research question, we designed a new parallel algorithm that solves the VRPs of Company X. This new algorithm is compared to the existing serial algorithm, known as the algorithm of Company X, which currently handles their VRPs. The existing algorithm operates in three phases: construction, local search, and ruin and recreate (R&R). Our research primarily focused on the construction phase but also considered the R&R phase. Since the algorithms used in OHD are very complex, new algorithms included in the parallel algorithm were created by modifying the parameter settings of the algorithm of Company X. This means that the problem we faced is a variant of the algorithm configuration problem which is typically solved by automated methods. However, these methods are unsuitable for our research due to the vast number of parameter settings and the lack of information about the working of a parameter setting. For these reasons, we have introduced a new solution design to tackle our problem.

Solution design

Our solution design consists of three steps which are visualised in Figure 1. In Step 1, the goal is to create good algorithms that outperform the algorithm of Company X for various types of cases of Company X such that they can be used in the new parallel algorithm. A data analysis is conducted to support the parameter setting changes. Subsequently, we investigated whether using multiple construction methods and starting solutions is beneficial in Step 2, and if so, determine the optimal number of construction methods and starting solutions that proceed to the local search and R&R phase. Finally, in Step 3, the algorithms included in our new parallel algorithm are selected.

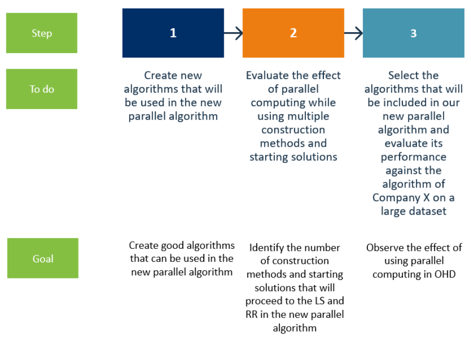

Step
1
2
3
To do
Create new algorithms that will be used in the new parallel algorithm
Evaluate the effect of parallel computing while using multiple construction methods and starting solutions
Select the algorithms that will be included in our new parallel algorithm and evaluate its performance against the algorithm of Company X on a large dataset
Goal
Create good algorithms that can be used in the new parallel algorithm
Identify the number of construction methods and starting solutions that will proceed to the LS and RR in the new parallel algorithm
Observe the effect of using parallel computing in OHD

Figure 1: A visualization of the solution design used to address our problem

Solution

In this way, a new parallel algorithm is constructed which is visualised in Figure 6.8. It includes 15 different construction methods of which all starting solutions proceed to the local search phase. Afterwards, the 3 best starting solutions after local search proceed to the R&R phase where 5 R&R methods are applied on them. In this way, 15 different final solutions are created and finally the best one is provided to Company X. Note that we were restricted to use a maximum of 15 CPUs due to the current contractual constraints with the cloud computing platform.

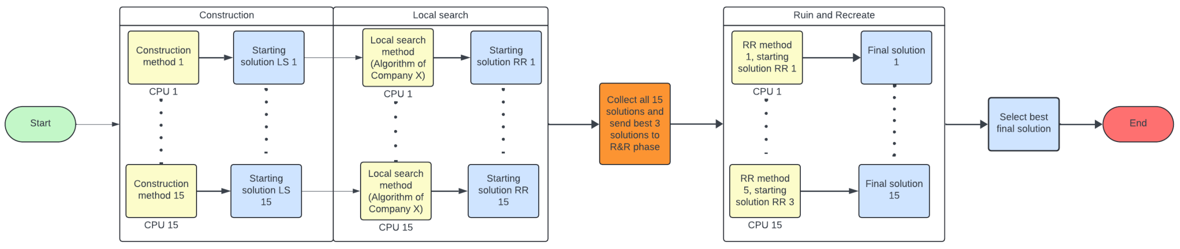


Figure 2: A visualization of the design of our new parallel algorithm including multiple construction and R&R methods.

Results

The main results of all experiments conducted in this research are as follows:

- **Multiple construction methods and starting solutions are beneficial.** We have created 653 algorithms which are made by modifying the parameter settings of the construction phase of the algorithm of Company X. It was observed that when running all 653 algorithms in parallel and only proceeding with the best starting solution after construction to the local search and R&R phase that the solution quality can be improved by 1.04% in terms of costs compared to the algorithm of Company X. Note that in these costs the computational costs are also included. On top of that, we observed that when we proceed with all starting solutions to the local search and R&R phase, the solution quality could be improved by 1.34%. This shows that the use of multiple construction methods and starting solutions is very beneficial. Furthermore, it shows that the best starting solution does not always result in the best final solution where we observed that the larger the case, the more important the starting solution. In case, we use 15 different construction methods since we are restricted to the use of 15 CPUs, an improvement of 1.18% is obtained.

- **Individual algorithms outperform the algorithm of Company X**. While creating the algorithms in Step 1 of our research design, individual algorithms were created that outperform the algorithm of Company X. The best individual algorithm that is also included our parallel algorithm outperformed the algorithm of Company X by 0.66% in terms of costs. This means that from the total improvement of 1.18%, 0.66% is caused by the updated parameter settings. Consequently, the implementation of parallel computing by using multiple construction methods and starting solutions proceeding to the local search and R&R phase caused an improvement of 0.54%.

- **Parallel computing is not useful when cases differ significantly.** In the data analysis and experiments, we have encountered that the cases from Country A and Country B differ significantly, with different types of algorithms performing well for each. For this reason, it is better to use different (parallel) algorithms for the cases of Country A and Country B instead of using one parallel algorithm.

- **Multiple R&R methods and starting solutions are beneficial.** We observed in the experiments that the optimal way of using 15 CPUs in the R&R phase of our parallel algorithm, is to apply 5 R&R methods on the best 3 solutions after local search. This parallel phase of the R&R outperforms the R&R method of the algorithm of Company X by 0.24% in terms of costs. The best individual R&R method included in the R&R phase of our parallel algorithm outperforms the R&R method of the algorithm of Company X by 0.18%. Consequently, the improvement caused by using multiple R&R methods and starting solutions in parallel is 0.06%.

- **New parallel algorithm outperforms the algorithm of Company X**. Our new parallel algorithm including multiple construction and R&R methods outperforms the algorithm of Company X by 1.57% in terms of costs. Consequently, ORTEC can save Company X a significant amount a year by using our new parallel algorithm instead of the algorithm of Company X which is mainly caused by the fact that computational costs are extremely low compared to the planning costs of Company X. However, our new parallel algorithm violates the maximum running time in 37.96% cases since a stopping criterion is removed in the experiments. Since most and the largest improvements are made at the beginning of the algorithm, we believe the inclusion of the stopping criterion will not have a large impact on the quality of the final solution.

Conclusions and recommendations

- **Parallel computing is highly beneficial.** This research has proven that the use of parallel computing is highly beneficial within OHD. Consequently, we recommend using it for every customer while using our research design to transform a serial algorithm into a parallel algorithm.

- **Implementation of our parallel algorithm.** We recommend implementing our new parallel algorithm for Company X after an agreement regarding the distribution of extra savings and computational costs has been made.

- **Update parameter settings of algorithms regularly.** We created individual algorithms that significantly outperform the algorithm of Company X by modifying parameter settings of the construction and R&R phase. For this reason, we recommend ORTEC to regularly update the parameter settings of algorithms with the support of a data analysis since data can change over time.

- **Scale up number of CPUs.** In this research, we were limited to the use of 15 CPUs, however, we have seen in the experiments that even more savings could be obtained by using more CPUs. For this reason, we would like to recommend to ORTEC to investigate if it is beneficial to scale up the number of CPUs in the contract with the cloud computing platform such that more CPUs are available for parallel algorithms.

- **Local search in parallel.** We recommend ORTEC to do further research regarding the parallelization of the local search phase of the algorithms since this research has proven that it is highly beneficial for the construction and R&R phase.

Preface

Dear reader,

With the completion of this master's thesis, I mark the culmination of my journey as a student in Industrial Engineering & Management at the University of Twente, and the commencement of my professional career. Over the past several months, I have had the privilege to conduct research at ORTEC, an experience that has profoundly shaped my academic and professional development. Before delving into the contents of this thesis, I would like to take a moment to thank those who have supported and guided me along this journey.

First and foremost, I extend my sincere appreciation to all my colleagues at ORTEC. Their willingness to assist and their positive energy has made my internship a successful and enjoyable experience. I am particularly grateful to my supervisors, Cynthia Slotboom and Nienke Jansen, for their guidance and mentorship throughout this research. They were really dedicated to ensuring the quality of my research and were always available to provide feedback. I also wish to acknowledge the contributions of Arjen Rietveld and Laura Simons, whose high-level guidance and expert insights have greatly influenced the direction and outcomes of my research.

Secondly, I would like to thank Breno Alves Beirigo, who was my first supervisor at the University of Twente, who provided insightful feedback that significantly improved the academic quality of my thesis. Moreover, I would like to thank Eduardo Lalla for being my second supervisor and enhancing the quality of my work.

Finally, I am grateful to my parents for shuttling me to and from the station which significantly reduced my travelling time making it possible to regularly visit the office at Zoetermeer. Besides, I would like to thank my friends, whose companionship and shared moments of joy during holidays and weekends provided a welcome distraction.

Tom Huizingh
Nieuwleusen
July 5, 2024

Contents

Acronyms

C collegial communication. 15, 16

CPU Central Processing Unit. iv, v, 26, 58, 64, 68, 70

CVRP Capacitated Vehicle Routing Problem. 7, 8

EC European Community. 9

EDRP E-grocery Delivery Routing Problem. 9

HMDVRP Heterogeneous Multi-Depot Vehicle Routing Problem. 9

HVRP Heterogeneous Vehicle Routing Problem. 8

I/F-Race Iterated F-Race. 18

ILS Iterated Local Search. 18

KC knowledge-based collegial communication. 15, 16

KPI Key Performance Indicator. xiv, 40, 42, 65, 66, 69

KPIs Key Performance Indicators. xiii, 21, 32, 40–44, 46, 49, 51–53, 59, 65, 69

KS Knowledge-based Synchronization. 14, 16

MDVRP Multi-Depot Vehicle Routing Problem. 9

MPDS Multiple initial Points/Populations, Different search Strategies. 15, 26, 64, 67, 70

MPSS Multiple initial Points/Populations, Same search Strategies. 15

MTVRP Multi-Trip Vehicle Routing Problem. 9

OHD ORTEC for Home Delivery. ii, iii, v, 1, 3–6, 25–32, 39, 48, 61, 67, 69, 70

PCI Parallel Cheapest Insertion. 59, 61–63

PoC Proof of Concept. 5

R&R Ruin and Recreate. ii–v, viii, ix, xiii–xv, 3–6, 19, 24, 26, 29, 30, 32, 39, 40, 46, 54, 56–65, 67–71, 89

RI Regret Insertion. 59, 61–63

RS Rigid Synchronization. 14, 15

SDTDMTHMDHVRPTWEC Site-Dependent, Time-Dependent, Multi-Trip, Heterogeneous Multi-Depot, Heterogeneous Vehicle Routing Problem with Time Windows and EC social legislation. 9

SDVRP Site-Dependent Vehicle Routing Problem. 9

SMAC Sequential Model-based Algorithm Configuration. 18, 19

SPDS Same initial Point/Population, Different search Strategies. 15

SPSS Same initial Point/Population, Same search Strategy. 15

TDVRP Time-Dependent Vehicle Routing Problem. 8

TSP Traveling Salesman Problem. viii, 7

VRP Vehicle Routing Problem. i–iii, viii, x, 1, 3, 4, 6–10, 12, 17, 19, 20, 28, 67, 68

VRP-EC Vehicle Routing Problem with EC social legislation. 9

VRP-HOS Vehicle Routing Problem with Hours-Of-Service regulations. 9

VRPTW Vehicle Routing Problem with Time Windows. 8

VRPX Vehicle Routing Problem of Company X. x, 9, 10, 19, 20, 29, 67

Glossary

Algorithm of Company X The algorithm in the OHD software that is currently used by ORTEC to solve the routing problem of Company X. 21

Company X A large supermarket that is offering a delivery service of groceries. viii, 1–7, 9, 17, 19, 21, 22, 25

Construction The part of the algorithm that builds a starting solution from scratch. iii, 11, 21, 67

Current solution The best-known solution found by the algorithm at a given time. 12, 14, 21

Local search An algorithm that attempts to improve the current solution by iteratively applying small modifications. iii, 21, 67

ORTEC A worldwide company that provides optimization software and analytical solutions to all kinds of problems. iii, 1–7, 10, 11, 13, 17, 19, 21, 25–27, 29–31, 47, 55, 57–60, 67, 70, 71

Ruin and Recreate A metaheuristic that is used to escape a local optimum by destroying and rebuilding a part of the solution. iii, 3, 21, 67

List of Figures

List of Tables

Chapter 1

Introduction

In this introductory chapter, we provide a comprehensive overview of the context, objectives, and structure of this Master thesis. Firstly, Section 1.1 discusses the background and context of the research, highlighting the importance of optimizing vehicle routing problems in modern business operations. Secondly, the vehicle routing problem of Company X is introduced in Section 1.2. Afterwards, the problem statement, outlining the specific challenges faced by ORTEC while solving the problem of Company X, is presented in Section 1.3. Subsequently, the research objectives of this master thesis are stated in Section 1.4. Moreover, the sub-research questions that guide in answering the main research question and achieving the research objectives are outlined in Section 1.5, aligning them with the content of subsequent chapters. Furthermore, we emphasize the significance of this study in addressing real-world logistics challenges in Section 1.6, while Section 1.7 acknowledges the scope and limitations of the research. Finally, Section 1.8 outlines the organization of this thesis.

1.1 Background and Context

Nowadays, people order groceries online instead of physically visiting the grocery market. This trend is accompanied by a multitude of advantages such as the elimination of queuing, no longer experiencing issues related to depleted inventory on shelves, the alleviation of carrying heavy bags and the absence of additional leisure time devoted to visits to the supermarket. Alternatively, you can just simply order your groceries online and you can schedule a delivery moment that aligns with your preferences. However, in striving to fulfil the desires of the customers, several challenges are encountered by the supermarket providing the delivery service. First of all, every customer should get their products on time in good condition. Secondly, cold products should be transported in different boxes than for example chips. In addition, some addresses can not be accessed with every vehicle type. These are just examples of the many constraints and restrictions that companies that are delivering groceries are facing. Company X is a large supermarket that is offering a delivery service of groceries. Company X is delivering orders internationally which makes the planning of routes very complicated. Consequently, they want to have a complete routing scheme that minimizes operational costs while satisfying all the needs of the customers. The problem of Company X belongs to the class of the Vehicle Routing Problem (VRP) which is a combinatorial optimization problem and will be introduced in the next section.

The company ORTEC is optimizing the routing problem for Company X. ORTEC B.V. stands for Operation Research Technology, in this report referred to as ORTEC. ORTEC was founded in 1981 by five Econometric students and is now a globally respected company. ORTEC provides optimization software and analytical solutions to a wide range of companies addressing diverse problems. To solve routing problems they created various products based on the characteristics of the problem. In this research, our focus will be exclusively on the product that is designed for companies offering home delivery services. This particular product is called ORTEC for Home Delivery (OHD) and is also employed by Company X.

1.2 The Vehicle Routing Problem of Company X

Company X has millions of customers spread over a large area in two different countries: Country A and Country B. As a consequence, the logistics and transportation network of Company X is extensive.

To serve every customer, Company X has many warehouses located throughout the whole serving area. They distinguish between two types of warehouses: main depots and subdepots. Main depots are assigned to a specific area and supply the subdepots located in this area. Customers are allocated to a single main depot but may also be served by all subdepots that are supplied by this main depot. Company X has 11 main depots in total and has separate plannings for the morning and evening. Consequently, every day 22 routing problems of Company X need to be solved by ORTEC. These routing problems include many restrictions and constraints which will be explained in the subsequent subsections.

1.2.1 Depots

Every main depot can have to supply up to five subdepots and are assumed to have unlimited capacity whereas subdepots have limited capacity. The capacity per subdepot may be different. Company X demands that all vehicles of the subdepots should be fully utilized before the vehicles of the main depot are used.

1.2.2 Vehicle types

Company X uses different types of vehicles to serve every customer such as priority 1 vans, priority 2 vans, priority 3 vans and normal vans. However, the number of vehicles per vehicle type that are available differs per depot. Logically, the different types of vehicles have different cost sets, speeds and capacities. Company X requires every priority 1 and priority 2 van that is available for a depot to be assigned to a route although this increases the overall costs since they are less efficient compared to normal vans. priority 1 and priority 2 vans are expected to be assigned to easier routes. In this context, easier routes are defined as those that visit customers close to a depot. Problem instances of Company X can have up to 220 vehicles in their vehicle fleet.

1.2.3 Capacity restrictions

The capacity constraints per vehicle are handled by volume and weight. The weight restriction is included since different driving rules are applied to vans above a certain weight. Customers are ordering multiple kinds of products however for the planning of the routes, we do not have to take this into account. The products are transported in boxes and the volume constraint ensures that vehicles will not be overloaded. In this way, for example, huge orders will not be assigned to small vans.

1.2.4 Area restrictions

Some customers can only be visited by certain types of vehicles. For example, in some city centers, only zero-emission vehicles are allowed to drive.

1.2.5 Multi-trips

Vehicles may be assigned to multiple trips and a maximum of ten trips per vehicle is allowed. All trips have to start and end at the same depot. Furthermore, priority 1 and priority 2 vans are excluded from the option to execute multi-trips.

1.2.6 Driver regulations

The total duration of all trips of a vehicle is restricted since employees are not allowed to work more than a specific amount of hours by law. Additionally, the breaks of the drivers also need to be considered while scheduling the routes. Besides, vehicles have also a time window in which all trips should be planned.

1.2.7 Delivery Time Windows

Customers booked a timeslot that suits them in where the groceries need to be delivered. These are hard-time windows and must be adhered to for customer satisfaction.

1.2.8 Congestion

To adhere to the time windows of the customers, realistic travelling times have to be used. For this reason, congestion is taken into account while planning the trips since travel times can vary significantly depending on the time of the day. Road sections have different travel times per interval of 15 minutes based on historical data to provide realistic plannings.

1.2.9 Input data

All input data are known before the planning is started which makes it a static and deterministic VRP. Input data consists of customers' information and the number of available vehicles per type per depot. Customers' information consists of the coordinates of their location, order quantity, time window, expected service time and the vehicle types that are allowed to deliver. A distance and driving time matrix per time interval of 15 minutes is constructed based on the locations of customers and depots with the use of a map service.

1.2.10 Objectives

The main objective is to plan as many orders as possible without violating any of the above-mentioned constraints and restrictions since these are hard constraints and restrictions. If more trucks are needed than available, they hire extra employees and trucks such that more customers can be served. These trips are called overflow routes and are logically assigned higher costs such that they are only used when strictly necessary. Plannings are made well in advance such that extra vehicles and drivers can be arranged. Secondly, the costs should be minimized. Costs are assigned to routes, working hours and travelled distance. These costs depend on the vehicle type used for the route. Finally, the maximum duration of the planning process per main depot is 20 minutes.

1.3 Problem Description

The OHD software has recently been made available in the cloud. In the past, customers were required to install an application on their local computers to access the routing solvers. Currently, customers can log in on their personalised page in the cloud to utilise the functionalities of the OHD software. The migration of the OHD software in the cloud enables the use of parallel computing which was not possible in the preceding application. The fundamental principle of parallel computing involves the concurrent utilization of multiple processors to solve a computational problem. Consequently, a greater number of potential solutions can be computed within the same computational time at the expense of higher computational costs. This approach typically yields improved solutions within the equivalent computational time. As a result, a better solution can probably be obtained in the same computational time.

For each customer, a tailored algorithm is constructed, but they all consist of the same three phases: construction, local search and ruin and recreate (R&R). In the software of ORTEC, many built-in construction, local search, and R&R methods, including various settings, are available for the implementation consultants to construct a tailored algorithm for a certain client. In the construction phase, one of the construction methods is used to create an initial solution from scratch. This solution is used as a starting point for the local search method to improve the solution. The local search method generates new solutions by making small adjustments to the current solution. Only solutions that improve the current solution are accepted in this phase. In case no improvements can be found, the algorithm proceeds to the final phase. In the final phase, a ruin method is used to destroy a part of the solution and a recreate method is used to rebuild the solution. In case the rebuilt solution is within a certain range of the current objective value, a quick local search method is applied to the solution to check whether a better global solution can be found. This process is repeated until a predetermined stopping criterion is met. More detail about the algorithm that is used for the case of Company X is provided in Chapter 3. The algorithm that is currently solving the routing problem of Company X in the OHD software of ORTEC is from now on referred to as the algorithm of Company X.

Given the recent transition of the ORTEC software to a cloud-based environment, the utilization of parallel computing is currently in its nascent stages. ORTEC can build many different algorithms by combining a large number of built-in methods including various settings but currently only one particular algorithm is used for each customer. However, now the use of parallel computing is enabled in the software, multiple distinct methods at the same time could be used to diversify the search and potentially improve the solutions. Therefore, ORTEC is interested in the effects of incorporating parallel computing within the OHD software. Since the creation of a tailor-made algorithm for customers is a very time-consuming process, the different algorithms in the parallel algorithm will be created by adjusting parameter settings within the algorithm of Company X. Consequently, the design of our new parallel algorithm is a variant of a well-known problem in the literature: the algorithm configuration problem.

In this thesis, the algorithm and data of Company X will be used as an example and validation of our research about the implementation of parallel computing within the OHD software. Since the algorithms of customers are unique, it is impossible to apply the research also to other cases in this thesis due to time limitations. However, considering that all algorithms of customers follow a similar structure, we decided to modify the data of the AH case a bit. In this way, the problem and data of Company X will be more like other customer cases. Consequently, this research is not strictly limited to the case of Company X but focuses on the OHD software in general. This has led to the following research question asked by ORTEC:

To what extent can parallel computing improve the performance of the OHD software?

1.4 Research Objectives

The research objectives of this thesis are as follows:

1. To provide ORTEC with relevant feedback on the implementation of parallel computing in the OHD software concerning the trade-off between computational costs and computational power.
2. To enhance the quality of solutions, measured in terms of the objective values, through the implementation of a new parallel algorithm within the same computational time.

1.5 Research Questions

To achieve the research objectives stated in the preceding section and to address the main research question stated in Section 1.3, sub-research questions have been formulated to guide this research. Each chapter of this thesis aligns with one or more sub-research questions addressing specific aspects of the main research question. These sub-research questions are as follows:

Chapter 2: Literature Study

- **RQ1: What can literature tell us about the various variants of the VRP and the heuristics used within the algorithm of Company X?**

ORTEC is solving a lot of different variants of the VRP including the one of Company X with algorithms that use construction, local search and Ruin and Recreate (R&R) methods. Consequently, enhancing knowledge about these variants and methods is essential.

- **RQ2: What is parallel computing and how can it be used to solve VRPs?**

We are conducting research about the effect of parallel computing, consequently, we need to have a better understanding of the concept of parallel computing and how it can be used to solve VRPs.

- **RQ3: What can literature tell us about the algorithm configuration problem and what solution methods are available?**

Since we will modify the parameters of the algorithm of Company X, we are dealing with an algorithm configuration problem. For this reason, we need to conduct literature research about this problem and its solving methods in order to formulate an appropriate solution approach.

Chapter 3: Problem Context

- **RQ4: What does the algorithm of Company X look like?**

Since we will adjust the parameters of the algorithm of Company X, we need to have a detailed understanding of the algorithm.

- **RQ5: How can parallel computing be implemented within the OHD software?**

Given that we will design a new parallel algorithm, we need to know what forms of parallel computing are available in OHD.

Chapter 4: Solution Design

- **RQ6: How can the mathematical formulation of the encountered algorithm configuration problem be defined?**

In this research, we are solving a variant of the algorithm configuration problem. However, before we can create a solution approach, we need to formulate our problem such that the problem is clear.

- **RQ7: What solution approach will be used to design a new parallel algorithm?**

After the problem we are facing is formulated, we need to design a solution approach to solve it.

Chapter 5: Data Analysis

- **RQ8: What are the characteristics of the input data of Company X that could be used by designing new algorithms?**

A data analysis will be conducted to gain a deeper understanding of the parameter settings of the algorithm of Company X and to introduce new algorithms.

Chapter 6: Experiments

- **RQ9: How should the construction and R&R phase of our new parallel algorithm look like?**

After the solution approach is formulated and the data analysis is conducted, the construction and R&R phase of our new parallel algorithm should be designed.

- **RQ10: How does the new proposed parallel algorithm perform when applied to real-world data from Company X, and how does it compare to the algorithm of Company X?**

After the new parallel is designed, it needs to be tested against the algorithm of Company X on the data of Company X in order to obtain the effect of parallel computing.

Chapter 7: Conclusions, recommendations and further research

- **RQ11: What are the practical implications of the results obtained, and what recommendations can be made to ORTEC for implementing parallel computing in their software?**

Finally, after all research has been conducted and the solution has been designed and evaluated, conclusions can be made and recommendations for ORTEC and further research are provided.

1.6 Significance of the Study

The successful implementation of parallel computing into the OHD software of ORTEC will provide better solutions in the same computational time to the routing problems of customers. This brings several advantages. Firstly, ORTEC frequently needs to provide a Proof of Concept (PoC) before securing contracts with clients. Sometimes, companies request PoCs from multiple route-optimizing service providers and ultimately sign contracts with the provider offering the best solution. Thus, improving ORTEC's solutions could lead to winning more PoC battles and acquiring more customers. Secondly, customers will benefit from reduced transportation costs, increased efficiency and lower CO2 emissions, resulting in enhanced satisfaction. Furthermore, due to value-sharing contracts, ORTEC stands to benefit from these savings as well. Consequently, successfully implementing parallel computing into OHD will result in increased profits for ORTEC. Additionally, this research contributes to the broader field of combinatorial optimization and logistics by addressing a real-world problem with practical implications.

1.7 Scope and Limitations

It is important to acknowledge the scope and limitations of this thesis. While we aim to provide a comprehensive solution to ORTEC's problem, certain simplifications or assumptions must be made due to the limited time to finish this master thesis assignment.

First of all, we will focus on the effect of using parallel computing in the construction and R&R phase of the algorithm. Although it would be intriguing to test the implementation of parallel computing in all three phases, ORTEC is mostly interested in the effect of using parallel computing in the construction and R&R phase, where the primary focus is directed towards the parallel construction of starting solutions.

Furthermore, the implementation of parallel computing in the OHD software is exclusively executed for the algorithm and datasets specific to Company X. Although the datasets are modified such that they align to other customers' datasets and some valid general observations can be made, we cannot state with full certainty hard conclusions for the OHD software in general.

Besides, it is not possible to evaluate our proposed algorithm against datasets and algorithms documented in the literature since in this research we are testing within the software of ORTEC. In the software of ORTEC, we can't reproduce an algorithm from the literature because we can only make use of their implementations. Simplifications and assumptions about the experimental settings will be discussed later in this report.

1.8 Thesis Structure

This thesis is structured as follows:

- **Chapter 2: Literature Study** - This chapter provides an in-depth review of the existing literature related to the different variants of VRP and its solution approaches. Furthermore, the concept of parallel computing is explained and its application on VRPs is reviewed. Moreover, the algorithm configuration problem will be introduced and an overview of the approaches used to solve it is provided.
- **Chapter 3: Problem Context** - In this chapter, we provide a thorough explanation of the algorithm of Company X, followed by a discussion on the possibilities of parallel computing within OHD.
- **Chapter 4: Solution Design** - This chapter presents the mathematical formulation of our variant of the algorithm configuration problem and the solution approach used to address this problem.
- **Chapter 5: Data Analysis** - This chapter provides insights into the characteristics of the input data of Company X which are crucial for solving the problem of Company X.
- **Chapter 6: Experiments Construction phase** - Here, we discuss the experimental design including the results obtained from testing our new proposed parallel construction phase on Company X's data.
- **Chapter 7: Experiments R&R phase** - This chapter outlines the experimental design including the results obtained from testing our new proposed parallel R&R phase on Company X's data.
- **Chapter 8: Conclusions, recommendations and further research** - This final chapter analyzes the results, discusses the implications, and provides the key findings, recommendations and future research directions for ORTEC.

Chapter 2

Literature Study

This chapter provides a comprehensive literature review such that we can answer the first three research questions that were stated in Section 1.5. The purpose of this literature study is to acquire the necessary knowledge and contextual understanding to develop a new parallel algorithm that can solve the routing problem of Company X. In Section 2.1 an overview of various variants of the VRP is presented. The solving methods used by ORTEC to solve all sorts of variants of the VRP are described in Section 2.2. Moreover, in Section 2.3, an extensive overview of parallel metaheuristics is provided and relevant applications of parallel computing for solving VRPs are reviewed in Section 2.4. Furthermore, the algorithm configuration problem, along with several solving methods, is outlined in Section 2.5. Finally, in Section 2.6 relevant insights obtained from the literature study are summarised.

2.1 Variants of the Vehicle Routing Problem

The vehicle routing problem is an extension of the well-known Traveling Salesman Problem (TSP) and was first introduced by Dantzig and Ramser (1959). The TSP aims to visit every node in a single route by minimizing the distance or costs. In the VRP the nodes represent customers that all satisfy a certain amount of demand. The goal of the VRP is to deliver the demanded goods from a depot to the customers in the most efficient way by minimizing costs or travelling time while allowing the use of multiple vehicles. All routes have to start and end at the depot location. The distinction between the TSP and VRP can be seen in Figure 2.1. There exist many variants and extensions of this classical VRP and some of them are discussed in the following subsections.

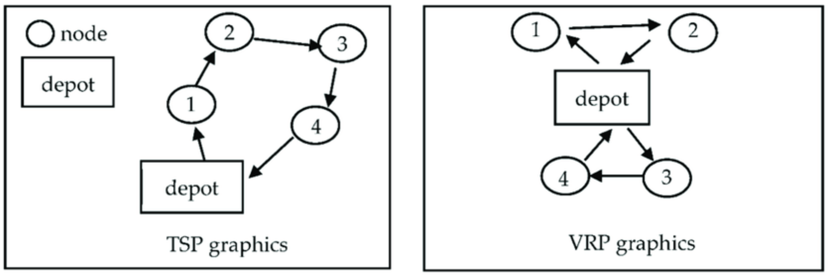


Figure 2.1: Difference between TSP and VRP (Lin et al., 2022)

2.1.1 Capacitated Vehicle Routing Problem

The first extension of the classical VRP that will be discussed is the Capacitated Vehicle Routing Problem (CVRP). In the CVRP a restriction is added on the capacity of the vehicles. The customers have to be served by a vehicle fleet that has limited capacity, contrary to the classical VRP where the vehicles have unlimited capacity. The fleet of vehicles is homogeneous, meaning that they all have the same characteristics and thus the same capacity. The goal is to determine the most efficient routes for a homogeneous fleet of vehicles to deliver goods from a depot to a set of customers while respecting the capacity constraints of the vehicles.

2.1.2 Heterogeneous Vehicle Routing Problem

In the classical VRP and the CVRP only a homogeneous fleet of vehicles is considered however in reality this is rarely the case. In the Heterogeneous Vehicle Routing Problem (HVRP) vehicles are not considered to be identical anymore and different types of vehicles can be used to serve all the customers. The term 'different types of vehicle' can be interpreted in various ways. The characteristics of the vehicles such as capacity, maximum driving speed, fixed and variable costs, serving areas etc. could diverge. An extensive literature review of the HVRP has been written by Soonpracha et al. (2014).

2.1.3 Vehicle Routing Problem with Time Windows

In the previous variants of the VRP, it was assumed that the products could be delivered to the customers at any time of the day. This is of course not realistic, especially not for the home delivery service of a grocery store. Customers want their products to be delivered at times when they are at home. The Vehicle Routing Problem with Time Windows (VRPTW) addresses this extra constraint. The time window assigned to a customer signifies the permissible timeframe for visiting the address of the customer. The problem can include hard or soft time windows. In case of hard time windows, the vehicle should arrive and leave within the time window of the customer. Note that handling time is also considered at the customer location. No violations are allowed. If for example, a vehicle arrives before the start of the time window, it has to wait until the start of the time window. In some cases, this waiting time is penalised with extra costs to improve the efficiency of the solution since in that case the driver and vehicle are idle. On the other hand, soft time windows may be violated but penalty costs will be assigned to penalise this violation and prevent the optimizer from ignoring the time window restrictions. The addition of the time window constraints can greatly impact the solution and its objective value. In Figure 2.2 a small vehicle routing problem is displayed. The dots with numbers are customers and the time windows of the customers' locations are provided within brackets. If time windows are ignored, the sequence of the customers in the routes of the optimal solution would have been in the same sequence as the corresponding numbers. We can see that including the time window restrictions has a relatively large impact on the solution. The total distance travelled that is needed to deliver the products while respecting the time windows is larger than the case where we exclude the time window constraint.

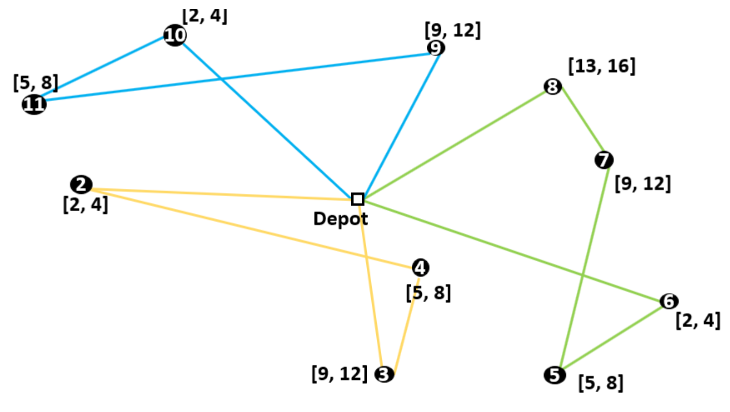


Figure 2.2: Impact of time windows restrictions in a VRP (AIMMS, 2020)

2.1.4 Time-Dependent Vehicle Routing Problem

Another variant of the VRP that includes a constraint related to time is the Time-Dependent Vehicle Routing Problem (TDVRP). In the classical VRP, we assume that it always takes the same time to move from Customer A to Customer B. However, the travelling time in rush hours is not equal to the travelling time in night hours. Besides, the delay in rush hours is also depending on the location since the traffic density is different in villages than in large cities. Consequently, congestion has to be taken into account to make the VRP more realistic. This could be done by applying a factor on the travelling time in rush hours or by using complex forecast models.

2.1.5 Multi-Trip Vehicle Routing Problem

In the basic variants of the VRP, each vehicle is only assigned to one route during a planning period which is typically a day. This is unrepresentative for many practical cases of companies where a vehicle makes multiple trips during a day. The Multi-Trip Vehicle Routing Problem (MTVRP) allows the use of a single vehicle for multiple routes. Brandão and Mercer (1998) showed that the number of vehicles used could be reduced by allowing vehicles to make multiple trips instead of only single trips in a real-life case at a company. As a consequence, the costs of delivering decreased.

2.1.6 Multi-Depot Vehicle Routing Problem

So far we have only considered VRPs that serve all the customers from one central depot. However, in real-life applications of the VRP, there are often multiple depots that are used to serve the customers (Lahyani et al., 2015). The Multi-Depot Vehicle Routing Problem (MDVRP) deals with this additional restriction. There are many variants of the MDVRP in the literature. In the standard MDVRP, every vehicle route must start and end at the same depot. There also exist extensions of the MDVRP where vehicles are allowed to visit other depots. Moreover, the customer allocations to depots could diverge between variants by allowing a customer to be served from only one specific depot or multiple depots. Furthermore, the depots could have different characteristics such as capacity and costs. This variant is called a Heterogeneous Multi-Depot Vehicle Routing Problem (HMDVRP). More variants and extensions can be found in Montoya-Torres et al. (2015) which conducted a literature review on the MDVRP.

2.1.7 Site-Dependent Vehicle Routing Problem

In the previously discussed versions of the VRP, it was assumed that every vehicle is allowed to visit every customer. However, in some real-life applications of the vehicle routing problem, some customers can only be visited by a subset of the vehicles. For example, it may be possible that customers in congested areas can only be visited by smaller types of vehicles. Furthermore, it could be the case that the order of a customer requires specific facilities in the vehicle such as freezing compartments. This variant of the VRP is called the Site-Dependent Vehicle Routing Problem (SDVRP) and Cordeau and Laporte (2001) presents an algorithm capable of solving this problem.

2.1.8 Vehicle Routing Problem with driving hours regulations

In the formed subsections, the routing problems did not take into account the European Community (EC) Regulation concerning driving and working hours also known as EC social legislation. In case this legislation is neglected, severe fines can be received. Therefore, these regulations have an enormous impact on the design of the routes Kok (2010). The Vehicle Routing Problem with EC social legislation (VRP-EC) does take into account the driving and working hours rules imposed by the European Union which are more restrictive than the Vehicle Routing Problem with Hours-Of-Service regulations (VRP-HOS) that takes into account the driving rules of the United States. Consequently, any solution method for the VRP-EC can also solve the VRP-HOS (Hans et al., 2010). These regulations include restrictions on the maximum length of a driving and working period. A working or driving period is ended by a long break. Furthermore, there is a restriction on daily and weekly driving, working and rest time.

2.1.9 Literature review on the VRP of Company X

In the preceding paragraphs, all features of the problem of Company X, introduced in Section 1.2, are examined individually. However, the problem of Company X encompasses all these characteristics collectively. Consequently, the problem of Company X, introduced in Section 1.2, could be described as a variant of the Site-Dependent, Time-Dependent, Multi-Trip, Heterogeneous Multi-Depot, Heterogeneous Vehicle Routing Problem with Time Windows and EC social legislation (SDTDMTHMDHVRPTWEC). In this thesis, we will refer to the problem of Company X as VRPX to improve readability. We conducted a systematic literature review to identify related articles that address VRP variants similar to the VRPX in this section. The VRPX could be described as a rich, real-life or multi-attribute VRP but also as a variant of the E-grocery Delivery Routing Problem (EDRP). The EDRP is a subclass of the VRP and captures a family of problems that an online grocery is commonly encountering. In Table 2.1 the findings of our systematic literature research are visualized.

The first column denotes which article is compared to the VRPX features that are listed in the headers of the other columns. An "✓" in a specific column indicates that the feature mentioned in the header is addressed in the problem discussed by the corresponding article. Analyzing the table, we observe that no other article comprehensively covers all features of the VRPX. On top of that, the systematic literature reviews conducted by Alcaraz et al. (2019), Caceres-Cruz et al. (2014), Liu et al. (2023), and Fan et al. (2021), which collectively examined 151 articles on VRP variants, did not identify a VRP variant with the same features as the VRPX. Consequently, we can conclude that to the best of our knowledge, the VRPX is not yet addressed in the literature.

Table 2.1: Systematic literature research on the VRPX in which VRPs addressed in the articles in literature are compared to the features of the VRPX. All features of the VRPX are listed in the column names and in case it is included in the VRP of the article listed in the first column, the corresponding cell in the table is assigned a "✓", otherwise it is left blank

Article	**Problem features**							
	SD	**TD**	**MT**	**MD**	**HMD**	**H**	**TW**	**EC**
Zhen et al. (2020)			✓	✓			✓	
Dondo and Cerdá (2007)				✓		✓	✓	
Rincon-Garcia et al. (2020)		✓					✓	✓
Dayarian et al. (2015)				✓		✓	✓	
Alcaraz et al. (2019)				✓		✓	✓	✓
Ruinelli (2011)	✓					✓	✓	✓
Viera and Tansini (2004)				✓			✓	
Zare-Reisabadi and Mirmohammadi (2015)	✓					✓	✓	
Fan et al. (2021)		✓		✓			✓	
Pan et al. (2021)		✓	✓				✓	
Zhang et al. (2022)			✓	✓	✓			
E-grocery routing problems								
Liu et al. (2021)				✓		✓	✓	
Zhou et al. (2016)				✓				
Bouwstra et al. (2021)							✓	
Vazquez-Noguerol et al. (2022)			✓	✓			✓	
Carrabs et al. (2017)	✓					✓	✓	
Liu et al. (2020)				✓		✓		
Ensafian (2023)			✓	✓				
Emeç et al. (2016)							✓	
VRPX	✓	✓	✓	✓	✓	✓	✓	✓

The following problem features are included in the table headers:

1. SD: Site-Dependent (Subsection 2.1.7)
2. TD: Time-Dependent (Subsection 2.1.4)
3. MT: Multi-Trip (Subsection 2.1.5)
4. MD: Multi-Depot (Subsection 2.1.6)
5. HMD: Heterogeneous Multi-Depot (Subsection 2.1.6)
6. H: Heterogeneous Vehicle Fleet (Subsection 2.1.2)
7. TW: Time Windows (Subsection 2.1.3)
8. EC: Driver Regulations (Subsection 2.1.8)

2.2 Heuristics for solving VRPs

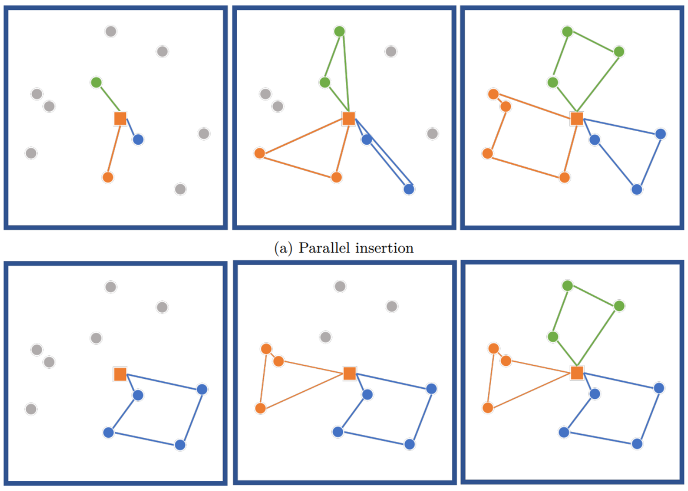
The classical VRP is an NP-hard problem which means that all variants of the VRP are also NP-hard by restriction. This means that the optimal solution cannot be guaranteed within polynomial time (Lenstra and Kan, 1981). Consequently, to be applicable to real-life problem instances, recent research concentrates on approximate algorithms that can find high-quality solutions in a limited time (Kumar and Panneerselvam, 2012). ORTEC uses an approximate algorithm that is created to solve the routing problems of customers in a reasonable time. For our research, it is important to get a better understanding of the solving approach of ORTEC. Therefore, we dive deeper into the concepts of construction heuristics, local improvement heuristics and ruin and recreate methods in this section.

2.2.1 Construction heuristics

Construction heuristics are used to create a solution to a problem from scratch and are often part of larger algorithms. The goal is to efficiently build an initial solution that can be improved by the heuristics in the subsequent phases. In this subsection, we dive deeper into the three main decisions that must be made while designing a construction heuristic within the software of ORTEC. First of all, we have to consider whether we want to use sequential- or parallel insertion. Secondly, a criterion to find a seed task needs to be selected. Finally, a strategy to insert the remaining tasks needs to be chosen.

Sequential- vs Parallel insertion

The difference between these two methods is that parallel insertion builds routes simultaneously, whereas sequential insertion builds per route. This is visualised in Figure 2.3. In Figure 2.3a, it can be seen that parallel insertion creates as many routes as a predetermined number of vehicles and a seed task is assigned to every route. Afterwards, the remaining tasks are inserted into the existing routes by a certain strategy simultaneously. This is contrary to the example of sequential insertion in Figure 2.3b, where one vehicle is selected and tasks are inserted until the insertion results in an infeasible solution. In that case, a new vehicle is selected to build a route. This process is repeated until either all tasks are assigned to a route or no more vehicles are available. The advantage of parallel insertion is that it provides a more global view of the solution space because it considers multiple vehicles and customers simultaneously whereas the sequential insertion focuses on the optimization of the current route. This leads to poor solution quality of the last constructed routes and thus hampers the overall performance of the sequential insertion (Pang, 2011). For example in Laporte et al. (2000), the parallel insertion method dominates the sequential insertion method in terms of the objective value. However, the disadvantage of parallel insertion is that it needs a predetermined number of vehicles that need to be used to build an initial solution. In practice, it is rarely the case that the number of vehicles that will be utilized is known in advance. For this reason, sequential insertion is used most often since it does not require the number of vehicles beforehand.


(a) Parallel insertion

(b) Sequential insertion

Figure 2.3: Difference between sequential and parallel insertion (Jansen, 2023)

Seed task selection

After the decision between sequential and parallel insertion has been made, we need to define a sorting criterion such that we can choose the seed tasks for the routes. There are many ways to do this, but most often in practice, the task that is most difficult to plan is chosen. This is done because there is a high chance that it could not be scheduled later in the process. To maximize the number of tasks planned, it needs to be planned first. Difficult task properties are for example the largest distance to the depot, largest quantity or smallest time window length. In case sequential insertion is used, the vehicles could also be sorted by for example their capacity or costs.

Insertion strategy

Finally, when the first two decisions have been made, we need to find a strategy to insert the remaining tasks into the routes. Often the task that is closest to the route is inserted into the route. However, we could also consider multiple criteria for selecting the task that should be inserted. For example, the task with the largest quantity within a certain distance to the route could be inserted. Moreover, if two tasks within this area have the same ordering quantity, the one with e.g. the smallest time window could be selected. In this way, many different insertion strategies could be created.

2.2.2 Local improvement heuristics

The initial solution that is created by the construction method can often be improved by applying local improvement heuristics. A local improvement heuristic explores the neighbourhood of the starting solution by making small changes. A list of candidate solutions is constructed and the new solution that has the largest improvement is selected and used as the starting solution for the next iteration. This process is repeated until either no improvement can be found or the maximum number of iterations is exceeded. Improvement heuristics could be searching for improvements within routes (intra-route) or between routes (inter-route). Multiple improvement heuristics can be used in recursion since one improvement heuristic may lead to the possibility of improvement by another improvement heuristic. In the following paragraphs, some improvement heuristics that are relevant to this research are explained and all of them can be used as well intra- and inter-route.

2-Opt

The 2-Opt improvement heuristic removes two connections from a route and by reconnecting the edges a new solution is created. This method tries to improve the current solution by removing crossings from the trips as can be seen in Figure 2.4.

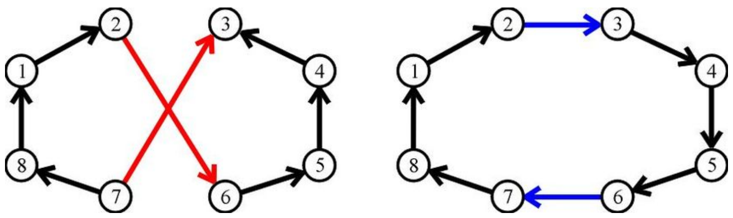


Figure 2.4: Example of 2-opt improvement (Zunic et al., 2017)

Move

This improvement heuristic relocates a task or a group of tasks to another place. An example is presented in Figure 2.5, where task i is moved between task j and $j+1$.

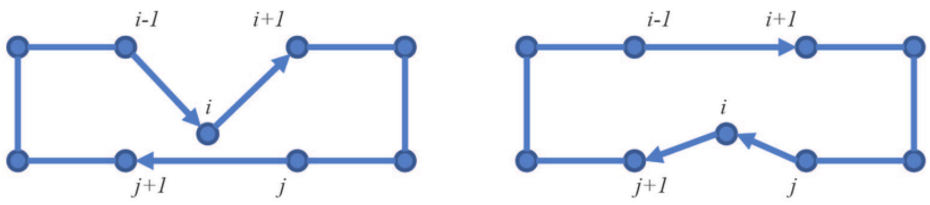


Figure 2.5: Example of a Move improvement (Tinarut and Leksakul, 2019)

Swap

The Swap improvement heuristic exchanges the place in a route between two sets of task(s). In Figure 2.6 task *i* is placed on the position of task *j* and vice versa.

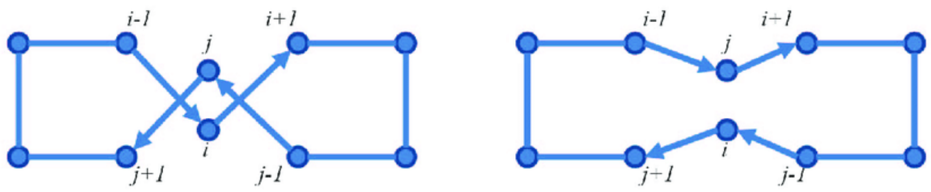


Figure 2.6: Example of a Swap improvement (Tinarut and Leksakul, 2019)

Cross Exchange

The Cross Exchange removes two times two crosses as is visualised in Figure 2.7. It can be seen as performing 2-Opt twice between routes that cross paths or a swap between two sets of two tasks.

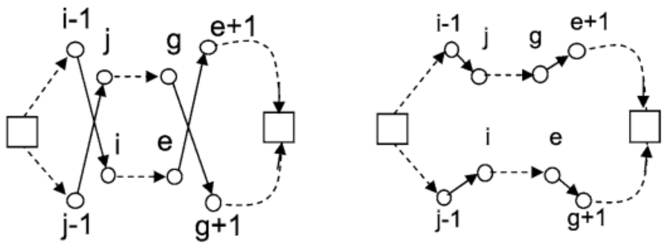


Figure 2.7: Example of a Cross Exchange improvement (Paraskevopoulos et al., 2008)

2.2.3 Ruin & Recreate methods

The main drawback of the local search is that it can get stuck in a local optimum that is significantly worse than the global optimum (Lourenço et al., 2003). Since it does not accept worse solutions and is only searching in a small neighbourhood, it cannot get out of this local optimum. This scenario is illustrated in Figure 2.8. Suppose the starting solution is the green dot in the graph. Since we are applying local search we only accept downwards moves. As a consequence, we get stuck in the local optimum. To overcome this problem and find the global optimum or, most likely, a better local optimum, the metaheuristic ruin and recreate is used. R&R is capable of making the move of the orange arrow by exploring a large neighbourhood. This is accomplished by destroying and rebuilding a part of the solution. The size of the destruction is an important decision that largely affects the performance of the R&R. If we destroy a too big part of the solution, the R&R may behave like a random restart, so the chance of finding a better solution is small. On the other hand, if the perturbation is too small, the R&R will often fall back into the same local optimum from which it is trying to escape. As a result, the diversification of the search space will be limited (Lourenço et al., 2003). If a rebuilt solution is better or within a certain threshold from the current solution, a short local search method is applied to intensify the search in the newly discovered solution space. Afterwards, the new solution is compared to the current best solution and the new solution is only accepted if it is better than the current one. In the following paragraphs, we will explain some ruin and recreate methods that are used by ORTEC.

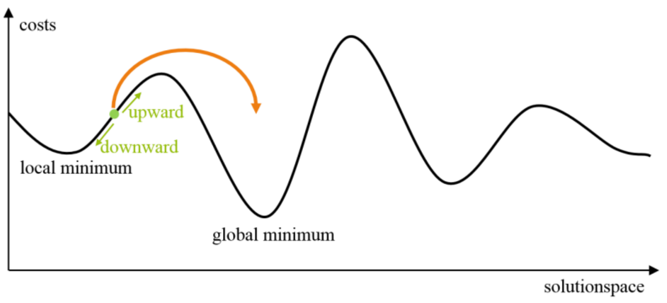


Figure 2.8: Illustration of a local search method that is stuck in a local optimum (Simons, 2017)

Ruin methods

There are many ways of destroying a part of a solution. We can choose to remove tasks randomly. This method is known as Random Removal. The Related Removal will randomly select a seed task, and tasks that are most related in terms of distance will be removed as well. The most expensive tasks will be removed in case the Worst Removal method is used. The Random Cluster Removal will randomly select some seed tasks and remove those and all tasks in the direct neighbourhood from the current solution. Similarly, the tasks in the neighbourhood of a predefined number of the most expensive tasks are removed from the current solution by using the Worst Cluster Removal. In the Route Removal method, entire routes are expunged from the current solution attempting to reduce the total number of routes. In this method, more empty routes are preferred above full routes while selecting the routes that will be removed.

Recreate methods

The construction heuristics mentioned in Subsection 2.2.1 are used as a recreating method to rebuild the destroyed solution. Note that the parallel insertion method is beneficial in this case since we now know the number of routes before rebuilding the solution.

2.3 Parallel computing

Parallel computing is defined as the application of two or more processing units to solve a single problem (Scott et al., 2005). Parallelism thus follows from a decomposition of the total computational load and the distribution of the resulting tasks to available processors (Crainic, 2019). This decomposition of the computational load can be done in numerous approaches, therefore a classification of parallel metaheuristic strategies is provided in Subsection 2.3.1. In Subsection 2.3.2, we briefly present the concept of low-level parallelism. However, for our research high-level parallelism is more relevant since we desire to find a better solution in the same computational time. Thus we review high-level parallelism in more depth in Subsection 2.3.3. Finally, we provide an overview of the advantages of parallel computing in Subsection 2.3.4.

2.3.1 Classification for parallel metaheuristic strategies

A frequently used classification method for the parallel metaheuristic strategies is presented by Crainic (2008) and includes three dimensions. The first dimension, *Search Control Cardinality*, indicates whether the global parallel search is controlled by a single processor (1-control, 1C) or distributed among several processors (p-control, pC). The second dimension, *Search Control and Communications*, denotes the way information is exchanged between the processors and distinguishes between synchronous and asynchronous communication. In the case of synchronous communication, information is shared between the processors on predefined moments (e.g. number of iterations or running time). This means that all involved processes have to stop and wait for every processor to arrive at the synchronization point. Afterwards, information is exchanged and the search continues. In Rigid Synchronization (RS), the processors only communicate at the end of the algorithm to identify the best overall solution. If communication is exchanged at predetermined intervals while executing the algorithm, it is called Knowledge-based Synchronization (KS). In the case of asynchronous communication, the processors autonomously undertake their search and initiate communications with other processors. The global search concludes upon the completion of each search. In the case of collegial communication (C), a single solution is sent whereas in knowledge-based collegial communication (KC) supplementary information is transmitted to the other processors. Finally, the third dimension *Search Differentiation*, accounts for the way different searches are executed. Solution methods can start from the same or different solutions and can make use of the same or different search strategies. Consequently, the following four classes can be derived: Same initial Point/Population, Same search Strategy (SPSS); Same initial Point/Population, Different search Strategies (SPDS); Multiple initial Points/Populations, Same search Strategies (MPSS); Multiple initial Points/Populations, Different search Strategies (MPDS). Logically, ″point″ is used for trajectory-based metaheuristics and ″population″ is used for evolutionary methods. In SPDS and MPDS, the searches may be performed by different algorithms or, more commonly, by the same algorithm with different parameters (Ghiani et al., 2003).

2.3.2 Low-level parallelism

Low-level parallelism is also known as acceleration strategy or functional parallelism since it aims to speed up a sequential metaheuristic and it does not modify the behaviour of the search trajectory. The implementation of low-level parallelism is mostly targeted in 1C/RS/SPSS designs with the use of the master-slave topology. In the classical master-slave approach the master executes a sequential metaheuristic but dispatches time-consuming tasks that will be executed by the slaves in parallel. In this way, computational time is saved. The tasks can be to evaluate the neighbourhood of a solution or calculate the objective value of a partial solution. The master receives and integrates the results obtained from their slaves and moves on to the next procedure. The impact of such low-level strategies has proved limited since the search trajectory of the parallel procedure is quite similar to its sequential counterpart (Crainic, 2008). However, when neighbourhoods are large, the reduction in computing time may become interesting while using a large number of processors.

2.3.3 High-level parallelism

In high-level parallelism, multiple search processes are executed simultaneously. These methods have generally offered better performances, in terms of solution quality and computing times, than the methods of low-level parallelism (Crainic, 2008). Most applications of multi-search methods belong to the pC category described in Subsection 2.3.1. In the following paragraphs, some high-level parallelism strategies will be discussed.

Domain decomposition methods

The basic idea of domain decomposition methods is to separate the search space into smaller and easier subproblems that will be addressed by applying a sequential metaheuristic. From these partial solutions, a global solution is constructed that tackles the entire problem. By fixing or discarding variables and constraints, a separation of the search space can be obtained. ″This separation may result in a partition (disjoint subsets) or a coverage (subsets may overlap) of the overall search space″ (Schryen, 2020). The partitioning reduces the size of the solution space but it also leaves large portions of the solution space unexplored. Consequently, the separation has to be repeated to explore the entire solution space. Domain decomposition is usually implemented using a master-slave 1C/RS scheme with MPSS or MPDS searching strategies. The master process determines the separation of the search space and assigns the partial subsets to the slaves. The slaves will simultaneously and independently execute their search on their distributed subset. Afterwards, the master process collects partial solutions, reconstructs the complete solution and modifies the separation. This process is continued until the termination criterion is met. In contrast to sequential metaheuristic methods, domain decomposition methods have different search behaviour and solution quality. These methodologies prove particularly advantageous when dealing with larger problem instances.

Independent multi-search methods

The independent multi-search methods were one of the first parallelization methods published in the literature because of its simplicity. This method belongs to the pC/RS class, in which MPSS, SPDS, and MPDS can be used as search differentiation. It parallelizes the multi-start strategy by initiating concurrent solvers that explore the search space starting from the same or different initial solutions. In this manner, the independent multi-search methods expedite the exploration of the search space aiming for an improved solution compared to sequential searching methods. No advantage of running multiple solvers in parallel is taken apart from the identification of the best solution among all searches at the final synchronization step. The independent multi-search method is rather easy to implement and generally offers an interesting performance (Crainic, 2019).

Cooperative multi-search methods

Cooperative multi-search has emerged as one of the most successful meta-heuristic methodologies to address hard optimization problems (Crainic and Toulouse, 2010). Cooperative strategies often offer superior performances compared to independent search because they take advantage of the parallel diversified exploration of the search space (Crainic, 2008). Information is shared between processors while the search is ongoing. The design of the information communication mechanism is a fundamental element to the strong performance of the cooperative multi-search methods. Questions relative to when and what information to exchange and among what processors are often asked. As mentioned in the classification provided in Subsection 2.3.1, we distinguish two moments of information exchange either synchronously (directly) or asynchronously (indirectly).

The synchronous cooperative strategies are generally used by population-based methods and belong to the pC/KS class. The goal is to re-generate a state of complete knowledge at specific points in the search by exchanging the global best solution to all solvers, it assumes to find a superior solution. This goal is generally only partially attained, although it generally outperforms the sequential versions as well as simpler parallelization strategies (Crainic, 2019), the approach has a few drawbacks. Firstly, sending the overall best solution to all cooperating solvers leads to the case that solvers are exploring the same parts of the search space. This rapidly decreases the efficiency and diversity of the search and causes an early convergence of the algorithm to a potentially not-so-good local optimum. To overcome this drawback, instead of using a fully connected communication graph, less densely connected communication topologies can be used. Twomey et al. (2010) presents an analysis of communication policies including the ring, hypercube and fully connected topologies. Moreover, in synchronization information sharing the communications are initiated when the slowest solver has completed his search. This results in idle time for the other processors and thus in significant time inefficiencies. Consequently, asynchronous information sharing seems more promising, guided by intuitive considerations.

As expected asynchronous information sharing generally outperforms synchronous methods (Crainic, 2019) and is used by most trajectory-based cooperative multi-search methods. The asynchronous exchanges of information are regulated by a central memory also referred to as solution- pool or warehouse. In case a processor desires to send out information, such as a new local optimum, it is dispatched to the central memory. Similarly, when a processor requires external information, perhaps to diversify the search, it retrieves this information from the central memory. Communication in both directions is initiated solely by individual processors. In case the central memory only stores complete solutions, the cooperative multi-search method belongs to the pC/C class. If mechanisms that can create new information and solutions based on the solutions stored in the central memory are included in the cooperative multi-search methods, they belong to the class pC/KC.

A general observation for both synchronous and asynchronous cooperative strategies is that exchanges should not be too frequent to avoid excessive communication overhead as well as premature convergence to local optima (Toulouse et al., 1999).

2.3.4 Advantages parallel computing

The main advantages of the implementation of parallel computing in metaheuristics are multi-faceted. Firstly, it accelerates the research procedure since multiple processors are working on the computational load instead of only one. In this way, a predetermined level of solution quality can be accomplished faster. Secondly, very large instances of complex optimization problems that can not be solved by a sequential machine could be solved by parallel metaheuristics. Moreover, the quality of the obtained solutions can be improved since a larger part of the search space could be explored in the same or less computational time compared to sequential methods. Last but not least, parallel metaheuristics can improve the robustness of the algorithm. In this context, robustness means that the algorithm performs equally well to a large and varied set of problem instances without excessive calibration.

2.4 Parallel computing and VRPs

Many parallel computing applications that solve VRPs in the literature use population-based methods. Since this would require a serious transformation of the software of ORTEC these applications are not addressed in this review. For our research, it is relevant to find applications of multi-start algorithms since we are interested in the effects of using more than one initial solution in the algorithm of Company X. According to Czapiński (2013), initial solutions have a small impact on the running time of the algorithm but they can have a significant influence on the quality of the solutions. In the following subsections, several methods to generate different initial solutions are discussed.

2.4.1 Different parameter settings

Polat (2017) proposes a parallel variable neighbourhood search that is initiated from different starting solutions. To construct multiple different initial solutions for each processor different parameter settings are used in the same construction heuristic. The parameters are randomly drawn from an interval.

2.4.2 Random seed values

In Cordeau and Maischberger (2012) each processor generates a unique initial solution by using different random seed values. The different seed values will result in different seed customers while building the routes. Customers are sorted in ascending order based on the angle they make with the depot and are inserted in this sequence.

In Bräysy et al. (2004) randomly an unrouted customer among the customers farthest from the depot or among the customers having the earliest end of the time window is selected as a seed for a new route. The selection criterion is randomly chosen when constructing a new starting solution. While inserting, only closely located customers are considered whereas insertion of customers that are located far away from the depot is favoured. Since they are usually considered the most challenging ones to plan.

2.4.3 Different construction heuristics

Le Bouthillier and Crainic (2005) presents a parallel cooperative meta-heuristic that uses two phases to produce initial solutions. First, four different fast-construction heuristics are used to create solutions from scratch. Afterwards, a route elimination method is applied to these solutions before they are used as starting points in the search.

2.4.4 Results

All above-discussed applications of multi-start methods performed well in terms of computational time, robustness and objective values on a large set of benchmark instances. This proves the usefulness of using multiple initial solutions in a parallel metaheuristic.

2.5 Algorithm configuration problem

Many high-performance algorithms designed for computationally hard problems have numerous parameters whose settings greatly impact their effectiveness (Hutter et al., 2009). Consequently, optimizing these parameters is an important task for algorithm developers (Hutter et al., 2010). Traditionally, the optimization of the parameters was carried out through manual experimentation which can be a tedious task, especially when dealing with a large number of parameters. Hence, automated approaches for finding good parameter settings have become available to address the algorithm configuration problem and have led to significant improvements in the state-of-the-art for solving computationally challenging problems (Hutter et al., 2012).

We are also dealing with a variant of the algorithm configuration problem because we will be creating different algorithms for the new parallel algorithm by modifying the parameter settings of the algorithm of Company X. For this reason, we will elaborate more on the algorithm configuration problem by providing the formal definition in Subsection 2.5.1. Subsequently, three state-of-the-art automatic algorithm configuration approaches will be reviewed in Subsection 2.5.2. Finally, in Subsection 2.5.3, a short discussion will be held to summarise the concepts of the solving methods that are relevant to this research.

2.5.1 Problem definition

The algorithm configuration problem involves the challenge of finding the optimal parameter settings of an algorithm that maximizes its performance on a given dataset. Hoos (2012) describes this problem formally as follows:

Given
- an algorithm A with parameters $p_1, \ldots, p_k$ that affect its behavior,
- a space C of configurations (i.e., parameter settings), where each configuration $c \in C$ specifies values for A's parameters such that A's behaviour on a given problem instance is completely specified (up to possible randomization of A),
- a set of problem instances I,
- a performance metric m that measures the performance of A on instance set I for a given configuration c,

find a configuration $c^* \in C$ that results in optimal performance of A on I according to metric m.

The algorithm A whose performance will be optimized is also referred to as the target algorithm and $A(c)$ is used to denote target algorithm A under a specific configuration c. The domain of p includes the set of values any given parameter p can take. The parameters of algorithms can be classified into several types: numerical, ordinal, categorical, boolean or conditional. Numerical parameters are characterized by integer or real number values. Ordinal and categorical parameters have a finite set of discrete values, but categorical parameters lack any meaningful order. Boolean parameters regulate the activation or deactivation of heuristic mechanisms while conditional parameters are only active under specific values of other parameters.

2.5.2 Automated algorithm configuration methods

In this subsection, three state-of-the-art automatic algorithm configuration approaches will be briefly reviewed. Namely, the Iterated F-race, ParamILS and Sequential Model-based Algorithm Configuration (SMAC).

Iterated F-race

The Iterated F-race, also known as I/F-Race, is a racing procedure and is a variant of the F-race. The concept of the F-race is simple: sequentially evaluate all the candidates on a set of instances and eliminate candidates as soon as they perform significantly worse than the candidate with the overall best performance at a given stage of the race. The F-race requires every possible configuration to be evaluated in the initial steps of a race, consequently, F-race is not suitable for large problems and discretization is needed for continuous parameters. To overcome this problem, an iterative application of F-Race was introduced by Balaprakash et al. (2007), namely the I/F-Race. Each iteration consists of two stages, in the first stage a set of candidate configurations is sampled from a probabilistic model. Subsequently, in the second stage, a standard F-Race is performed on the candidate set, where the configurations that survived this race are used to update the probabilistic model that will be used in the next iteration. This iterative process continues until a stopping criterion, such as reaching the computational budget, is met. A more detailed explanation of the I/F-Race including applications can be found in Hoos (2012).

ParamILS

The core of the ParamILS framework is the Iterated Local Search (ILS), a well-known and versatile method that has been successfully applied to a variety of difficult combinatorial problems (Rasku et al., 2019). Within ParamILS, a one-exchange neighbourhood is used to search the space of all algorithm parameter value combinations, altering only one parameter value at a time. The ParamILS algorithm starts its search by identifying the best-performing parameter configuration from a pool comprising the default configuration and randomly generated configurations. This best-performing configuration serves as the starting point for the iterated local search process. Each iteration involves a perturbation phase where small random changes are made to explore the solution space. This is followed by a local search phase, which is aimed at reaching a new local optimum. Subsequently, an acceptance criterion is used to determine whether to continue the search process from this newly discovered local optimum or not. The iterated local search process concludes upon meeting a specified stopping criterion, at which point it returns the best configuration found. Hutter et al. (2009) provides a more elaborate description of the ParamILS along with its variants and extensions.

SMAC

Hutter et al. (2011) introduced Sequential Model-based Algorithm Configuration (SMAC), an iterative framework that operates by alternating between fitting a regression model to observed performance data and using this model to predict the performance of new candidate configurations. SMAC starts with an initial configuration set, which is evaluated on the instance set. Subsequently, it fits a predictive model (such as Gaussian processes or random forests) on this performance data. Using this model, an Expected Improvement function is formulated and local optimization techniques are applied to identify the configuration that maximizes this function. Once the most promising configuration is identified, it is evaluated on the instance set, and the results are used to update the regression model. This iterative process continues until a specified stopping criterion, such as reaching a maximum running time, is met. Finally, SMAC returns the best-evaluated configuration found during the process.

2.5.3 Discussion

The automated algorithm configuration systems discussed in the previous section have demonstrated remarkable performance improvements across a broad range of applications (Geschwender et al., 2014). However, these often lack transparency in their decision-making process, providing limited insight into why specific parameter configurations are selected. This poses a significant drawback, particularly in academic research, where understanding the reasoning behind optimization strategies is essential (Rasku et al., 2019). This aspect holds significance for ORTEC as well, as the findings of this research should be applicable to future projects.

Furthermore, these automated algorithm configuration methods are typically applied to single serial algorithms, whereas our research focuses on a parallel algorithm comprising a set of algorithms. Additionally, the size of this parallel algorithm as well as the number of starting solutions to proceed to the local search and R&R phase are variable. In theory, an automated configuration method could be applied to this problem by treating the whole parallel algorithm as one configuration. However, the number of possible configurations will explode as will be shown in Chapter 4.

The performance of the automated algorithm configuration methods relies on the ability to evaluate a large number, ideally thousands, of configurations (Styles and Hoos, 2013). Since the average running time of the algorithm of Company X is approximately 5 minutes, evaluating, for example, 5000 configurations on a single instance would already take 17.4 days. Consequently, computational times for a representative dataset would be prohibitively long and costly. Cloud computing could be used to reduce this computational time but it would incur additional expenses.

For these reasons, we have opted not to utilize an automated algorithm configuration method in our research. However, we will integrate the main concepts of the automated tuning framework presented in Gunawan et al. (2011). Their framework consists of three phases: screening, exploration and exploitation. Firstly, in the screening phase, an experiment is conducted to determine which parameters are significantly unimportant. These unimportant parameters are set to a constant value to reduce the search space. Furthermore, ranges of the important parameters are updated based on the results of this first experiment. Secondly, in the exploration phase, these new ranges are explored and reduced such that they can be used as a starting point for an automated tuning procedure such as ParamILS to find the optimal parameter setting in the exploitation phase. In Chapter 4 our solution approach will be explained in detail.

2.6 Conclusions

This chapter has presented a comprehensive literature study on relevant concepts related to this research. Firstly, we presented various variants of the VRP that address features of the VRPX individually. Moreover, a literature review was conducted on the VRPX in which we found out that to the best of our knowledge, the vehicle routing problem of Company X does not yet exist in the literature. Furthermore, different construction-, local improvement heuristics and R&R methods are introduced which are the three core elements of the algorithms of ORTEC that are solving routing problems.

Secondly, an overview of a classification for parallel metaheuristics based on the communication between processors, the number of starting solutions and the number of different search strategies is given. We have seen that multi-search methods in general improve the quality and robustness of the solutions. Furthermore, we have seen that different initial solutions for parallel metaheuristics solving VRPs were created by different parameter settings, seed values or construction heuristics.

Finally, a formal definition of the algorithm configuration problem and automated solving approaches are presented and discussed. Unfortunately, the automated solving methods are not suitable for this research however the main concepts of the automated tuning framework of Gunawan et al. (2011) will be used to construct a new parallel algorithm to address the VRPX in the subsequent chapters.

Chapter 3

Problem Context

In this chapter, we provide a more in-depth analysis of the problem context. In Section 3.1, we explain how the algorithm of Company X is constructed, whereas this algorithm is explained in detail in Section 3.2. Finally, the options of using parallel computing within the software of ORTEC are elaborated on in Section 3.3.

3.1 Design process of the algorithm of Company X

When Company X entered into a contract with ORTEC, an optimisation project was started to design a tailored algorithm for Company X. Extensive discussions were conducted to identify the most important Key Performance Indicators (KPIs) for Company X. Subsequently, Company X provided about 20 sets of data from 2 or 3 main depots, enabling the implementation consultants of ORTEC to develop a new algorithm for Company X that meets their needs and performs well for their cases. This new algorithm is also used for the cases of all the other main depots, despite the absence of data for those depots beforehand. Afterwards, the algorithm was only modified in case Company X introduced new requirements. For example, when they started to use priority 3 vehicles, the algorithm was extended such that it was able to handle priority 3 vehicles. In the next section, the algorithm of Company X is explained in detail.

3.2 The algorithm of Company X

In this section, we explain the algorithm of Company X in more detail which is solving the problem of Company X, explained in Sections 1.2 and 2.1. This algorithm consists of three phases as can be seen in Figure 3.1: construction, local search and ruin and recreate. In the following subsections, the methods used in these phases to obtain the objectives stated in Subsection 1.2.10 will be elaborated on. Note that in none of these phases, an infeasible or worse solution than the current solution is accepted as a new solution.

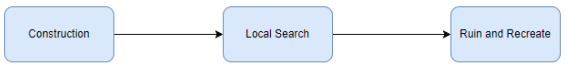


Figure 3.1: Flowchart of the algorithm of Company X

3.2.1 Construction method

Since Company X requires that all vehicles of subdepots should be used, the vehicles of the subdepots are planned first and the vehicles of the main depot last. The subdepots are decreasingly ordered based on the distance to the main depot. Following this sequence, the construction is done separately per depot but they use the same construction method. To comply with the business rules of Company X about priority 1 and priority 2 vans, the routes are scheduled in the following order: priority 1, priority 2 and finally the normal and priority 3 vans. In this way, the customers close to the depot are not taken up by the normal vans and all priority 1 and priority 2 vans will be used. The construction of the routes for the different types of vehicles follows the same structure: seed task selection, insertion of tasks, short intra-route local search, insertion of unplanned tasks and if possible multi-trip construction. Afterwards, when all vehicles of a depot are planned, a local search procedure is applied to all the routes assigned to this specific depot. Finally, unplanned tasks, if any, are trying to be inserted into the routes. The above-mentioned structure of the construction is visualised in a flowchart in Figure 3.2. In the subsequent paragraphs, the main elements of the flowchart are explained in detail. Although the whole structure includes some local search procedures, it is considered as the construction phase.

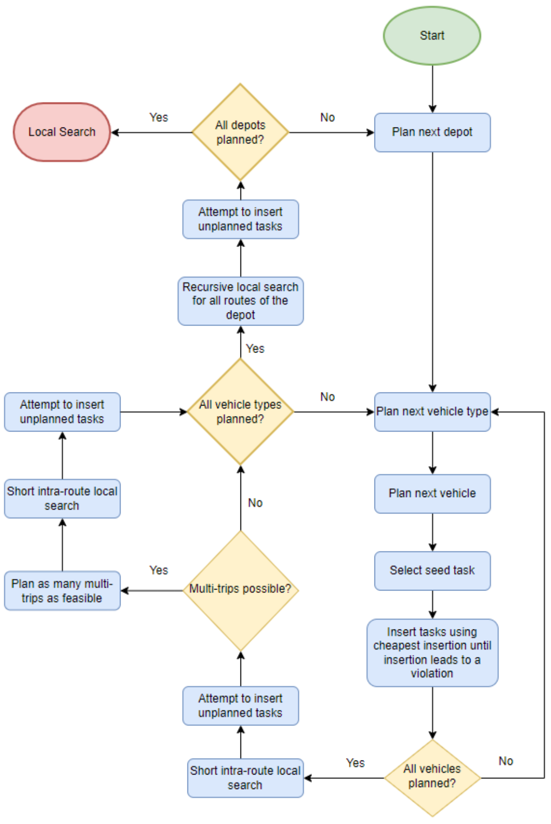

Start
Local Search
Yes
All depots planned?
No
Plan next depot
Attempt to insert unplanned tasks
Recursive local search for all routes of the depot
Yes
Attempt to insert unplanned tasks
All vehicle types planned?
No
Plan next vehicle type
Short intra-route local search
No
Plan next vehicle
Plan as many multi-trips as feasible
Yes
Multi-trips possible?
Select seed task
Insert tasks using cheapest insertion until insertion leads to a violation
Attempt to insert unplanned tasks
Short intra-route local search
Yes
All vehicles planned?
No

Figure 3.2: Flowchart of the construction phase of the algorithm of Company X

Seed task selection

For the priority 1 and priority 2 van routes, the seed task is the closest unassigned customer to the selected depot. The seed task of priority 3 and normal vans is based on the largest difference between the distance to the nearest depot that is not the depot of the selected vehicle and the depot of the selected vehicle. In this criterion, a scale of 10 kilometres is used. This means that customers are sorted in scales of 10 kilometres based on the difference between the distance to the nearest depot that is not the depot of the selected vehicle and the depot of the selected vehicle. The first scale corresponds to the customers with a difference between 0 and 10 kilometres, the second scale to the customers with a difference between 10 and 20 kilometres and so on. The customer that is assigned to the largest scale is used as a seed task. In case multiple customers are assigned to this scale, the one that is located furthest away from the depot of the selected vehicle is selected as a seed task. Additionally, there is also a third sorter based on the quantity included however this one is at the moment not used since the scale of the second sorter is 0. In case a scale is 0, the tasks are ranked and either the first or the last one is selected depending on the sorting direction. Finally, for multi-trip routes, the seed task is the same for all vehicles. Namely, the closest unassigned customer to the depot because it is assumed that there is not enough time to do another long trip.

Insertion of tasks

After the seed task is selected, customers are inserted in the route one by one. The remaining customers are sorted increasingly based on their distance relative to the seed task with a scale of 1 kilometre. This means that we could have selected multiple customers. Consequently, if needed the selected customers are sorted based on the start of the time window with a scale of 1 hour. If these two criteria are not strict enough to select one single customer, the customer of the remaining ones that is located closest to the seed task, is inserted. The selected customer is inserted using the cheapest insertion method while respecting the time window of the customer. This procedure is repeated until insertion leads to a violation. This insertion method is used for all types of vehicles.

Short intra-route local search

If all vehicles of a certain type are planned, a short local search method is applied that only allows changes within a route. This is done to minimize the waiting time per route. The 2-Opt and Move improvement heuristics are applied to every route of the specific vehicle type. These two improvement heuristics are explained in Subsection 2.2.2.

Insertion of unplanned tasks

The intra-route local search method could result in modifications in the trips, creating opportunities for the inclusion of new customers in the route. Therefore, unplanned tasks close to one of the stops in a route are tried to be inserted into the route of a certain vehicle type with the use of sequential cheapest insertion.

Multi-trip construction

After the first trip for every vehicle of a certain type is constructed, multi-trips are considered for all types of vehicles except for priority 1 and priority 2 vans. If feasible, an extra trip is added to a vehicle following the same construction process as the first route. However, before applying the short local search method, an extra trip is constructed and added to the vehicle if feasible. This is done for all vehicles of a certain type until the addition of a new trip leads to a violation. Subsequently, the same short intra-route local search is applied and unplanned tasks are tried to be inserted.

Local search after construction of one depot

After each vehicle of a depot is fully utilised or all customers are inserted, a local search procedure is applied to all routes of this depot. The local search method contains five different improvement heuristics that run recursively: 2-Opt, Cross Exchange, Move&Swap, Move and a route elimination method that tries to shift all tasks of a route to a different route. In Subsection 2.2.2 the Move, Swap, 2-Opt and Cross Exchange improvement heuristics are explained. The Move&Swap improvement heuristic is a combination of the Move and Swap improvement heuristics. The local search procedure stops either after a recursion without improvement or after five recursions.

Planning remaining tasks

Similar to the previous applications of local search procedures, successively unplanned customers are trying to be inserted into the existing routes. However, in this case, parallel cheapest insertion is used. First, customers are tried to be inserted into non-empty trips. If the insertion of an unplanned customer is not feasible in any non-empty trip, a new trip is created if any vehicle is available. Otherwise, it remains unplanned.

3.2.2 Local search

The initial solution, created by the construction method explained in the previous subsection, will be improved via some local search procedures that are allowed to modify routes that belong to different depots. The flowchart of the local search phase of the algorithm of Company X can be seen in Figure 3.3. First of all, the number of trips is attempted to be decreased by moving all tasks from one trip completely to another trip. Secondly, the short intra-route local search method, explained in Paragraph 3.2.1 is applied to optimize the sequence of the routes. Subsequently, the recursive local search, described in Paragraph 3.2.1, is applied to the current solution. Afterwards, a new attempt to insert unplanned customers is made. First, they are tried to be inserted in non-empty routes. If all insertions of a customer will lead to a violation, a new trip is created if any vehicle is available. Otherwise, the customer remains unplanned. If at least one unplanned customer is successively inserted into a route, the recursive local search is again applied to the new solution. Finally, a new effort is made to decrease the number of trips via a route elimination method. These methods together form the local search phase of the algorithm of Company X.

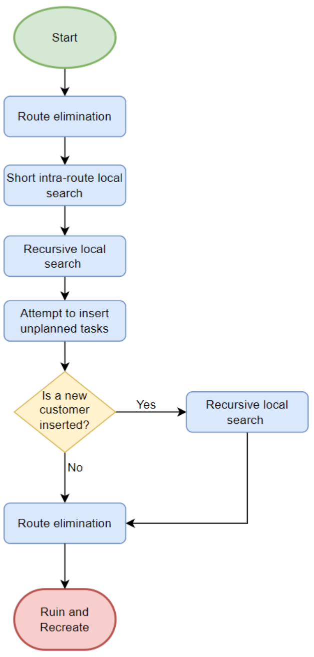


Figure 3.3: Flowchart of the local search phase of the algorithm of Company X

3.2.3 Ruin and Recreate

The algorithm of Company X incorporates the six ruin methods mentioned in Subsection 3.2.3, coupled with the parallel cheapest insertion method as the recreation method. Consequently, there are six possible combinations of R&R methods. To guarantee that the priority 1 and priority 2 vans are still assigned to the customers located close to the depots, the ruin methods are not applied to the routes of priority 1 and priority 2 vans. However, customers who are removed from the solution are allowed to be inserted into priority 1 and priority 2 van routes. Moreover, routes starting at subdepots are favoured over routes starting at the main depot for the insertion of removed customers. Note that the removed customers from the solution by the ruin method will be added to the list of unplanned customers. If it is not possible to plan all customers into the existing routes, overflow routes can be used. The ruin and recreate phase is presented with a flowchart in Figure 3.4 and works with a roulette wheel that is used for 40 iterations in 5 recursions. The roulette wheel chooses one of the six R&R methods that will be used at each iteration. In the first iteration, every method has the same probability of being chosen. If the newly obtained solution by the chosen R&R method is within the threshold margin of 5% of the current solution, a fast local search method is applied. This fast local search method consists of the 2-Opt and Move improvement heuristics that are allowed to make changes inter-route as well as intra-route. If this new solution is better than the current solution, the probability of getting chosen in the next iteration is higher for the used R&R method. On the other hand, if the newly obtained solution is not better than the current solution, the probability of getting chosen in the next iteration decreases for the used R&R method. In this way, the probabilities are updated after each iteration. After each recursion, the probabilities are reset. In case one recursion does not result in an improved solution, the recursion will stop. The short intra-route- and recursive local search are once again applied to the best solution obtained after the recursion of the roulette wheel. Finally, the route elimination method is applied to see whether the number of vehicles used could be decreased or that routes could be assigned to cheaper vehicles.

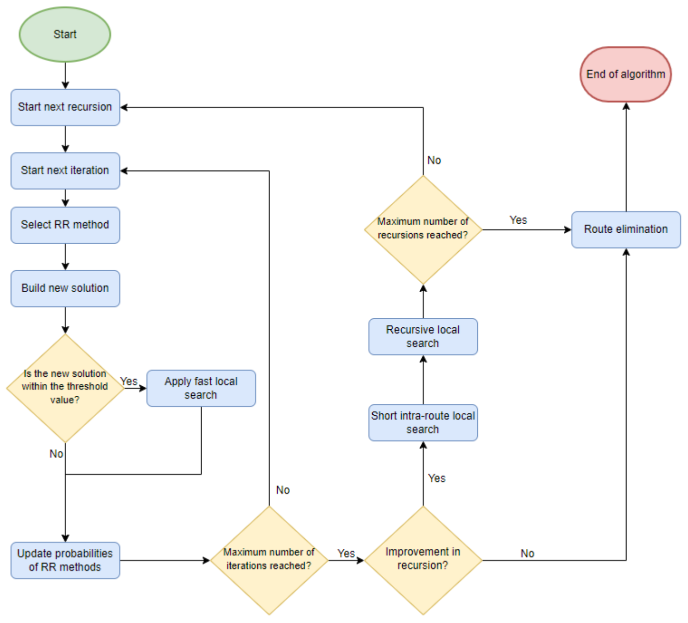

Start
Start next recursion
Start next iteration
Select RR method
Build new solution
Is the new solution within the threshold value?
Yes
Apply fast local search
No
Update probabilities of RR methods
Maximum number of iterations reached?
No
Yes
Improvement in recursion?
Yes
Short intra-route local search
Recursive local search
Maximum number of recursions reached?
No
Yes
Route elimination
No
End of algorithm

Figure 3.4: Flowchart of the ruin and recreate phase of the algorithm of Company X

3.2.4 Final solution

The final solution that will be provided to Company X is either the solution obtained after the whole algorithm was run or the solution that is obtained when the maximum running time is reached. The algorithm will expire well before the maximum duration for the smaller cases. However, for big instances, the algorithm has to be terminated to avoid violating the maximum running time. In the latter case, the best solution achieved up till that point will be provided to Company X.

3.3 Parallel computing at ORTEC

In this section, the experiences and opportunities with the use of parallel computing at ORTEC are discussed. In the OHD software, it is not possible to speed up calculations by the use of low-level parallel computing which concept is explained in Subsection 2.3.2. However, for this thesis high-level parallel computing is relevant and fortunately, that is possible to implement into an algorithm. In the following paragraphs, the use of domain decomposition and multi-search methods within the OHD software will be examined.

3.3.1 Domain decomposition methods

The OHD software offers the ability to make use of the in Subsection 2.3.3 clarified domain decomposition methods. Problems can be split via multiple splitting strategies into disjunct parts and these can be separately optimized in parallel. This type of high-level parallel computing is already used by ORTEC in the algorithm of some customers. The problems of these customers are too large to be solved by a sequential algorithm in a reasonable time. Consequently, the problem is split into disjunct subproblems that are then solved in parallel. This is repeated several times with different splitting criteria to explore a larger search space. Hence, customers located on the edge of multiple areas are included in routes of all of these areas. Afterwards, it can be concluded as to which route it is most optimal to be assigned. In this way, large problems of customers are solved in a reasonable time.

3.3.2 Multi-search strategies

In the OHD software, it is also possible to apply multiple search strategies concurrently. However, this is not yet used in practice in a customer case because it is not clear how much better the solutions will be if multi-search strategies are included. Therefore, ORTEC has not been using multi-search strategies because it would only increase computational costs while no extra revenue is generated. The computational cost per Central Processing Unit (CPU) amounts to 16 cents per hour.

In case multi-search strategies are used in the OHD software, it is only possible to continue with one solution to the next phase of the algorithm. For example, if four different initial solutions are constructed in the construction phase of the algorithm of Company X, only the best solution of those four generated starting solutions may proceed to the local search phase. This logic applies also to the use of parallel computing in the other phases. One exception to use multiple starting solutions is the fact that it is possible to execute complete algorithms in parallel however processors are not able to communicate with each other. In this way, it is still possible to employ the MPDS search differentiation but each implemented algorithm must be executed independently from start to finish, with the optimal solution selected upon completion of each algorithm's run. Moreover, the construction-, local search- and R&R methods are inherently tied and cannot be interchanged during execution. Consequently, while technically feasible, the utilization of the MPDS search differentiation is limited.

3.4 Conclusions

This chapter has provided an in-depth understanding of the problem context. We have explained how the algorithm of Company X was constructed by the employees of ORTEC. Subsequently, a detailed explanation of the three phases of the algorithm of Company X is provided. We have seen that it includes a lot parameters of which it settings could be modified. Moreover, we have seen that both the seed task selection as the insertion method consist of different sorting criteria with scales. Finally, the current opportunities for implementing parallel computing into an algorithm in the OHD software are explored. We found out that communication between processors is not possible but independent multi-search methods are possible in case you either use one starting solution or all.

Chapter 4

Solution Design

We have seen in the literature study in Chapter 2 that the use of multiple construction methods by changing parameter settings led to good results. Furthermore, it can be concluded from Subsection 3.2.1 that the entire construction method of the algorithm of Company X is quite complex. For this reason, we are going to create different construction methods by modifying the parameter settings of the seed task selection- and insertion method of the algorithm of Company X instead of designing completely new construction methods. The latter method would require a lot of construction work compared to the former one. Consequently, it would then cost ORTEC a lot of time and thus money to adapt algorithms into a parallel version. Section 4.1 provides an overview of the available parameter settings of the seed task selection- and insertion method of the algorithm of Company X. Subsequently, as mentioned in Section 1.3 the problem of Company X is slightly modified to align closer with other customers problems. The modification made to the problem of Company X will be explained in Section 4.2. Furthermore, the mathematical formulation of the problem we are facing is given in Section 4.3. Finally, the solution approach employed to tackle this problem is outlined in Section 4.4.

4.1 Parameters construction method

In this section, we elaborate in-depth on the seed task selection procedure and the insertion method since the parameters of these will be modified to construct multiple construction methods for our new parallel algorithm.

4.1.1 Seed task selection methods

As mentioned in Section 3.2.1 currently, the seed task for the normal and priority 3 vans is selected considering three different criteria:

1. The difference between the nearest depot that is not the depot of the selected vehicle and the depot of the selected vehicle
2. The distance of the customer with respect to the starting depot of the selected vehicle
3. The order quantity of a customer in terms of weight

Currently, the first sorter operates on a scale of 10 kilometers, decreasingly. Similarly, the second sorter also operates decreasingly, with a scale of 0, indicating that the third sorter, which sorts based on the order quantity of customers in decreasing order, is not utilized. This setup allows us to experiment with adjusting the scales to alter the significance of other sorters. For example, reducing the scale of the first sorter would increase its importance, as a smaller scale would result in greater differentiation between customers based on the distance between their nearest depot which is not the depot of the selected vehicle and the selected vehicle's depot, and vice versa.

Moreover, we have the flexibility to modify the direction of the scales and introduce additional or alternative sorting criteria. In total, there are 24 different sorting criteria available in OHD, such as the start, duration and finish times of time windows, x- and y-coordinates, and distance to the selected depot. Each criterion includes a scale that can take positive real values and can be used in either decreasing or increasing order. This versatility allows us to create a wide range of seed task selection strategies.

4.1.2 Insertion methods

In Section 3.2.1 is explained that we use three sorting criteria to select the customer that will be inserted in the route:

1. The distance of the customer with respect to the seed task
2. The start of the time window of the customer
3. The distance of the customer with respect to the seed task

Currently, the scale of the first sorter is 1 kilometer and increasing. The second sorter also operates increasingly, with a scale of 1 hour. The third sorter is also increasing and uses a scale of 0. As described in the previous subsection, we have the flexibility to experiment with adjusting the scales and directions of the sorters, as well as adding or replacing sorting criteria. However, in the insertion method, there are even more parameter settings that can be modified.

Firstly, tasks are at the moment inserted with a batch size of 1, meaning that customers are inserted one by one. However, we have the option to insert multiple tasks at once, for example, inserting 5 tasks simultaneously. The batch size is variable, as is the number of batches created. Furthermore, tasks are now inserted by the cheapest insertion method however a parallel cheapest insertion method is also available. Moreover, tasks could be resorted after each insertion of a batch such that the customers are not only selected based on the distance to the seed task but also on the distance to every other task that is already in the route. Additionally, batch sizes larger than 1 could be sorted while using cheapest insertion. Finally, the estimator used in the cheapest and parallel cheapest insertion could be changed. As a result, an exorbitant number of insertion methods could be created by experimenting with above-mentioned parameter settings.

4.1.3 Degree of complexity

In the preceding subsections, it could be obtained that an immense number of different construction methods could be created by adjusting the parameter settings of the seed task selection- and insertion methods of the algorithm of Company X in OHD. To illustrate the complexity of the problem consider the following example: if we allow each sorter's scale to have only 10 values, each sorting criterion can be varied in 20 different ways (since it can be sorted in both increasing and decreasing directions), then with 24 different sorting criteria, we have a total of 480 unique sorting criterion variants (assuming no sorter is used twice). Consequently, 480! unique seed task selection methods could be created, resulting in a staggering 1.9×10^{1080} possible methods.

Note that even more insertion methods could be created since it concludes more parameter settings. On top of that, the combination of seed task selection- and insertion methods that lead to a unique construction method will be even larger. Furthermore, note that we restricted the scale of the sorters to only 10 values whereas all positive real numbers are in fact allowed. This proves the extremely large solution space of the problem and its high level of complexity.

4.2 Modification of the problem of Company X

In Chapter 1, we mentioned that Company X makes use of different types of vehicles: priority 1, priority 2, priority 3 and normal vans. In this research, the priority 1 routes are removed from the datasets and the priority 2 routes are replaced with normal routes. This has been done since we can not judge for Company X whether a solution is better than another if we include the priority 1 and priority 2 routes. These routes should all be used and assigned to the relatively easier routes, however, we cannot decide for Company X if they would be satisfied with a solution in which a priority 2 route is replaced by a normal route to improve the costs substantially.

Furthermore, the term 'relatively easier routes' is subjective too thus we cannot decide for Company X what easier routes are and if they are satisfied with the provided solution.

Moreover, by removing the priority 1 and priority 2 routes the problem will look more like VRPs from other customers since those types of routes are not used by other customers. In this way, our research findings are more likely to be applicable to other customers using the OHD software.

For these reasons, we have decided to remove the priority 1 routes and replace the priority 2 routes with normal routes such that we can decide based on the number of planned tasks and total costs which solution is the best. Besides, it barely changes the VRPX as we maintain a heterogeneous vehicle fleet for almost all of the main depots. This is the only modification made to the problem of Company X that is explained in Section 1.2.

4.3 Mathematical formulation

In order to provide ORTEC valuable insights into the utilization of parallel computing within OHD, we will develop a parallel algorithm. As mentioned before, the algorithms included in the parallel algorithm will be created by adjusting the parameter settings of the seed task selection and insertion methods which are introduced in Section 4.1. Both the number of algorithms included and the number of starting solutions that will proceed to the local search and R&R phase are variable. This problem can be framed as an algorithm configuration problem, akin to the one outlined in Section 2.5.1:

Given

- the algorithm of Company X with parameters $s_1, \ldots, s_k$ that affect its behavior,
- a space A of algorithms, where each algorithm $a \in A$ is a unique variant of the algorithm of Company X where a_0 denotes the algorithm of Company X,
- a set of problem instances I,
- a space P of configurations (i.e. parallel algorithms), where each $p \in P$ consists of algorithms $a_1, \ldots, a_n$, where a subset s of starting solutions S will proceed to the local search and RR phase,
- a performance metric m that measures the performance of p on instance set I,

find a configuration $p* \in P$ including a set of algorithms $\{a_1, \ldots, a_n\}^* \in A$ where a subset $s* \in S$ starting solutions will proceed to the local search and R&R that results in optimal performance on I according to performance metric m. The design of the new parallel algorithm is visualized in Figure 4.1

The performance is evaluated firstly on the number of planned tasks and secondly on the total costs which includes the computational costs as well as the planning costs. Furthermore, while solving the above-formulated problem, research about the importance of the initial solution and the relationship between input data and parameter settings should be conducted.

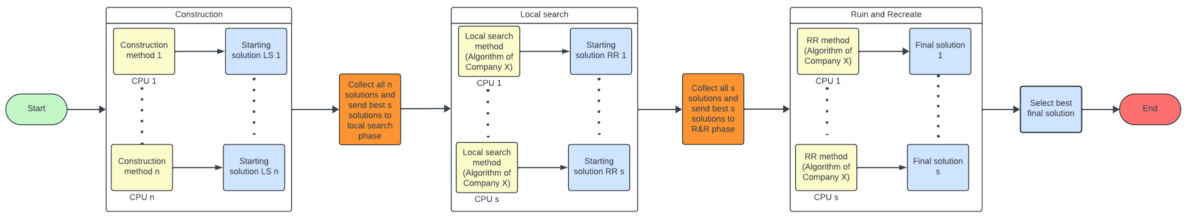

Start
Construction
Construction method 1
Starting solution LS 1
CPU 1
Construction method n
Starting solution LS n
CPU n
Collect all n solutions and send best s solutions to local search phase
Local search
Local search method (Algorithm of Company X)
Starting solution RR 1
CPU 1
Local search method (Algorithm of Company X)
Starting solution RR s
CPU s
Collect all s solutions and send best s solutions to R&R phase
Ruin and Recreate
RR method (Algorithm of Company X)
Final solution 1
CPU 1
RR method (Algorithm of Company X)
Final solution s
CPU s
Select best final solution
End

Figure 4.1: A visualization of the design of the new parallel algorithm that uses multiple construction methods and starting solutions.

4.3.1 Degree of complexity

Suppose we continue the example given in Subsection 4.1.3, where we discovered that we can create 1.9×10^{1800} unique seed task selection methods assuming that the scales of the sorters are limited to only 10 values. Additionally, the number of insertion methods is even larger since it involves more parameters. However, for this new example, we will assume that the number of different seed task selection and insertion methods are equal. This implies that we are able to create $(1.9 \times 10^{1080})^2$ thus 3.5×10^{2160} unique algorithms. Since the number of algorithms n included in the parallel algorithm p is variable, we can create $2^{3.5 \times 10^{2160}}$ different parallel algorithms. This is already an extremely large number however since we also restricted the scales to 10 values and assumed that the number of insertion methods is equal to the number of seed task selection methods, we can say that we can create, without restrictions, a myriad number of parallel algorithms p. Besides, we also ignored the fact that we have to select a subset of starting solutions to proceed to the local search and R&R phase. This underwrites again the high degree of complexity of the problem we are facing.

4.4 Solution Design

The primary objective of this research is to provide ORTEC with relevant insights regarding the impact of parallel computing within OHD. We are going to address this goal by constructing a new parallel algorithm that incorporates multiple construction methods and evaluating its performance against the algorithm of Company X. To do so, we have to address the problem, formulated in the previous section. In this section, we will present our solution approach for solving this problem and gather relevant information for ORTEC about the use of parallel computing with the use of multiple construction methods during the process. We have seen in Section 4.3, that a myriad number of unique parallel algorithms could be created within OHD and in Section 2.5 that automated algorithm configuration methods are not suitable for our problem. For this reason, we have to devise an efficient way of solving this problem, which involves breaking down the design process into multiple steps. The solution design is visualised in Figure 4.2 and explained in the subsequent paragraphs.

Firstly, we are going to create multiple algorithms that include different construction methods in which our goal is to find good construction methods for various types of cases of Company X. This means that it is crucial to not solely evaluate configurations based on their average performance but also on their performance for specific types of cases. Since if an algorithm only performs well for specific cases and very poorly for others, it remains relevant for parallel computing. This search for good construction methods will be combined with a data analysis, enabling us to provide ORTEC with information about the relationship between input data and construction strategies. Since the intrinsic complexity of the cases of Company X for any configuration of the algorithm of Company X may differ substantially, comparisons between different configurations are always based on the same dataset and seed number (Hoos, 2012).

Secondly, we will investigate whether using multiple starting solutions is beneficial, and if so, determine the optimal number of construction methods for generating these solutions and how many should proceed to the local search and R&R phase.

Finally, after we have determined the number of construction methods and solutions that will proceed to the local search, we have to select the construction methods that will be used in the new parallel algorithm that optimizes the performance and robustness. In the following subsections, the solution approach for the three steps will be explained in more detail.

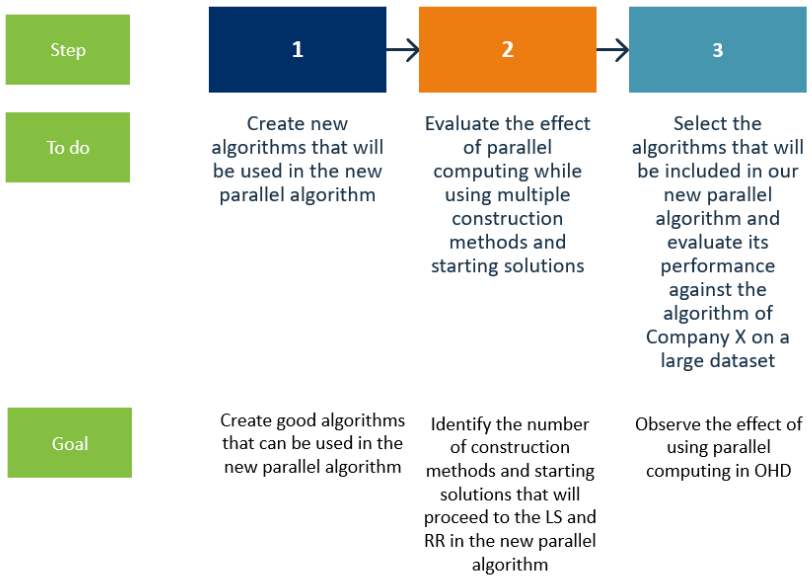

Step
1
2
3
To do
Create new algorithms that will be used in the new parallel algorithm
Evaluate the effect of parallel computing while using multiple construction methods and starting solutions
Select the algorithms that will be included in our new parallel algorithm and evaluate its performance against the algorithm of Company X on a large dataset
Goal
Create good algorithms that can be used in the new parallel algorithm
Identify the number of construction methods and starting solutions that will proceed to the LS and RR in the new parallel algorithm
Observe the effect of using parallel computing in OHD

Figure 4.2: A visualization of the solution design used to address our problem

4.4.1 Step 1: Identifying good construction methods

In the first step, it is important that we create good construction methods for various types of cases of Company X such that we are able to select a robust set of construction methods that will be used in the parallel algorithm. This will be done using the framework of Gunawan et al. (2011) which consists of three phases: screening, exploration and exploitation and is visualised in Figure 4.3.

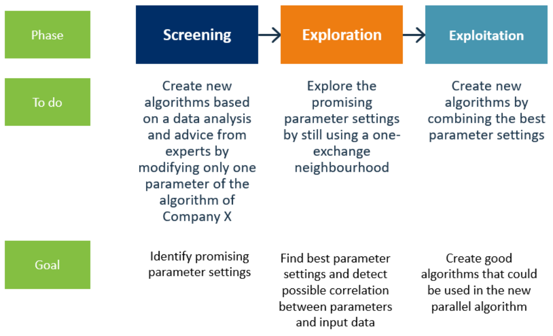

Phase
Screening
Exploration
Exploitation
To do
Create new algorithms based on a data analysis and advice from experts by modifying only one parameter of the algorithm of Company X
Explore the promising parameter settings by still using a one-exchange neighbourhood
Create new algorithms by combining the best parameter settings
Goal
Identify promising parameter settings
Find best parameter settings and detect possible correlation between parameters and input data
Create good algorithms that could be used in the new parallel algorithm

Figure 4.3: A visualization of the first step of the solution design used to address our problem

Screening

The screening phase aims to identify parameters with insignificant impact on algorithm performance, termed as unimportant parameters. These parameters are then set as constants to reduce the solution space. To achieve this, we consulted the creators of the algorithm of Company X and experts from ORTEC to detect parameter settings worthy of variation, leveraging their extensive knowledge of OHD and the algorithm of Company X. They recommended not varying much with the sorting criteria since the first sorting criterion for the seed task was specifically constructed for the cases of Company X. Furthermore, most insertion methods include as first scale the distance and the second scale is chosen based on customer-specific information. In this way, our solution space is reduced significantly.

To screen the solution space of the sorting criteria, some new sorting criteria are introduced based on the data analysis which is presented in the next chapter. These are tested in the screening experiment together with various values for the scales of the first sorter in both seed task selection and insertion method. Notably, changes to the algorithm of Company X are made via a one-exchange neighborhood as has been done in Rasku et al. (2019), allowing us to observe the effect of the single change to the algorithm of Company X.

For the experiments, a subset of the dataset was selected based on their characteristics to facilitate the detection of potential relationships between construction strategies and input data. The most promising parameter settings identified will be further explored on the complete dataset in the subsequent phase: exploration.

Exploration

In the second phase, we are going to explore the promising ranges of scales found in the screening phase. Additionally, we will experiment with other parameters suggested by the creators of the algorithm of Company X that will be introduced in Section 6.2. These configurations will be tested on the full dataset in the exploration experiment. The goal of this phase is to detect the best parameter settings. Note that in our case, best parameter settings may comprise multiple settings per parameter.

Exploitation

In the concluding phase, exploitation, we are no longer using the one-exchange neighbourhood structure and instead leverage the solution space insights obtained from previous experiments. We will implement multiple changes to the algorithm of Company X simultaneously, combining the best settings identified in earlier phases. Priority will be given to parameter settings that individually yield the most significant improvement in solution quality, while those with the least impact will receive less emphasis. The objective of the exploitation experiment is to develop highly effective construction methods to facilitate the creation of an efficient parallel algorithm.

4.4.2 Step 2: Effect of parallel computing

In this step, we will assess the effect of parallel computing. In Step 1, we will create a lot of algorithms with different construction methods and from these experiments on the dataset we will store the KPIs of all construction solutions and final solutions. This approach enables us to simulate the outcomes if all solutions were executed concurrently. Additionally, we can analyze the effects of employing multiple starting solutions that proceed to the local search and R&R phase. As a result, we can determine the optimal number of different construction methods to include in our new parallel algorithm, as well as the number of resulting starting solutions that should proceed to the local search and R&R phase.

4.4.3 Step 3: Design of new parallel algorithm

In the final step of our solution design, we will design the new parallel algorithm based on the outcomes of the previous phase, where we determined the number of different construction methods to include and the number of starting solutions that proceed to the local search and R&R phase. The new parallel algorithm should encompass algorithms that consistently perform well on average, providing a strong lower bound for every case of Company X. Additionally, it should incorporate construction methods that excel for specific types of cases, allowing us to capitalize on the benefits of parallel computing. In this way, we aim to create a robust and high-performing parallel algorithm that enhances the solution quality of the algorithm of Company X. The performance of this new parallel algorithm compared to the algorithm of Company X is evaluated on a new testing dataset that contains a week of data of all main depots of Company X which is executed in the final experiment.

4.5 Conclusions

In this chapter, we have provided a detailed overview of our solution approach, which will be used to answer the main research question stated in Section 1.3. We presented the various options of different seed task selection- and insertion methods within OHD. Subsequently, the modification of the problem of Company X is explained and the mathematical formulation of the problem at hand is given. Finally, we outlined the solution approach employed in our research to address this problem which consist of three steps. Firstly, in Step 1 we will create new algorithms that perform well such that they could be used in the new parallel algorithm. Secondly, the effect of parallel computing while using multiple construction methods and starting solution will be evaluated such that the size of the parallel algorithm could be determined in Step 2. Finally, in Step 3, we will select the specific algorithms that will be included in the new parallel algorithm. Afterwards, this new parallel algorithm is evaluated against the algorithm of Company X on a large dataset to observe the effect of using parallel computing within OHD.

Chapter 5

Dataset Analysis

In this chapter, we provide an analysis of the input data of cases of Company X. The goal of this data analysis is to detect potentially difficult tasks which could be selected as seed tasks. Furthermore, it is used to detect promising ranges for the scales of the first sorter in the seed task selection and insertion method. Besides, it is used to classify the cases such that we can find potential relationships between construction strategies and input data. In Section 5.1, we discuss how we selected and obtained the datasets for the experiments and the main characteristics of the dataset are visualized. In the following sections, we dive deeper into the characteristics of the input data of the customers regarding their order quantities (5.2), distance to depots (5.3), distance to other customers (5.4) and time windows (5.5). Finally, in Section 5.6 we explain how the cases are classified based on their input data among the above-mentioned characteristics.

5.1 Dataset experiments

For the screening, exploration, and exploitation experiments, we have curated a diverse dataset of Company X's cases to reflect a typical week of operational activities. Seven main depots out of a total of twelve were selected, chosen to encompass a range of scenarios. These depots vary in size, with some catering to only a few hundred customers while others serve several thousand. Additionally, the number of subdepots, the vehicle fleet and the usage of priority 3 vehicles vary among these selected depots. Our dataset comprises morning and evening cases for three different days of the week: Monday, Wednesday, and Saturday, totalling 42 cases. These days were specifically chosen for their contrasting characteristics in terms of order volumes and operational complexity. Mondays typically witness large orders as companies stock up for the week ahead, while Wednesdays tend to be less demanding. Saturdays, on the other hand, usually present a mix of both high and low-order volumes.

Table 5.1 provides an overview of the average number of tasks, number of vehicles, vehicle types, priority 3 vehicles, and subdepots per main depot. Furthermore, the corresponding minimum, average and maximum values across the entire dataset are presented. The significant difference between the minimum and maximum values indicates the diversity of the dataset. Notably, the cases of main depots 3 and 4 incorporate significantly fewer customers and vehicles compared to others. These two main depots serve customers in Country B whereas the alternative main depots serve customers in Country A.

Table 5.1: High-level characteristics presented per main depot, encompassing the average number of customers, vehicles, types of vehicles, priority 3 vehicles and subdepots. Additionally, the minimum, average, and maximum values for these characteristics across the dataset cases are shown. This dataset consists of six cases per main depot, including morning and evening cases for three days of the week. Note that ranges are used instead of absolute values for confidential issues.

Main depot (Country)	**Customers**	**Vehicles**	**Types of vehicles**	**Priority 3 vehicles**	**Subdepots**
1 (A)	3000-3500	150-200	2	0-10	4
2 (A)	2500-3000	150-200	4	30-40	3
3 (B)	0-500	0-50	2	0-10	1
4 (B)	500-1000	0-50	2	0-10	2
5 (A)	2500-3000	150-200	3	10-20	1
6 (A)	3000-3500	150-200	1	0-10	4
7 (A)	3000-3500	150-200	3	0-10	3
Min	0-500	13	1	0-10	1
Average	2000-2500	128.9	2.4	0-10	2.6
Max	4000-4500	227	4	30-40	4

5.2 Order quantity

The first characteristic of the input data that we are going to analyze is the order quantity. As previously noted, vehicle capacities encompass both volume and weight, which aligns with the volume and weight attributes of customer orders. As mentioned before in Subsection 2.2.1, ideally we want a seed task to be a difficult task. To observe the difficult tasks concerning their order quantities, we are going to look at the ratio between the largest order relative to the capacity of the largest vehicle. This problem feature is introduced by Steinhaus (2015).

We have examined the utilization concerning volume and weight relative to the largest vehicle capacity for the largest order as well as for the averages of the largest 1%, 4%, and 10% orders across all cases in the dataset. This has been done since on average approximately 25 orders are included in a route which means that in a planning the routes needed to serve all customers are about 4 percent of the number of tasks. As a result, we need to select approximately 4 percent of the total tasks as seed tasks. Consequently, we opted to include the largest 4 percent of orders in the analysis. Additionally, we included the top 1 and 10 percent to offer a more comprehensive perspective on the peak values of the orders. A pseudocode of the calculations for the top 1, 4 and 10 percent is provided in Figure 1. The same code is also used for the calculations of the characteristics in the following sections. Table 5.2 presents the corresponding minimum, average, and maximum values per category in the dataset. For instance, the largest order by weight in the dataset occupies 93% of the largest truck's capacity. Note that the capacity of the largest truck is equal for every case. Furthermore, on average the largest order in a case by volume utilizes 26% of the capacity of the largest truck. Conversely, the smallest largest order of a case regarding weight in the dataset covers only 11% of the largest vehicle.

In Table 5.2, it's evident that the ratios between the maximum order quantity and the capacity of the largest vehicle in terms of weight are higher for peak values compared to volume-based order quantities. This suggests that order weight is more likely to cause violations during route insertion than order volume. Furthermore, the maximum utilization rate of the largest order by weight in the dataset (0.93) indicates the presence of very large orders. However, on average, the largest orders across all cases only cover 33% of truck capacities. Furthermore, the maximum ratio of the largest 1, 4 and 10% orders is also relatively low compared to the maximum utilization. This indicates that orders with extremely high quantities and thus difficult orders are rare. Additionally, the case in the dataset that has the lowest largest order in terms of weight only utilizes 11% of the capacity of the largest vehicle in that case. This means that there is a lot of variety across all cases in the dataset concerning the order quantities.

Table 5.2: The minimum, average and maximum value of the utilization rates of the largest order, the largest 1, 4, 10% orders of cases across the dataset are visualised in terms of volume and weight.

	Max		Top 1%		Top 4%		Top 10%	
	Volume	Weight	Volume	Weight	Volume	Weight	Volume	Weight
min	0.11	0.11	0.09	0.09	0.07	0.07	0.06	0.06
average	0.26	0.33	0.13	0.16	0.10	0.11	0.08	0.08
max	0.59	0.93	0.26	0.29	0.18	0.20	0.14	0.15

Algorithm 1: Pseudo code calculations top x% of KPIs

```
Input: $KPI$ and $Percentage$
Output: $minimum$, $average$ and $maximum$
1  Function Calc($KPI$, $Percentage$)
2      $V = \emptyset$
3      if $KPI$ = 'Distance to Depots' or 'Order quantity' then
4          $direction$ = 'largest'
5      else
6          $direction$ = 'smallest'
7      for case in dataset do
8          $value$ = average of $direction$ $Percentage$ values of $KPI$ in case
9          $V \leftarrow value$
10     minimum = min($V$)
11     average = average($V$)
12     maximum = max($V$)
13     return minimum, average, maximum
```

5.3 Distance to Depots

In this section, we will delve into the distance and driving time of customers to the nearest and second nearest depots. This analysis helps us gain insights into the spatial distribution of customers in relation to depot locations and scale values of the sorters of the algorithm of Company X. This will help us identify difficult customers regarding their location. While there is typically a strong correlation between the shortest distance and shortest driving time to the nearest depot, differences can arise due to factors such as route characteristics. For instance, a customer may be relatively far from a depot in terms of distance, but if the route primarily traverses highways, the driving time could be considerably lower compared to a customer located closer but requiring travel through congested city centers. To account for these nuances, we examine distance and driving time separately.

The 'AvgShortDistToDepot' column in Table 5.3 presents the minimum, average, and maximum distances in kilometers for the average distance of customers to the closest depot across the dataset cases. For example, customers in the case with the shortest distance to the nearest depot are, on average, 12.47 kilometers away from the nearest depot. Additionally, other columns display the minimum, average, and maximum average distances of the x% furthest customers from the nearest depot across dataset cases. As an illustration, there is a case where the top 1% of the furthest customers are, on average, 83.86 kilometers away from the nearest depot. It is notable that the maximum value is nearly three times larger than the minimum value of the average shortest distance to a depot over all columns. This indicates substantial variation across dataset cases regarding the distance of each customer to the nearest depot. This discrepancy is reasonable as some depots primarily serve rural areas, while others serve customers in urban centers, resulting in significant differences in the shortest distance to the nearest depot for each customer.

Table 5.3: 'AvgDistToNearestDepot' denotes the average distance to the nearest depot of every customer in a case whereas Top 1, 4, and 10% denote the average of the x% customers with the largest distance to the nearest depot in case. The minimum, average and maximum values of the cases in the dataset are visualised. The distances presented in the tables are expressed in kilometers.

	AvgDistToNearestDepot	Top1%	Top4%	Top10%
min	12.47	28.60	27.01	22.92
average	20.17	50.09	44.27	39.84
max	35.18	83.86	71.84	68.36

The same trend is observed in the difference in distance between the second nearest depot and the nearest depot, as depicted in Table 5.4 where the maximum values are significantly larger than the minimum values. The column 'AvgDiffDepot2To1' denotes the average difference in distance between the nearest and second nearest depot for all customers within a case, with only the minimum, average, and maximum values of the dataset presented in the table. Similarly, the other columns denote the average of the x% largest differences in distances between the nearest and second-nearest depot within a case, again with only the minimum, average, and maximum values of the dataset displayed in the table. Furthermore, we observe relatively high maximum values in both tables, with a marginal discrepancy between the largest 1 and 10 percent distances. This suggests that there aren't just a few outliers in the dataset, but rather clusters of customers located far away from the depots. Additionally, the stark contrast between the minimum and maximum values of the difference between the second nearest and nearest depot highlights the significant variance in the importance of depot assignments for customers within each case.

Table 5.4: 'AvgDiffDepot2To1' denotes the average of the difference in distance between the nearest and second nearest depot for every customer in a case whereas Top 1, 4, 10% denote the largest x% differences in distance between the nearest and second nearest depot in a case. The minimum, average and maximum values of the cases in the dataset are shown. The distances presented in the tables are expressed in kilometers.

	AvgDiffDepot2To1	Top1%	Top4%	Top10%
min	7.29	26.87	25.15	21.60
average	25.75	49.69	47.63	44.98
max	46.29	71.13	70.10	68.40

Table 5.5 shows the same characteristics as in Table 5.3 however in this case the driving time without congestion is used instead of the distance. This immediate comparison reveals that the factor between the maximum and minimum values is approximately 2, as opposed to 3 in the case of distance. This suggests that there is less variability across cases when considering driving time compared to distance. However, it's worth noting that customers located furthest away from the closest depot have driving times extending up to over an hour which is relatively long. In Table 5.6, differences between the nearest and second-nearest depot are explored in terms of driving times. Here, we observe similar proportions as in Table 5.4, which focuses on the difference in distance between depots.

Table 5.5: 'AvgTimeToNearestDepot' denotes the average driving time to the nearest depot of every customer in a case whereas Top 1, 4, and 10% denote the average of the x% customers with the largest driving time to the nearest depot in case. The minimum, average and maximum values of the cases in the dataset are visualised. The driving times presented in the tables are expressed in minutes.

	AvgTimeToNearestDepot	Top1%	Top4%	Top10%
min	15.01	31.36	29.36	25.48
average	21.60	45.07	40.22	36.61
max	32.31	66.18	58.81	55.27

Table 5.6: 'AvgTimeDepot2To1' denotes the average of the difference in driving time between the nearest and second nearest depot for every customer in a case whereas Top 1, 4, 10% denote the largest x% differences in driving time between the nearest and second nearest depot in a case. The minimum, average and maximum values of the cases in the dataset are shown. The driving times presented in the tables are expressed in minutes.

	AvgTimeDepot2To1	Top1%	Top4%	Top10%
min	6.61	17.49	16.95	16.18
average	17.20	35.71	33.72	31.69
max	29.64	57.48	51.93	47.22

5.4 Distance to other customers

In this section, we will examine the statistics regarding the distances between customers. This will help us understand the density of the cases in the dataset and give us insights into the scale of the first sorter of the insertion method. We have a look at the average distances among the k-nearest neighbours of customers in the cases of the dataset. This subset includes the closest customer (Rasku et al., 2016), as well as the closest 1%, 4%, and 10% of customers within each case. Table 5.7 presents the minimum, average, and maximum values of the cases in the dataset. Once again, substantial variations between cases in the dataset are apparent, as evidenced by the large differences between the minimum and maximum values. These disparities underscore the diverse nature of the dataset, with cases exhibiting significantly different customer densities. Furthermore, considering the driving time between customers, as shown in Table 5.8, reveals a lower factor compared to distance estimates. Nevertheless, considerable differences in density persist across cases in the dataset.

Table 5.7: ’AvgClosestDist’ denotes the average distance for all customers in a case to its closest neighbour where the Top 1, 4, 10% denote the average distance of all customers to the x% closest neighbours. The minimum, average and maximum values of cases across the dataset are presented. The distances presented in the tables are expressed in kilometers.

	AvgClosestDist	Top1%	Top4%	Top10%
min	0.24	0.69	1.71	2.55
average	0.61	1.74	4.39	6.84
max	1.76	4.34	11.30	18.59

Table 5.8: ’AvgClosestTime’ denotes the average driving time for all customers in a case to its closest neighbour where the Top 1, 4, 10% denote the average driving time of all customers to the x% closest neighbours. The minimum, average and maximum values of cases across the dataset are presented. The driving times presented in the tables are expressed in minutes.

	AvgClosestTime	Top1%	Top4%	Top10%
min	0.67	1.81	4.01	5.58
average	1.43	3.69	7.87	10.81
max	3.45	7.74	15.80	21.95

5.5 Time Windows

In this section, we will have a closer look at the time windows that are set by the customers in the input data. Table 5.9 provides the minimum, average and maximum values of the shortest time windows, average time windows and the shortest 1%, 4% and 10% time windows across all cases in the dataset. Since the shorter the time window, the more difficult the task, we are interested in the length of the shortest time windows. It is important to note that the service time is subtracted from the time window length in the values presented in the table. This has been done since the delivery should start and end within the designated time window specified by the customers. Consequently, this reduces the space to schedule the task in a route. The average service time of all customers in the dataset is about 8 minutes, with variations ranging from 5 to 45 minutes.

We can conclude from Table 5.9 that the time windows of the customers of Company X are relatively short since on average they have a length of about 3 hours. Furthermore, it can be seen that some cases include customers with a time window shorter than an hour, indicating tight scheduling constraints. Moreover, the differences between the shortest 1% and 10% time windows are not that large, suggesting a significant proportion of customers with relatively small time windows across the cases. Finally, it can be observed that the differences between the minimum and maximum are relatively large, implying substantial variability among cases in the dataset in terms of the time window lengths.

Table 5.9: ’ShortestTW’ implies the shortest time window of a case whereas ’AvgTWLength’ denotes the average length of all time windows in a case. The Top 1, 4, 10% stands for the average length of the x% shortest time windows of a case. The minimum, average and maximum values of the cases in the dataset for each characteristic are shown in the table. The time window lengths are expressed in minutes.

	ShortestTW	AvgTWLength	Top1%	Top4%	Top10%
min	5.57	123.05	38.00	43.42	47.81
average	68.02	171.42	75.45	81.86	91.67
max	107.12	235.51	108.21	108.91	109.51

Moreover, it can be concluded from Table 5.10 that the maximum percentage in the dataset of tasks finishing within 5 hours after the start of the shift is 30.91. Besides, some cases have no tasks with time windows ending within 5 hours. Additionally, the average start of a time window after the start of a shift is approximately 4 hours. This suggests that tasks typically commence relatively late compared to the beginning of the shift, indicating that a relatively small number of tasks require immediate completion at the onset of the shift.

Table 5.10: 'TasksFinishin5hours' denotes the percentage of customers that need to be served within 5 hours after the shift start of a case and 'AvgTWStartAfterShiftStart' stands for the average start of the time window of customers after the shift has started of a case. The minimum, average and maximum values of the cases in the dataset are visualised.

	TasksFinishin5hours(%)	AvgTWStartAfterShiftStart(h)
min	0.00	2.50
average	13.63	3.97
max	30.91	5.33

5.6 Classification of the cases

To facilitate the detection of relationships between the input data and the newly designed construction methods in the subsequent chapters, the datasets will be categorized based on the above-discussed categories. Each category will be assigned a label of 'Small', 'Average', or 'Large'. For the categories pertaining to time windows, the labels 'Short', 'Average', and 'Large' will be used. This classification will be performed by dividing the data into quartiles. The first quartile will correspond to the 'Small' or 'Short' label, the fourth quartile will correspond to the 'Large' or 'High' label, and the quartiles in between will denote the 'Average' label. Subsequently, cases may be reassigned to a different label if they are outliers in their current classification and are better suited to another label. For instance, if the largest value in the 'Average' label is twice as high as the second-largest value in the 'Average' label and is nearly as large as the smallest value in the 'High' label, this case will be shifted from the 'Average' label to the 'High' label.

5.7 Conclusions

In this chapter, we have conducted a thorough examination of the cases from Company X included in our dataset, covering every aspect of the input data. We observed significant variability between cases concerning order quantities, task density, and time windows within the dataset. The substantial diversity observed across these aspects implies that the use of multiple starting solutions could be highly advantageous for the algorithm of Company X. By incorporating diverse starting solutions, the algorithm can better adapt to the varying characteristics of different cases within the dataset, thereby enhancing its overall effectiveness.

Chapter 6

Experiments

In this chapter, all experiments that are executed in order to design the new parallel algorithm of Company X are presented where we start with the experiments regarding the construction phase. This chapter is structured by the steps of our research design which is presented in Figure 4.2. The first three sections correspond to the three phases of Step 1 of the research design which is visualised in Figure 4.3. As a result, the experiments conducted with respect to the screening phase in which we create new algorithms based on the data analysis conducted in Chapter 5 are presented in Section 6.1. Subsequently, the results of the experiments regarding the exploration phase in which we will explore the most promising parameter settings found in the screening phase are shown in Section 6.2. Finally, in Section 6.3, the results of the experiments of the last phase of Step 1 of the research design are presented. In this final phase, the best-found parameter settings in the previous phases will be combined to create good algorithms that could be used in the new parallel algorithm. Note that every newly created algorithm will be compared against the algorithm of Company X which is explained in Section 3.2. Section 6.4 presents the outcomes of the experiments related to Step 2 of the research design which include the evaluation of the use of multiple construction methods and starting solutions. Finally, in Section 6.5, the new parallel algorithm using multiple construction methods is created and evaluated against the algorithm of Company X to obtain the effect of the use of multiple construction methods in parallel within OHD which corresponds to Step 3 of our research design.

Afterwards, we apply the research design to the R&R phase of the algorithm of Company X. At ORTEC, it is known that the current R&R method scarcely improves the solution after local search. Due to time constraints and recent research at ORTEC, modifications have been made to both the research design and the R&R method of the algorithm of Company X. These modifications are detailed in Section 6.6. Following this, Section 6.7 presents the experimental results and the decisions made in developing a new parallel version of the R&R phase that will be used in our new parallel algorithm. Finally, the complete new parallel algorithm including multiple construction and R&R methods is evaluated against the algorithm of Company X to obtain the effect of parallel computing within OHD.

6.1 Screening experiment

In this section, we examine the experimental outcomes of the screening experiment, which entail the algorithms formulated based on data analysis. The primary aim of this experiment is to identify promising parameter settings that will be further tested on a larger scale in the exploration experiment. We commence with Subsection 6.1.1, where we explore experiments centred around construction methods focused on order quantity in terms of weight. Following that, in Subsection 6.1.2, we undertake the creation and testing of multiple construction methods based on the distance between customers and the depot(s). Additionally, in Subsection 6.1.3, we develop and evaluate various construction methods based on the distance between customers. Finally, Subsection 6.1.4 encompasses the creation and testing of construction methods related to the time windows of customers. Note that we will not evaluate the performances of new algorithms against the data characteristics with respect to driving times since the difference in classification is that small with respect to the distance. Each subsection follows a similar structure: first, the new algorithms are explained, and then the experimental results of these new algorithms are discussed.

6.1.1 Order quantity

Algorithms explanation In Section 5.2, we observed that some cases include very large orders which underwrite the potential value of selecting large orders as seed tasks. Consequently, we designed a seed task strategy primarily focusing on the weight of customer orders. This new seed task strategies sort customers firstly based on their order weight, followed by utilizing the same sorting criteria employed by the algorithm of Company X which are explained in detail in Section 4.1.1. We established scales of 0, 10, and 30 for sorting order quantity in terms of weight. A scale of 0 ensures selection based solely on order quantity, while a scale of 10 emphasizes order quantity but also considers the other sorters. The scale of 30, chosen as the average value of all order quantities in the dataset, distinguishes tasks only if the gap is notably large. These three new construction methods are then applied to a subset of dataset cases selected based on their order quantity characteristics. This approach enables us to investigate potential correlations between seed task selection strategies based on order quantity and the order quantity characteristics of each case. We have chosen not to design an insertion strategy based on order quantity since this will result in very disorganized routes with a lot of crossovers since time windows and travel times between nodes are ignored.

Experimental results Table 6.1 summarizes the results of the experiment by presenting the average performance of the three new algorithms compared to the algorithm of Company X across the 12 cases for the most significant KPIs per algorithm. Among these 12 cases, 6 cases feature orders with a relatively high utilization rate of the capacity of the largest vehicles. For both low and average utilization rates of the largest orders in the case, 3 cases are selected. As previously stated, the number of planned tasks is the most important KPI, and from the table, it is evident that all algorithms perform equally well on this criterion. However, when comparing costs, it becomes apparent that the algorithm of Company X clearly outperforms the newly constructed algorithms that select the seed task based on order quantity. Additionally, the solutions generated by the new algorithms utilize significantly more routes and trips to serve all customers, logically resulting in higher costs. It's worth noting that the number of routes corresponds to the number of vehicles used in the solution, and each vehicle can make multiple trips. Finally, we can see that the calculation time is also a lot higher which could imply that the chosen seed tasks cause a bad starting solution. As a consequence, the local search and R&R require more time to fix these routes. All in all, we can conclude that the new algorithms score on average a lot worse than the algorithm of Company X.

Table 6.1: Results of the experiments for the new algorithms that use the order weight as the first sorter for their seed task selection method, followed by the same sorting criteria used by the algorithm of Company X. The first column denotes the scale of the order weight sorter whereas the other columns display the percentile difference with respect to the algorithm of Company X for the number of planned tasks, costs, number of routes and trips and finally the calculation time (Calc time). (1)

Scale	Planned tasks (%)	Costs (%)	Routes (%)	Trips (%)	Calc time (%)
0	0.00	2.01	5.57	2.06	23.31
10	0.00	1.82	5.19	2.14	36.45
30	0.00	1.77	4.92	1.61	30.59

However, one advantage of parallel computing is the ability to select algorithms that excel in certain cases while performing worse in others. In the parallel algorithm, an algorithm's solution is only utilized if it proves to be exceptionally good; otherwise, a better solution from another algorithm is chosen. Consequently, while the average performance of an algorithm may be lower than that of the algorithm of Company X, it can still be valuable for the new parallel algorithm. Table 6.2 illustrates the instances where an algorithm outperforms the Company X algorithm, categorized by label type, with the number in brackets indicating the number of cases of each type used in the experiment. Notably, the new algorithms demonstrate superior performance in cases where the largest orders have a relatively low utilization rate of the largest vehicle. Interestingly, these are all cases from Country B that include a relatively low number of tasks as shown in Table 5.1. Despite their overall poorer average performance, these new algorithms outperform the algorithm of Company X in the cases of Country B. However, it is noteworthy that the new algorithms do not exhibit better performance in cases with larger orders, which is contrary to expectations. Although the dataset analysis in Chapter 5 revealed the presence of very large orders, the seed task strategies based on the largest orders do not seem to function well. As a result, we have decided to introduce two new algorithms with significantly larger scales, ensuring that only the very large orders are selected as seed tasks. If no large orders are remaining, the seed task selection method of the algorithm of Company X is applied. The new scales are 300 and 500.

Table 6.2: An overview of the number of times the solution of the new algorithms, which use the order weight as the first sorter for the seed task selection method, is better than the solution of the algorithm of Company X visualized per case type in terms of the utilization rate of the largest orders in the case. The first column denotes the scale of the order weight sorter and the number in brackets behind the classification indicates the total number of cases of the corresponding type used in the experiments. (1)

	Case classification of utilization rate of largest orders			
Scale	Low (3)	Average (3)	High (6)	Total (12)
0	3	0	0	3
10	2	0	0	2
30	2	0	0	2

Table 6.3 includes the results of the two new algorithms. These new algorithms perform significantly better across all KPIs compared to the initial three algorithms based on order quantity. However, the algorithm of Company X still performs the best on average, although the differences are much smaller than those in Table 6.1. In Table 6.4, we observe the number of instances where each algorithm equals or outperforms the algorithm of Company X, categorized by case type. The number in brackets behind each label type represents the number of cases used in this experiment per label type. Additionally, the number in brackets within the table denotes the number of solutions equal to those of the algorithm of Company X. In cases where there are no orders exceeding a weight of 300 or 500, the algorithms perform identically to the algorithm of Company X, resulting in solutions equal to those of the algorithm of Company X. From Table 6.4, we can see that the new algorithms perform relatively well in cases with larger orders. However, we also observe instances where the algorithm of Company X outperforms the new algorithms with larger scales for cases with relatively large orders. All in all, we see that the new algorithms are on average a bit worse than the algorithm of Company X but do outperform the algorithm of Company X for some cases that include larger orders. This suggests that very large orders could indeed serve as effective seed tasks, while the remaining seed tasks might be selected using a different strategy. Therefore, we need to determine the threshold for defining when a task is large enough to be considered a seed task. This will be explored in the exploration experiment detailed in the next section, where we will test scales corresponding to a utilization rate of the largest vehicle of 0.2, 0.3, 0.4, and 0.5. These scales will be set at 190, 285, 380, and 475, respectively.

Table 6.3: Results of the experiments for the new algorithms that use the order weight as the first sorter for their seed task selection method, followed by the same sorting criteria used by the algorithm of Company X. The first column denotes the scale of the order weight sorter whereas the other columns display the percentile difference with respect to the algorithm of Company X for the number of planned tasks, costs, number of routes and trips and finally the calculation time (Calc time).

Scale	Planned tasks (%)	Costs (%)	Routes (%)	Trips (%)	Calc time (%)
0	0.00	2.01	5.57	2.06	23.31
10	0.00	1.82	5.19	2.14	36.45
30	0.00	1.77	4.92	1.61	30.59
300	0.00	0.11	0.37	0.54	-4.70
500	0.00	0.05	0.19	0.98	-2.83

6.1.2 Distance to depots

Algorithms explanation In Section 5.3, it was observed that certain tasks exhibit a significant disparity between the distance to the nearest and second nearest depot. This suggests a preference for selecting such tasks as seed tasks for the nearest depot to optimize routing. Otherwise, scheduling these tasks in routes of the second nearest depot may not be optimal due to the considerable difference in distance between the two depots. Moreover, the range between the minimum and maximum values of the difference between the nearest and second nearest depot is relatively large. Hence, it becomes intriguing to investigate whether using different scales could yield better results and if for example, smaller scales work better for cases in which the difference in distance between the nearest and second nearest depot is relatively small. To explore this hypothesis, we will alter the scale of the first sorter of the seed task selection method of the algorithm of Company X, which is based on the difference in distance between

Table 6.4: An overview of the number of times the solution of the new algorithms, which use the order weight as the first sorter for the seed task selection method, is better than or equal to the solution of the algorithm of Company X visualized per case type in terms of the utilization rate of the largest orders in the case. The numbers in brackets in the table denote the number of times a solution is equal to the ones of the algorithm of Company X. The first column denotes the scale of the order weight sorter and the number in brackets behind the classification indicates the total number of cases of the corresponding type used in the experiments. (2)

	Case classification of utilization rate of largest orders			
Scale	Low (3)	Average (3)	High (6)	Total (12)
0	3	0	0	3
10	2	0	0	2
30	2	0	0	2
300	3 (3)	2 (1)	2	7 (4)
500	3 (3)	3 (1)	3	9 (4)

the nearest depot and the second nearest depot. Consequently, six algorithms with different scales of the first sorter of the seed task selection method of the algorithm of Company X were constructed, with three estimated based on distance and three based on driving time. These scales include 0, 5, and 20 kilometers, as well as 0, 10, and 20 minutes. Recall that the algorithm of Company X employs a scale of 10 kilometers. These 6 new algorithms are tested across 19 different cases. Among these, five cases were labelled 'Small', two were labelled 'Average', and 12 were labelled 'Large', based on the largest 4% difference in distance between the nearest depot and the second nearest depot of the customers within each case. Similar to the algorithms designed for handling customer order quantities, insertion strategies based on customer distance to the depot were not developed to avoid generating messy routes.

Additionally, there are cases where only two depots exist, with one operating as the main depot and the other as a subdepot. However, the main depot primarily supplies the subdepot, and vehicles from the main depot are not used unless the subdepot cannot fulfil all customer demands. Consequently, these cases essentially represent vehicle routing problems with only one depot. To address this specific problem, a seed task selection strategy was devised which is selecting the customer furthest away from the subdepot as a seed task. Two new algorithms were developed based on this strategy: one using distance as an estimator and the other using driving time. This seed task selection strategy is called `SeedFarAway`. These two new algorithms are tested on 6 cases where a single subdepot is serving most customers.

Experimental results The outcomes of the experiments regarding the 6 new algorithms based on the different scale values of the first sorter of the seed task selection method of the algorithm of Company X are presented in Table 6.5. This table lists the average performance of these new algorithms compared to the algorithm of Company X for the most important KPIs over the 19 cases. As previously mentioned, the most critical KPI for Company X is the number of planned tasks, and it is noteworthy that the algorithm with a scale of 5 kilometer performs the worst as it is unable to plan all tasks across the 19 different cases contrary to the other scales. However, the presence of unplanned tasks occurs only in one case, and with the use of parallel computing, it is not detrimental if an algorithm fails to plan all tasks for every case, as another algorithm can compensate. Consequently, if this algorithm yields favourable results for other cases, it could still be valuable for inclusion in the new parallel algorithm. It is important to note that the presence of unplanned tasks has a slightly positive influence on costs, routes, and trips. However, the percentual difference in planned tasks is so small that it will not affect the rankings in the table. Furthermore, every new algorithm outperforms the algorithm of Company X on average, indicating that varying the scale of the difference between the distance of the nearest and second-nearest depot could hold significant potential.

Table 6.6 displays the number of instances where an algorithm performs better than or equal to the algorithm of Company X per label. The number in brackets within the table indicates the occurrences where a solution is equal to the one of algorithm of Company X, while the number outside the brackets represents the instances where the solution is better, including those where it is equal. Additionally, the number in brackets behind the label type indicates the number of cases used per type. We can conclude from the table that all algorithms outperform the algorithm of Company X in more than half of the total number of cases. This reaffirms the potential impact of varying the scales for the difference in distance between the nearest and second-nearest depot. To explore this further, we have created

Table 6.5: Results of the experiments for the new algorithms that use different scales for the first sorter of the seed task selection method of the algorithm of Company X which is the difference between the nearest depot that is not the depot of the selected vehicle and the depot of the selected vehicle. The first column denotes the scale of this sorter where the scales based on driving time are presented with "min" and the scales based on the distance with "km". The other columns show the percentile difference with respect to the algorithm of Company X for the number of planned tasks, costs, number of routes and trips and finally the calculation time (Calc time).

Scale	Planned tasks (%)	Costs (%)	Routes (%)	Trips (%)	Calc time (%)
0 min	0.00	-0.37	-1.11	0.27	-13.34
10 min	0.00	-0.35	-0.83	-1.19	-20.71
20 min	0.00	-0.35	-0.83	-1.19	-18.46
5 km	-0.02	-0.26	-0.46	0.27	-0.46
0 km	0.00	-0.22	-0.83	0.73	-4.81
20 km	0.00	-0.06	-0.28	-0.73	-5.18

numerous algorithms with varying parameters for this scale. In the exploration experiment, we will use scales of 0, 2.5, 5, 7.5, 15, 20, 40 and 60 kilometers for distance estimation, and 0, 1, 5, 10, 30 and 60 minutes for driving time estimation.

Table 6.6: An overview of the number of times the solution of the new algorithms, which differ in scales of the first sorter of the seed task selection method of the algorithm of Company X, is better than or equal to the solution of the algorithm of Company X visualized per case type in terms of the difference in distance between the nearest and second nearest depot. The numbers in brackets in the table denote the number of times a solution is equal to the ones of the algorithm of Company X. The first column denotes the scale of the first sorter of the algorithm of Company X where the scales based on driving time are presented with "min" and the scales based on the distance with "km". The number in brackets behind the classification indicates the total number of cases of the corresponding type used in the experiments.

	Case classification of difference between nearest and second nearest depot			
Scale	Small (5)	Average (2)	Large (12)	Total (19)
0 min	3	1	6	10
10 min	4	2	7	13
20 min	4	2	7	13
0 km	3	1	7	11
20 km	3	0	7 (1)	10 (1)
5 km	4	1	8	13

Table 6.7 shows the results of the experiments regarding the algorithms that use the SeedFarAway seed task selection strategy based on distance and driving time. It can be concluded that both algorithms significantly outperform the algorithm of Company X on almost all KPIs. For this reason, both algorithms will also be added to the exploration experiment to see how they perform on a larger scale for other types of cases that include more depots.

Table 6.7: Results of the experiments for the algorithms that use the SeedFarAway seed task selection strategy based on distance (km) and driving time (min). The corresponding names are listed in the first column and the other columns show the percentile difference with respect to the algorithm of Company X for the number of planned tasks, costs, number of routes and trips and finally the calculation time (Calc time).

Algorithm	Planned tasks (%)	Costs (%)	Routes (%)	Trips (%)	Calc time (%)
SeedFarAway (km)	0.00	-1.04	-1.61	-1.61	3.95
SeedFarAway (min)	0.00	-1.80	-1.61	-1.61	-10.66

6.1.3 Distance to other tasks

Algorithms explanations In Section 5.4, we observed significant variation in task density across different cases. Given this variability, it would be interesting to explore the effects of using different scales for the first scale of the insertion strategy method used by the Company X algorithm, particularly in relation to customer density. Currently, a scale of 1 kilometer is utilized as the first sorter. We have created four new algorithms with scales of 0.2, 0.5, 3, and 5 kilometers. Additionally, four similar algorithms were created, but with driving time used as an estimator instead of distance. The scales for these algorithms are 1, 2.5, 5, and 7.5 minutes. These eight new algorithms are evaluated across 22 cases, each varying in customer density. The results will be presented in the following paragraph.

Experimental results Table 6.8 presents the performance of these eight new algorithms compared to the algorithm of Company X for the most important KPIs. The insertion strategies with lower scale values for both distance and driving time outperform, on average, those with larger scales. Additionally, their calculation times are significantly lower compared to the larger scales.

Table 6.8: Results of the experiments for the new algorithms that use different scales for the first sorter of the insertion method algorithm of Company X which is the difference compared to the seed task. The first column denotes the scale of this sorter where the scales based on driving time are presented with ″min″ and the scales based on the distance with ″km″. The other columns show the percentile difference with respect to the algorithm of Company X for the number of planned tasks, costs, number of routes and trips and finally the calculation time (Calc time).

Scale	Planned tasks (%)	Costs (%)	Routes (%)	Trips (%)	Calc time (%)
0.5 km	0.00	-0.07	-0.42	0.07	-3.88
0.2 km	0.00	-0.03	-0.21	0.41	-14.91
1 min	0.00	0.00	0.00	0.61	-7.26
2.5 min	0.00	0.04	0.28	0.14	4.83
5 min	0.00	0.62	1.96	0.20	-0.31
5 km	0.00	0.76	2.80	0.95	47.05
3 km	0.00	0.78	2.73	0.68	9.45
7.5 min	0.00	1.31	3.15	1.15	17.74

In Table 6.9, we can see that the smaller scales outperform the larger scales, especially for cases where customers are located close to each other. Furthermore, the scale of the algorithm of Company X proves effective for cases where customers are averagely close to each other, as most algorithms were unable to find a better solution for the three average cases. Additionally, many algorithms surpass the algorithm of Company X for cases where customers are situated far from each other. It is worth noting that the 12 cases where the distance between customers is relatively large are all cases from Country B that include a relatively small number of tasks. Moreover, considering Table 6.10, which illustrates the percentage difference for each new algorithm compared to the algorithm of Company X for cases where customers are located far from each other, it can be concluded that the algorithms with larger scales, such as 3 kilometer and 7.5 minutes, perform well. This suggests a potential correlation between the scale of the insertion strategy and the distance between customers in the case, given that we also observed strong performance from the smaller scales in cases where the distance between customers is small. However, further testing is required to validate this hypothesis on a larger scale in the next experiment. Since both smaller and larger scales demonstrated promising results, we will introduce numerous new scales to test in the exploration experiment. These scales will include: 0, 0.1, 0.25, 0.5, 0.75, 1.25, 1.5, 1.75, 2, 2.5, 3, 4, 5, 7.5, and 10 kilometers, as well as 0, 0.5, 1, 1.5, 2, 2.5, 5, 7.5, and 10 minutes.

Table 6.9: An overview of the number of times the solution of the new algorithms, which differ in scales of the first sorter of the insertion method of the algorithm of Company X, is better than the solution of the algorithm of Company X visualized per case type in terms of the difference in distance between customers. The first column denotes the scale of the sorter where the scales based on driving time are presented with ″min″ and the scales based on the distance with ″km″. The number in brackets behind the classification indicates the total number of cases of the corresponding type used in the experiments.

	Case classification of difference between customers			
Scale	Small (7)	Average (3)	Large (12)	Total (22)
0.2km	5	1	6	12
0.5km	5	0	6	11
3km	0	0	8	8
5km	1	0	6	7
1min	3	1	7	11
2.5min	3	1	5	9
5min	0	1	3	4
7,5min	0	0	6	6

Table 6.10: Results of the experiments for the new algorithms that use different scales for the first sorter of the insertion method algorithm of Company X in case only the cases where customers are located relatively far away from each other are considered. The first column denotes the scale of this sorter where the scales based on driving time are presented with "min" and the scales based on the distance with "km". The second column shows the percentile difference with respect to the algorithm of Company X in terms of the costs.

Scale	Costs (%)
7,5 min	-0.32
3 km	-0.29
1 min	-0.26
0.5 km	-0.14
2.5 min	-0.10
5 km	-0.03
0.2 km	0.04
5 min	0.30

6.1.4 Time windows

Algorithms explanation We have seen in Section 5.5 that some cases have relatively short time windows. For this reason, we devised three algorithms aimed at selecting seed tasks primarily based on the shortest length of the time window of the customers. We utilized the length of the time window of the customers as the new first sorter, with scales of 0, 15, and 30 minutes. Additionally, we created three algorithms wherein tasks with the earliest start of the time window are chosen as seed tasks. In these algorithms, the start of the time window is employed as the first sorter, with scales of 0, 5, and 15 minutes. It is important to note that in these six algorithms, the sorters of the algorithm of Company X are utilized in case multiple tasks fall within the first scale. These six new algorithms were evaluated across 14 cases varying in the length of time windows and earliest finish time of tasks.

In addition to these new seed task selection strategies, we developed six additional algorithms that use the length or the start of the time window as the first sorter of the insertion method. Three algorithms were created using scales of 0, 30, and 60 minutes for both the first sorter regarding the length of the time window and the one sorting based on the start of the time window. Similarly, as for the new seed task selection strategies, the sorters of the Company X algorithm are utilized in these algorithms if the first scale of the new insertion method includes multiple tasks. These six new algorithms were tested across 20 different cases with varying time window characteristics.

Experimental results Table 6.11 presents the results of the experiments conducted with the algorithms incorporating seed task selection strategies based on the length and start of the time window. It is evident from the table that the new algorithms perform significantly worse than the algorithm of Company X in terms of costs, number of routes, and trips. Furthermore, it is notable that the costs, number of planned tasks, routes, and trips are exactly the same for all three scales, as well as for the strategy based on the start of the time window and the strategy based on the length of the time window. This suggests that the scales may not differ significantly enough to yield differences in solutions. It could also be caused by many time windows starting and ending at the same time.

Table 6.11: Results of the experiments for the new algorithms that use different seed task selection methods based on the length and start of the time window. The first column denotes the sorter and scale of the algorithm where "TW Length" stands for the length of the time window and "TWStart" for the start of the time window. The scales are presented in minutes. The other columns show the percentile difference with respect to the algorithm of Company X for the number of planned tasks, costs, number of routes and trips and finally the calculation time (Calc time).

Algorithm	Planned tasks (%)	Costs (%)	Routes (%)	Trips (%)	Calc time (%)
TWLength Scale 0	0.03	0.96	2.69	0.54	13.20
TWLength Scale 15	0.03	0.96	2.69	0.54	4.82
TWLength Scale 30	0.03	0.96	2.69	0.54	3.27
TWStart Scale 0	0.00	1.30	3.61	1.35	23.84
TWStart Scale 5	0.00	1.30	3.61	1.35	19.27
TWStart Scale 15	0.00	1.30	3.61	1.35	21.28

In Table 6.12 and Table 6.13, we observe that although the algorithms selecting the seed task based on the time window length or start perform on average notably worse than the algorithm of Company X, they still outperform the algorithm of Company X for 3 and 5 cases out of 14, respectively. Remarkably, these cases correspond to those with the easiest time window characteristics, and once again, their main depots are located in Country B and thus include a relatively small number of tasks. Additionally, we can conclude that the location of customers with respect to the depots poses a more restrictive characteristic than the time windows. This is evidenced by the algorithm of Company X that selects the most difficult customers regarding their location with respect to the depots outperforming the new algorithms that are selecting customers based on their time window features.

Table 6.12: An overview of the number of times the solution of the new algorithms, which use the length of the time window as the first sorter of the seed task selection method, is better than the solution of the algorithm of Company X visualized per case type in terms of the length of the time windows. The first column denotes the scale of the sorter in minutes. The number in brackets behind the classification indicates the total number of cases of the corresponding type used in the experiments.

	Case classification of time window length			
Scale	Short (5)	Average (3)	Large (6)	Total (14)
0	0	0	3	3
15	0	0	3	3
30	0	0	3	3

Table 6.13: An overview of the number of times the solution of the new algorithms, which use the start of the time window as the first sorter of the seed task selection method, is better than the solution of the algorithm of Company X visualized per case type in terms of the percentage of tasks that needs to be completed within 5 hours after the start of the shift. The first column denotes the scale of the sorter in minutes. The number in brackets behind the classification indicates the total number of cases of the corresponding type used in the experiments.

	Case classification of % tasks that finish in 5 hours			
Scale	Few (7)	Average (4)	Many (3)	Total (14)
0	3	2	0	5
5	3	2	0	5
15	3	2	0	5

We see similar results for the insertion strategies based on the start and length of the time windows as for the seed task selection strategies in Table 6.14, where the algorithm of Company X outperforms the new algorithms in all KPIs. Furthermore, it stands out that the calculation times of the new algorithms are much larger than the calculation time of the algorithm of Company X. This implies that the insertion of tasks based on time window characteristics produces relatively bad initial solutions and the local search and R&R need extra time to fix these routes however in the end, the final solutions are still worse.

Table 6.14: Results of the experiments for the new algorithms that use different insertion methods based on the length and start of the time window. The first column denotes the sorter and scale of the algorithm where "TW Length" stands for the length of the time window and "TWStart" for the start of the time window. The scales are presented in minutes. The other columns show the percentile difference with respect to the algorithm of Company X for the number of planned tasks, costs, number of routes and trips and finally the calculation time (Calc time).

Algorithm	Planned tasks (%)	Costs (%)	Routes (%)	Trips (%)	Calc time (s)
TWLength Scale 30	-0.01	1.71	3.94	1.85	92.34
TWLength Scale 60	-0.01	1.76	4.10	2.09	96.38
TWLength Scale 0	0.00	1.86	4.27	2.25	60.54
TWStart Scale 60	-0.02	1.97	5.00	2.73	92.69
TWStart Scale 0	-0.02	2.01	5.17	2.89	72.10
TWStart Scale 30	-0.02	2.01	5.17	2.89	89.80

In Table 6.15 and Table 6.16, we once again observe that the new algorithms outperform the algorithm of Company X for the cases that have the easiest time window characteristics. It is noteworthy that all these cases are once more cases of which their main depot is located in Country B and includes a relatively small number of tasks. Since the results for both the seed task selection strategies and the insertion strategies were subpar, we decided to leave them out of the exploration experiment.

Table 6.15: An overview of the number of times the solution of the new algorithms, which use the length of the time window as the first sorter of the insertion method, is better than the solution of the algorithm of Company X visualized per case type in terms of the length of the time windows. The first column denotes the scale of the sorter in minutes. The number in brackets behind the classification indicates the total number of cases of the corresponding type used in the experiments.

	Case classification of time window length			
Scale	Short (6)	Average (2)	Large (12)	Total (20)
0	0	0	4	4
30	0	0	3	3
60	0	0	3	3

Table 6.16: An overview of the number of times the solution of the new algorithms, which use the start of the time window as the first sorter of the insertion method, is better than the solution of the algorithm of Company X visualized per case type in terms of the percentage of tasks that needs to be completed within 5 hours after the start of the shift. The first column denotes the scale of the sorter in minutes. The number in brackets behind the classification indicates the total number of cases of the corresponding type used in the experiments.

	Case classification of % tasks that finish in 5 hours			
Scale	Few (10)	Average (8)	Many (2)	Total (20)
0	2	2	0	4
30	2	2	0	4
60	2	2	0	4

6.1.5 Summary

In the preceding section, we investigated the outcomes of the screening experiment. Subsection 6.1.1 illustrated that relying solely on order quantity for selecting seed tasks yielded inferior solutions compared to the algorithm of Company X. Nonetheless, a hybrid approach wherein only the largest orders were chosen as seed tasks and the selection strategy of the algorithm of Company X was employed for the remaining seed tasks, displayed promising outcomes. In the exploration experiment, various scales will be explored to test when a task is large enough to be selected as a seed task. In Subsection 6.1.2 different scales of the initial sorter of the seed task selection method of the algorithm of Company X, which sorts based on the difference in distance between the nearest and second nearest depot, were evaluated. Furthermore, we introduced a seed task selection strategy that selects the task that is located furthest away from the selected depot. This new strategy showed some good results compared to the algorithm of Company X and will be tested on a larger scale in the exploration experiment. Moreover, in Subsection 6.1.3, multiple scales of the first sorter of the insertion method of the algorithm of Company X, sorting tasks based on their distance from unplanned tasks to the seed task, are inspected. These alternative strategies exhibit promising results compared to the algorithm of Company X, prompting their evaluation on a larger scale in the next experiment. Conversely, diverse seed task selection and insertion methods grounded on the start and duration of the time window are assessed in Subsection 6.1.4. Regrettably, these approaches yielded subpar performance compared to the algorithm of Company X, leading to their exclusion from the exploration experiment. Finally, it stands out that many algorithms despite performing badly on average, still surpassed the algorithm of Company X for the cases in which the main depot is located in Country B and thus has a relatively small number of tasks.

6.2 Exploration experiment

In the exploration experiment, various construction methods that are created based on the data analysis and advice from the creators of the algorithm of Company X are tested on the full dataset which is explained in Section 5.1. In this way, the results are more reliable. As mentioned in Section 4.4, the goal is to detect the most promising seed task selection and insertion strategies that will be used in the exploitation experiment where we are going to test combinations of changes instead of testing with single changes to the algorithm of Company X. Furthermore, we hope to find a correlation between the input data and certain strategies such that employees of ORTEC know in the future beforehand which strategies and scales potentially works well. In Subsection 6.2.1, we will explain how we came up with the different construction methods that are used in this experiment. Moreover, we will introduce the categories that are used to split up the analysis to detect per category the most promising ones in Subsection 6.2.2. Afterwards, we will present the outcomes of the experiment per category in the subsequent sections. Finally, the main findings of the exploration experiment are summarised in Subsection 6.2.8.

6.2.1 Construction methods of the exploration experiment

In the exploration experiment, we are testing with a total of 113 algorithms that all have different construction methods that are created by changing a single parameter setting of the algorithm of Company X. The corresponding parameter settings of the new algorithms can be found in Table A.1 where the settings of the algorithm of Company X are presented in bold. In the previous section, 42 algorithms are already introduced which are all based on the results of the screening experiment that was supported by the data analysis of Chapter 5. The remaining algorithms are created by interesting parameters that could be changed which are mentioned by the creators of the algorithm of Company X and experts in OHD.

First of all, different batch sizes and number of batches are used in combination with different insertion methods. The algorithm of Company X uses batch sizes of 1 and a total of 40 batches are created and these batches are inserted with the use of cheapest insertion. The following combinations of batch sizes and number of batches are tested: 1x40, 2x20, 3x14, 4x10, 5x8, 6x7, 7x6, 8x5, 10x4, 14x3, 20x2, 40x1 where the first number denotes the batch size and the second number the number of batches that are created. Note the product of the batch size and number of batches does not exceed 42. This has been done since there are never routes that contain more than 42 customers. Consequently, creating more batches is useless and increases the calculation time. These batches are inserted with the use of cheapest insertion which inserts the tasks in the batch based on the task ID, cheapest insertion which inserts the tasks based on the closest distance to one of the tasks already present in the route and parallel cheapest insertion. It is important to note that after each batch is inserted, the batches are updated contrary to what occurs in the algorithm of Company X. This means that if for example, we use batch sizes of 2 that after we inserted the first two tasks, the next batch is created for the updated route including the first two inserted tasks. These combinations of different batch sizes and insertion methods result in 36 new algorithms.

Secondly, five different estimators could be interesting to use for the cheapest insertion according to the creators of the algorithm of Company X. Besides the current estimator, `CostAndTimeWindowOrdering (CostsTW)`, `Distance`, `DrivingTime`, `Costs` and `WaitTimeCosts` could be used. In case the distance is used as an estimator, the task is inserted at the place where the increase in distance of the route is minimized. Similarly, the other estimators are used where the `WaitTimeAndCosts` estimator is a weighted average between the increase in waiting time and costs in the route. This results in an extra 4 algorithms.

Furthermore, different scales for the second sorter of the seed task selection could be used in which we could also change the direction of the sorting. In Section 4.1.1 the consequences of changing the scales and direction of the second sorter are explained. We have chosen 0, 1, 2.5, 5, and 7.5 kilometers and 0, 1, 2.5, 5, and 10 minutes as scales both sorted increasingly as decreasingly. This results in an extra 20 algorithms.

Finally, the second sorter in the insertion strategy of the algorithm of Company X that sorts based on the start of the time window of the tasks is tested with different scales. The current scale is 60 minutes and the following scales are experimented with: 0, 20, 80, 120 minutes. This adds another four algorithms to this experiment.

In addition, some algorithms with bad construction methods are added to compare the final solutions of the algorithms to the ones that come from bad initial solutions. In this way, we can see the effect of a good initial solution compared to a bad initial solution. We have included an algorithm that inserts customers based on the largest order quantities, an algorithm that selects the largest orders as a seed tasks, an algorithm that uses the task ID as the first sorter for both the seed task selection as the insertion method and an algorithm that chooses the task with the shortest time window length as seed task. In case the latter one has multiple tasks with the same shortest time window, a seed task is chosen based on the task ID. Consequently, an extra four algorithms are added to the exploration experiment which function as benchmarks.

6.2.2 Categories exploration experiment

To make fair comparisons, we are going to split the new algorithms into several categories where 3 categories belong to the seed task selection methods and 4 belong to the insertion methods. The 3 categories of the seed task selection methods are the strategies that select customers based on their locations to depot(s) as seed task, the different scales of the second sorter of the seed task selection method of the algorithm of Company X and the seed task selection strategies that focus on the orders with the largest quantity. The four categories of the insertion methods include the first and second sorters of the insertion method of the algorithm of Company X which are the sorters based on distance to the seed task and the start of the time window. Moreover, it includes the different estimators of the cheapest insertion. Finally, it also includes the different batch sizes and number of batches in combination with different insertion methods. Note that these 7 categories are listed in the first column in Table A.1 where the `SeedFarAway` is joined together with the scale of the first sorter of the seed task selection method into one category and "Batch size x number of batches" and "Insertion procedure" are merged.

6.2.3 Performances of categories

In order to decide which parameter changes will be used in the exploitation experiment, where we are going to use combinations of changes, we look at a few different KPIs. First of all, we look at the algorithms that have the best performance on average, however, as mentioned before a big advantage of parallel computing is the ability to compensate for algorithms that work really well for particular cases but bad for other cases. For this reason, we are also going to look at the peak values of algorithms thus the best 3 and best 25% rankings. Note that the ranking is determined firstly based on the highest number of planned tasks. In case this results in a tie, the algorithm with lower costs will be ranked higher. Finally, we look at the potential of an algorithm and if some particular algorithms work very well for specific sorts of cases. In this way, we hope to create a diverse set of construction methods that perform well over a varied set of data of Company X.

If we have a look at the performances of the algorithms over the whole dataset in terms of average costs in Table A.2, we see that the best-performing algorithms are the ones with changes to the first sorter of the seed task selection method and the batch size in combination with the number of batches and the insertion method. Furthermore, we can see that the third category that is relatively highly ranked is the first sorter of the insertion method of Company X. Note that we sorted in this table only on the costs and ignored the fact that some algorithms have a really small difference in planned tasks. This could of course lead to a slightly lower amount of costs and thus in a higher rank but in total there are 92823 tasks in the dataset and in the top 50, the highest number of unplanned tasks is 6. Consequently, we believe the effect of these unplanned tasks on the costs is negligible.

If we have a look at Table A.3, that is sorted based on the average of the best 25% rankings of the algorithms, we see a similar performance of the categories except that the first sorter of the insertion method of the algorithm of Company X has some more top rankings. If we zoom in even more on the peak values by looking at the 3 best performances of the algorithms on the dataset, and we sort increasingly on the third rank, we see similar performances of the categories in Table A.4 as in Table A.2 for the average performance. In which the top is dominated by the combination of batch sizes, number of batches and insertion methods and the first sorter of the seed task selection method. In the following sections, we are going to dive deeper into the performance of the different categories concerning specific cases to detect some potentially good parameter settings and strategies.

6.2.4 Seed task selection strategies

Order quantity In Figure 6.1, the gap in percentage between the best individual solutions found and solutions found by the algorithms per case type concerning the utilization rates of the largest orders is provided. We can see that the algorithm with the largest scale of 475, which is half the capacity of the largest truck, performs in general the best. For the cases, with lower utilization rates of the largest orders it provides exactly the same solutions as the algorithm of Company X, for the cases with average utilization rates of the largest orders almost the same and for the cases with high utilization rates of the largest orders it provides the best solutions for the algorithms that select the seed task primarily on the order quantity. Consequently, this algorithm does exactly for which it is created, namely, to tackle the largest orders. This graph also shows us that a relatively large scale in terms of the order quantity as the first sorter of the seed task selection method is also useful in a non-parallel algorithm since it performs better for cases that include large orders and provides the same results as the algorithm of Company X for cases that do not include orders above this scale.

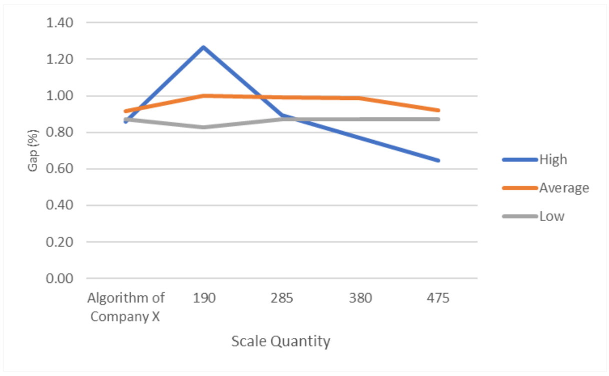


Figure 6.1: A visualization of the performance of algorithms that primarily select the seed task based on order quantity per case classification, showing the percentage gap to the best individual solutions found. The case classification used is the utilization rate of the largest orders in the case. The x-axis represents the scale of the sorter, with the algorithm of Company X labelled by its name since it doesn't use such a scale. The y-axis represents the percentage gap. Additionally, the case classification agenda is presented on the right side of the graph.

Difference in distance between nearest and second nearest depot In Figure 6.2 and 6.3, the gap in percentage between the best individual solutions found and solutions found by the algorithms per case type concerning the difference in distance and driving time between the nearest and second nearest depot is provided. From these graphs, it can be concluded that the seed tasks should be selected not only on the difference in distance between the nearest and second nearest depot since the scale 0 is outperformed for each case type by another scale. Furthermore, it could be seen that the scale of the algorithm of Company X is outperformed by many different scales for each case type. Moreover, we can see that the relatively larger scales of 40 kilometers and 30 minutes provide good results for each type of case and that the small scale of 2.5 kilometers performs really well for the cases where the difference between the nearest and second nearest depot is small. Finally, it is difficult to detect hard relationships between the input data characteristics and the scales based on this so further research needs to be conducted for this.

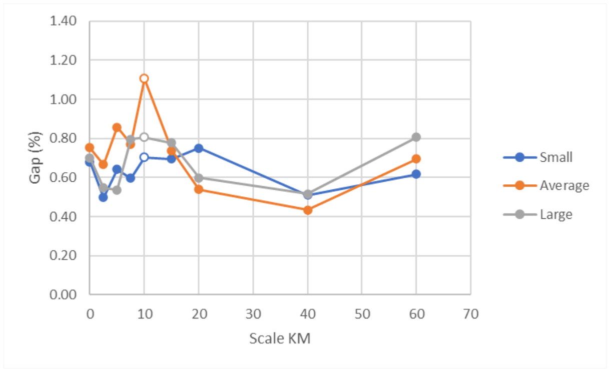


Figure 6.2: A visualization of the performance of algorithms that primarily select the seed task based on the difference in distance between the selected depot and the nearest depot that is not the selected depot per case classification, showing the percentage gap to the best individual solutions found. The case classification used is the difference in distance between the nearest and second nearest depot of the customers in the case. The x-axis represents the scale of the sorter, where the scale of the algorithm of Company X is shown with white dots. The y-axis represents the percentage gap. Additionally, the case classification agenda is presented on the right side of the graph.

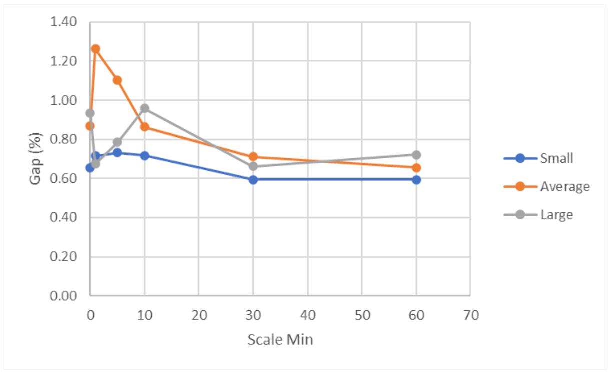


Figure 6.3: A visualization of the performance of algorithms that primarily select the seed task based on the difference in driving time between the selected depot and the nearest depot that is not the selected depot per case classification, showing the percentage gap to the best individual solutions found. The case classification used is the difference in distance between the nearest and second nearest depot of the customers in the case. The x-axis represents the scale of the sorter and the y-axis represents the percentage gap. Additionally, the case classification agenda is presented on the right side of the graph.

Parameter selection exploitation experiment In case we only consider the seed task selection strategies for the average performance, the best 25% rankings and the best 3rd rank in Tables A.5, A.6 and A.7, we can see that the first sorter of the seed task selection strategy dominates the top rankings.

Furthermore, we could see the perfect example of an algorithm that is very suitable for a parallel algorithm but not for the current non-parallel algorithm of Company X, namely the `SeedFarAwayDistance` which selects the customer that is furthest away in terms of distance from the selected depot as seed task. We can see from Table A.5, that it performs relatively badly on average however the average of the 25% best rankings is extremely low as could be seen in Table A.6. This means that this algorithm either performs extremely well or bad. If we had only considered the average performances of the algorithms, we would never have determined that this algorithm performed for some cases extremely well. For the above-mentioned reasons, the `SeedFarAwayDistance` is the first seed task selection method that we will select for the exploitation experiment.

Secondly, we will use `SeedTaskFarToNearDiffDepot2,5KM` and `SeedTaskFarToNearDiffDepot40KM` because these are performing really well for the KPIs mentioned in the start of Subsection 6.2.3. We can also see that the scales 60 kilometers, 30 and 60 minutes for the first sorter of the algorithm of Company X score also quite well behind the first two chosen parameter settings. We have only chosen to go for the scale of 30 minutes since the algorithms with such large scales are kind of similar to the `SeedFarAwayDistance` algorithm since there are quite some cases that do not include such large differences between the distance or driving time between the nearest and second nearest depot. Consequently, the second sorter of the algorithm of Company X will be used mainly which is the sorter that selects the customer that is furthest away from the selected depot. In order to have a good but also varied set of seed task selection methods, we have decided to choose only the scale of 30 minutes.

Finally, we have selected the scale of 20 kilometers since this algorithm also performs relatively well on the KPIs. Furthermore, we only had selected the scale of 2.5 and 40 kilometer which are relatively low and high scales. For this reason, to increase the variety of the selected set of seed task selection methods we have chosen to include the scale of 20 kilometers. This means that we have chosen 5 different seed task selection methods that will all be used in combination with the scale of 475 in terms of the order quantity as the first sorter. Consequently, we have in total 10 different seed task selection strategies that will be used in the exploitation experiment.

Remark that we have not chosen to use any different scales for the second sorter of the algorithm of Company X. First of all, from Table A.2 can be concluded that the decreasing scales work in general better than the increasing ones and the current scale is also decreasingly. Moreover, by increasing the scale, the third sorter will be become more important which is in this case the order quantity. By using the 475 scale as first sorter, we will filter out all difficult tasks in terms of order quantity and we no longer need the third sorter to be used. Additionally, this will also help to not let the number of combinations explode in the exploitation experiment.

6.2.5 Insertion methods

Batch sizes, number of batches and insertion procedure In case we only consider the insertion methods for the average performance, the best 25% rankings and the best 3rd rank in Tables A.8, A.9 and

A.10, we can see that for all KPIs the combination of different batch sizes, number of batches and insertion methods are by far the best. Consequently, we are going to select significantly more parameter settings from this category than from the others.

We have compared all the algorithms within this category for the main KPIs which can be found in Tables A.11, A.12 and A.13. It stands out that the smaller batch sizes in combination with a larger number of batches outperform the algorithms that include larger batch sizes in combination with a lower number of batches. This is also logical since the first batch is created purely based on the seed task. In case this batch is very large, we are inserting a lot of tasks with respect to the seed task while ignoring that the route is growing. Consequently, it is more logical to insert smaller batch sizes since we then observe more often what the nearest tasks are with respect to all already inserted customers in the route.

Furthermore, it could be seen that resorting after inserting a batch improves the results which could be concluded if we compare the `CloseToSeedBatch1x40ResortTrue` with the algorithm of Company X in the Tables. The only difference between those two algorithms is that we resort after each inserted batch. We can see that this algorithm outperforms the algorithm of Company X at all KPIs and is on average 0.32% better in terms of costs than the algorithm of Company X. However, this result comes at the cost of a 40% larger running time at the construction phase and a 20% larger running time while executing the whole algorithm since the resorting costs logically extra time. Nevertheless, in case we use for this case parallel cheapest insertion instead of cheapest insertion, the increase in running time while executing the whole algorithm is only 5% and the same solution will be achieved. In Table A.14, the average running times for all the phases in this category are presented. From this Table, it can be concluded that the running times are almost always lower than the algorithm of Company X in case we use parallel cheapest insertion. On top of that, most of the algorithms including parallel cheapest insertion provide good solutions so this is a win-win situation.

Moreover, we can see that while inserting with cheapest insertion, it is most of the time better to sort the batch based on distance instead of inserting them on task ID except for the batch sizes of 2 and 3. An explanation for this could be that these batch sizes are so small that the sequence of insertion matters less and of course the chance of inserting them in the 'right' sequence is a lot higher for smaller batch sizes since there are fewer options. Logically, resorting the batches increases the running time. However, it could be seen that the average running times are not even close to the limit of 20 minutes thus a small increase in running time is not insurmountable. The limit is only reached for some large cases in which the local search and R&R have to do a lot of work in improving a very bad initial solution. Since the worst construction methods will not be selected for the new parallel algorithm, an improvement in terms of solution value can come at the cost of an increase in running time since there is enough space towards the maximum running time limit of 20 minutes. We have chosen in total 9 different parameter settings from this category that perform well on the KPIs and these are listed in Table 6.19. In order to diversify the search, we have chosen combinations of batch sizes and number of batches from the three different insertion methods.

Distance between customers Although the top rankings in Tables A.8, A.9 and A.10 are dominated by the insertion methods including the different batch sizes and number of batches in combination with different insertion methods, some scales of the first sorter of the insertion method of the algorithm of Company X also perform relatively well for the KPIs. Furthermore, these good-performing scales outperform by far the insertion methods that include the scale of the second sorter of the algorithm of Company X and the different estimators of the insertion procedure.

From Figures 6.4 and 6.5, it could be concluded that the scale of the algorithm of Company X is performing for all case types decently but some scales perform better. Moreover, it stands out from both graphs that, in general, the lower the scales, the better the solutions. However, we can see that a scale of 0 is never the best scale, except for the case where customers are located close to each other and the scale is estimated with driving time. This implies that the time window of customers is relevant while inserting but not too much as could be seen in the performance of the larger scales in which the time window is becoming more important while inserting. Furthermore, it could be seen that the best scale value based on distance decreases with the customer density of the case. This suggests a correlation between the scale of the first sorter of the insertion method of the algorithm of Company X and the customer density in a case. Besides, we can see the scales estimated with distance outperform the scales based on the driving time. This can also be obtained from Tables A.15, A.16 and A.17 which include the performance of the algorithms that use different scales of the first sorter of the insertion method of the algorithm of Company X.

We can see in the tables that the following three scales dominate the top rankings of the KPIs: 0.1, 0.25, and 1.5 kilometers. We have chosen to use these scales in the exploitation experiment since these were also the best-performing scales for the 3 different types of cases labelled based on the density of customers. In this way, we have a small but diverse set of scales that is suitable for the different types of cases in terms of customer density.

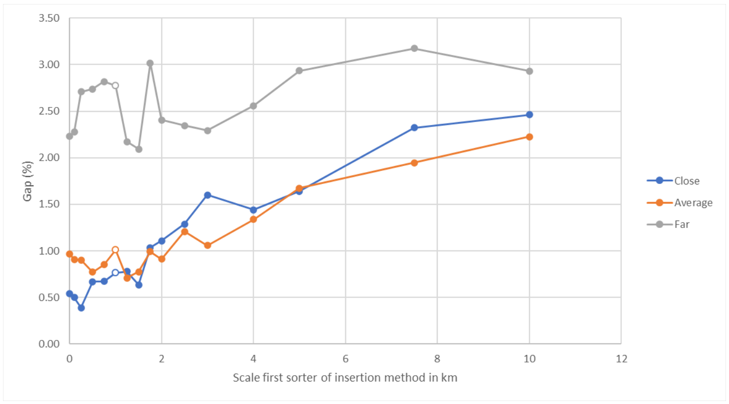


Figure 6.4: A visualization of the performance of algorithms that use different scales of the first sorter of the insertion method of the algorithm of Company X per case classification, showing the percentage gap to the best solutions found. The case classification used is the difference in distance between the customers in the case. The x-axis represents the scale of the sorter in kilometers and the y-axis represents the percentage gap. The scale of the algorithm of Company X is denoted with white dots in the graph. Additionally, the case classification agenda is presented on the right side of the graph.

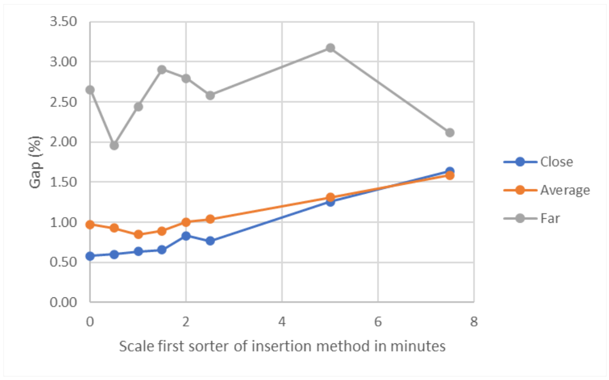


Figure 6.5: A visualization of the performance of algorithms that use different scales of the first sorter of the insertion method of the algorithm of Company X per case classification, showing the percentage gap to the best solutions found. The case classification used is the difference in distance between the customers in the case. The x-axis represents the scale of the sorter in minutes and the y-axis represents the percentage gap. Additionally, the case classification agenda is presented on the right side of the graph.

Insertion method estimator and second scale of insertion method The selected parameter settings result already in 10 * 9 * 3 = 270 different algorithms. In order to prevent the number of algorithms and thus the number of runs in the exploitation experiment from exploding, we have decided to choose only two parameter settings from one of the two remaining categories such that the number of algorithms will be 540 in total.

In Tables A.18, A.19 and A.20, the results in terms of the KPIs of both categories are provided. We have chosen the `Distance` estimator and the current estimator (`CostsTW`) of the cheapest insertion as the last parameter settings for the exploitation experiment. First of all, we believe that this improves the variety of the new algorithms and thus the robustness of the new parallel algorithm the best. We can see from Table A.18, that the solutions of the scales of 0, 20, 60 and 120 minutes for the second sorter of the algorithm of Company X are almost identical which implies that a change in this scale barely changes the solutions. Furthermore, the estimator of the cheapest insertion method is used for every insertion of a task and the second sorter of the insertion method of the algorithm of Company X is only used in case the first sorter is not able to filter out one task. Since we have chosen two relatively small scales for the first sorter of the insertion method, respectively 0.1 and 0.25 kilometers, the second sorter will be used even less. Moreover, in the algorithm of Company X, the estimator is the weighted average of the costs and the placement of the task with respect to its time window. Thus we expect that inserting based on distance will have quite a large impact on the insertion place of a task although the distance is of course also a bit included in the costs. Finally, in Table A.19, it can be seen that the absolute peak values of the `Distance` estimator are the best. Likewise, in Table A.18, it can be seen that the `CostsTW` estimator performs the best on average which is also logical since this estimator was specially created for the algorithm of Company X. For these reasons, we have chosen the `Distance` and `CostsTW` estimators as the last parameter settings for the exploitation experiment.

6.2.6 Comparisons against benchmarks

As mentioned in Subsection 6.2.1, we have added a few bad algorithms that function as benchmarks such that we can see how important the starting solution is with respect to the quality of the final solution. We can see in Table 6.17 that these algorithms indeed perform poorly considering their high costs and low rankings. Additionally, the running times are also really high since the local search and R&R have to do a lot of work to fix these bad constructions. On the other hand, we can see in Table A.21 that the local search and RR are able to transform the very bad initial solution into a good final solution in some occasions considering some high rankings. `SeedTaskShortestTWLength` even finds the best solution for all 113 algorithms for one case. If we have a closer look at these higher rankings, we can see that they all occur for cases of Country B which include a relatively low number of customers. For the larger cases, the highest ranking of these benchmark algorithms is 63 implying that they provide very poor solutions for the larger cases. Since the local search and R&R can explore a relatively large part of the solution space for the cases of Country B compared to the larger cases where the number of possible solutions is extremely larger, the initial solution is of less importance. However, these good solutions initiate from pure randomness and occur on a very rare base thus are not useful for our new parallel algorithm. All in all, we can conclude that the initial solution is quite important for the algorithm of Company X where the larger the problem, the more important the initial solution.

Table 6.17: Results of the benchmark algorithms where the percentile difference in costs to the algorithm of Company X is shown in the second column. The third column denotes the average ranking of the final solution and the last column shows the average calculation time in seconds.

Algorithm	**Diff (%)**	**Avg Ranking**	**Calc Time (s)**
Algorithm of Company X	0.00	42.00	337.97
SeedTaskShortestTWLength	1.22	102.53	430.71
SeedTaskLargestOrderQuantity	1.41	104.63	445.61
InsertionLargestOrderQuantity	2.40	111.23	641.74
SeedTask&InsertionTaskID	3.09	112.10	662.08

6.2.7 Parallel vs algorithm of Company X

Table 6.18 includes the results in case we would use all 113 algorithms in parallel compared to the solutions of the algorithm of Company X. For this evaluation, the parallel design displayed in Figure 4.1 is used where n equals in this case thus 113. The first column of the table denotes how many of the best initial solutions will proceed to the local search and R&R phase which corresponds to the p in the figure. The initial solutions are ranked firstly based on the number of planned tasks and in case this number is equal then the solution with the lower costs will be ranked higher. The second column shows the total costs of all solutions in the dataset per number of starting solutions that proceed to the local search and RR. The third column shows us the difference in costs to the solution of the algorithm of Company X. The fourth column denotes the difference in percentage compared to the case where we would run all 113 algorithms fully in parallel. For example, we would lose 0.32% if we only proceed with the best starting solution instead of proceeding with all 113 starting solutions. The fifth column visualizes the extra computational time in hours compared to the algorithm of Company X. Note that no matter how many starting solutions will be used in the local search and RR phase, first the constructions of all 113 have to be executed. For example, the extra calculation time of using 3 starting solutions is calculated by the sum of the total construction time of all 113 algorithms over all 42 cases and the local search and RR calculation time of the 3 best starting solutions for each case. The sixth column shows us the savings compared to the solution of the algorithm of Company X. The seventh column visualizes the costs of the extra computational time that was denoted in the fifth column where each hour of computational time costs 16 cents as mentioned in Section 3.3.2. The last column denotes the net savings for Company X in case they would pay for all extra computational costs.

Remark that it is already possible in the software of ORTEC to proceed with either the best or all starting solutions to the local search and RR phase. Consequently, by single adjustments to the algorithm of Company X, large savings could already be obtained. Furthermore, we can see that the values of net savings are increasing with the number of starting solutions which is a consequence of the proportion of the calculating costs and the costs of route planning of Company X. Note that we have to improve an extremely small percentage of the total solution value of this dataset to make an extra hour of computational time beneficial. On top of that, the average running time of the algorithm of Company X is about 5 minutes implying that an extra hour of running time is a lot. This explains the proportions of the savings versus the extra computational time and results in the case that it is optimal to proceed with every starting solution.

Table 6.18: The results of using the 113 single change algorithms in parallel compared to the algorithm of Company X, where the net savings are presented in proportions to the total planning costs while using the algorithm of Company X.

Nr Solutions to LS&RR	Diff algorithm of Company X (%)	Diff best (%)	Extra Calc Time (h)	Extra Calc Costs (€)	Net Savings (%)
1	-0.76	0.32	167.41	26.78	0.76
3	-0.94	0.14	169.63	27.14	0.94
5	-0.97	0.11	171.88	27.50	0.97
10	-1.01	0.06	174.09	27.86	1.01
25	-1.03	0.05	176.37	28.22	1.03
50	-1.04	0.04	178.74	28.60	1.04
75	-1.05	0.03	181.30	29.01	1.05
100	-1.07	0.01	183.57	29.37	1.07
113	-1.08	0.00	185.87	29.74	1.08

Furthermore, it stands out that the best starting solution does not always result in the best final solution since the costs are 0.32% higher if we only proceed with the best starting solution instead of proceeding with all 113 starting solutions. The starting solution of the best final solution is on average 1.08% worse than the best starting solution and has an average ranking of 27.31 after construction. This confirms the numbers in Table 6.18 that it is beneficial to proceed with multiple starting solutions to the local search and RR phase. Additionally, it means that the initial solution is not extremely important in case we include relatively good construction methods contrary to the bad benchmark algorithms in Subsection 6.2.6.

Moreover, we obtained that the percentual savings are a lot higher for the cases of Country B than the cases of Country A. The average saving for the Country B cases is 3.18% whereas for the Country A cases, this saving is 0.95%. Consequently, we can conclude that the algorithm of Company X does not perform well for these smaller cases which also matches with the observations in Subsection 6.1.5. Besides, the average ranking of the starting solution resulting in the best final solution is lower for the Country A cases than for the Country B cases, respectively 20.03 vs 45.50. This also aligns to the conclusions made in Subsection 6.2.6 about the importance of starting solutions for the smaller cases.

6.2.8 Summary

In this subsection, we have analysed the results of the exploration experiment and have selected the parameter settings of the seed task selection and insertion methods that will be used in the exploitation experiment. We have chosen a varied set of parameter settings based on the average performance in terms of costs and the best rankings in terms of the final solution for all cases. Where we have selected more parameter settings from the best-performing categories than the categories that performed worse. The best-performing categories were the first sorter of the seed task selection method, the batch size and number of batches in combination with different insertion methods and the scale of the first sorter of the insertion method. The chosen strategies and corresponding parameter settings can be found in Table 6.19. In Subsection 6.2.4, we saw that the additional sorter on top of the first sorter of the algorithm of Company X that selects very large orders as a seed task has on average a positive effect on the solution values of cases that include large orders. Additionally, we detected some correlation between the scales of the first sorter of the insertion method of Company X and the difference in distance between customers in a case. For the case types where the difference between customers is relatively small, the smaller scales outperform the somewhat larger scales. For the cases where the difference between customers is larger, we saw that the somewhat larger scales outperform the other scales. Besides, we saw that the insertion methods based on distance outperform the insertion methods based on driving time. Since the scales above 2 kilometer performed really poorly, it could also be concluded that the insertion of customers should be mainly done by distance since the larger the scale the less important the distance and the more important the start of the time window of the customer. In Subsections 6.2.6 and 6.2.7, we saw that the initial solution is quite important for the algorithm of Company X, especially for the larger cases and that is beneficial to use multiple construction methods and also to proceed with multiple starting solutions to the local search and R&R phase. As a result, we observed that we could already save a lot of costs by running multiple algorithms in parallel that are created from a single change to the algorithm of Company X. Finally, we can conclude that the algorithm of Company X does not work well for the smaller cases of Company X.

6.3 Exploitation experiment

In Section 6.2, we have chosen the parameter settings that will be used in the exploitation experiment where we will create algorithms with all combinations of these parameter settings. The chosen parameters per category are listed in Table 6.19. The second row denotes that we are including and excluding the order quantity sorter in the seed task selection strategies. In case the sorter is included, a scale of 475 will be used. In the row of the seed task selection strategies, we can see that we have the `SeedFarAwayDistance` and four different scales of the difference between the selected depot and the nearest not-selected depot as our five seed task selection methods. We have nine different combinations of batch sizes, number of batches and insertion methods and these are listed in the fourth row. Finally, we have three different scales for the first sorter of the insertion method and two different estimators that will be used for the insertion procedures. Thus, we have 2*5*9*3*2 = 540 different combinations of parameter settings. These 540 algorithms are also tested on the dataset while using the parallel design of Figure 4.1 and the results are shown in Table 6.20. From the table, we can see that these new combination algorithms provide even better solutions for the 42 cases in the dataset than the algorithms of the exploration experiment in which only single changes were made to the algorithm of Company X. Furthermore, we can see just as in Table 6.18 that it is more profitable to use multiple starting solutions because the higher the number of starting solutions the higher the profit.

Table 6.19: Overview of all selected parameters settings per category that will be used in the exploitation experiment

Category	**Parameter setttings**
Order quantity sorter (2)	475, none
Seed task selection strategies (5)	SeedFarAwayDistance, DiffDepot2.5KM, DiffDepot20KM, DiffDepot30Min, DiffDepot40KM
Insertion strategies (9)	
BatchSize x NumberOfBatches + Insertion method	5x8 Parallel Cheapest Insertion, 3x14 Parallel Cheapest Insertion, 2x20 Parallel Cheapest Insertion, 1x40 Parallel Cheapest Insertion, 4x10 Parallel Cheapest Insertion, 3x14 Random Cheapest Insertion, 2x20 Random Cheapest Insertion, 3x14 Sorted Cheapest Insertion, 6x7 Sorted Cheapest Insertion
Scale first sorter insertion method (m) (3)	100, 250, 1500
Insertion Estimators (2)	CostAndTimeWindowOrdering, Distance

Table 6.20: The results of using the 540 algorithms of the exploitation experiment in parallel compared to the algorithm of Company X, where the net savings are presented in proportions to the total planning costs while using the algorithm of Company X.

Nr Solutions to LS&RR	**Diff algorithm of Company X (%)**	**Diff best (%)**	**Extra Calc Time (h)**	**Extra Calc Costs (€)**	**Net Savings (%)**
1	-1.04	0.30	782.94	125.27	1.03
5	-1.13	0.21	791.51	126.64	1.13
10	-1.16	0.18	801.81	128.29	1.15
50	-1.26	0.08	889.31	142.29	1.25
100	-1.29	0.05	1001.63	160.26	1.28
200	-1.31	0.02	1233.38	197.34	1.30
400	-1.32	0.02	1737.06	277.93	1.30
541	-1.34	0.00	2181.52	349.04	1.32

6.4 Exploration and exploitation experiments

In the exploration and exploitation experiments, we have applied a total of 653 unique algorithms on the dataset. In this section, we are going to analyse the outcomes if we would run all these algorithms in parallel on this dataset. In this way, we can answer the questions of Chapter 4 and design our new parallel algorithm.

6.4.1 Single changes vs Combination algorithms

In Table 6.21, the results of these 653 unique algorithms while using the parallel design presented in Figure 4.1 are shown. We can see that the results are slightly better as in Table 6.20, implementing that the algorithms that only include single changes do not add many extra savings to the algorithms that include multiple changes. Consequently, we can say that the algorithms with single changes provide relatively good results but the algorithms that include multiple changes provide even better results since they explore the solution space more extensively. Furthermore, this could also be seen from the rankings of the 653 algorithms based on the average costs on this dataset where the best single change algorithm is at the 128th rank. All in all, this shows us that the algorithms with multiple changes are performing better than the single change algorithms. This is also logical since we expected that multiple changes that worked well individually, would perform even better together.

Table 6.21: The results of using the 653 algorithms of the exploration and exploitation experiments in parallel compared to the algorithm of Company X, where the net savings are presented in proportions to the total planning costs while using the algorithm of Company X.

Nr Solutions to LS&RR	Diff algorithm of Company X (%)	Diff best (%)	Extra Calc Time (h)	Extra Calc Costs (€)	Net Savings (%)
1	-1.04	0.31	947.81	151.03	1.03
5	-1.14	0.22	956.33	152.39	1.13
10	-1.17	0.18	966.75	154.06	1.16
50	-1.28	0.07	1054.52	168.10	1.27
100	-1.30	0.05	1167.37	186.16	1.29
200	-1.32	0.03	1398.53	223.14	1.30
400	-1.33	0.02	1884.44	300.89	1.31
651	-1.35	0.00	2649.33	423.27	1.32

6.4.2 Starting solutions vs profit

In the previous experiments, we have seen that the profit increased by using more construction methods and starting solutions. This is illustrated by Figure 6.6, where we plotted the number of starting solutions against the net savings. Note that we first run all 653 construction methods and afterwards the x best starting solutions proceed to the local search and R&R. The arrow in the graph denotes the maximum value of the net savings and is located at 600 starting solutions. Consequently, it can be concluded that it is profitable for ORTEC to work with plenty of construction methods and starting solutions caused by the fact that the computational costs are very low compared to the costs of a daily planning of Company X. Furthermore, it can be obtained that the slope is the highest at the start of the graph. This means that the most savings are gained while using more than one starting solution that will proceed to the local search and R&R phase.

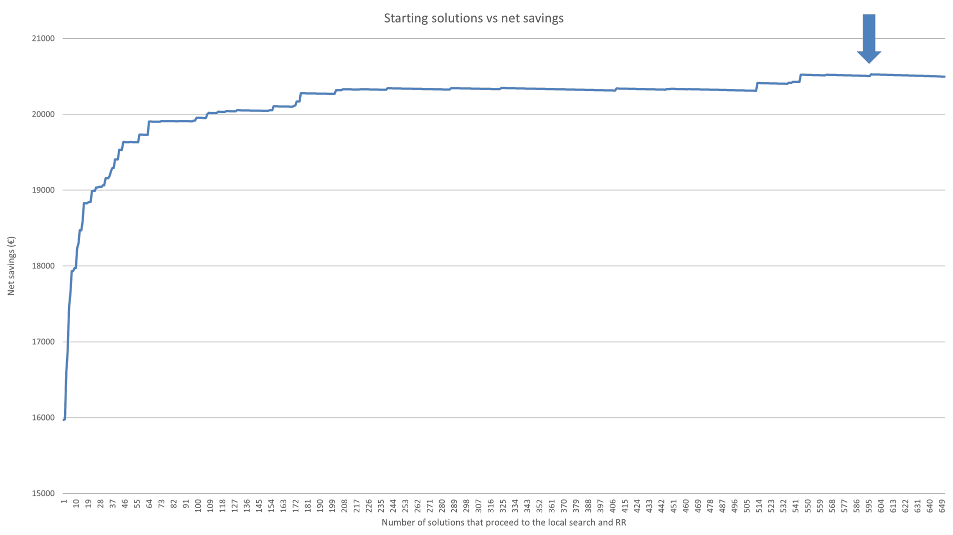


Figure 6.6: Number of starting solutions that will proceed to the local search and R&R phase plotted against the net savings where the arrow points out the maximum net savings

6.4.3 Country A vs Country B cases

We have seen in Section 6.1 and 6.2 that the algorithm of Company X performs poorly for the Country B cases since very many different algorithms outperformed it. In this subsection, we are going to dive deeper into this observation. In Subsection 6.2.6, we have seen that the average savings for the Country B cases are 3.18% and for the Country A cases 0.95%. Furthermore, we saw that the initial solutions are less important for the Country B cases than for the Country A cases because they profit a lot more from randomness. Since they are that much smaller, a relatively larger part of the solution space can be explored by the algorithms. Moreover, the difference between the Country B and Country A cases could also be obtained in the algorithms that work well for both cases. The 10 best algorithms on average for the Country A cases have an average ranking of 318.1 for the Country B cases and vice versa the 10 best algorithms on average for the Country B cases have an average ranking of 263.9 for the Country A cases. This means that the algorithms that perform well for the Country B cases, do not perform well for the Country A cases and vice versa. Consequently, we can conclude that the cases of Country B and Country A differ too much from each other and therefore different (parallel) algorithms should be used on each.

The algorithm of Company X is constructed for the main depots located in Country A and is afterwards also applied to the Country B cases. This means that if we construct a new parallel algorithm for the Country B cases too, we will not get a good perspective of the effect of parallel computing since savings are also gained from the fact that the algorithm of Company X was not constructed for the Country B cases and does not perform well on these cases. As a result, we are going to design and test our new parallel algorithm against the algorithm of Company X only for the Country A cases such that the conclusions of our research are reliable.

6.5 Construction phase of new parallel algorithm

In this section, we are first constructing the construction phase of our new parallel algorithm in Subsection 6.5.1. Afterwards, its performance is evaluated against the algorithm of Company X and the effect of parallel computing in the construction phase is determined in Subsection 6.5.2.

6.5.1 Design of construction phase of new parallel algorithm

Ideally, we would want to use as many construction methods as possible and proceed with all starting solutions to the local search and RR phase as could be seen in Figure 6.6. However, this is not possible in practice since ORTEC does not have that many processors available. The new parallel algorithm is restricted to the use of 15 CPUs concurrently. Since we have seen in the previous experiments that it is very beneficial to use a lot of construction methods and starting solutions that proceed to the local search and R&R, we are going to use 15 different construction methods of which all starting solutions will proceed to the local search and RR in our new parallel algorithm. This means that n and p equal 15 for our new parallel algorithm in Figure 4.1. In order to select these 15 construction methods, we are going to use the same criteria as in Subsection 6.2.3 where we selected the parameter settings that were going to be used in the exploitation experiment. Namely, the average costs, best 3rd rank and the average of the best 25% rankings with respect to all the cases in the dataset from Country A. We have selected the top 15 of all categories and these are listed in Table A.22. Note that the names of the algorithms are constructed by merging the parameter settings of Table A.1 where a "_" is used as a separator.

It stands out that many algorithms in this list are included with both the 475 scale of order quantity and without. Remark that if a case does not have an order larger than 475, the solution of the two versions of the algorithms is exactly the same. Since only 8 out of the 30 cases in the dataset have orders that are larger than 475 and we are only allowed to use 15 different construction methods, we have decided that we will only include one of these two versions of an algorithm in the construction phase of our new parallel algorithm. In this way, we will have a varied set of starting solutions. Note that the dataset of the final experiment includes a whole week of data and that the proportion of cases with orders larger than 475 will be even smaller since most of the large orders are from Mondays. Furthermore, the algorithms including the 475 scale are on average better than the ones without but not per se for every single case. Consequently, not only the algorithms including the 475 scale of order quantity will be selected. In order to have a good balance between algorithms from these 3 criteria in the new parallel algorithm, we start by selecting the best algorithm per criteria. If an algorithm is selected, it is removed from the ranking as well as its 475 order quantity scale equivalent. This process is repeated 5 times such that we have selected 15 algorithms. In this way, we have a good mix of algorithms that perform well on average and algorithms that perform well for only a subset of cases. These 15 algorithms are bold in Table A.22 and will be included in the construction phase of our new parallel algorithm and will be tested on a large set of data in the final experiment in the next section.

6.5.2 Performance of new parallel algorithm

In this section, we are going to evaluate the performance of our new parallel algorithm that uses 15 different construction methods against the algorithm of Company X on a week of data from all the main depots located in Country A. This dataset consists of 108 cases. The results of this experiment are shown in Table 6.22. We can see that our new parallel algorithm clearly outperforms the algorithm of Company X on the two most important KPIs. First of all, our new proposed parallel algorithm is able to increase the number of planned customers by 0.004% and decrease the costs by 1.18%. Logically, our new proposed parallel algorithm uses significantly more calculation time than the algorithm of Company X. However, the corresponding extra calculation costs are negligible with respect to the total costs. Consequently, by taking into account as well the planning costs as the computational costs, the new parallel algorithm outperforms the algorithm of Company X still by 1.18%.

Every algorithm in the parallel algorithm provides at least three times the best solution for a case, implying that we have selected a well-varied set of algorithms for our new parallel algorithm. Furthermore, the average ranking after construction for the best final solution is 5.6 and in one case the worst starting solution ends up in the best final solution. This ratifies the choice of using all starting solutions to proceed to the local search en R&R phase.

An important note has to be made on the effects of parallel computing with respect to the total savings of 1.18% since some savings are down to the creation of new algorithms with the help of data analysis and experiments. The best algorithm out of the 653, logically also included in the new parallel algorithm, outperforms individually the algorithm of Company X by 0.66% in terms of costs. This means that we can conclude that the savings caused by the parallelism of the algorithm are 0.52%.

Table 6.22: Results final experiment where the percentile difference between our new parallel algorithm and the algorithm of Company X is presented for the most important KPIs.

Costs (%)	Planned customers (%)	Calculation time (%)	Calculation costs (%)	Total costs (%)
-1.18	0.004	1343.68	1343.68	-1.18

6.6 Modified Research design for the R&R phase

In this section, the changes made to the R&R are explained in Subsection 6.6.1. Subsequently, Subsection 6.6.2 explains the modifications made to our research design.

6.6.1 New RR

During our research, the R&R described in Section 3.2 was modified based on internal research within ORTEC to enhance its performance. Consequently, the R&R used for the experiments with multiple construction methods differs from the R&R used in experiments on parallel R&R methods in this chapter. This means that the R&R for which parameter settings will be modified also differs. The new R&R method is similar to the previous one which is visualised in the flowchart in Figure 3.4, however, the R&R methods included in the roulette wheel and the number of iterations and recursions have been changed.

Recall that the former R&R method used six ruin methods (Random Removal, Related Removal, Route Removal, Random Cluster Removal, Worst Removal, Worst Cluster Removal) in combination with one recreate method (Parallel Cheapest Insertion (PCI), totalling six R&R methods. The new roulette wheel includes eight R&R methods, featuring Random Removal, Related Removal, Route Removal, and the new Route and Adjacent Task Removal as removal methods where Regret Insertion (RI) and PCI procedures are used as recreate methods. Route and Adjacent Task Removal randomly removes a predetermined number of routes and adjacent tasks from the solution. Moreover, the number of recursions has been set to 2 and the number of iterations to 125, instead of the previous 5 recursions of 40 iterations.

We will modify the parameter settings of the following four categories: removal percentage of ruin methods, the ruin and recreate methods included in the roulette wheel, the estimator of the recreate method, and the combination of recursions and iterations.

6.6.2 Research design

Due to the limited time available for researching the R&R of the algorithm of Company X, we modified the research design described in Chapter 4. We will use the coordinate descent method to tune the parameters of the four categories. The coordinate descent method is simple but efficient because it breaks down complex multidimensional problems into simpler one-dimensional problems (Wright, 2015).

It optimizes one parameter at a time while keeping others constant where the optimized parameter is updated before moving to the next. This means that we will conduct screening and exploration experiments for the first parameter, select the most promising settings for the exploitation experiment, and then proceed to the next parameter using the best setting of the previous parameter. This process is repeated for all four categories. These are the only changes made to the research design described in Chapter 4.

6.7 RR experiments

In this section, we present all experiments related to the creation of a new R&R phase for our new parallel algorithm. First, the experimental setup is discussed in Subsection 6.7. Next, Subsection 6.7.1 covers the screening and exploration phase of Step 1 of the research design, with each subsection addressing one of the four categories: removal percentage of ruin methods, the ruin and recreate methods included in the roulette wheel, the estimator of the recreate method, and the combination of recursions and iterations. These categories are used in this sequence in our research design based on advice from experts within ORTEC. In Subsection 6.7.2, we combine the best parameter settings from these experiments to create effective R&R methods for the R&R phase of our new parallel algorithm. Subsequently, Subsection 6.7.3 executes experiments for Steps 2 and 3 of our research design, constructing the new R&R phase of our parallel algorithm. Finally, Subsection 6.7.4 compares the new R&R phase of our parallel algorithm to the R&R method of the algorithm of Company X and evaluates the complete parallel algorithm against the complete algorithm of Company X. Note that in all experiments except the final evaluation of our new complete parallel algorithm against the algorithm of Company X, only a single seed value is used due to time and budget restrictions.

Experimental set-up

In Step 1 of the research design, we will use a single starting solution per case for all R&R methods, as the primary objective is to develop good R&R methods. This starting solution will be generated using the construction and local search phase of the best individual algorithm identified in Chapter 6. The Country A cases from the dataset described in Chapter 5 will be used for all experiments except the final one, resulting in a training dataset of 30 cases. Additionally, we will remove the stopping criterion triggered when a complete recursion of the roulette wheel yields no improvement, ensuring each R&R method has the same number of iterations for fair comparisons. Lastly, the number of planned tasks will not be shown in the tables since they are zero, except for the final experiment.

6.7.1 Screening and exploration experiments RR

Removal percentage

In a previous study on the R&R method at ORTEC, Simons (2017) recommended using a removal percentage between 5% and 20%. Since the removal percentage must be an integer, we tested the following values: 5, 7, 10, 12, 15, 17, and 20. Note that 5% is the current removal percentage used for the algorithm of Company X. The Route and Adjacent Task Removal method in the roulette wheel operates with absolute numbers rather than percentages. We adjusted its parameter settings to approximate the correct number of tasks removed, using the average number of tasks in the dataset for calculations. The settings used per removal percentage are detailed in Table A.23.

The results of this experiment are shown in Table 6.23. It is notable that removal percentages greater than 7 do not improve the starting solution, while lower percentages rarely provide improvements.

Additionally, larger removal percentages increase calculation time, as more routes need to be recreated per iteration. Given the poor performance of the tested removal percentages and the relatively better results with smaller percentages, we decided to also test percentages below 5: specifically, 1, 2, 3, and 4.

Table 6.23: The results of various removal percentages used for the removal methods included in the roulette wheel of the algorithm of Company X where in the table the percentual difference between the solution before and after the R&R is presented for the costs, number of routes and trips. The "Nr of improvements" column denotes the number of times that the R&R was able to improve the starting solution. The last column shows the average calculation time of the R&R in seconds

Removal Percentage	Costs (%)	Routes (%)	Trips (%)	#Improved (Out of 30)	CalcTime (s)
5	-0.04	-0.06	-0.06	3	576.18
7	-0.02	-0.03	-0.03	1	660.28
10	0.00	0.00	0.00	0	797.96
12	0.00	0.00	0.00	0	879.89
15	0.00	0.00	0.00	0	995.31
17	0.00	0.00	0.00	0	1049.52
20	0.00	0.00	0.00	0	1135.61

Table 6.24 presents the results of experiments with smaller removal percentages. It can be observed that smaller percentages yield better solutions, with removal percentage values of 1% and 2% improving almost every starting solution. Since 1% provides the best results, it will be used for subsequent experiments. To identify the most promising settings for the exploitation phase, we examined which percentage yields the best results when run in parallel with 1%. It turns out that a removal percentage of 3% is the most complementary. Consequently, removal percentages of 1% and 3% will be used in the exploitation experiment.

Table 6.24: The results of various removal percentages used for the removal methods included in the roulette wheel of the algorithm of Company X where in the table the percentual difference between the solution before and after the R&R is presented for the costs, number of routes and trips. The "Nr of improvements" column denotes the number of times that the R&R was able to improve the starting solution. The last column shows the average calculation time of the R&R in seconds

Removal Percentage	Costs (%)	Routes (%)	Trips (%)	#Improved (Out of 30)	CalcTime (s)
1	-0.15	0.00	-0.25	29	437.26
2	-0.11	-0.03	-0.14	26	470.54
3	-0.09	-0.06	-0.14	14	499.44
4	-0.06	-0.06	-0.08	9	495.38
5	-0.04	-0.06	-0.06	3	576.18
7	-0.02	-0.03	-0.03	1	660.28
10	0.00	0.00	0.00	0	797.96
12	0.00	0.00	0.00	0	879.89
15	0.00	0.00	0.00	0	995.31
17	0.00	0.00	0.00	0	1049.52
20	0.00	0.00	0.00	0	1135.61

Removal and recreate methods

In this experiment, we are going to experiment with the removal and recreate methods included in the roulette wheel. As discussed in Subsection 6.6.1, the current roulette wheel includes four different removal methods and two recreate methods. While more methods are available in OHD, recent internal research indicated that adding them does not improve the performance of the R&R. Thus, we will only investigate using fewer removal and recreate methods. We have created all unique roulette wheels containing two and three removal methods and include two algorithms that use only one recreate method. Roulette wheels including a single removal method are excluded based on Simons (2017), which showed multiple removal methods are superior. Using 1% as removal percentage, the results in Table 6.25 indicate several modifications to the removal methods included in the current roulette wheel lead to better solutions. Notably, using only PCI as the recreate method improves results, whereas using only RI worsens them. For this reason, we have opted for further testing of all different removal method combinations in combination with PCI. These results are provided in Table 6.26 where we can see that most changes in removal methods lead to decreased performance. The best-performing roulette wheel excluded the related removal method and used both recreate methods, which will be used in subsequent experiments. Due to many improvements from changes to the R&R methods included in the roulette wheel, we will include four different roulette wheels in the exploitation phase. Evaluating all combinations of four in parallel is very time-consuming, so we select the two best-performing wheels on average and the two with the most top-three rankings. This approach ensures a varied set of roulette wheels: `NoRelated (PCI & RI)`,`Related-Route (PCI & RI)`,`Random-Adjacent (PCI)`, and `NoRandom (PCI)`, where changes in removal methods are noted outside the brackets and recreate methods used are in brackets. For example, `NoRelated (PCI & RI)` defines the roulette wheel including all four removal methods except for the Related Removal and both recreate methods: Parallel Cheapest Insertion (PCI) and Regret Insertion (RI)

Table 6.25: The results of various removal and recreate methods included in the roulette wheel of the algorithm of Company X where in the table the percentual difference between the solution before and after the R&R is presented for the costs, number of routes and trips. The first column denotes the change to the roulette wheel methods of the algorithm of Company X. The "Nr of improvements" column denotes the number of times that the R&R was able to improve the starting solution. The last column shows the average calculation time of the R&R in seconds

RR change	Costs (%)	Routes (%)	Trips (%)	#Improved (Out of 30)	CalcTime (s)
NoRelated	-0.19	-0.06	-0.31	30	444.73
PCI	-0.18	0.00	-0.20	30	448.61
Random-Adjacent	-0.16	-0.06	-0.20	29	460.72
Random-Route	-0.16	0.00	-0.14	30	435.03
Route-Adjacent	-0.16	-0.03	-0.36	23	406.73
Related-Adjacent	-0.16	-0.03	-0.25	23	414.35
NoRoute	-0.16	0.00	-0.22	29	436.41
NoAdjacent	-0.15	-0.03	-0.28	29	418.05
No Change	-0.15	0.00	-0.25	29	426.79
RI	-0.14	0.00	-0.20	29	429.21
Random-Related	-0.14	0.00	-0.06	29	426.49
Related-Route	-0.14	0.00	-0.14	24	368.36
NoRandom	-0.13	0.00	-0.25	22	402.85

Table 6.26: The results of various removal methods included in the roulette wheel of the algorithm of Company X while using the parallel cheapest insertion as recreate method where in the table the percentual difference between the solution before and after the R&R is presented for the costs, number of routes and trips. The first column denotes the change to the removal methods in the roulette wheel of the algorithm of Company X. The "Nr of improvements" column denotes the number of times that the R&R was able to improve the starting solution. The last column shows the average calculation time of the R&R in seconds

Removal methods	Costs (%)	Routes (%)	Trips (%)	#Improved (Out of 30)	CalcTime (s)
Random-Adjacent	-0.18	0.00	-0.14	29	447.44
No Change	-0.18	0.00	-0.20	30	433.90
Related-Adjacent	-0.17	-0.03	-0.20	24	411.79
Route-Adjacent	-0.16	-0.06	-0.33	22	379.05
NoAdjacent	-0.16	-0.03	-0.20	30	421.80
NoRandom	-0.16	-0.03	-0.22	25	388.63
NoRelated	-0.15	0.00	-0.25	29	441.45
Random-Route	-0.15	-0.03	-0.20	30	403.10
Random-Related	-0.15	0.00	-0.14	29	444.45
Related-Route	-0.14	-0.03	-0.22	24	362.96
NoRoute	-0.11	0.00	-0.11	29	324.63

Estimators

In this subsection, we discuss the results of experiments using different estimators for the recreate methods used in the roulette wheel where the current estimator used is `DrivingTime`. The results, shown in Table 6.27, indicate that the `WaitTimeCosts` estimator significantly outperforms the others, at the expense of higher computational time. The `Costs`, `CostsTW`, and `Distance` estimators even deteriorate the solutions compared to the current estimator. For the next experiment, we will use the `WaitTimeCosts` estimator. Since the `DrivingTime` estimator provides the best results when used together in parallel with `WaitTimeCosts`, these two estimators are selected for the exploitation experiment.

Table 6.27: The results of using various estimators for the recreate methods included in the roulette wheel of the algorithm of Company X where in the table the percentual difference between the solution before and after the R&R is presented for the costs, number of routes and trips. The first column denotes the estimator used and the "Nr of improvements" column denotes the number of times that the R&R was able to improve the starting solution. The last column shows the average calculation time of the R&R in seconds

Estimator	Costs (%)	Routes (%)	Trips (%)	#Improved (Out of 30)	CalcTime (s)
WaitTimeCosts	-0.22	0.00	-0.17	30	604.00
DrivingTime	-0.19	-0.06	-0.31	30	450.01
Costs	-0.18	0.00	-0.31	30	476.35
CostsTW	-0.16	-0.03	-0.17	21	434.34
Distance	-0.14	0.00	-0.17	25	409.40

Number of iterations vs number of recursions

In Subsection 6.6.1, it is explained that the R&R of the algorithm of Company X uses a total of 250 iterations. Different configurations achieve this number due to the roulette wheel's learning layer with recursions. After each recursion, the probabilities for a R&R method to be selected are reset and a local search is executed. In this experiment, we tested various ways to reach 250 iterations.

The results, shown in Table 6.28, indicate that resetting probabilities and performing local searches after recursions benefit the R&R since the configurations with the smallest number of recursions provided the worst results on average. Additionally, a higher number of recursions increased calculation time, as local searches were performed more frequently. The 63x4 configuration provided the best solutions on average. When run in parallel with other configurations, 63x4 and 125x2 together provided the best results. Therefore, 63x4 and 125x2 are selected for the exploitation experiment.

Table 6.28: The results of using various combinations of recursions and number of iterations per recursions while using a total of 250 iterations for the roulette wheel of the algorithm of Company X. The percentual difference between the solution before and after the R&R is presented for the costs, number of routes and trips. The first column denotes the number of iterations and recursions used and the "Nr of improvements" column denotes the number of times that the R&R was able to improve the starting solution. The last column shows the average calculation time of the R&R in seconds

Iterations x Recursions	Costs (%)	Routes (%)	Trips (%)	#Improved (Out of 30)	CalcTime (s)
63x4	-0.230	0.00	-0.22	30	675.28
84x3	-0.225	-0.03	-0.20	30	663.29
50x5	-0.218	0.00	-0.20	30	703.48
125x2	-0.216	0.00	-0.17	30	632.69
250x1	-0.199	-0.03	-0.20	30	554.22

6.7.2 Exploitation experiment RR

In this experiment, we will create R&R methods by combining the selected parameter settings of the experiments in the previous subsection which are presented in Table 6.29. By combining all settings, we can create 32 unique R&R methods. Additionally, we included roulette wheels including removal methods with both 1 and 3 as removal percentage values resulting in 48 different R&R methods. These methods are tested on the 30 Country A cases. Afterwards, the best 15 R&R methods are selected in the same manner as in 6.4 where we selected the construction methods based on the average performance, best 3rd rank and the average of the best 25% rankings. These 15 R&R methods are presented in Table A.24 and will be used in the next experiment to determine the optimal number of starting solutions and R&R methods to include in the R&R phase of our new parallel algorithm.

Table 6.29: Overview of all selected parameters settings per category that will be used in the exploitation experiment

Category	Parameter settings
Removal percentage (2)	1,3
Roulette wheel (4)	NoRelated (PCI & RI), Related-Route (PCI & RI), Random-Adjacent (PCI) and NoRandom (PCI)
Estimator (2)	WaitTimeCosts, DrivingTime
Iterations x Recursions (2)	63x4, 125x2

6.7.3 Creation of parallel RR

In this subsection, we are going to determine how many starting solutions and R&R methods will be used in the R&R phase of our new parallel algorithm such that we optimally make use of the 15 CPUs that we are allowed to use. We will test this by applying the construction and local search phase of our new parallel algorithm created in the previous chapter on the dataset consisting of 30 Country A cases. As a result, 450 starting solutions are created which need to go the R&R phase and the 15 best R&R methods determined in the previous section will be applied on these starting solutions. This will generate 225 final solutions per case and thus a total of 6750 final solutions. Afterwards, we are able to determine what the best combination of the number of starting solutions and R&R methods is. Since we are restricted to using 15 CPUs, we have the following options for the combination of the number of starting solutions and R&R methods: 1x15, 2x7, 3x5, 5x3, 7x2, 15x1. For the starting solutions, we will use the *x* best starting solutions since in Table A.25 can be seen that on many occasions the best starting solution also results in the best final solution. The average ranking of the starting solution of the best final solutions is 2.67. This indicates that in general the better the starting solution, the better the final solution. For this reason, we have decided to use the *x* best starting solutions. For the selection of the R&R methods, we will use the same selection procedure as in Subsection 6.7.2 where we will use the following sequence of importance of the categories: average performance, average of best 25% rankings and best 45th rank. Note that we now use the 45th rank since we now have 15 times as many starting solutions. The design of our new parallel algorithm including multiple R&R methods is visualised in Figure 6.7 where we want to find the optimal values for *r* and *k*.

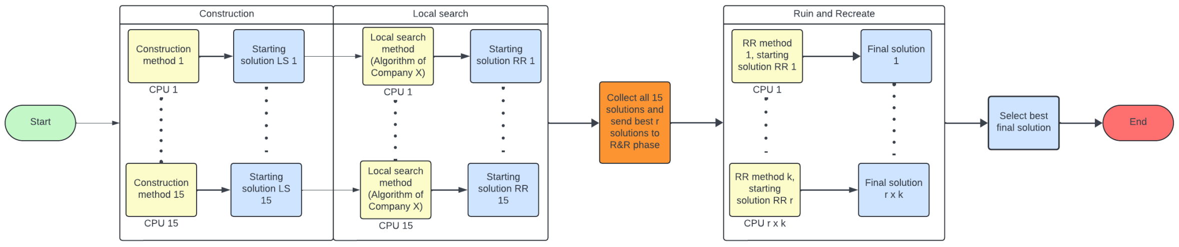


Figure 6.7: A visualization of the solution design used to address our problem

The outcomes of all combinations of the number of starting solutions and R&R methods are presented in Table 6.30. It can be concluded that the combination of three starting solutions and five R&R methods provides the best results. Consequently, the R&R phase of our new parallel algorithm will include five R&R methods which will all be applied to the three best solutions after local search. This means that in Figure 6.7 *r* equals 3 and *p* equals 5. This makes our new parallel algorithm an independent multi-search method using MPDS as search differentiation. In the next subsection, the R&R phase of our new parallel algorithm will be tested on a larger dataset to evaluate its performance against the R&R method of the algorithm of Company X. On top of that, our new complete parallel algorithm including multiple construction- and R&R methods will be evaluated against the algorithm of Company X.

Table 6.30: Comparison of the different combinations of the number of starting solutions and R&R methods where their performances are compared with respect to the R&R method of the algorithm of Company X

Starting solutions x RR methods	Difference with R&R of the algorithm of Company X (%)
15x1	-0.201
7x2	-0.238
5x3	-0.244
3x5	-0.252
2x7	-0.248
1x15	-0.249

6.7.4 Final experiment RR

In this experiment, we will first evaluate the R&R phase of our new parallel algorithm against the R&R phase of the algorithm of Company X where both make use of the 15 starting solutions created by the algorithm constructed in the previous chapter. In addition, we have also applied the best-found individual R&R method of the preceding experiments on the 15 starting solutions to determine what part of the improvement actually belongs to the use of multiple different R&R methods. This evaluation is conducted on a dataset that consists of a week of data of Company X for all main depots in Country A. The results of this experiment can be found in Table 6.31.

We can clearly see that our R&R method outperforms the R&R method of the algorithm of Company X by 0.24% in terms of costs. However, we can also see that most improvements come from the fact that we improved the parameter settings of the R&R method of Company X since the best-found individual R&R method outperforms the R&R method of the algorithm of Company X by 0.18% in terms of costs. Consequently, the use of multiple different R&R methods only caused a 0.06% in terms of costs and 0.001% in terms of planned tasks improvement over the best individual R&R method.

Table 6.31: Results of the experiment regarding the comparison of the new R&R phase of our new parallel algorithm against the R&R method of the algorithm of Company X and the best individual R&R method. In the table, the percentual difference with respect to the R&R method of the algorithm of Company X per KPI is presented.

Algorithm	Planned tasks (%)	Costs (%)	CalcTime (%)
New RR phase	-0.001	-0.24	11.84
Best individual RR	-0.002	-0.18	6.88

We have also evaluated our new complete parallel algorithm including multiple construction and R&R methods, visualised in Figure 6.8, against the complete algorithm of Company X which is explained in Chapter 3. For this final evaluation, we used five different seed values to ensure that our observed improvements are not attributable to randomness. The average outcomes of the five runs are presented in Table 6.32 and it stands out that our new parallel algorithm outperforms the algorithm of Company X by 1.56% in terms of costs. Consequently, ORTEC can save Company X a significant amount a year by using our new parallel algorithm instead of the algorithm of Company X to generate all the routing schedules. However, our new parallel algorithm performs slightly worse on the number of planned tasks, namely 0.003%. This worse performance in the KPI of the number of planned tasks is caused by one case in the dataset where the solution of our new parallel algorithm includes significantly fewer unplanned tasks than the solution of the algorithm of Company X. For all other 107 cases in the dataset, our new parallel algorithm outperforms the algorithm of Company X for both KPIs.

However, our new parallel algorithm violates the time limit of 20 minutes in 41 out of the 108 cases. This is mainly caused by the fact that we removed the stopping criterion that is used when a recursion of a roulette wheel without improvement is executed. We have removed this stopping criterion in order to make fair comparisons between the roulette wheels that have the same number of total iterations but differ in the number of recursions. Note that the algorithm is not stopped after 20 minutes since we have run the first two phases, construction and local search, separately from the R&R phase. Furthermore, the most and largest improvements are made in the first iterations of the R&R method thus we believe that time limit violations do not have a very large impact on the result of the best final solution.

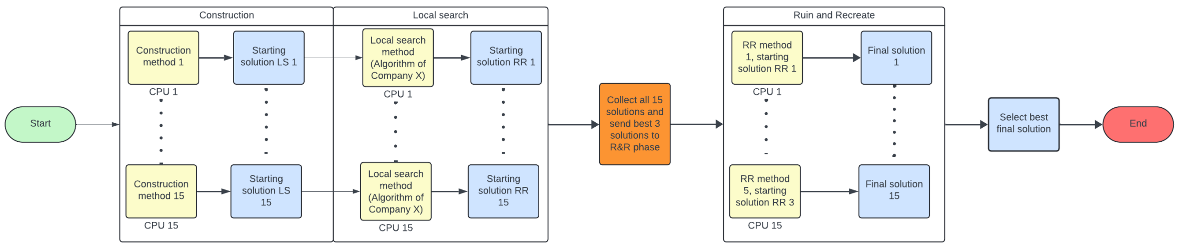


Figure 6.8: A visualization of the design of our new parallel algorithm including multiple construction and R&R methods.

Table 6.32: Results of the final experiment where our new complete parallel algorithm is evaluated against the algorithm of Company X on a full week of data of Company X where we have used 5 different seed values. The percentile difference compared to the algorithm of Company X of the average of these 5 runs is presented for each KPI in the table.

Planned tasks (%)	**Costs (%)**	**Calculation time (%)**	**Calculation costs (%)**	**Total Costs (%)**
-0.003	-1.56	2796.49	2796.49	-1.56

Chapter 7

Conclusions, recommendations and further research

This final chapter concludes the research conducted in the previous chapters where in Section 7.1 the most important conclusions are presented. Recommendations for ORTEC are provided in Section 7.2 whereas recommendations for further research are provided in Section 7.3.

7.1 Conclusions

This section includes a recap of the key findings from our research conducted to answer the following main research question stated in Section 1.3:

To what extent can parallel computing improve the performance of the OHD software?

To answer this question, the sub-research questions formulated in Section 1.5 were answered and the data of Company X is used. Currently, the algorithm of Company X is solving the problem of Company X within OHD and consists of three phases: construction, local search and ruin and recreate (R&R). In the literature research in Chapter 2, multiple variants and solving methods of the VRP are reviewed however the variant of Company X, Vehicle Routing Problem of Company X (VRPX), is novel to the best of our knowledge. Furthermore, we introduced several parallel computing approaches for solving VRPs of which the high-level parallelism method Multiple initial Points/Populations, Different search Strategies (MPDS) is used in this research to develop a new parallel algorithm. The multiple different starting solutions and search strategies are created by adjusting the parameter settings of the algorithm of Company X. Consequently, our problem became a variant of a well-known problem in the literature, namely the algorithm configuration problem. Automated methods typically solve this problem, but they were unsuitable for our research due to the vast number of parameter settings and the lack of information about the working of a parameter setting.

Solution design

To address our problem, we created a solution design consisting of 3 steps. In Step 1, the goal is to create good algorithms that work well for various types of cases of Company X such that they can be used in the new parallel algorithm. Subsequently, we will investigate whether using multiple construction methods and starting solutions is beneficial in Step 2, and if so, determine the optimal number of construction methods and starting solutions that will proceed to the local search and R&R phase. Finally, in Step 3, the algorithms included in our new parallel algorithm will be selected.

To create good algorithms in Step 1, we used a modified version of the framework of Gunawan et al. (2011) which consists of three phases: screening, exploration, and exploitation. Firstly, in the screening phase, we reduced the number of parameter settings by identifying the least promising parameter settings with the help of experts from ORTEC. Moreover, we have conducted a data analysis on the cases of Company X in Chapter 5 to detect difficult tasks that could fit as a seed task and to find possible correlations between the parameter settings and data characteristics. In the data analysis, we have seen a lot of variety across the cases in the dataset in terms of the number of customers, order quantities and distances between customers and depots. This variety indicated the potential benefit of a parallel algorithm, as in the experiments of this phase different parameter settings performed well for different types of cases.

In these experiments, we only modified a single parameter setting of the algorithm of Company X to observe its effects and detect promising ranges. Subsequently, in the exploration experiment, the solution space of the promising parameter setting ranges is examined to find the best setting(s) per parameter. It is important to note that in our context, "best" does not always mean the best on average. Instead, it could also refer to the best for particular cases since the main advantage of parallel computing is its ability to leverage different algorithms simultaneously. This way, if one algorithm produces an exceptionally good solution for a specific case, that solution can be utilized. If an algorithm's solution is mediocre, it can be compensated by a better solution from another algorithm running in parallel. In this way, multiple settings per parameter could be selected. Afterwards, in the exploitation experiment, new algorithms were created by combining these best parameter settings to develop good algorithms that could be used in the new parallel algorithm.

Experimental results construction phase

From these experiments, several insights on the parameter settings of the construction method of the algorithm of Company X were gained such as that the smaller batch sizes enhance better solutions than the larger batch sizes and that there is a correlation between the scale of the first sorter of the insertion method and the customer density of a case. In Chapter 6, relevant observations about the usage of certain parameters are provided however more research is needed for definitive conclusions. Furthermore, we have seen in the data analysis that the Country B cases differ significantly from the Country A cases which caused different sorts of algorithms to perform well for the Country A cases than for the Country B cases. As a consequence, it can be concluded that if cases differ significantly from each other parallel computing is not the appropriate solution, however, different (parallel) algorithms should be designed and used. Since the algorithm of Company X was originally only created for Country A cases and the Country B cases were added later without changing the algorithm, the performance of the algorithm of Company X for the Country B cases is relatively poor. As a consequence, some part of the improvements of our new parallel algorithm would be because of the fact that the algorithm of Company X was not made for the Country B cases if we use both Country B and Country A cases in our testing dataset. For this reason, we have concluded to construct a new parallel algorithm solely for Country A cases to make sure the results of our research are reliable and valid.

Up to this stage, we have constructed a total of 653 different algorithms by modifying parameter settings in Step 1 which we all run in parallel to observe the effect of using multiple construction methods and starting solutions that proceed to the local search and R&R phases. From this experiment, we obtained that proceeding with only the best starting solution to the local search and R&R can improve the solution quality in terms of costs by up to 1.04%. On top of that, we observed that in case more than just one starting solution that proceeds to the local search and R&R is used, the solution quality improved further, with a 1.34% improvement when all starting solutions are used. Consequently, we can conclude that the use of many construction methods as well as the use of multiple starting solutions that will proceed to the local search and R&R phase is very beneficial mainly because of the extremely low hourly computational costs assigned to a single Central Processing Unit (CPU) in relation with the relatively high routing costs of Company X. As a result, the more construction methods and starting solutions proceeding to the local search and R&R, the better the objective value which includes the extra computational costs. Besides, this proves that the best initial solution does not always result in the best final solution. However, we saw that the initial solution is important for the cases of Company X since the very bad initial solutions also result in poor final solutions, where we saw that the larger the VRP, the more important the initial solution.

Ideally, we would design a large parallel algorithm since this resulted in a good performance in our experiments. However, the number of CPUs available for the new parallel algorithm was restricted to 15 by ORTEC due to the current contractual constraints with the cloud computing platform. Consequently, we have developed a new parallel algorithm that uses 15 different construction methods of which all starting solutions will proceed to the same local search and R&R. Our parallel algorithm includes algorithms that perform well on average and algorithms that only perform well for specific types of cases. In this way, we have selected a varied set of algorithms for our parallel algorithm, ensuring robustness across the varied set of cases of Company X.

Our new parallel algorithm using multiple construction methods has been evaluated on a full week of Company X's data against the algorithm of Company X and it increases the number of planned tasks by 0.004% and decreases the costs by 1.18% which are the most important Key Performance Indicators (KPIs) for Company X. Note that in these costs also the extra computational costs are included. As a result of the data analysis and the experiments, we created an algorithm that individually outperformed the algorithm of Company X by 0.66% in terms of costs. This means that from the total improvement of 1.18%, 0.66% is caused by the updated parameter settings. Consequently, the implementation of parallel computing by using multiple construction methods and starting solutions proceeding to the local search and R&R phase caused an improvement of 0.54%.

Experimental results R&R phase

Subsequently, due to the limited time, we have applied a slightly modified version of our research design to the R&R phase of the algorithm of Company X aiming to improve it even more. We have used the coordinate descent method to optimize the parameter settings implying that we first optimize one parameter and afterwards continue to the next parameter while keeping the best parameter setting of the previously optimized parameter setting(s). Similar to the experiments of the construction phase, some observations about the parameter settings could be made such as that the smaller the removal percentages, the better the solutions and the smaller the computational time. By combing the best-found parameter settings of these experiments, 48 unique R&R methods were created and the best 15 were selected to be used in the next experiment which will evaluate the number of starting solutions and R&R methods to include in the R&R phase of our new parallel algorithm. To do so, we have applied these 15 R&R methods on all starting solutions created by the 15 different construction methods on the dataset of the Country A cases. Note that local search is also applied to these starting solutions. In this experiment, we found out that on many occasions the best starting solution also results in the best final solution. Furthermore, we found out based on this experiment that it is optimal to use the best three starting solutions in combination with five different R&R methods. To evaluate the use of multiple R&R methods, this new parallel R&R phase of our parallel algorithm is applied to the starting solutions of a week of data of Company X resulting from the solutions after local search from the 15 different construction methods. The outcomes are compared against the current R&R method of the algorithm of Company X and to the best-found individual R&R method. From this experiment, it can be concluded that the new R&R phase of our parallel algorithm outperforms the R&R method of the algorithm of Company X by 0.24% in terms of costs. However, the gap with the best individual R&R method is only 0.06%. This means that most improvements of the new R&R phase of our parallel algorithm phase are down to the modified settings instead of the use of multiple different R&R methods.

Performance of new parallel algorithm

This resulted in our new parallel algorithm that uses multiple construction- and R&R methods which is evaluated against the complete algorithm of Company X on a full week of Country A cases with five different seed values. From the results, it could be obtained that our new parallel algorithm outperformed the algorithm of Company X in terms of costs by 1.57%. However, the algorithm of Company X slightly outperforms our new parallel algorithm in terms of planned tasks by 0.003%. This relatively worse performance in the KPI of planned tasks is caused by a single case in the dataset where the algorithm of Company X finds a significantly better solution than our new parallel algorithm. In all other 107 cases in this dataset containing a full week of data of Company X, our new parallel algorithm clearly outperforms the algorithm of Company X in terms of both costs and the number of planned tasks. Our new parallel algorithm violates the running time in 41 out of the 108 cases mostly due to the removal of the stopping criterion related to a recursion of a roulette wheel without improvement. Since most improvements in the R&R phase are made in the beginning, we believe that a strict cut-off after 20 minutes does not have a large effect on the solutions. Above that, the largest improvements are gained from the use of multiple construction methods. All in all, we can conclude that parallel computing is very beneficial for ORTEC and can improve the performance of OHD significantly.

7.2 Recommendations

The first and most important recommendation to ORTEC is to implement our new parallel algorithm for the Country A cases of Company X, as it significantly outperforms the algorithm of Company X.

However, an agreement must first be reached with Company X regarding the distribution of extra savings and computational costs.

Secondly, we recommend applying our research design to serial algorithms of other customers to transform them into parallel versions. Our research demonstrated its effectiveness for the algorithm of Company X, and we believe it will yield similar benefits for other algorithms. However, given the limited time availability of ORTEC employees compared to our research, we propose a modified version of our research design to reduce the implementation time of new parallel algorithms. We suggest in the construction phase to focus only on experimenting with the parameter settings of the first sorter both the seed task selection and insertion method, as well as the insertion procedure combined with the number of batches and batch size, since these categories showed the best performance. For the R&R phase, we recommend using our modified research design used in Sections 6.6 and 6.7. Additionally, we observed that a parallel algorithm including solely algorithms that are based on a modification of a single parameter setting of the algorithm of Company X already yields good results. As a result, the development of a new parallel algorithm can be further expedited by using algorithms that incorporate only a single change. This approach skips experiments involving combinations of parameter changes, thus reducing the overall construction time of a parallel algorithm.

Moreover, while developing our parallel algorithm, we created new algorithms that individually outperformed the algorithm of Company X. The best construction algorithm achieved a 0.66% improvement over the algorithm of Company X whereas the best R&R algorithm had a 0.18% improvement. The parameter settings for the algorithm of Company X were based on test cases provided at the project's inception a few years ago. Therefore, we recommend that ORTEC update parameter settings of algorithms regularly, as data can change over time. This practice would be especially beneficial if real data becomes available post-implementation, providing comprehensive data for every main depot far beyond the initial test cases. Additionally, conducting a data analysis to determine the optimal parameter settings, as demonstrated in Chapter 5, would be highly advantageous.

Furthermore, our experiments revealed significant differences between the Country B and Country A cases, with different types of algorithms performing well for each. Notably, the algorithm of Company X performs poorly in the Country B cases. Therefore, we recommend that ORTEC develops a new (parallel) algorithm specifically for the Country B cases of Company X.

Finally, we recommend that ORTEC adopts the strategy of adding a sorter to the seed task selection method of the algorithm of Company X, which prioritizes the largest orders as seed tasks. This strategy can also be applied to other data characteristics, such as the start or length of time windows. By doing so, tasks with specific challenging data characteristics can be prioritized as seed tasks before using the standard seed task selection method. This approach allows the seed task strategy to consider multiple data characteristics simultaneously, rather than being fixated on a single one.

7.3 Further research

In this research, we have demonstrated that parallel computing within OHD is highly beneficial, though our study focused on utilizing multiple construction- and R&R methods. Therefore, our primary suggestion for further research is to explore the parallel use of various local search procedures to further enhance the solutions.

Secondly, in our research, we implemented the independent multi-search variant Multiple initial Points/Populations, Different search Strategies (MPDS). However, the parallel algorithm could benefit from a cooperative multi-search approach, where processors communicate with each other. This method has shown promising results in the literature. Therefore, we recommend investigating the use of a cooperative parallel algorithm, particularly for the local search and R&R phases of the algorithms.

Moreover, we found that using a large number of different construction methods remains beneficial due to the very low computational costs. However, we were limited to using only 15 CPUs because the current contract with the cloud computing platform does not permit more. Therefore, our third recommendation for further research to ORTEC is to investigate the potential benefits of scaling up the number of processors to allow more CPUs to be utilized in parallel algorithms.

Additionally, in our research design, we used a one-exchange neighbourhood, modifying only a single parameter setting at a time to observe its effect on solution quality without considering parameter interactions. Therefore, our research framework could be extended by examining the interactions between parameters as well.

Furthermore, we have excluded the use of different seed values for generating various R&R methods in this research. The difference between using only the best-found R&R method and multiple R&R methods was just 0.06%. Preliminary tests experimenting with seed values showed promising results. Therefore, we recommend ORTEC further investigate the option of using R&R methods that vary only by seed value and potentially in combination with multiple starting solutions.

Finally, we observed some correlation between scale values and data characteristics in the cases of Company X. However, it is too early to draw definitive conclusions. Therefore, our final recommendation for further research to ORTEC is to test this correlation on a larger dataset, including cases from other customers. This will help reduce the time needed to construct an algorithm for new customers.

Appendix A

Large tables Chapter 6

This appendix includes all tables that were too large to present in Chapter 6.

Table A.1: An overview of the different parameter settings used in the exploration experiment where the first column denotes the parameter and the second column the corresponding parameter settings that are tested. The new strategies for the seed task selection method introduced in the screening experiment are listed in the top rows. The rows below denote all parameters and sorters present in the Algorithm of Company X where the settings of the algorithm of Company X are presented in bold. The last column introduces the name of the algorithm for the specific parameter settings which will be used in the following sections

Parameter	**Settings**	**Example algorithm name**
New strategies		
Scale first sorter order quantity	190, 285, 380, 475	SeedTaskQuantityScale475
SeedFarAway	Distance and Driving time	SeedFarAwayDistance
Algorithm of Company X		
Scale first sorter seed task selection method	0, 2.5, 5, 7.5, **10**, 15, 20, 40, 60 kilometers and 0, 1, 5, 10, 30, 60 minutes	SeedTaskFarToNearDiffDepot10Min
Scale second sorter seed task selection method (Distance to selected depot)	**0**, 1, 2.5, 5, 7.5 kilometers and 0, 1, 2.5, 5, 10 minutes (**decreasing** and increasing)	SeedTaskFarToNearNearestAddress1KMIncreasing
Batch size x number of batches	**1x40**, 2x20, 3x14, 4x10, 5x8, 6x7, 7x6, 8x5, 10x4, 14x3, 20x2, 40x1	CloseToSeedBatch8x5 (task id), CloseToSeedBatch8x5SortingBatches (distance), CloseToSeed8x5ParallelInsertion
Insertion procedure	**Cheapest insertion (task id)**, cheapest insertion (distance) and Parallel cheapest insertion	
Estimator of insertion procedure	**CostAndTimeWindowOrdering**, Distance, Driving time, Costs and WaitTimeAndCosts	CloseToSeedCheapestInsertionCosts
Scale first sorter of insertion method	0.1, 0.25, 0.5, 0.75, **1**, 1.25, 1.5, 1.75, 2, 2.5, 3, 4, 5, 7.5, 10 kilometers and 0, 0.5, 1, 1.5, 2, 2.5, 5, 7.5, 10 minutes	CloseToSeed2KM
Scale second sorter insertion method (TW Start)	0, 20, **60**, 80, 120 minutes	CloseToSeedStartTW20Min

Table A.2: An overview of the average costs for each algorithm used in the exploration experiment. The table is sorted increasingly. This table includes confidential data, therefore, the data is removed from the table.

Algorithm	**Costs (€)**

Table A.3: An overview of the average of the best 25% Rankings of the algorithms used in the exploration experiment where the table is presented increasingly

Algorithm	Average Top 25% Ranking
SeedFarAwayDistance	2.6
CloseToSeedBatch3x14	3.9
CloseToSeedBatch2x20	4.2
SeedTaskFarToNearDiffDepot2,5KM	4.3
CloseToSeed2x20ParallelInsertion	5.2

Continued on next page

Table A.3 – continued from previous page

Algorithm	Average Top 25% Ranking
CloseToSeed5x8ParallelInsertion	5.4
SeedTaskFarToNearDiffDepot40KM	6
CloseToSeed1x40ParallelInsertion	6.1
SeedTaskFarToNearDiffDepot60KM	6.6
CloseToSeed0,1KM	6.8
CloseToSeed4x10ParallelInsertion	6.8
CloseToSeedBatch1x40ResortTrue	7.1
CloseToSeed3x14ParallelInsertion	8
SeedTaskFarToNearDiffDepot30Min	8
CloseToSeedBatch3x14SortingBatches	8.1
SeedFarAwayDrivingTime	8.1
CloseToSeedBatch7x6	8.5
SeedTaskFarToNearDiffDepot60Min	8.5
CloseToSeedBatch6x7SortingBatches	8.6
CloseToSeed0,25KM	8.9
CloseToSeed6x7ParallelInsertion	9
CloseToSeedBatch5x8SortingBatches	9.1
SeedTaskFarToNearDiffDepot5Min	9.4
CloseToSeedBatch5x8	9.5
SeedTaskFarToNearDiffDepot20KM	9.5
CloseToSeedBatch4x10SortingBatches	9.8
CloseToSeed1,5KM	10.2
CloseToSeedBatch2x20SortingBatches	10.2
SeedTaskFarToNearNearestAddress1KMDecreasing	10.2
CloseToSeed1,25KM	10.3
SeedTaskFarToNearDiffDepot7,5KM	10.4
CloseToSeed7x6ParallelInsertion	10.5
SeedTaskFarToNearDiffDepot5KM	10.7
CloseToSeed14x3ParallelInsertion	11.1
CloseToSeed8x5ParallelInsertion	11.3
CloseToSeed0KM	11.7
SeedTaskFarToNearNearestAddress1MinDecreasing	11.9
SeedTaskFarToNearDiffDepot0Min	12.3
SeedTaskFarToNearNearestAddress1MinIncreasing	12.5
CloseToSeedBatch4x10	12.6
SeedTaskFarToNearNearestAddress2,5MinDecreasing	12.6
SeedTaskFarToNearDiffDepot0KM	12.9
CloseToSeedBatch7x6SortingBatches	13.2
CloseToSeedBatch8x5SortingBatches	13.3
SeedTaskFarToNearDiffDepot1Min	13.3
SeedTaskFarToNearDiffDepot15KM	13.9
SeedTaskFarToNearDiffDepot10Min	14.1
CloseToSeed1,5Min	14.2
SeedTaskFarToNearNearestAddress0MinIncreasing	14.2
CloseToSeed0Min	14.6
SeedTaskFarToNearNearestAddress5KMDecreasing	14.6
CloseToSeed0,5KM	14.7
CloseToSeed0,75KM	14.9
CloseToSeed2,5Min	15
CloseToSeedBatch6x7	15
CloseToSeedBatch10x4SortingBatches	15.2
SeedTaskFarToNearNearestAddress10MinDecreasing	15.6
CloseToSeed1Min	15.7
CloseToSeed0,5Min	15.8
CloseToSeedBatch14x3SortingBatches	15.9

Continued on next page

Table A.3 – continued from previous page

Algorithm	Average Top 25% Ranking
CloseToSeed10x4ParallelInsertion	16.2
CloseToSeedStartTW80Min	16.8
SeedTaskFarToNearNearestAddress1KMIncreasing	16.8
SeedTaskFarToNearNearestAddress10MinIncreasing	17.4
SeedTaskFarToNearNearestAddress0KMIncreasing	17.7
SeedTaskFarToNearNearestAddress2,5KMDecreasing	18
SeedTaskFarToNearNearestAddress0MinDecreasing	18.1
CloseToSeed3KM	18.2
CloseToSeedCheapestInsertionDistance	18.3
SeedTaskFarToNearNearestAddress2,5KMIncreasing	18.3
CloseToSeedCheapestInsertionWaitTimeAndCost	18.4
SeedTaskFarToNearNearestAddress2,5MinIncreasing	18.9
SeedTaskFarToNearNearestAddress7,5KMDecreasing	19.1
SeedTaskFarToNearNearestAddress5MinDecreasing	19.4
CloseToSeed2KM	19.6
CloseToSeed2Min	20
CloseToSeedBatch10x4	20.9
CloseToSeed1,75KM	22.3
CloseToSeedStartTW0Min	23.2
SeedTaskQuantityScale190	23.8
CloseToSeedStartTW120Min	24.1
CloseToSeedBatch8x5	24.3
CloseToSeedStartTW20Min	24.4
CloseToSeed40x1ParallelInsertion	24.5
CloseToSeed20x2ParallelInsertion	24.6
SeedTaskFarToNearTWDurationIncreasingScale0	25.2
Algorithm of Company X	25.6
SeedTaskQuantityScale475	25.7
SeedTaskFarToNearNearestAddress5KMIncreasing	26.6
SeedTaskFarToNearNearestAddress7,5KMincreasing	26.7
SeedTaskQuantityScale380	26.7
CloseToSeedBatch20x2SortingBatches	27.2
CloseToSeed2,5KM	27.8
CloseToSeedCheapestInsertionCosts	28.4
CloseToSeedBatch14x3	29
CloseToSeed4KM	29.2
SeedTaskFarToNearNearestAddress5MinIncreasing	29.8
CloseToSeed7,5Min	30.5
SeedTaskQuantityScale285	31.4
CloseToSeed5Min	37.3
CloseToSeedCheapestInsertionDrivingTime	37.7
CloseToSeedBatch20x2	43.9
PlanSeedTaskFarToNearQuantityDecreasingScale0	44
CloseToSeedBatch40x1SortingBatches	45.4
CloseToSeed5KM	48.9
CloseToSeed7,5KM	54.3
CloseToSeed10KM	59.2
CloseToSeedQuantityDecreasingScale0	63.1
CloseToSeed10Min	65.6
CloseToSeedBatch40x1	66.2
PlanTasksCloseToSeedTaskIDSorting	73.1

Table A.4: Increasing list of the best 3rd Rank of the algorithms used in the exploration experiment

Algorithm	Best 3rd Rank
SeedFarAwayDistance	1
CloseToSeed2x20ParallelInsertion	1
CloseToSeed5x8ParallelInsertion	1
CloseToSeedBatch3x14	2
CloseToSeedBatch2x20	2
CloseToSeedBatch3x14SortingBatches	2
SeedTaskFarToNearDiffDepot2,5KM	3
SeedTaskFarToNearDiffDepot40KM	3
CloseToSeedBatch6x7SortingBatches	3
CloseToSeed1x40ParallelInsertion	4
SeedTaskFarToNearDiffDepot60KM	4
CloseToSeed4x10ParallelInsertion	4
SeedTaskFarToNearDiffDepot30Min	4
SeedFarAwayDrivingTime	4
SeedTaskFarToNearDiffDepot60Min	4
CloseToSeed0,25KM	4
CloseToSeed6x7ParallelInsertion	4
CloseToSeed0KM	4
SeedTaskFarToNearNearestAddress0MinIncreasing	4
SeedTaskFarToNearNearestAddress5KMDecreasing	4
CloseToSeed0,1KM	5
CloseToSeedBatch1x40ResortTrue	5
CloseToSeedBatch5x8SortingBatches	5
SeedTaskFarToNearDiffDepot5Min	5
CloseToSeed1,5KM	5
CloseToSeedBatch2x20SortingBatches	5
CloseToSeed8x5ParallelInsertion	5
SeedTaskFarToNearNearestAddress1MinIncreasing	5
SeedTaskFarToNearNearestAddress2,5MinDecreasing	5
CloseToSeedCheapestInsertionDistance	5
CloseToSeed3x14ParallelInsertion	6
CloseToSeedBatch4x10SortingBatches	6
SeedTaskFarToNearNearestAddress1KMDecreasing	6
CloseToSeed1,25KM	6
SeedTaskFarToNearDiffDepot7,5KM	6
SeedTaskFarToNearDiffDepot5KM	6
CloseToSeedBatch8x5SortingBatches	6
SeedTaskFarToNearDiffDepot1Min	6
CloseToSeedBatch14x3SortingBatches	6
CloseToSeedBatch7x6	7
SeedTaskFarToNearDiffDepot20KM	7
CloseToSeed7x6ParallelInsertion	7
SeedTaskFarToNearDiffDepot0Min	7
SeedTaskFarToNearDiffDepot15KM	7
CloseToSeed10x4ParallelInsertion	7
CloseToSeedCheapestInsertionWaitTimeAndCost	7
CloseToSeedBatch5x8	8
CloseToSeed14x3ParallelInsertion	8
CloseToSeedBatch4x10	8
SeedTaskFarToNearDiffDepot10Min	8
SeedTaskFarToNearNearestAddress10MinIncreasing	8
SeedTaskFarToNearNearestAddress2,5KMIncreasing	8
SeedTaskFarToNearTWDurationIncreasingScale0	8

Continued on next page

Table A.4 – continued from previous page

Algorithm	Best 3rd Rank
CloseToSeed2,5KM	8
SeedTaskFarToNearNearestAddress1MinDecreasing	9
SeedTaskFarToNearDiffDepot0KM	9
CloseToSeed0Min	9
SeedTaskFarToNearNearestAddress10MinDecreasing	9
CloseToSeed1Min	9
CloseToSeed0,5Min	9
CloseToSeedStartTW80Min	9
SeedTaskFarToNearNearestAddress0KMIncreasing	9
CloseToSeed2KM	9
CloseToSeedCheapestInsertionCosts	9
CloseToSeedBatch7x6SortingBatches	10
CloseToSeed1,5Min	10
CloseToSeed0,75KM	10
CloseToSeed2,5Min	10
CloseToSeedBatch6x7	10
SeedTaskFarToNearNearestAddress1KMIncreasing	11
CloseToSeed0,5KM	12
CloseToSeed3KM	12
SeedTaskQuantityScale190	12
CloseToSeed20x2ParallelInsertion	12
CloseToSeedBatch20x2SortingBatches	12
CloseToSeedBatch10x4SortingBatches	13
SeedTaskFarToNearNearestAddress2,5KMDecreasing	13
SeedTaskFarToNearNearestAddress7,5KMDecreasing	13
CloseToSeedBatch10x4	13
CloseToSeed1,75KM	14
CloseToSeedStartTW0Min	14
CloseToSeedBatch8x5	14
CloseToSeed2Min	15
CloseToSeedStartTW20Min	15
CloseToSeedBatch40x1SortingBatches	15
SeedTaskFarToNearNearestAddress0MinDecreasing	16
SeedTaskFarToNearNearestAddress5MinDecreasing	16
Algorithm of Company X	16
SeedTaskFarToNearNearestAddress2,5MinIncreasing	17
CloseToSeed4KM	17
CloseToSeed5Min	17
CloseToSeedStartTW120Min	18
SeedTaskQuantityScale475	18
CloseToSeedBatch14x3	18
SeedTaskFarToNearNearestAddress5KMIncreasing	19
CloseToSeed7,5Min	19
CloseToSeed40x1ParallelInsertion	21
SeedTaskFarToNearNearestAddress7,5KMincreasing	21
SeedTaskQuantityScale380	23
SeedTaskFarToNearNearestAddress5MinIncreasing	24
CloseToSeedBatch40x1	26
PlanSeedTaskFarToNearQuantityDecreasingScale0	28
SeedTaskQuantityScale285	29
CloseToSeedCheapestInsertionDrivingTime	31
CloseToSeedBatch20x2	36
CloseToSeed7,5KM	36
CloseToSeed5KM	37
CloseToSeed10KM	50

Continued on next page

Table A.4 – continued from previous page

Algorithm	Best 3rd Rank
CloseToSeed10Min	51
CloseToSeedQuantityDecreasingScale0	52
PlanTasksCloseToSeedTaskIDSorting	60

Table A.5: An overview of the average costs for each algorithm with a different seed task selection method used in the exploration experiment. The table is sorted increasingly. This table includes confidential data, therefore, the data is removed from the table.

Algorithm	**Costs (€)**

Table A.8: An overview of the average costs for each algorithm with a different insertion method used in the exploration experiment. The table is sorted increasingly. This table includes confidential data, therefore, the data is removed from the table.

Algorithm	**Costs (€)**

Table A.9: Increasing list of the best 3rd Rank of the algorithms with a different insertion method used in the exploration experiment

Algorithm	**Best 3rd rank**
CloseToSeed5x8ParallelInsertion	1
CloseToSeed2x20ParallelInsertion	1
CloseToSeedBatch3x14SortingBatches	2
CloseToSeedBatch3x14	2
CloseToSeedBatch2x20	2
CloseToSeedBatch6x7SortingBatches	3
CloseToSeed6x7ParallelInsertion	4
CloseToSeed4x10ParallelInsertion	4
CloseToSeed1x40ParallelInsertion	4
CloseToSeed0KM	4
CloseToSeed0,25KM	4
CloseToSeedCheapestInsertionDistance	5
CloseToSeedBatch5x8SortingBatches	5
CloseToSeedBatch2x20SortingBatches	5
CloseToSeedBatch1x40ResortTrue	5
CloseToSeed8x5ParallelInsertion	5
CloseToSeed1,5KM	5
CloseToSeed0,1KM	5
CloseToSeedBatch8x5SortingBatches	6
CloseToSeedBatch4x10SortingBatches	6
CloseToSeedBatch14x3SortingBatches	6
CloseToSeed3x14ParallelInsertion	6
CloseToSeed1,25KM	6
CloseToSeedCheapestInsertionWaitTimeAndCost	7
CloseToSeedBatch7x6	7
CloseToSeed7x6ParallelInsertion	7
CloseToSeed10x4ParallelInsertion	7
CloseToSeedBatch5x8	8
CloseToSeedBatch4x10	8
CloseToSeed2,5KM	8
CloseToSeed14x3ParallelInsertion	8
CloseToSeedStartTW80Min	9
CloseToSeedCheapestInsertionCosts	9
CloseToSeed2KM	9

Continued on next page

Table A.9 – Continued from previous page

Algorithm	Best 3rd rank
CloseToSeed1Min	9
CloseToSeed0Min	9
CloseToSeed0,5Min	9
CloseToSeedBatch7x6SortingBatches	10
CloseToSeedBatch6x7	10
CloseToSeed2,5Min	10
CloseToSeed1,5Min	10
CloseToSeed0,75KM	10
CloseToSeedBatch20x2SortingBatches	12
CloseToSeed3KM	12
CloseToSeed20x2ParallelInsertion	12
CloseToSeed0,5KM	12
CloseToSeedBatch10x4SortingBatches	13
CloseToSeedBatch10x4	13
CloseToSeedStartTW0Min	14
CloseToSeedBatch8x5	14
CloseToSeed1,75KM	14
CloseToSeedStartTW20Min	15
CloseToSeedBatch40x1SortingBatches	15
CloseToSeed2Min	15
Algorithm of Company X	16
CloseToSeed5Min	17
CloseToSeed4KM	17
CloseToSeedStartTW120Min	18
CloseToSeedBatch14x3	18
CloseToSeed7,5Min	19
CloseToSeed40x1ParallelInsertion	21
CloseToSeedBatch40x1	26
CloseToSeedCheapestInsertionDrivingTime	31
CloseToSeedBatch20x2	36
CloseToSeed7,5KM	36
CloseToSeed5KM	37
CloseToSeed10KM	50
CloseToSeed10Min	51
CloseToSeedQuantityDecreasingScale0	52
PlanTasksCloseToSeedTaskIDSorting	60

Table A.10: An overview of the average of the best 25% Rankings of the algorithm with a different insertion method used in the exploration experiment where the table is presented increasingly

Algorithm	Best 25% rank
CloseToSeedBatch3x14	3.9
CloseToSeedBatch2x20	4.2
CloseToSeed2x20ParallelInsertion	5.2
CloseToSeed5x8ParallelInsertion	5.4
CloseToSeed1x40ParallelInsertion	6.1
CloseToSeed0,1KM	6.8
CloseToSeed4x10ParallelInsertion	6.8
CloseToSeedBatch1x40ResortTrue	7.1
CloseToSeed3x14ParallelInsertion	8
CloseToSeedBatch3x14SortingBatches	8.1
CloseToSeedBatch7x6	8.5
CloseToSeedBatch6x7SortingBatches	8.6
CloseToSeed0,25KM	8.9

Continued on next page

Table A.10 – Continued from previous page

Algorithm	**Best 25% rank**
CloseToSeed6x7ParallelInsertion	9
CloseToSeedBatch5x8SortingBatches	9.1
CloseToSeedBatch5x8	9.5
CloseToSeedBatch4x10SortingBatches	9.8
CloseToSeed1,5KM	10.2
CloseToSeedBatch2x20SortingBatches	10.2
CloseToSeed1,25KM	10.3
CloseToSeed7x6ParallelInsertion	10.5
CloseToSeed14x3ParallelInsertion	11.1
CloseToSeed8x5ParallelInsertion	11.3
CloseToSeed0KM	11.7
CloseToSeedBatch4x10	12.6
CloseToSeedBatch7x6SortingBatches	13.2
CloseToSeedBatch8x5SortingBatches	13.3
CloseToSeed1,5Min	14.2
CloseToSeed0Min	14.6
CloseToSeed0,5KM	14.7
CloseToSeed0,75KM	14.9
CloseToSeed2,5Min	15
CloseToSeedBatch6x7	15
CloseToSeedBatch10x4SortingBatches	15.2
CloseToSeed1Min	15.7
CloseToSeed0,5Min	15.8
CloseToSeedBatch14x3SortingBatches	15.9
CloseToSeed10x4ParallelInsertion	16.2
CloseToSeedStartTW80Min	16.8
CloseToSeed3KM	18.2
CloseToSeedCheapestInsertionDistance	18.3
CloseToSeedCheapestInsertionWaitTimeAndCost	18.4
CloseToSeed2KM	19.6
CloseToSeed2Min	20
CloseToSeedBatch10x4	20.9
CloseToSeed1,75KM	22.3
CloseToSeedStartTW0Min	23.2
CloseToSeedStartTW120Min	24.1
CloseToSeedBatch8x5	24.3
CloseToSeedStartTW20Min	24.4
CloseToSeed40x1ParallelInsertion	24.5
CloseToSeed20x2ParallelInsertion	24.6
Algorithm of Company X	25.6
CloseToSeedBatch20x2SortingBatches	27.2
CloseToSeed2,5KM	27.8
CloseToSeedCheapestInsertionCosts	28.4
CloseToSeedBatch14x3	29
CloseToSeed4KM	29.2
CloseToSeed7,5Min	30.5
CloseToSeed5Min	37.3
CloseToSeedCheapestInsertionDrivingTime	37.7
CloseToSeedBatch20x2	43.9
CloseToSeedBatch40x1SortingBatches	45.4
CloseToSeed5KM	48.9
CloseToSeed7,5KM	54.3
CloseToSeed10KM	59.2
CloseToSeedQuantityDecreasingScale0	63.1
CloseToSeed10Min	65.6

Continued on next page

Table A.10 – Continued from previous page

Algorithm	**Best 25% rank**
CloseToSeedBatch40x1	66.2
PlanTasksCloseToSeedTaskIDSorting	73.1

Table A.22: Top 15 algorithms for the 3 categories: Average costs, best 3rd rank and the average of the best 25% rankings

Average costs	**Best 3rd rank**	**Average of best 25% rankings**
SeedDiffDepot40KM_2x20Parallel_250m_CostTW	**SeedDiffDepot30Min_6x7SortingBatches_100m_CostTW**	**SeedDiffDepot30Min475_1x40Parallel_100m_CostTW**
SeedDiffDepot40KM475_2x20Parallel_250m_CostTW	**SeedDiffDepot40KM475_2x20_250m_CostTW**	**SeedDiffDepot40KM475_4x10Parallel_250m_CostTW**
SeedDiffDepot40KM475_2x20_250m_CostTW	**SeedDiffDepot40KM475_6x7SortingBatches_250m_CostTW**	SeedDiffDepot40KM_4x10Parallel_250m_CostTW
SeedDiffDepot40KM475_4x10Parallel_250m_CostTW	SeedDiffDepot40KM_6x7SortingBatches_250m_CostTW	SeedDiffDepot30Min_1x40Parallel_100m_CostTW
SeedDiffDepot40KM_3x14SortingBatches_250m_CostTW	**SeedDiffDepot40KM475_4x10Parallel_250m_CostTW**	**SeedDiffDepot40KM475_2x20_250m_CostTW**
SeedDiffDepot40KM475_3x14_100m_CostTW	**SeedDiffDepot40KM475_3x14Parallel_250m_CostTW**	**SeedFarAway475_2x20Parallel_250m_CostTW**
SeedDiffDepot40KM_4x10Parallel_250m_CostTW	SeedDiffDepot30Min_1x40Parallel_100m_CostTW	SeedFarAway_2x20Parallel_250m_CostTW
SeedDiffDepot40KM475_1x40Parallel_250m_CostTW	**SeedDiffDepot30Min475_1x40Parallel_100m_CostTW**	SeedDiffDepot30min475_6x7SortingBatches_100m_CostTW
SeedDiffDepot40KM_6x7SortingBatches_250m_CostTW	**SeedFarAway475_2x20Parallel_250m_CostTW**	**SeedDiffDepot40KM475_6x7SortingBatches_250m_CostTW**
SeedDiffDepot30Min475_2x20Parallel_250m_CostTW	**SeedDiffDepot2,5KM475_4x10Parallel_250m_CostTW**	SeedDiffDepot40KM_6x7SortingBatches_250m_CostTW
SeedDiffDepot40KM475_3x14SortingBatches_250m_CostTW	SeedFarAway_2x20Parallel_250m_CostTW	**SeedFarAway475_3x14SortingBatches_250m_CostTW**
SeedDiffDepot40KM_2x20_250m_CostTW	**SeedFarAway_5x8Parallel_250m_CostTW**	**SeedDiffDepot2,5KM475_3x14Parallel_250m_CostTW**
SeedDiffDepot40KM475_3x14Parallel_250m_CostTW	SeedDiffDepot40KM_4x10Parallel_250m_CostTW	SeedDiffDepot40KM475_1x40Parallel_250m_CostTW
SeedDiffDepot40KM_3x14Parallel_250m_CostTW	SeedDiffDepot30min475_6x7SortingBatches_100m_CostTW	SeedFarAway_3x14SortingBatches_250m_CostTW
SeedDiffDepot40KM_5x8Parallel_100m_CostTW	SeedDiffDepot30Min_2x20Parallel_100m_CostTW	SeedDiffDepot2,5KM475_2x20_100m_CostTW

Table A.6: An overview of the average of the best 25% Rankings of the algorithm with a different seed task selection method used in the exploration experiment where the table is presented increasingly

Algorithm	**Average Top 25% Ranking**
SeedFarAwayDistance	2.6
SeedTaskFarToNearDiffDepot2,5KM	4.3
SeedTaskFarToNearDiffDepot40KM	6.0
SeedTaskFarToNearDiffDepot60KM	6.6
SeedTaskFarToNearDiffDepot30Min	8.0
SeedFarAwayDrivingTime	8.1
SeedTaskFarToNearDiffDepot60Min	8.5
SeedTaskFarToNearDiffDepot5Min	9.4
SeedTaskFarToNearDiffDepot20KM	9.5
SeedTaskFarToNearNearestAddress1KMDecreasing	10.2
SeedTaskFarToNearDiffDepot7,5KM	10.4
SeedTaskFarToNearDiffDepot5KM	10.7
SeedTaskFarToNearNearestAddress1MinDecreasing	11.9
SeedTaskFarToNearDiffDepot0Min	12.3
SeedTaskFarToNearNearestAddress1MinIncreasing	12.5
SeedTaskFarToNearNearestAddress2,5MinDecreasing	12.6
SeedTaskFarToNearDiffDepot0KM	12.9
SeedTaskFarToNearDiffDepot1Min	13.3
SeedTaskFarToNearDiffDepot15KM	13.9
SeedTaskFarToNearDiffDepot10Min	14.1
SeedTaskFarToNearNearestAddress0MinIncreasing	14.2
SeedTaskFarToNearNearestAddress5KMDecreasing	14.6
SeedTaskFarToNearNearestAddress10MinDecreasing	15.6
SeedTaskFarToNearNearestAddress1KMIncreasing	16.8
SeedTaskFarToNearNearestAddress10MinIncreasing	17.4
SeedTaskFarToNearNearestAddress0KMIncreasing	17.7
SeedTaskFarToNearNearestAddress2,5KMDecreasing	18.0
SeedTaskFarToNearNearestAddress0MinDecreasing	18.1
SeedTaskFarToNearNearestAddress2,5KMIncreasing	18.3
SeedTaskFarToNearNearestAddress2,5MinIncreasing	18.9
SeedTaskFarToNearNearestAddress7,5KMDecreasing	19.1
SeedTaskFarToNearNearestAddress5MinDecreasing	19.4
SeedTaskQuantityScale190	23.8
SeedTaskFarToNearTWDurationIncreasingScale0	25.2
Algorithm of Company X	25.6
SeedTaskQuantityScale475	25.7
SeedTaskFarToNearNearestAddress5KMIncreasing	26.6
SeedTaskFarToNearNearestAddress7,5KMincreasing	26.7
SeedTaskQuantityScale380	26.7
SeedTaskFarToNearNearestAddress5MinIncreasing	29.8
SeedTaskQuantityScale285	31.4
PlanSeedTaskFarToNearQuantityDecreasingScale0	44.0

Table A.7: Increasing list of the best 3rd Rank of the algorithms with a different seed task selection method used in the exploration experiment

Algorithm	**Best 3rd rank**
SeedFarAwayDistance	1
SeedTaskFarToNearDiffDepot2,5KM	3
SeedTaskFarToNearDiffDepot40KM	3
SeedFarAwayDrivingTime	4
SeedTaskFarToNearDiffDepot30Min	4
SeedTaskFarToNearDiffDepot60KM	4
SeedTaskFarToNearDiffDepot60Min	4
SeedTaskFarToNearNearestAddress0MinIncreasing	4
SeedTaskFarToNearNearestAddress5KMDecreasing	4
SeedTaskFarToNearDiffDepot5Min	5
SeedTaskFarToNearNearestAddress1MinIncreasing	5
SeedTaskFarToNearNearestAddress2,5MinDecreasing	5
SeedTaskFarToNearDiffDepot1Min	6
SeedTaskFarToNearDiffDepot5KM	6
SeedTaskFarToNearDiffDepot7,5KM	6
SeedTaskFarToNearNearestAddress1KMDecreasing	6
SeedTaskFarToNearDiffDepot0Min	7
SeedTaskFarToNearDiffDepot15KM	7
SeedTaskFarToNearDiffDepot20KM	7
SeedTaskFarToNearDiffDepot10Min	8
SeedTaskFarToNearNearestAddress10MinIncreasing	8
SeedTaskFarToNearNearestAddress2,5KMIncreasing	8
SeedTaskFarToNearTWDurationIncreasingScale0	8
SeedTaskFarToNearDiffDepot0KM	9
SeedTaskFarToNearNearestAddress0KMIncreasing	9
SeedTaskFarToNearNearestAddress10MinDecreasing	9
SeedTaskFarToNearNearestAddress1MinDecreasing	9
SeedTaskFarToNearNearestAddress1KMIncreasing	11
SeedTaskQuantityScale190	12
SeedTaskFarToNearNearestAddress2,5KMDecreasing	13
SeedTaskFarToNearNearestAddress7,5KMDecreasing	13
Algorithm of Company X	16
SeedTaskFarToNearNearestAddress0MinDecreasing	16
SeedTaskFarToNearNearestAddress5MinDecreasing	16
SeedTaskFarToNearNearestAddress2,5MinIncreasing	17
SeedTaskQuantityScale475	18
SeedTaskFarToNearNearestAddress5KMIncreasing	19
SeedTaskFarToNearNearestAddress7,5KMincreasing	21
SeedTaskQuantityScale380	23
SeedTaskFarToNearNearestAddress5MinIncreasing	24
PlanSeedTaskFarToNearQuantityDecreasingScale0	28
SeedTaskQuantityScale285	29

Table A.11: Increasing list of the best 3rd Rank of the algorithms with different batch sizes, number of batches and insertion procedures in the exploration experiment

Algorithm	**Best 3rd rank**
CloseToSeed2x20ParallelInsertion	1
CloseToSeed5x8ParallelInsertion	1
CloseToSeedBatch3x14	2
CloseToSeedBatch2x20	2
CloseToSeedBatch3x14SortingBatches	2
CloseToSeedBatch6x7SortingBatches	3
CloseToSeed1x40ParallelInsertion	4
CloseToSeed4x10ParallelInsertion	4
CloseToSeed6x7ParallelInsertion	4
CloseToSeedBatch1x40ResortTrue	5
CloseToSeedBatch5x8SortingBatches	5
CloseToSeedBatch2x20SortingBatches	5
CloseToSeed8x5ParallelInsertion	5
CloseToSeed3x14ParallelInsertion	6
CloseToSeedBatch4x10SortingBatches	6
CloseToSeedBatch8x5SortingBatches	6
CloseToSeedBatch14x3SortingBatches	6
CloseToSeedBatch7x6	7
CloseToSeed7x6ParallelInsertion	7
CloseToSeed10x4ParallelInsertion	7
CloseToSeedBatch5x8	8
CloseToSeed14x3ParallelInsertion	8
CloseToSeedBatch4x10	8
CloseToSeedBatch7x6SortingBatches	10
CloseToSeedBatch6x7	10
CloseToSeed20x2ParallelInsertion	12
CloseToSeedBatch20x2SortingBatches	12
CloseToSeedBatch10x4SortingBatches	13
CloseToSeedBatch10x4	13
CloseToSeedBatch8x5	14
CloseToSeedBatch40x1SortingBatches	15
Algorithm of Company X	16
CloseToSeedBatch14x3	18
CloseToSeed40x1ParallelInsertion	21
CloseToSeedBatch40x1	26
CloseToSeedBatch20x2	36

Table A.12: An overview of the average of the best 25% Rankings of the algorithm with different batch sizes, number of batches and insertion procedures in the exploration experiment where the table is presented increasingly

Algorithm	Best 25% rank
CloseToSeedBatch3x14	3.9
CloseToSeedBatch2x20	4.2
CloseToSeed2x20ParallelInsertion	5.2
CloseToSeed5x8ParallelInsertion	5.4
CloseToSeed1x40ParallelInsertion	6.1
CloseToSeed4x10ParallelInsertion	6.8
CloseToSeedBatch1x40ResortTrue	7.1
CloseToSeed3x14ParallelInsertion	8.0
CloseToSeedBatch3x14SortingBatches	8.1
CloseToSeedBatch7x6	8.5
CloseToSeedBatch6x7SortingBatches	8.6
CloseToSeed6x7ParallelInsertion	9.0
CloseToSeedBatch5x8SortingBatches	9.1
CloseToSeedBatch5x8	9.5
CloseToSeedBatch4x10SortingBatches	9.8
CloseToSeedBatch2x20SortingBatches	10.2
CloseToSeed7x6ParallelInsertion	10.5
CloseToSeed14x3ParallelInsertion	11.1
CloseToSeed8x5ParallelInsertion	11.3
CloseToSeedBatch4x10	12.6
CloseToSeedBatch7x6SortingBatches	13.2
CloseToSeedBatch8x5SortingBatches	13.3
CloseToSeedBatch6x7	15.0
CloseToSeedBatch10x4SortingBatches	15.2
CloseToSeedBatch14x3SortingBatches	15.9
CloseToSeed10x4ParallelInsertion	16.2
CloseToSeedBatch10x4	20.9
CloseToSeedBatch8x5	24.3
CloseToSeed40x1ParallelInsertion	24.5
CloseToSeed20x2ParallelInsertion	24.6
Algorithm of Company X	25.6
CloseToSeedBatch20x2SortingBatches	27.2
CloseToSeedBatch14x3	29.0
CloseToSeedBatch20x2	43.9
CloseToSeedBatch40x1SortingBatches	45.4
CloseToSeedBatch40x1	66.2

Table A.13: An overview of the average costs for each algorithm with different batch sizes, number of batches and insertion procedures in the exploration experiment. The table is sorted increasingly. This table includes confidential data, therefore, the data is removed from the table.

Algorithm	Costs (€)

Table A.14: An overview of the calculation times per phase for the algorithms using different batch sizes, number of batches and insertion procedures. The calculation times per phase are presented in seconds and in the "Diff (%)" column the percentile difference compared to the calculation time of the algorithm of Company X is given.

Algorithm	Avg Construction Time (s)	Diff (%)	Total Calc Time (s)	Diff (%)
CloseToSeedBatch40x1	145.98	9.30	593.01	75.46
CloseToSeedBatch40x1SortingBatches	183.30	37.24	477.75	41.36
CloseToSeedBatch20x2	161.22	20.70	437.57	29.47
CloseToSeedBatch1x40ResortTrue	185.71	39.04	405.27	19.91
CloseToSeedBatch2x20SortingBatches	161.63	21.01	402.11	18.98
CloseToSeedBatch14x3	150.40	12.60	397.95	17.75
CloseToSeedBatch8x5SortingBatches	153.04	14.58	386.55	14.37
CloseToSeedBatch14x3SortingBatches	147.71	10.58	379.58	12.31
CloseToSeedBatch6x7SortingBatches	151.83	13.67	374.71	10.87
CloseToSeedBatch5x8SortingBatches	148.94	11.51	371.02	9.78
CloseToSeedBatch20x2SortingBatches	154.91	15.98	370.73	9.69
CloseToSeedBatch3x14SortingBatches	157.15	17.66	370.10	9.50
CloseToSeedBatch7x6SortingBatches	150.42	12.62	369.98	9.47
CloseToSeedBatch10x4SortingBatches	146.79	9.90	364.00	7.70
CloseToSeedBatch2x20	144.58	8.24	363.64	7.59
CloseToSeed40x1ParallelInsertion	135.32	1.31	362.63	7.30
CloseToSeedBatch10x4	143.97	7.79	361.94	7.09
CloseToSeedBatch3x14	136.48	2.18	359.81	6.46
CloseToSeedBatch4x10SortingBatches	145.80	9.16	359.64	6.41
CloseToSeedBatch8x5	141.05	5.60	357.94	5.91
CloseToSeed1x40ParallelInsertion	164.34	23.04	357.37	5.74
CloseToSeedBatch6x7	136.39	2.12	350.48	3.70
CloseToSeedBatch7x6	134.36	0.59	350.32	3.65
CloseToSeedBatch5x8	132.90	-0.50	345.95	2.36
CloseToSeedBatch4x10	133.90	0.25	342.04	1.20
Algorithm of Company X	133.57	0.00	337.97	0.00
CloseToSeed2x20ParallelInsertion	141.79	6.15	333.55	-1.31
CloseToSeed10x4ParallelInsertion	117.01	-12.40	326.78	-3.31
CloseToSeed3x14ParallelInsertion	124.83	-6.54	322.12	-4.69
CloseToSeed4x10ParallelInsertion	126.70	-5.14	320.18	-5.26
CloseToSeed5x8ParallelInsertion	122.27	-8.46	318.48	-5.77
CloseToSeed20x2ParallelInsertion	115.85	-13.27	318.08	-5.89
CloseToSeed7x6ParallelInsertion	115.51	-13.52	309.62	-8.39
CloseToSeed8x5ParallelInsertion	112.85	-15.51	302.39	-10.53
CloseToSeed6x7ParallelInsertion	115.30	-13.68	301.44	-10.81
CloseToSeed14x3ParallelInsertion	111.76	-16.33	301.19	-10.88

Table A.15: An overview of the average costs for each algorithm with different scales of the first sorter of the insertion method of Company X in the exploration experiment. The table is sorted increasingly. This table includes confidential data, therefore, the data is removed from the table.

Algorithm	Costs (€)

Table A.16: Increasing list of the best 3rd Rank of the algorithms with different scales of the first sorter of the insertion method in the exploration experiment

Algorithm	Rank
CloseToSeed0,1KM	4
CloseToSeed1,5KM	4
CloseToSeed0,25KM	5
CloseToSeed1,25KM	5
CloseToSeed0KM	6
CloseToSeed1Min	8
CloseToSeed0,5KM	9
CloseToSeed0,5Min	9
CloseToSeed0,75KM	9
CloseToSeed0Min	9
CloseToSeed1,5Min	10
CloseToSeed2,5Min	10
CloseToSeed2KM	10
CloseToSeed2Min	12
CloseToSeed1,75KM	12
CloseToSeed3KM	14
CloseToSeed2,5KM	15
CloseToSeed4KM	16
CloseToSeed5Min	17
CloseToSeed7,5Min	17
CloseToSeed5KM	19
CloseToSeed7,5KM	36
CloseToSeed10KM	37
CloseToSeed10Min	50
Algorithm of Company X	51

Table A.17: An overview of the average of the best 25% Rankings of the algorithms with a different scale of the first sorter of the insertion method in the exploration experiment where the table is presented increasingly

Algorithm	Average Rank
CloseToSeed0,1KM	6.8
CloseToSeed0,25KM	8.9
CloseToSeed1,5KM	10.2
CloseToSeed1,25KM	10.3
CloseToSeed0KM	11.7
CloseToSeed1,5Min	14.2
CloseToSeed0Min	14.6
CloseToSeed0,5KM	14.7
CloseToSeed0,75KM	14.9
CloseToSeed2,5Min	15
CloseToSeed1Min	15.7
CloseToSeed0,5Min	15.8
CloseToSeed3KM	18.2
CloseToSeed2KM	19.6
CloseToSeed2Min	20.0
CloseToSeed1,75KM	22.3
Algorithm of Company X	25.6
CloseToSeed2,5KM	27.8
CloseToSeed4KM	29.2
CloseToSeed7,5Min	30.5
CloseToSeed5Min	37.3
CloseToSeed5KM	48.9
CloseToSeed7,5KM	54.3
CloseToSeed10KM	59.2
CloseToSeed10Min	65.6

Table A.18: An overview of the average costs in proportion to the algorithm of Company X for each algorithm with a different estimator of the insertion procedure and different scales of the second sorter of the insertion method in the exploration experiment. The table is sorted increasingly.

Algorithm	**Costs (%)**
CloseToSeedStartTW80Min	-0.01
CloseToSeedStartTW120Min	0
CloseToSeedStartTW0Min	0
CloseToSeedStartTW20Min	0
Algorithm of Company X	0
CloseToSeedCheapestInsertionDistance	0.14
CloseToSeedCheapestInsertionWaitTimeAndCost	0.26
CloseToSeedCheapestInsertionCosts	0.29
CloseToSeedCheapestInsertionDrivingTime	0.36

Table A.19: Increasing list of the best 3rd Rank of the algorithms with a different estimator of the insertion procedure and different scale of the second sorter of the insertion method in the exploration experiment

Algorithm	**MinRank3FinalSolution**
CloseToSeedCheapestInsertionDistance	5
CloseToSeedCheapestInsertionWaitTimeAndCost	7
CloseToSeedStartTW80Min	9
CloseToSeedCheapestInsertionCosts	9
CloseToSeedStartTW0Min	14
CloseToSeedStartTW20Min	15
Algorithm of Company X	16
CloseToSeedStartTW120Min	18
CloseToSeedCheapestInsertionDrivingTime	31

Table A.20: An overview of the average of the best 25% Rankings of the algorithms with a different estimator of the insertion procedure and different scale of the second sorter of the insertion method in the exploration experiment where the table is presented increasingly

Algorithm	**AverageTop25%**
CloseToSeedStartTW80Min	16.8
CloseToSeedCheapestInsertionDistance	18.3
CloseToSeedCheapestInsertionWaitTimeAndCost	18.4
CloseToSeedStartTW0Min	23.2
CloseToSeedStartTW120Min	24.1
CloseToSeedStartTW20Min	24.4
Algorithm of Company X	25.6
CloseToSeedCheapestInsertionCosts	28.4
CloseToSeedCheapestInsertionDrivingTime	37.7

Table A.21: Overview of the rankings for the benchmark algorithms per case in the dataset

Case (Country	SeedTask&InsertionTaskID	InsertionLargestOrderQuantity	SeedTaskLargestOrderQuantity	SeedTaskShortestTWLength
1 (A)	113	110	107	108
2 (A)	113	110	111	109
3 (A)	113	112	109	110
4 (A)	111	112	101	108
5 (A)	112	113	97	63
6 (A)	113	105	112	110
7 (B)	76	70	84	37
8 (B)	9	82	39	71
9 (B)	80	111	41	30
10 (B)	98	13	100	4
11 (B)	105	88	100	31
12 (B)	111	84	39	10
13 (A)	110	112	76	102
14 (A)	109	110	106	103
15 (A)	109	111	112	103
16 (A)	113	112	107	103
17 (A)	109	112	111	108
18 (A)	113	112	109	108
19 (A)	113	111	106	107
20 (A)	110	111	108	109
21 (A)	113	112	84	110
22 (A)	112	113	111	104
23 (A)	112	113	106	107
24 (A)	113	112	108	107
25 (A)	112	109	105	103
26 (A)	113	111	106	105
27 (A)	113	109	101	96
28 (B)	6	8	74	1
29 (B)	60	52	21	8
30 (B)	88	53	28	38
31 (B)	105	113	36	30
32 (B)	104	78	68	110
33 (B)	110	103	18	93
34 (A)	113	112	107	82
35 (A)	112	113	110	109
36 (A)	113	111	105	109
37 (A)	113	111	110	107
38 (A)	113	112	103	101
39 (A)	113	112	107	101
40 (A)	113	109	105	89
41 (A)	111	113	95	92
42 (A)	113	112	104	103

Table A.23: Settings per removal percentage for the Route And Adjacent Task Removal method

Removal percentage	Number of routes	Adjacent tasks	Sequence length
1	1	1	2
2	2	2	2
3	3	2	2
4	3	4	2
5	5	5	2
7	5	2	4
10	5	6	3
12	4	6	5
15	6	5	5
17	7	6	4
20	9	5	4

Table A.24: The 15 best RR methods found in the exploitation experiment

RR method
NoRelated_1%_DrivingTime_125x2
NoRelated_1%_WaitTimeCosts_125x2
NoRelated_1%_WaitTimeCosts_63x4
NoRelated_1-3%_WaitTimeCosts_125x2
NoRelated_1-3%_WaitTimeCosts_63x4
PCI_NoRandom_1%_DrivingTime_125x2
PCI_NoRandom_1%_WaitTimeCosts_125x2
PCI_Random-Adjacent_1%_DrivingTime_125x2
PCI_Random-Adjacent_1%_DrivingTime_63x4
PCI_Random-Adjacent_1%_WaitTimeCosts_125x2
PCI_Random-Adjacent_1%_WaitTimeCosts_63x4
PCI_Random-Adjacent_1-3%_WaitTimeCosts_125x2
PCI_Random-Adjacent_1-3%_WaitTimeCosts_63x4
Related-Route_1%_DrivingTime_125x2
Related-Route_1-3%_DrivingTime_125x2

Table A.25: This table shows per case in the dataset what ranking the starting solution of the best final solution had. In this case, the starting solution is the solution before the R&R phase and thus after the local search phase. Since we are using 15 different algorithms to produce the starting solution for the R&R methods, the maximum ranking is 15.

Case	Best final solution ranking starting solution
1	9
2	1
3	1
4	4
5	4
6	1
7	1
8	1
9	14
10	2
11	1
12	5
13	1
14	1
15	1
16	1
17	1
18	1
19	3
20	1
21	2
22	1
23	6
24	1
25	1
26	5
27	3
28	3
29	2
30	2
Average	**2.67**

Bibliography

AIMMS (2020). Time Windows &x2014; AIMMS How-To — how-to.aimms.com. [Accessed 01-12-2023].

Alcaraz, J. J., Caballero-Arnaldos, L., and Vales-Alonso, J. (2019). Rich vehicle routing problem with last-mile outsourcing decisions. *Transportation Research Part E: Logistics and Transportation Review*, 129:263–286.

Balaprakash, P., Birattari, M., and Stützle, T. (2007). Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In *Hybrid Metaheuristics: 4th International Workshop, HM 2007, Dortmund, Germany, October 8-9, 2007. Proceedings 4*, pages 108–122. Springer.

Bouwstra, P. S., Correia, G., Bijl, P., Negenborn, R. R., and Atasoy, B. (2021). Stochastic and dynamic routing with flexible deliveries for an e-grocer. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 3354–3359. IEEE.

Brandão, J. C. S. and Mercer, A. (1998). The multi-trip vehicle routing problem. *Journal of the Operational research society*, 49:799–805.

Bräysy, O., Hasle, G., and Dullaert, W. (2004). A multi-start local search algorithm for the vehicle routing problem with time windows. *European Journal of Operational Research*, 159(3):586–605.

Caceres-Cruz, J., Arias, P., Guimarans, D., Riera, D., and Juan, A. A. (2014). Rich vehicle routing problem: Survey. *ACM Computing Surveys (CSUR)*, 47(2):1–28.

Carrabs, F., Cerulli, R., and Sciomachen, A. (2017). An exact approach for the grocery delivery problem in urban areas. *Soft Computing*, 21:2439–2450.

Cordeau, J.-F. and Laporte, G. (2001). A tabu search algorithm for the site dependent vehicle routing problem with time windows. *INFOR: Information Systems and Operational Research*, 39(3):292–298.

Cordeau, J.-F. and Maischberger, M. (2012). A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research*, 39(9):2033–2050.

Crainic, T. (2019). Parallel metaheuristics and cooperative search. *Handbook of metaheuristics*, pages 419–451.

Crainic, T. G. (2008). *Parallel Solution Methods for Vehicle Routing Problems*, pages 171–198. Springer US, Boston, MA.

Crainic, T. G. and Toulouse, M. (2010). *Parallel meta-heuristics*. Springer.

Czapiński, M. (2013). An effective parallel multistart tabu search for quadratic assignment problem on cuda platform. *Journal of Parallel and Distributed Computing*, 73(11):1461–1468. Novel architectures for high-performance computing.

Dantzig, G. B. and Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, 6(1):80–91.

Dayarian, I., Crainic, T. G., Gendreau, M., and Rei, W. (2015). A column generation approach for a multi-attribute vehicle routing problem. *European Journal of Operational Research*, 241(3):888–906.

Dondo, R. and Cerdá, J. (2007). A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows. *European journal of operational research*, 176(3):1478–1507.

Emeç, U., Çatay, B., and Bozkaya, B. (2016). An adaptive large neighborhood search for an e-grocery delivery routing problem. *Computers & Operations Research*, 69:109–125.

Ensafian, H. (2023). *Route optimization of autonomous vehicles in the last-mile delivery operations*. PhD thesis.

Fan, H., Zhang, Y., Tian, P., Lv, Y., and Fan, H. (2021). Time-dependent multi-depot green vehicle routing problem with time windows considering temporal-spatial distance. *Computers & Operations Research*, 129:105211.

Geschwender, D., Hutter, F., Kotthoff, L., Malitsky, Y., Hoos, H. H., and Leyton-Brown, K. (2014). Algorithm configuration in the cloud: A feasibility study. In *Learning and Intelligent Optimization: 8th International Conference, Lion 8, Gainesville, FL, USA, February 16-21, 2014. Revised Selected Papers 8*, pages 41–46. Springer.

Ghiani, G., Guerriero, F., Laporte, G., and Musmanno, R. (2003). Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. *European journal of operational research*, 151(1):1–11.

Gunawan, A., Lau, H. C., and Lindawati (2011). Fine-tuning algorithm parameters using the design of experiments approach. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pages 278–292. Springer.

Hans, E., Schutten, J., Zijm, W., et al. (2010). Vehicle routing with traffic congestion and drivers' driving and working rules.

Hoos, H. H. (2012). Automated algorithm configuration and parameter tuning. In *Autonomous search*, pages 37–71. Springer.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pages 507–523. Springer.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2012). Parallel algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 55–70. Springer.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Murphy, K. (2010). Time-bounded sequential parameter optimization. In *Learning and Intelligent Optimization: 4th International Conference, LION 4, Venice, Italy, January 18-22, 2010. Selected Papers 4*, pages 281–298. Springer.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. (2009). Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306.

Jansen, N. (2023). Applying machine learning in route optimization — TU Delft Repositories — resolver.tudelft.nl. `http://resolver.tudelft.nl/uuid:33ba6733-cd14-44e2-a051-5504b2fa7541`. [Accessed 11-12-2023].

Kok, A. L. (2010). Congestion avoidance and break scheduling within vehicle routing. *University of Twente School of Management and Governance*, 166.

Kumar, S. N. and Panneerselvam, R. (2012). A survey on the vehicle routing problem and its variants. *Intelligent Information Management*, 04(03):66–74.

Lahyani, R., Khemakhem, M., and Semet, F. (2015). Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research*, 241(1):1–14.

Laporte, G., Gendreau, M., Potvin, J.-Y., and Semet, F. (2000). Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, 7(4):285–300.

Le Bouthillier, A. and Crainic, T. G. (2005). A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Computers & Operations Research*, 32(7):1685–1708.

Lenstra, J. K. and Kan, A. R. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227.

Lin, I.-C., Lin, T.-H., and Chang, S.-H. (2022). A decision system for routing problems and rescheduling issues using unmanned aerial vehicles. *Applied Sciences*, 12(12):6140.

Liu, D., Deng, Z., Mao, X., Yang, Y., and Kaisar, E. I. (2020). Two-echelon vehicle-routing problem: optimization of autonomous delivery vehicle-assisted e-grocery distribution. *IEEE Access*, 8:108705–108719.

Liu, D., Yan, P., Pu, Z., Wang, Y., and Kaisar, E. I. (2021). Hybrid artificial immune algorithm for optimizing a van-robot e-grocery delivery system. *Transportation Research Part E: Logistics and Transportation Review*, 154:102466.

Liu, X., Chen, Y.-L., Por, L. Y., and Ku, C. S. (2023). A systematic literature review of vehicle routing problems with time windows. *Sustainability*, 15(15):12004.

Lourenço, H. R., Martin, O. C., and Stützle, T. (2003). Iterated local search. pages 320–353.

Montoya-Torres, J. R., López Franco, J., Nieto Isaza, S., Felizzola Jiménez, H., and Herazo-Padilla, N. (2015). A literature review on the vehicle routing problem with multiple depots. *Computers Industrial Engineering*, 79:115–129.

Pan, B., Zhang, Z., and Lim, A. (2021). Multi-trip time-dependent vehicle routing problem with time windows. *European Journal of Operational Research*, 291(1):218–231.

Pang, K.-W. (2011). An adaptive parallel route construction heuristic for the vehicle routing problem with time windows constraints. *Expert Systems with Applications*, 38(9):11939–11946.

Paraskevopoulos, D. C., Repoussis, P. P., Tarantilis, C. D., Ioannou, G., and Prastacos, G. P. (2008). A reactive variable neighborhood tabu search for the heterogeneous fleet vehicle routing problem with time windows. *Journal of Heuristics*, 14(5):425–455.

Polat, O. (2017). A parallel variable neighborhood search for the vehicle routing problem with divisible deliveries and pickups. *Computers Operations Research*, 85:71–86.

Rasku, J., Kärkkäinen, T., and Musliu, N. (2016). Feature extractors for describing vehicle routing problem instances. In *OASICS*. Dagstuhl Publishing.

Rasku, J., Musliu, N., and Kärkkäinen, T. (2019). On automatic algorithm configuration of vehicle routing problem solvers. *Journal on Vehicle Routing Algorithms*, 2:1–22.

Rincon-Garcia, N., Waterson, B., Cherrett, T. J., and Salazar-Arrieta, F. (2020). A metaheuristic for the time-dependent vehicle routing problem considering driving hours regulations–an application in city logistics. *Transportation Research Part A: Policy and Practice*, 137:429–446.

Ruinelli, L. (2011). *Column generation for a rich vrp*. PhD thesis, Master's thesis, Department of Innovative Technologies, SUPSI, Manno, Scuola ....

Schryen, G. (2020). Parallel computational optimization in operations research: A new integrative framework, literature review and research directions. *European Journal of Operational Research*, 287(1):1–18.

Scott, L. R., Clark, T., and Bagheri, B. (2005). *Scientific parallel computing*. Princeton University Press.

Simons, L. (2017). Adaptive Large Neighborhood Search for Rich and Real-World Vehicle Routing Problems — TU Delft Repositories — resolver.tudelft.nl. `http://resolver.tudelft.nl/uuid:f9372533-0be0-4b25-ab26-b580c71e6b9a`. [Accessed 09-12-2023].

Soonpracha, K., Mungwattana, A., Janssens, G. K., and Manisri, T. (2014). Heterogeneous vrp review and conceptual framework. 2:1052–1059.

Steinhaus, M. (2015). *The application of the self organizing map to the vehicle routing problem*. University of Rhode Island.

Styles, J. and Hoos, H. H. (2013). Using racing to automatically configure algorithms for scaling performance. In *Learning and Intelligent Optimization: 7th International Conference, LION 7, Catania, Italy, January 7-11, 2013, Revised Selected Papers 7*, pages 382–388. Springer.

Tinarut, P. and Leksakul, K. (2019). Hybrid self-organizing map approach for traveling salesman problem. *Chiang Mai University Journal of Natural Sciences*, 18.

Toulouse, M., Crainic, T. G., and Sansó, B. (1999). An experimental study of systemic behavior of cooperative search algorithms. *Meta-heuristics: advances and trends in local search paradigms for optimization*, pages 373–392.

Twomey, C., Stützle, T., Dorigo, M., Manfrin, M., and Birattari, M. (2010). An analysis of communication policies for homogeneous multi-colony aco algorithms. *Information Sciences*, 180(12):2390–2404.

Vazquez-Noguerol, M., Comesaña-Benavides, J., Poler, R., and Prado-Prado, J. C. (2022). An optimisation approach for the e-grocery order picking and delivery problem. *Central European Journal of Operations Research*, 30(3):961–990.

Viera, O. and Tansini, L. (2004). Adapted clustering algorithms for the assignment problem in the mdvrptw. *Reportes Técnicos 04-13*.

Wright, S. J. (2015). Coordinate descent algorithms. *Mathematical programming*, 151(1):3–34.

Zare-Reisabadi, E. and Mirmohammadi, S. H. (2015). Site dependent vehicle routing problem with soft time window: Modeling and solution approach. *Computers & Industrial Engineering*, 90:177–185.

Zhang, Q., Wang, Z., Huang, M., Yu, Y., and Fang, S.-C. (2022). Heterogeneous multi-depot collaborative vehicle routing problem. *Transportation Research Part B: Methodological*, 160:1–20.

Zhen, L., Ma, C., Wang, K., Xiao, L., and Zhang, W. (2020). Multi-depot multi-trip vehicle routing problem with time windows and release dates. *Transportation Research Part E: Logistics and Transportation Review*, 135:101866.

Zhou, L., Wang, X., Ni, L., and Lin, Y. (2016). Location-routing problem with simultaneous home delivery and customer's pickup for city distribution of online shopping purchases. *Sustainability*, 8(8):828.

Zunic, E., Besirevic, A., Skrobo, R., Hasic, H., Hodzic, K., and Djedovic, A. (2017). Design of optimization system for warehouse order picking in real environment.