KU LEUVEN

FACULTY OF SCIENCE

Predicting the performance of an algorithm for the Capacitated Vehicle Routing Problem with Time Windows

(in collaboration with ORTEC)

Muratzhan Kyranbay and Johan Rogiers

Supervisor: Prof. R. Leus
ORSTAT

Cosupervisor: Dr. T. Hellemans
ORTEC

Master thesis submitted in fulfillment
of the requirements for the degree in
Master of Science in Statistics and Data Science

Academic year 2022-2023

Preface

The primary motivation behind writing this thesis was to research how traditional statistical techniques and machine learning models could be applied in operations research. However, taking into account our background in statistics, in one year, there is no possibility of doing extensive research on this broad topic. So, we limited ourselves to the *Capacitated Vehicle Routing Problem with Time Windows*. We were able to build predictive models for different performance measures of the optimization algorithm for this class of problem instances. Besides the predictive model, there were other statistical methods we did not expect to be helpful in our thesis, but they turned out to be crucial. They are the techniques of *Design of Experiments* and *Simulation*. We hope that, based on our work, there will be more academic papers on the applications of statistics and machine learning in other operations research problems.

We want to thank our supervisor Prof. Roel Leus for his genuine trust in our work and timely feedback, and our daily supervisor Dr. Tim Hellemans from ORTEC, for his technical help and the time he has devoted to being the contact person at the company. We also would like to thank Dr. Ernst Roos for his help in accessing the data, explaining the algorithm, and organizing monthly presentation sessions where we learned a lot from other students' theses. We appreciate the effort of ORTEC making it possible for students to work on their theses in collaboration with them.

Lastly, we thank our family and friends for their unconditional love and constant support.

Contribution statement

Prof. Roel Leus contributed to our thesis by being our contact person when we needed academic references on specific topics. He also suggested the idea of simulating instances.

Dr. Tim Hellemans has contributed to our instance simulation part by providing some ideas on how that could be done. He has also been a big help in optimizing the code that computed the values of the input features of the predictive model.

Dr. Ernst Roos provided us with the data we used in exploratory data analysis and some suggestions on what features to use in building the predictive model. He also explained the details of the algorithm that we used.

We have contributed equally to all the work, except the individual part of the thesis, which had to be written separately for assessment purposes. Muratzhan Kyranbay has worked on individually and wrote the Section 5.7 Johan Rogiers has worked on individually and wrote the Section 5.8.

Summary

Over the years, ORTEC has built several optimization algorithms for Capacitated Vehicle Routing Problems with Time Windows. In this problem, orders from a set of customers must be delivered by vehicles during the Time Windows specified by the customers. The specific algorithm we are working with is a heuristic algorithm that usually finds near-optimal solutions for the instances of the Capacitated Vehicle Routing Problem with Time Windows. In our thesis, we have aimed to build a predictive model for different performance measures of this optimization algorithm.

First, we have built a model to classify if the algorithm can feasibly solve the problem instance. Then, for the instances that the algorithm can achieve a feasible solution, we have built a model that can predict the time it takes for the algorithm to reach a reasonably good solution. Finally, for the infeasible instances, we have built a model to predict the percentage of customers whose orders were delivered. The models we have used are Regularized Logistic Regression, Linear Discriminant Analysis, Quadratic Discriminant Analysis, K-nearest Neighbors, Polynomial Kernel SVM, RBF Kernel SVM, Random Forest Classifier, Linear Regression, and Random Forest Regressor. The features used in the model are given in the main text as that requires some knowledge of the details of the Capacitated Vehicle Routing Problem with Time Windows. However, they are generally the summary measures of the Time Windows, customers' locations, number of customers, and vehicles.

Before building predictive models, we simulated our instances of the Capacitated Vehicle Routing Problem with Time Windows. We wanted to enrich our data by simulating instances where the feature values we extract are varied enough. That is important in assessing their impact on the performance of the algorithm. Therefore, the values of the features we would like to extract from the simulated instances' are decided by the Design of Experiments.

Of course, before the main analysis and modelling, we have put a significant effort into learning about the Capacitated Vehicle Routing Problem with Time Windows, researching the techniques to solve them, and understanding the data and algorithm we are working with.

Glossary

μ_{CTW} Customer Time Window Mean.

μ_D Distance Matrix Mean.

μ_{VTW} Vehicle Time Window Mean.

μ_{cd} Centroid Depot Mean Distance.

c Number of Clusters.

l_c Customer Demand Location.

l_v Vehicle Availability Location.

m Number of Vehicles.

n Number of Customers.

s_{CTW} Customer Time Window Standard Deviation.

s_D Distance Matrix Standard Deviation.

s_{VTW} Vehicle Time Window Standard Deviation.

Acronyms

CVRPTW Capacitated Vehicle Routing Problem with Time Windows.

DoE Design of Experiments.

KPI Key Performance Indices.

LNS Large Neighborhood Search.

MAE Mean Absolute Error.

MST Minimum Spanning Tree.

OHD ORTEC Home Delivery.

PCA Principal Component Analysis.

RMSE Root Mean Square Error.

RR Ruin and Recreate.

SA Simulated Annealing.

SPTWCC Shortest Path Problem with Time Windows and Capacity Constraints.

TDP Truck Dispatching Problem.

TSP Traveling Salesman Problem.

VRP Vehicle Routing Problem.

Contents

Chapter 1

Introduction

The Vehicle Routing Problem (VRP) was first introduced in Dantzig and Ramser (1959) with the name Truck Dispatching Problem (TDP). In this problem, identical vehicles with the same capacities deliver goods of specific sizes to a number of customers. The vehicles start from the depot and return to it. Of course, at any given time, the total size of the goods the trucks are loaded with must be, at most, the vehicle's capacity. And the objective that the authors had to achieve was to minimize the total distance travelled. In practice, however, businesses might also have other objectives, such as minimizing the driving duration, number of vehicles used, etc.

Along with the variations of it, VRPs are one of the most studied combinatorial problems in Operations Research. Nearly all logistics services, such as mail delivery, school bus routing, and cargo ship routing, can be considered (variations of) VRP problems (Vidal et al., 2020). According to the estimations, 70% of the value-added costs for businesses on food and drinks come from distribution costs. In addition, 15% of the national expenditures of the UK come from transportation (Bräysy and Gendreau, 2005). However, solving VRP would not be enough for many real-life applications. Therefore, there has been extensive research on the variations of the VRP. For example, if we add time window constraints for mail delivery, bus pick-up time, and cargo ship arrival time, we get a Capacitated Vehicle Routing Problem with Time Windows (CVRPTW).

Unfortunately, VRPs (including the CVRPTW) are proven to be NP-hard, and finding the exact optimal solutions becomes more challenging as the instance size increases. In fact, there is no algorithm known that can solve VRPs in polynomial time for the arbitrary size of the instance. Thus, CVRPTWs are solved heuristically in practice. Heuristic algorithms provide a good balance between the quality of the solution and the time it takes to reach this near-optimal solution.

As an Operations Research consulting company, ORTEC offers the services of optimizing CVRPTW instances their clients face in their businesses. Over the years, ORTEC has developed several algorithms to solve CVRPTW, one of the most successful ones being the algorithm used in ORTEC Home Delivery (OHD) service. Besides the general purpose configuration, it can be fine-tuned to achieve the best performance considering clients' specific needs. To mention a few examples, delivery in densely populated cities differs from delivery in rural areas, or some companies might have particular time windows of specific lengths. For the sake of simplicity, we will refer to the general-purpose version of the algorithm as the OHD algorithm.

In the original definition of the CVRPTW, if a solution does not exist that respects all time windows and visits all customers simultaneously, the CVRPTW problem instance is

considered infeasible. However, OHD is a heuristic algorithm where it is possible that during the optimization steps, some customers are not served either due to the instance's infeasibility or, in case it is a feasible problem, the algorithm's inability to find a feasible solution in a limited number of steps that it consists of. For the OHD algorithm, the first objective is to deliver to as many customers as possible, followed by the user-defined objective, which is a linear combination of total cost and vehicle wait time. It is a minor deviation from the original definition of a "solution" for the CVRPTW, and the reader might confuse this now with problems of type "Routing with Profits", where the main objective is to make as much profit as possible by visiting only a profitable subset of the customers. However, the biggest difference here is that OHD still tries to deliver goods for all customers, and the cost comes as the second/third objective, whereas for "Routing with Profits", the cost is the only objective. From now on, the outputs of the OHD algorithm where all customers are served will be called a "(complete) solution", and the outputs where only some customers are served will be called an "incomplete solution".

This thesis aims to use statistical/machine learning techniques to understand the performance of the general-purpose version of the algorithm. To be precise, we are interested in analyzing the performance of the OHD algorithm given the characteristics of the CVRPTW instance, e.g., tightness of the time windows and clusteredness of the delivery addresses. To that end, we will use several CVRPTW instances from ORTEC's customers and simulate our instances to enrich our dataset. When simulating CVRPTW instances, we would like to obtain as many varied instances as possible regarding their feature values. So, our idea to achieve it is to have a set of feature values we want our simulated instances to have. Preferably, those values need to be as varied as possible. The concept of Design of Experiments will be used to select those values so that the effects of the individual features on the algorithm's output are better assessed. Knowing the feature values at which the OHD algorithm performs better or worse can be highly beneficial in developing its new versions. Moreover, it is an excellent opportunity to bridge Operations Research with Statistics and push the boundaries of CVRPTW research by introducing a new methodology.

One paper with the closest research goal is by Arnold and Sörensen (2019). The authors assess the importance of several solution features in differentiating optimal solutions from non-optimal ones. Some solution features considered include compactness, width, the span in radians of routes, and the number of intersecting edges. Also, they were shown to help guide the optimization algorithm. Finally, Lucas et al. (2019) further extend the mentioned paper by employing Principal Component Analysis (PCA) to better handle the features' inter-dependencies/multicollinearity.

What makes our work different from the one mentioned above is that we are researching the impact of the input features instead of the solution features on the performance. That means that we will describe an instance's input characteristics instead of a solution's characteristics. In addition, unlike just differentiating the optimal solutions from non-optimal ones, we will measure the impact of the input features on several performance metrics. Namely, first, we will make a classification model that predicts whether the OHD's solution output is complete or incomplete. Then, the percentage of the tasks delivered will be predicted for the incomplete ones. As for the complete ones, the time it takes to reach the plateau in the optimization process of the OHD algorithm will be predicted. Both values will be introduced in the Chapter 5.

Chapter 2 introduces a typical CVRPTW instance faced by ORTEC's clients and the outputs of the OHD algorithm on those instances. In Chapter 3, we will give a literature review of

existing algorithms for solving CVRPTW, potential input features to explore. Then we will explain the details of the OHD algorithm in Chapter 4. The main analysis is presented in Chapter 5, followed by Chapter 6, where we will give the conclusions of our work and highlight the future work that could be done in this field.

Chapter 2

Introduction of the problem and data

We have introduced the concept of CVRPTWs in general, but we would now like to show a working example of what type of problem it is and how OHD solves it. This section is dedicated to this purpose and clearly explains the type of data we have access to. It is essential to define the scope of what we have access to such that the effective choice of features explained in Chapter 5 is understandable.

2.1 Input instance

ORTEC has several clients worldwide, which are suppliers of goods in a given area. Each day, the supplier sends the orders from their various customers to ORTEC in a specific file format. The information provided in this file is divided into three parts and are summarized here.

2.1.1 Vehicles

In this section of the file, we have all the data related to the vehicles available for this instance. Each vehicle is characterized by the following:

- Id of the vehicle;
- The earliest time of departure of the vehicle; linked to the id of the departure depot
- The latest time to end the route for the vehicle; linked to the id of the arrival depot
- The preparation duration;
- The priority of the vehicle;
- The capacities of the vehicle;

  Expressed in given units.
- The maximum duration of the trip;

- The break rules;

  Information about the duration of the breaks and the time window in which they can be taken.

- The cost associated with the vehicle;

  By route, kilometre, task, hour or overtime hour.

- The travel time factor;

  Describes the experience of the driver at driving. A higher time factor means lower speed.

- The service time factor when handling;

  Describes the experience of the driver at handling packages at customers. A higher time factor means a lower speed of service.

2.1.2 Tasks

In this section of the input file, information about the different orders of the customers is shown. We thus have access to the following:

- Id of the order;
- Address of the order;

  In the form of a mapLocationId encoded in base64 and/or latitude and longitude.

- The amount that has been ordered;

  Expressed in the specific unit values of ORTEC's customers.

- The start and end time of the time window of delivery;

  Expressed as the date format.

- Duration of the service at the address;

  Expressed in seconds

2.1.3 Depots

In this section, the file contains all the information related to the different depots available for this particular instance. Some customers of ORTEC have one main depot only, while others have one and several sub-depots. Information that can be found in these files is:

- Id of the depot;
- Address of the depot;

  In the form of a mapLocationId encoded in base64 and/or latitude and longitude.

- Loading duration at the depot;
- Unloading duration at the depot;

2.1.4 Example of input

The geographical information of an instance can be summarized on a map, as in Figure 2.1.

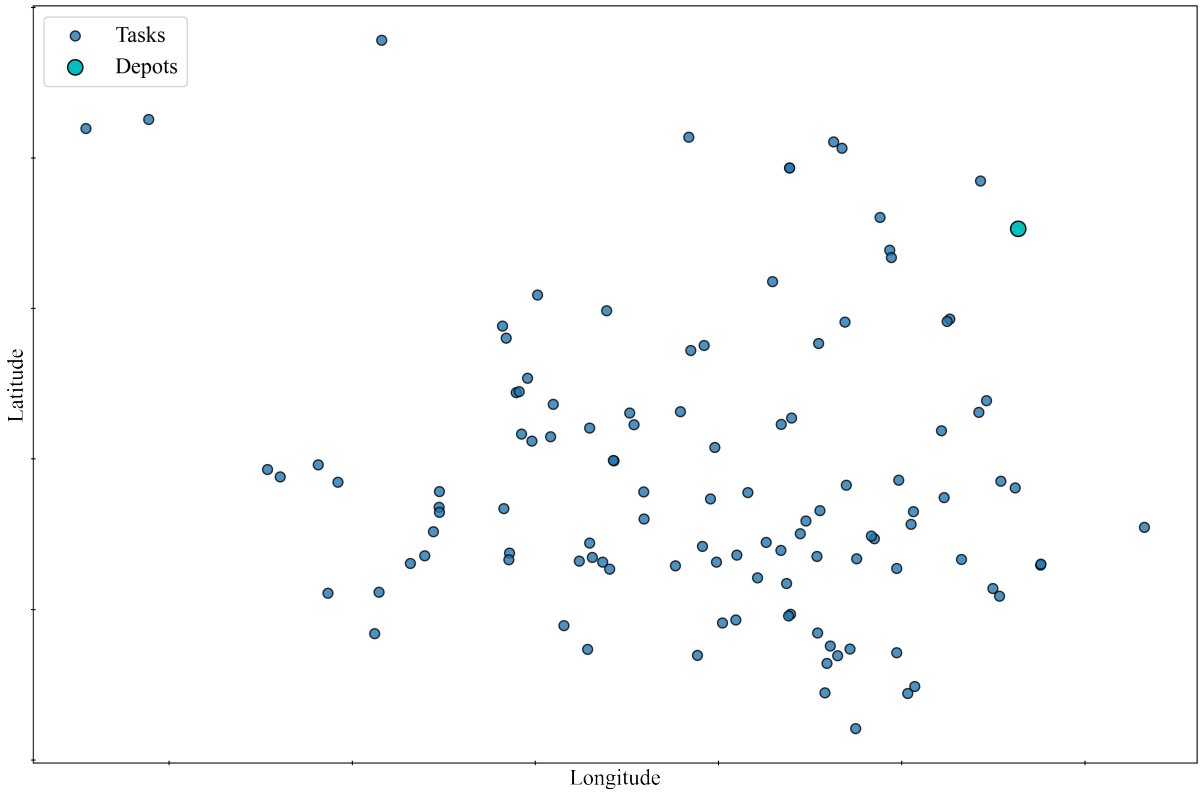

Figure 2.1: Spatial distribution of the customers and depot

As we can see, some customers are close to each other while others are far apart. Also, we can distinguish the position of the supplier's depot, which might not be in the centre of the instance, as is the case here.

Figure 2.2 shows an example of the different time windows available for the customer to deliver. In that example, we had 2888 customers, and the number of customers that chose a specific time window is reported in each cell. The different time windows available vary between each ORTEC's customer. For example, ORTEC's customer 1, which is a supplier, offers 9 different time windows to choose from, while other ones might propose a different number of distinct time windows.

The cost per route is reported in Figure 2.3 for an example instance. In that example, we see a total of almost 300 vehicles available; among them, 25 have a fixed cost of 0 (arbitrary units) to start the route, and 175 have a fixed cost of 2000 (arbitrary units) to start the route. Around 100 have a cost per route of almost 4000 (arbitrary units). Of course, the discrepancy in the cost might be different for other cases. Similarly, the number of different costs can also be different.

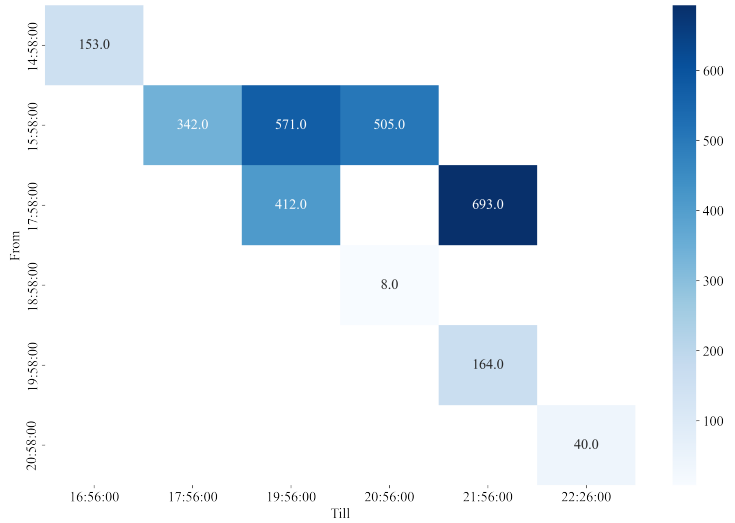

Figure 2.2: Example of distribution of time windows for delivering a customer

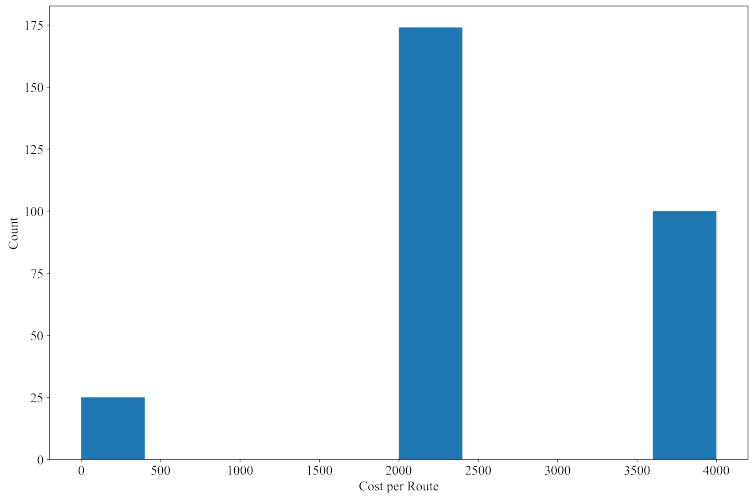

Figure 2.3: Example of the distribution of the different costs per route.

2.2 Output file

From the input information, ORTEC's work is to optimize the delivery routes to meet the different objectives set. There is a total of 8 objectives that are optimized from most to least important. The first objective that must be maximized is the number of planned tasks. It is important to specify this as a first objective; otherwise, we would always say that the best solution that costs less is to not deliver to anyone. However, then we would not have solved anything as these customers would not be served. Having this objective as the first objective also ensures that we are in the realm of general VRPs problems and that reducing the number of tasks to be done is not a possibility to improve the solution. The second objective is the user-defined objective of total cost and vehicle wait time combination. The third objective is the total cost of the delivery plan, which needs to be minimized. This cost is calculated taking into account the costs related to vehicles and the drivers previously introduced. The fourth objective that the algorithm tries to optimize is the total distance travelled. The fifth one is the total number of vehicles used. We then have the total route duration (taking into

account the service time, the pause times, etc.), the total driving time, and the total wait time that all need to be minimized. This optimization is done in several steps that will be better-understood thanks to a thorough literature review in Chapter 3 and a detailed discussion of OHD in Chapter 4. Once the optimizer has found a complete/incomplete solution, it outputs information in several specific files. Two types of files contain the information that we need. The first is a report on each algorithm step with a given timestamp. The second one is a response file structured in several different sections. The following parts of the section explain the details of the second output.

2.2.1 KPIs

In this file section, several Key Performance Indices (KPI) of the general output solution are given. These include :

- Additional costs;
- Break duration in seconds;

  Total break duration in seconds, summed over the fleet of vehicles used.
- Penalty of capacity;

  Penalty if the capacity of a truck in the final solution is exceeded. It should be 0 due to the constraints but is sometimes relaxed for specific clients' needs.
- Cost;

  The total cost of delivering to every customer. The output that comes partially as the second objective and fully as the third.
- Distance in meters;

  Total distance travelled by all trucks together.
- Driving duration in seconds;

  Total time of driving.
- Maximum distance per trip penalty;

  Penalty received if a trip exceeds the maximum distance allowed. It should be 0 due to the constraints but is sometimes relaxed for specific clients' needs.
- Number of planned tasks;

  The total number of tasks planned. For a complete solution, it should be equal to the total amount of initial orders.
- Number of used routes;

  The total number of vehicles used to deliver to every customer.
- Overtime in seconds;
- Duration of the route in seconds;

- Penalty of the finish time of route;

  Penalty received if a trip exceeds the end time allowed. It should be 0 due to the constraints but is sometimes relaxed for specific clients' needs.

- Penalty over time-window;

  Penalty received if a trip exceeds the allowed time window. It should be 0 due to the constraints but is sometimes relaxed for specific clients' needs.

- Duration of the wait in seconds;

- Penalty of the work time;

  Penalty received if a trip exceeds the maximum work time allowed. It should be 0 due to the constraints but is sometimes relaxed for specific clients' needs.

2.2.2 Not planned tasks

This short section reports an array containing the unplanned/unassigned tasks after optimization. As the first objective of the optimizer is to assign all tasks, this array should be empty. However, it may not be the case because of too strict constraints within the instance. The non-delivered tasks are then stored in that array.

2.2.3 Routes

In this section of the file, the different optimized routes are reported. Each route is associated with several different characteristics.

- Route id;

- Summary of the steps of the route step by step;

  Only depotId, taskId and activity types are reported here.

- Extensive detail about the steps of the route;

  Includes the truck's arrival time at a task, starting time of the task, finish time of the task, break duration, driving time, waiting time, and distance travelled since the last point.

- Any violations of the constraints that should be reported;

- KPIs for one route only;

  Same information as the general KPI but for a given route.

2.2.4 Example of output

An example of a solution for the input given in Figure 2.1 is shown in Figure 2.4. This is the best solution the optimizer has found in the given time.

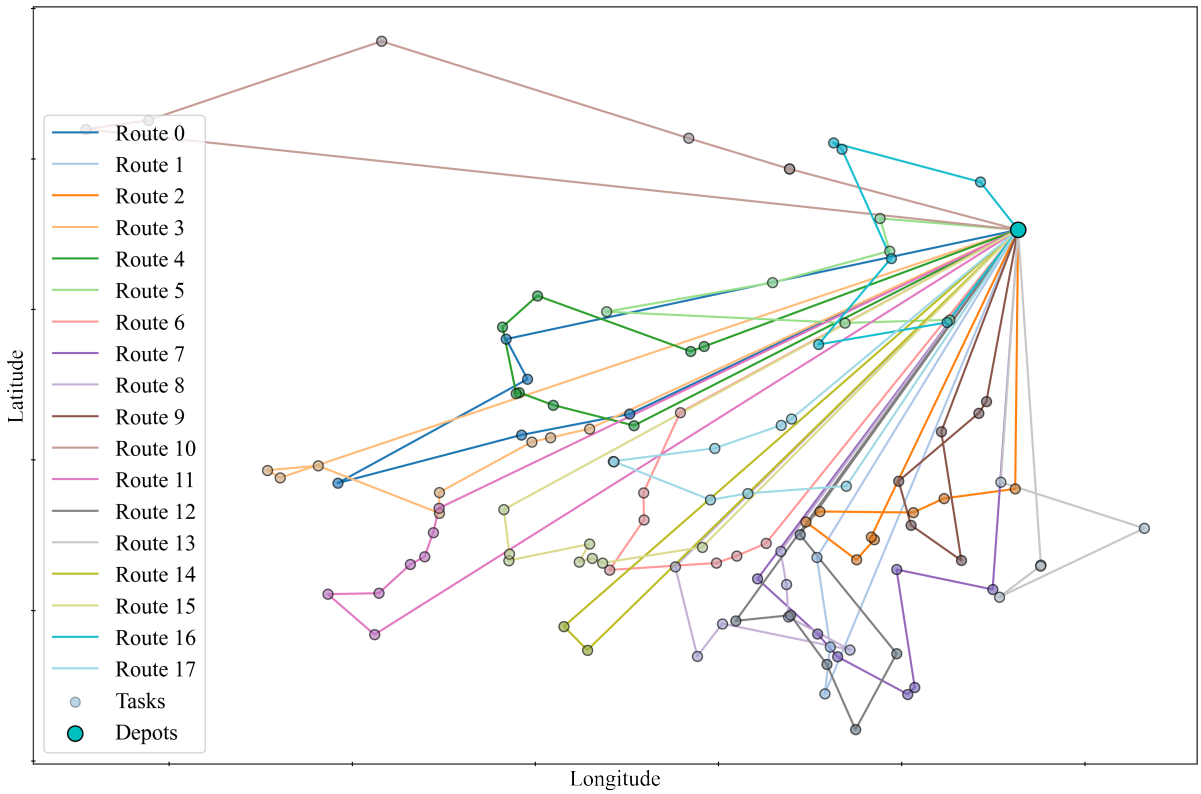

Figure 2.4: Spatial distribution of the customers, depots, and the routes output by OHD

Table 2.1 shows an example of the output KPIs we might end up with. As we can see, this solution did not receive any penalty, and all input tasks were planned. A total of 18 vehicles were used when there were 24 vehicles available.

Finally, as mentioned before, it sometimes happens that the algorithm cannot find a complete solution in the sense that not all tasks are planned. This leads to so-called incomplete solutions, and an example is shown in Figure 2.5.

Table 2.1: Example of values output by the algorithm

KPI's	Value
Additional costs	0
Break Duration in Seconds	0
Capacity Penalty	0
Costs	4921.4354
Distance in Meters	849566
Driving Duration in Seconds	94733
Max Distance per Trip Penalty	0
Number of Planned Tasks	113
Number of Used Routes	18
Overtime in Seconds	0
Route Duration in Seconds	162673
Route Finish Time Penalty	0
Time Window Penalty	0
Wait Duration in Seconds	140
Work Time Penalty	0

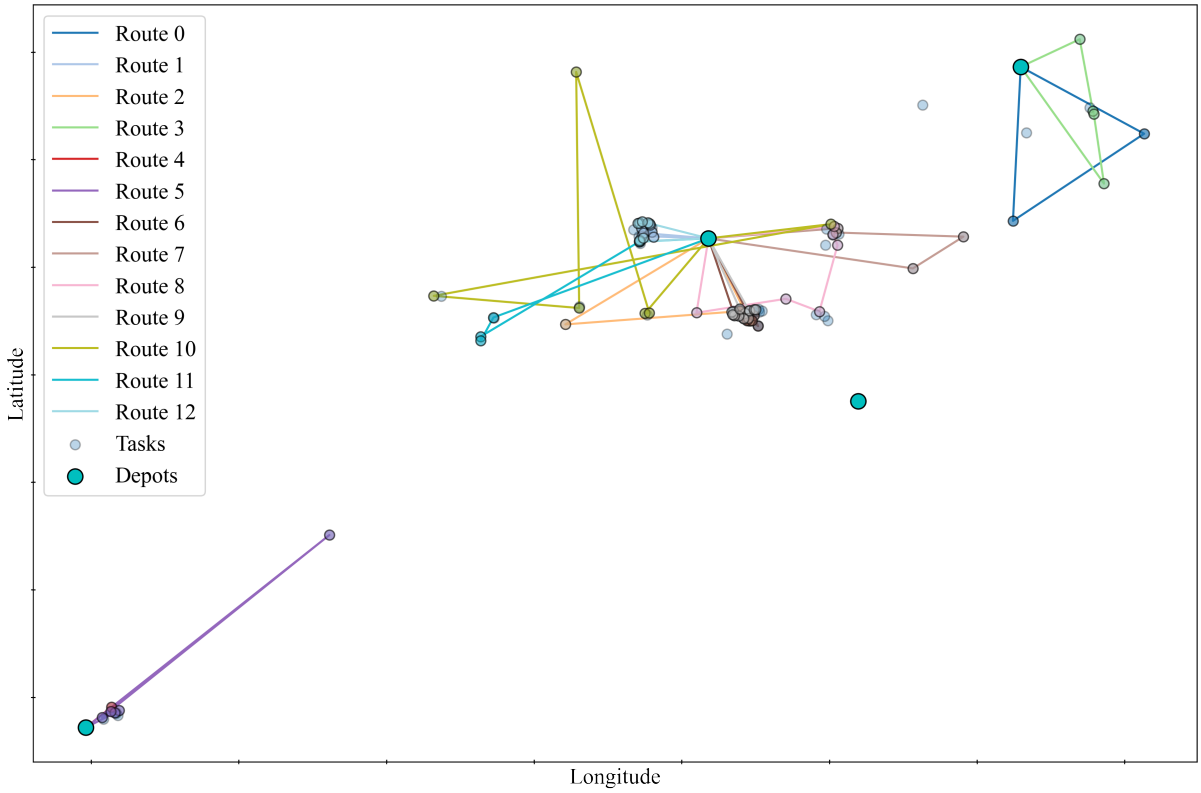

Figure 2.5: Example of an incomplete solution given by OHD

Chapter 3

Literature review

3.1 Existing algorithms

This section will give an overview of some existing techniques used in solving the CVRPTWs. Generally, the techniques can be divided into two major classes: exact and heuristic. Historically, the exact methods were of primary interest. However, as the sizes of real-life problem instances grow, the current research focus has shifted towards the methods that use heuristics (Bräysy and Gendreau, 2005). For that reason, we will discuss the heuristic methods in greater detail, and it also serves the purpose of preparing the reader for the steps in the OHD algorithm that will be explained later in Chapter 4.

For the consistency of notations used, we will define the CVRPTW as in Bräysy and Gendreau (2005):

In graph-theoretic terms, an instance of CVRPTW is a connected, bidirectional graph $G = (V, E)$ with $n+1$ vertices. n of those vertices are the customers and 1 depot. The weight of the edge connecting vertex i and j is denoted by d_{ij}. In reality, this corresponds to the distance between the vertices. An edge connecting any i and j also has t_{ij}, corresponding to the travel times, including the service time at the customer. There is also a time window when the customer can be served. In general, the driver can arrive early and wait until the time window starts at no cost, but they cannot arrive after the end of the time window. Finally, every customer has a service requirement q_i, which can signify, for example, the volume or weight of the goods being delivered/picked up, and all the vehicles have capacity Q. The capacity of any vehicle at any time cannot exceed Q. In many articles, the first objective is to minimize the number of vehicles/routes used followed by travel time or distance. This sequence of objectives might be different in practice. For example, as discussed before, at ORTEC, the number of orders delivered is maximized. Then the total cost and vehicle wait time combination, total cost are minimized, followed by the other six objectives.

Although this definition of CVRPTW might seem simple, many of the techniques discussed in this chapter are more general and will be the essential parts of the OHD algorithm.

3.1.1 Exact methods

One of the earliest attempts to solve CVRPTW exactly is by Kolen et al. (1987). They use the well-known Branch and Bound method that solves the problem by solving smaller sub-problems. Their algorithm solved instances with 6-15 customers and vehicles with the same capacity in 1 minute on average.

Column generation technique from linear programming was used by Desrochers et al. (1992). First, they formulate the CVRPTW as a set partitioning problem. Then, linear programming relaxed set covering problem is solved using column generation. Finally, they use a branch and bound to generate a feasible solution for the set partitioning problem. It performed best for instances with clustered customer locations and up to 100 customers, significantly improving from the previous paper. Similar to the Kolen et al. (1987) paper, the capacities of the vehicles are taken to be the same.

A new approach using Lagrange relaxation, another well-known optimization technique, was introduced by Kohl and Madsen (1997). To be precise, their method uses the Lagrange relaxation of the assignment condition, which states that one customer can be served by one vehicle only. It works by reducing the problem into one Shortest Path Problem with Time Windows and Capacity Constraints (SPTWCC) thanks to the assumption that the vehicles have the same capacity, which could be realistic in real-life scenarios. As a result, the authors solved previously unsolved instances with up to 100 customers, including those not solved by the Desrochers et al. (1992). They also improved the time for some of the already solved instances.

Fisher et al. (1997) introduce two methods. Both of them use the Lagrange relaxation techniques. The first one utilizes variable splitting, which leads to two subproblems, one of which is an easy semi-assignment problem. The second one, similar to Kohl and Madsen (1997), can be solved using SPTWCC. Another approach the authors took is to extend the previously known 1-tree approach, which is used to solve Traveling Salesman Problem (TSP) to K-tree. Similar to the previous papers, both of them were able to solve instances with up to 100 customers; however, the vehicles having different capacities this time.

3.1.2 Heuristic methods

As mentioned above, all of the methods we see in practice use heuristic methods in solving CVRPTW due to the big sizes of instances in real-life applications. They are also good at balancing the time it takes to reach approximate solutions and the solution quality.

In the following parts of the literature review, we will first discuss how the initial solution (possibly not complete or optimal) can be constructed and how it is improved using two major classes of techniques: *Local Search* and *Ruin and Recreate*.

Solution construction

In our opinion, the paper by Solomon (1987) is the most important in discussing the solution construction as it introduces four different ways to achieve that. Moreover, the significance of the article can be seen from the number of subsequent articles written by other authors that build their algorithms on top of the ones given by Solomon (1987). So, we will briefly overview all the methods provided in the paper.

Savings Heuristic

It is an extension of an algorithm from the paper of Clarke and Wright (1964). The original algorithm was developed to solve classical VRP without time windows. The algorithm starts by assigning each customer to its own route. Then, the routes are merged so that the first customer i in one route comes after the last customer j in another route (Figure 3.1). Routes with the highest cost savings (regarding the distance travelled) are

merged first. Mathematically cost saving is:

$$S_{ij} = d_{i0} + d_{0j} - \mu d_{ij} \text{ where } \mu \geq 0$$

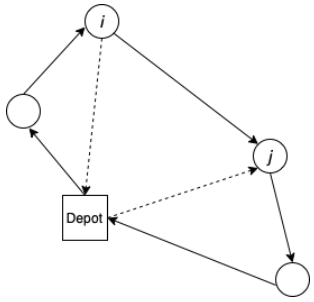

Figure 3.1: Savings heuristics

Here d_{ij} is the distance between the customers i and j as described at the beginning of the chapter, and μ controls how conservative we are in accepting the merge. In case μ is large, we are less likely to get big cost savings, thus, less likely to accept the merge.

The merging can be done in a sequential or parallel fashion. In sequential merging, we consider one route to merge into at a time, and in a parallel version, the best routes over all existing ones are merged. These steps are done until no more feasible merge is possible. Solomon (1987) extends this to CVRPTW by using *push forward*, which he defines as the time difference between the start of the customers before the merge and after the merge. This helps in efficiently testing the time feasibility.

In computations, the author also sets some restrictions on the wait times at the customers.

Nearest-Neighbor Heuristic

This sequential algorithm is quite an intuitive method that anyone with some quantitative education can devise. In this heuristic, one customer who is *closest* to the depot will initialize a route. Then other unrouted customers who are *closest* to the last customer in the route will be added given that it is feasible, respecting the time window and capacity constraints. When we can no longer add new customers to the current route, a new route is initialized similarly. This process is repeated until all customers have been assigned to routes.

The *closest* customer is the customer having the smallest value of a particular convex combination of geographical distance, the temporal closeness between customers i, j, and the urgency of the delivery j.

Insertion Heuristics

It is another set of sequential methods. The author provides three different ways of initializing the routes and three ways of inserting subsequent customers. In literature, the insertion techniques are commonly called $I1$, $I2$, and $I3$.

Assuming the routes were initialized (we will explain the process later), the following is how the insertions work:

I1

Assume that the current route we are looking at is $(i_0, i_1, ..., i_m)$ where $i_0 = 0$ and $i_m = 0$ (0 is the depot index). First, for every customer, the best position where the customer can be feasibly inserted is identified. Similar to the Savings heuristics, push forwards are used to

check time-window feasibility efficiently. For each user u, we solve the following:

$$\begin{aligned} \underset{p}{\operatorname{argmin}}\, c_1(i_p, u, i_{p+1}) \text{ for } p = 0, .., m-1 \\ c_1(i,u,j) = \alpha_i(d_{iu} + d_{uj} - \mu d_{ij}) + \alpha_2(b_{ju} - b_j) \\ \mu, \alpha_1, \alpha_2 \geq 0 \text{ and } \alpha_1 + \alpha_2 = 1 \end{aligned}$$

Here b_j is the scheduled arrival time at customer j before the insertion of the user u and b_{ju} after.

In other words, the best position to insert the customer u is calculated by taking the minimum of a particular convex combination of the extra distance travelled and the extra time required to visit the customer u.

After we have selected the best insertion place as the one between customers i and j for the customer u, we select the best customer to insert. That is done by maximizing the cost saving we can get when inserting customer u compared to if it was scheduled on its own. Mathematically:

$$\begin{aligned} \underset{u}{\operatorname{maximize}}\, c_2(i,u,j),\ u \text{ unrouted} \\ c_2(i,u,j) = \lambda d_{0u} - c_1(i,u,j) \\ \lambda \geq 0 \end{aligned}$$

As with the μ parameter in *I1*, λ controls how much importance we want to give to d_{0u} in the cost saving.

I2

This insertion approach is similar to the $I1$: the best insertion position computation stays the same. However, $c_2(i,u,j)$ is now chosen to minimize the convex combination of total route distance (R_t) and time when customer u is inserted R_d. Mathematically, we need to solve the following:

$$\begin{aligned} \underset{u}{\operatorname{minimize}}\, c_2(i,u,j),\ u \text{ unrouted} \\ c_2(i,u,j) = \beta_1 R_t(u) + \beta_2 R_d(u) \\ \beta_1, \beta_2 \geq 0 \text{ and } \beta_1 + \beta_2 = 1 \end{aligned}$$

I3

This insertion method can be seen as the generalization of the Nearest-Neighbor heuristic as $c_1(i,u,j)$ now also includes the urgency measure of j. The difference is, however, now we are not restricted to insertions only to the ends of the routes. Also, $c_2(i,u,j) = c_1(i,u,j)$

In all the insertion methods, if there are remaining customers and the route cannot take any more customers, a new route is initialized, and the insertion steps are repeated.

The routes are initialized in three different ways:

- Customer farthest from the depot ($I1$,$I2$, $I3$)
- Customer with the earliest deadline for delivery ($I1$,$I2$, $I3$)

- Customer with the least average distance to the depot and the travel time ($I2$, $I3$)

Sweep Heuristic
This insertion heuristic consists of two stages: clustering and scheduling. First, the customers are clustered depending on the polar angle from the depot; after that, any initialization and insertion heuristics can be used for each cluster. The original paper uses $I1$ to solve the problem for each cluster. Then, if unscheduled customers remain, we repeat the process.

There is a large number of articles written on extensions of these algorithms. We will mention a couple of them here that are interesting and frequently appear in the literature reviews of many other articles.

Potvin and Rousseau (1993) parallelize the $I1$ heuristics by selecting m customers as the initializations of m routes. These customers are found by running the $I1$ algorithm once and selecting the farthest customer in each route. Thus m is found automatically. Then for each customer u, they calculate the best position to insert in each route in the same way as in $I1$. Now, to select the best customer, they use regret measures. Thus, $c_1(i, u, j)$ is defined as before, but they select the user with the maximum regret $c_2(i, u, j)$, which is defined as the sum of differences between the cheapest insertion and all other insertions over all routes. The regret measure is intuitive to consider here as we should be inserting the customer into the route, which would result in a high cost when inserting somewhere else.

Ioannou et al. (2001) propose another type of sequential insertion heuristic. There, a customer farthest from the depot initializes a route. Then, the feasible customer with the lowest impact on the time windows of all customers (routed, unrouted, and themselves) is inserted. If the route is full, initialize a new route in the same way and repeat the insertion steps. These steps are repeated until no customer is left to be scheduled.

Local Search

Now that we have discussed some ways of constructing initial (partial) solutions, we will summarize the techniques that can be used to improve the current solution locally. Local search is an application of some *operator* on the solution to find a neighbouring solution that structurally does not differ from the current one dramatically. Examples of an operator can be moving some customers from one route to another, exchanging customers' orders in the route, and so on. If the operator leads to a better solution depending on the chosen criteria, we can accept it and repeat those operators iteratively. Usually, applying the operator once can lead to multiple new solutions, not just one. Depending on whether we choose the first solution we see, which is better or the best one from the set, there are two ways of accepting the new solution: first-accept (FA) or best-accept (BA).

Some noteworthy *operators* include:

Move/Relocate
This basic operator moves a group of customers to another place in the same or another route.

Swap/Exchange
It is another basic operator that swaps two groups of customers from two different routes.

2-opt

This edge-exchange method was first introduced by Croes (1958) in the context of TSP. The intuition is to eliminate the self-intersections in the single route by removing the two intersecting edges and adding two edges in another place (Figure 3.2). The 2-opt can be used in an inter-route manner in CVRPTW, as there can be more than one route.

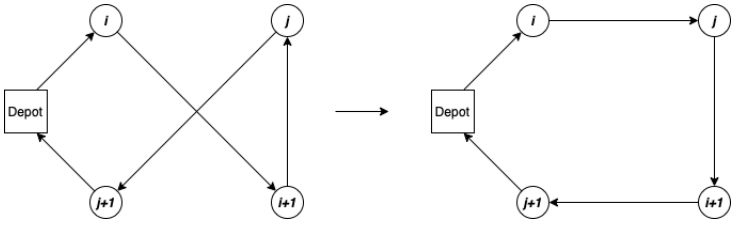

Figure 3.2: 2-opt

In Figure 3.2 the edges $(i, i+1)$ and $(j, j+1)$ were replaced by (i, j) and $(i+1, j+1)$.

3-opt

Lin (1965) generalized 2-opt to k-opt, where other k edges replace selected k edges. In particular, he introduces the application of 3-opt in TSP.

Or-opt

Now instead of exchanging edges or moving any group of customers, we can move the chains of customers to another place (inter/intra-route). One operator used extensively in practice is or-opt introduced by Or (1976). It considers only chains of at most three customers. Figure 3.3 shows an example of Or-opt.

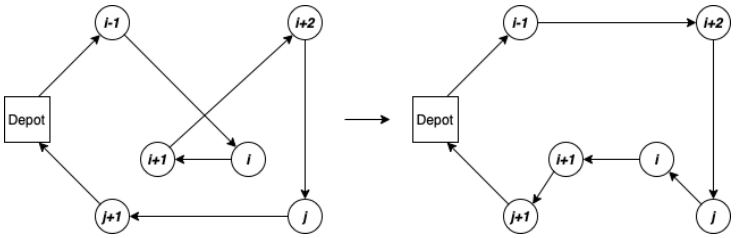

Figure 3.3: Or-opt

*2-opt**

We have seen that in the example we gave for 2-opt, the order of the customers i and $i+1$ has been changed. To be precise, before the 2-opt, customer i was served after $i+1$, but after the 2-opt, $i+1$ is served after i. So, there is a high likelihood that time window feasibility might be violated. To account for that, Potvin and Rousseau (1995) proposed inter-route 2*-opt, which does not allow order reversals. Figure Figure 3.4 gives an example of 2-opt*. In this example, there is no case of order reversal; the original order is kept the same wherever applicable.

CROSS-exchange

It was first introduced by Taillard et al. (1997), and similar to 2-opt*, this inter-route operator can preserve the order of customers. First, edges $(i, i+1)$ and $(j, j+1)$ are removed from the first route, and edges $(k, k+1)$ and $(l, l+1)$ are removed from the second route. Now that we have an isolated chain of customers between i and j in the first route and between

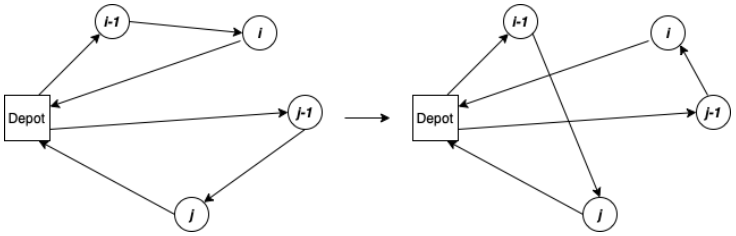

Figure 3.4: 2-opt*

k and l in the second route, we can simply exchange these chains of customers between the two routes (Figure 3.5).

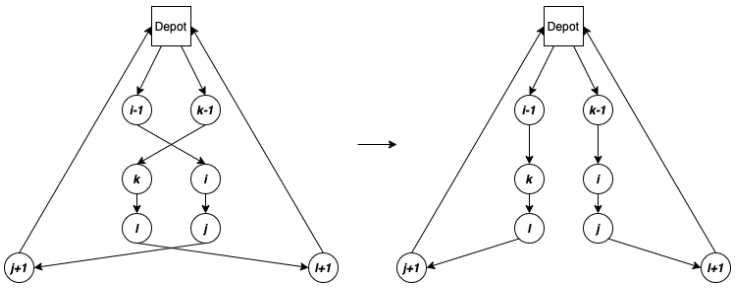

Figure 3.5: CROSS-exchange

Prosser and Shaw (1996) have extensively researched how different operators influence the algorithm's performance. The operators they have tested include intra-route 2-opt, relocate, and exchange. The authors claim that the operators' performance is quite good, considering their lack of sophistication.

Besides introducing 2-opt*, Potvin and Rousseau (1995) have reported some computational results based on a hybrid approach of 2-opt*/or-opt. Relating to the solution construction part of the section, the authors also use $I1$ heuristics from Solomon (1987).

As the name suggests, local search works well when we want to reach the local optimum of the problem by making minor adjustments to the current solution. So, a local search may not be enough to find the global minimum. One way to "escape" the local minimum is to occasionally accept worse solutions instead of accepting better ones in a greedy manner, as we did in the local search. That will help us move away from the local minimum in small steps. This method of "escaping" is commonly referred to as metaheuristics. Another alternative is to use Large Neighborhood Search (LNS), which focuses on searching for solutions that might differ from the current one dramatically and "far" in the solution space.

In the following sections, we will introduce one metaheuristic called Simulated Annealing and Ruin and Recreate (RR), an example of LNS.

Simulated Annealing

Simulated Annealing (SA) is a method that selects worse solutions at every step of the search with a certain probability. To be precise, let $\Delta C = C_{new} - C_{current}$ be the difference in the costs of current and new solutions. If $\Delta C \leq 0$, then we accept the solution, else we accept the new solution (which is further from optimality) with the probability of $f(\Delta C, T) = exp(-\Delta C/T)$. Here, the new, worse solution is accepted with a high probability if ΔC is small or T is large.

Usually, T is a function of time/iteration number $T(i)$, so we can control the probability of acceptance after different numbers of iterations. At the beginning of the search, we would like to explore the solution space, so T needs to be kept high. Then, as the number of iterations increases, we decrease the T to be more conservative in accepting less optimal solutions. In simpler terms, T controls the phases of exploring and exploiting part of the technique. $T(i)$ can be linear, exponential, hyperbolic, or logarithmic, depending on how we want $T(i)$ to decrease.

Ruin and Recreate

In this LNS method, the main idea is to remove some customers (Ruin) from the solution and try to reinsert them again (Recreate). Christiaens and Vanden Berghe (2016) makes a good connection between the local search operator we have introduced before and the RR framework. Operators such as k-opt, Or-opt, and CROSS-exchange can be considered small-scale RRs where we make minor local changes to a small number of customers/chains by removing (ruining) and relocating (recreating) them. According to Christiaens and Vanden Berghe (2016), the initial idea of *Rip-Up and Reroute* was already introduced by Dees and Smith (1981) in the context of wiring algorithms. Then LNS was introduced by Shaw (1998), essentially the RR application in VRP. However, we will review RR given in Schrimpf et al. (2000) where the authors introduce RR and provide its application in CVRPTW.

Ruin

Assuming that we have already constructed an initial solution and possibly used a local search, we can ruin some parts of our solution. There are several ways to do that: random ruin, radial ruin, and sequential ruin.

- Random ruin removes each customer with the fixed probability p
- For the radial ruin, select a random customer u in the solution, and remove pn closest customers to the customer u. Where p is the predefined fraction parameter
- Sequential ruin selects a random route and a customer on that route. Then remove the following pn customers on the same route

Pisinger and Ropke (2007) also introduces many new ways of ruining the solution, which the authors call "removal." One of them is *time-oriented removal*, where the customers with similar time windows are removed together, making it more realistic that reinserting them will be easier.

Recreate

As mentioned in Schrimpf et al. (2000), there are many ways of reconstructing a solution, and the article focuses on building the framework. For example, in our case, we could use different insertion heuristics introduced earlier in this section. However, the paper gives only one way of inserting *best insertion*, which has been shown to perform much better than previous heuristics despite its simplicity. In best insertion, we randomly pick one of the removed customers and insert them into the cheapest position in the solution. We repeat that until there are no customers left. If there is no feasible position to insert the customer, we initialize a new route.

Pisinger and Ropke (2007) also provides their regret measure according to which the customers are inserted.

Accept/Reject rule
After another solution was generated by Ruin and Recreate steps, we have the freedom to choose how we will accept or reject it. For example, one method could be Simulated Annealing, as the article suggests. However, in the computational part of the article, the authors have used Threshold Accepting, which accepts the new solution if $\Delta C \leq T$ where T is adjusted during iterations.

In this section, we have given an extensive literature review of the exact and state-of-the-art heuristic methods, some of which are essential for understanding the OHD algorithm's steps. Next, we will explore some of the features of the CVRPTW problem instances that could influence the performance of the heuristic algorithms.

3.2 CVRPTW instance features

As in every problem, we would want to keep all the information possible from a given instance to describe it. However, that is not a practical way of building models as it takes a lot of storage space. We are thus required to find clever ways to describe the instance. In that sense, we will look at features that have great explainability such that they are easily grasped by any non-expert reader or decision-making person at ORTEC. This is where feature engineering comes into play. Feature engineering aims to develop new ways to combine our initial data into fewer variables that bring the same amount of information. Some features are related to a broader class of VRP problems, while others describe the capacities or time windows in the problem as identified by Rasku et al. (2016). The extraction of different features in VRP instances has been extensively researched in the literature. This section will be dedicated to a literature review of these features and developing their computation.

3.2.1 Location based features

Probably the most logical type of feature that comes to mind to describe a capacitated vehicle routing problem with a time window is the description of the location of the different customers. As explained by Jiang et al. (2021), we can roughly divide the features extracted from locations into customer distribution, cluster, and graph-based features.

Customer distribution features

Smith-Miles and van Hemert (2011) first introduced a set of features that are derived from the TSP instance. They consist in :

- the number of customers n (Mersmann et al., 2012; Smith-Miles and van Hemert, 2011; Hutter et al., 2014; Pihera and Musliu, 2014; Kanda et al., 2016; Rasku et al., 2016; Steinhaus, 2015) ;

- the standard deviation of the distance matrix (Smith-Miles and van Hemert, 2011; Hutter et al., 2014; Pihera and Musliu, 2014; Rasku et al., 2016; Kanda et al., 2016; Steinhaus, 2015);

This work defines the distance matrix D as a matrix containing each distance d_{ij} between customer i and customer j.

- the coordinates of the instance centroid (Smith-Miles and van Hemert, 2011; Hutter et al., 2014; Pihera and Musliu, 2014; Rasku et al., 2016; Steinhaus, 2015);

  It is expressed as a (x,y) tuple containing the centroid's x-coordinate and y-coordinate.

$$x_{centroid} = \frac{\sum_{n=1}^{N} x_n}{N}$$

$$y_{centroid} = \frac{\sum_{n=1}^{N} y_n}{N}$$

- the radius of the instance (Smith-Miles and van Hemert, 2011; Hutter et al., 2014; Pihera and Musliu, 2014; Rasku et al., 2016; Steinhaus, 2015);

  It is defined as the mean distance between each point and the centroid.

- the distance between the centroid of the customer's locations and the depot. (Rasku et al., 2016)

  In Rasku's work, the distance between the centroid of the customer's locations and the depot is taken. However, in our case, we have instances with several depots, so the initial formula would not suit our problem anymore. We thus decided to take the mean of the distances from the depots to the centroid of the customer's locations.

- the fraction of distinct distances existing in the distance matrix (Smith-Miles and van Hemert, 2011; Hutter et al., 2014; Pihera and Musliu, 2014; Steinhaus, 2015);

  This is done by rounding the distances in the distance matrix to 2 decimals and dividing the number of distinct values by the total number of values.

- the rectangular area that frames the instance (Smith-Miles and van Hemert, 2011; Hutter et al., 2014; Pihera and Musliu, 2014; Steinhaus, 2015);

- the variance of the normalized nearest neighbour distances (nNND) (Smith-Miles and van Hemert, 2011; Hutter et al., 2014; Pihera and Musliu, 2014; Rasku et al., 2016; Steinhaus, 2015);

  It is calculated by taking the variance of the distances between each customer and its nearest neighbour for an instance that has been normalized beforehand.

- the coefficient of variation of the normalized nearest neighbour distances (Smith-Miles and van Hemert, 2011; Hutter et al., 2014; Pihera and Musliu, 2014; Steinhaus, 2015);

  This one is the same as above but takes the square root of the variance divided by the mean.

Intermediate customer distribution features

On top of these features, Mersmann et al. (2012) added a few intermediate features based on these initial ones. They mainly added the computation of minimum, maximum, and the median of distance matrix (Mersmann et al., 2012; Hutter et al., 2014; Rasku et al., 2016; Kanda et al., 2016) and nNND's (Mersmann et al., 2012; Hutter et al., 2014). However, they also added new types of features:

- Expected tour length for a random tour (Mersmann et al., 2012);

$$\frac{2}{N-1}\sum_{i=1}^{N}\sum_{j=1}^{N}d_{ij}$$

- Mean, frequency and quantity of modes of the distance matrix (Mersmann et al., 2012; Rasku et al., 2016);
- Minimum, mean, maximum distance of the customer's location from the centroid Mersmann et al. (2012);
- Summary statistics (min, max, mean, median) of the angle formed by a customer and its two nearest neighbours (Mersmann et al., 2012; Pihera and Musliu, 2014; Rasku et al., 2016; Kerschke et al., 2018);
- Summary statistics of the cosine of the angle mentioned above (Pihera and Musliu, 2014; Rasku et al., 2016);
- Convex hull area (Mersmann et al., 2012; Pihera and Musliu, 2014; Rasku et al., 2016; Kerschke et al., 2018);

  It is defined as the envelope framing every customer in the instance. Outside customers are defining the boundary of the convex hull.

- Fraction of customers that define the convex hull (Mersmann et al., 2012; Pihera and Musliu, 2014; Rasku et al., 2016).

Advanced customer distribution features - Probing features

With the knowledge of these features, a new set of advanced characteristics of the instances has been proposed by several authors to describe the problem instances further. These features are related to small local search or branch and cut algorithms used to grasp the difficulty of the instance. The algorithm and the number of steps used vary between authors. Still, the principle stays the same: allow only a few steps in each algorithm, repeat the algorithm a given number of times and take summary statistics. These probing features are thus extracted from routing algorithms. So it is expected that the values gathered from those are both a characteristic of the instance and the algorithm used. However, as we assume that the algorithm used is fixed for the whole feature extraction process, the value of that feature can still be used as a general knowledge of how the given routing algorithm performed on that instance. This can still give us an insight into how our complete algorithm would perform on the same instance, as it usually includes such routing algorithms. Here is a detailed list of the parameters extracted from it.

- Tour cost from construction heuristics: mean, coefficient of variation and skewness (Hutter et al., 2014; Pihera and Musliu, 2014);

  Cost, usually expressed in distance, of the solution found by the algorithm in the given number of steps.

- Local minimum tour length (Hutter et al., 2014; Pihera and Musliu, 2014; Rasku et al., 2016);

  The minimum distance achieved by the algorithm for this instance.

- Improvement per step (Hutter et al., 2014; Pihera and Musliu, 2014; Rasku et al., 2016);

  Quality improvement of the tour for each step of the algorithm.

- Mean, standard deviation and skewness of number of steps to local minimum (Hutter et al., 2014; Pihera and Musliu, 2014; Rasku et al., 2016);

  The number of steps to reach a local minimum.

- Mean, standard deviation and skewness of the distance between local minimum (Hutter et al., 2014; Pihera and Musliu, 2014);

  Hamming distance between two local minima. The Hamming distance is defined as $D_H = \sum_{i=1}^{k} |x_i - y_i|$.

- Probability of edges in local minima (Hutter et al., 2014; Pihera and Musliu, 2014; Rasku et al., 2016);

  The probability of edges appearing in any local minima met during the research.

- Number of improving steps (Hutter et al., 2014; Pihera and Musliu, 2014);

- Mean, coefficient of variation and skewness of the improvement per cut in a branch and cut algorithm (Hutter et al., 2014; Pihera and Musliu, 2014; Rasku et al., 2016);

  Based on the branch and cut algorithm, the measure of the improvement of the lower bound per cut

- Ratio of upper bound to lower bound of the branch and cut algorithm (Hutter et al., 2014; Pihera and Musliu, 2014; Rasku et al., 2016);

  The ratio of upper and lower bounds found at the end of the search.

- Percentage of integer values and non-integer values in the final solution of the branch and cut algorithm (Hutter et al., 2014; Pihera and Musliu, 2014);

- Mean, standard deviation and skewness of the number of tour intersections in a plane (Hutter et al., 2014; Pihera and Musliu, 2014);

- Autocorrelation coefficient (Hutter et al., 2014; Pihera and Musliu, 2014; Rasku et al., 2016).

  Used to assess the ruggedness of the function we are looking to optimize.

These features represent quite an extensive but non-exhaustive list of what has been done in the literature about location-based features. However, two other visualization types of these instances allow us to extract other features. Those will be detailed in the next two subsections.

3.2.2 Cluster-based features

Cluster-based features are extracted from the clustering of the different customer locations. There exist several clustering algorithms out there, but in the VRP literature, DBSCAN (or GDBSCAN) is the one that is usually used. Before going into more detail about the extracted features, we will first explain how the DBSCAN algorithm works.

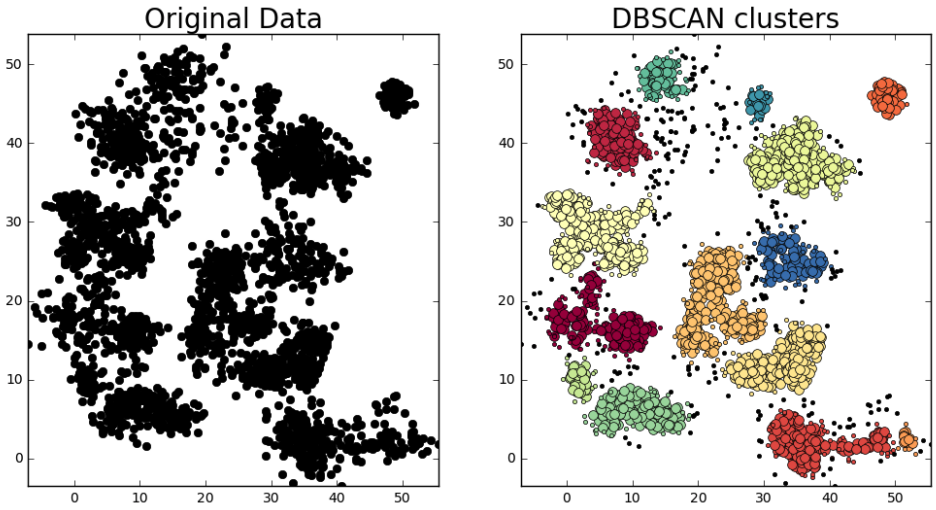

Figure 3.6: DBSCAN clusters (Chris Wernst (2017))

The DBSCAN algorithm is a clustering algorithm based on the density of the points needing to be clustered. Thus, it can find any arbitrary shape of a cluster in the data (Figure 3.6). This algorithm works in two separate steps. First of all, it looks at each point and its neighbourhood. A given point with a number of neighbours that exceeds a given threshold is considered a core point. Otherwise, this point is considered a non-core point. When this step is done, we start by taking a random core point and assigning it to a cluster. The neighbouring points are also assigned to this cluster. If the neighbouring points are core points, they can also extend the cluster to their neighbourhood. However, a non-core point cannot extend the cluster to its neighbourhood. When no more points can be added, we assign another unassigned core point to a new cluster and repeat until every core point is assigned to a cluster. The non-core points left unassigned after this are unclustered points, also called outliers. Two parameters are user-defined when using DBSCAN. First, the distance at which we consider the neighbourhood of a point also called *Eps* parameter. The second is the minimum number of points in the neighbourhood of a point for it to be considered a core point, also called *MinSample* or *MinPts* parameter.

The automatic choice of *Eps* and *MinPts* based on the problem instance is a topic of great research in the literature (Steinhaus, 2015; Sawant, 2014; Sander et al., 1998). Sander et al. (1998) have proposed an algorithm to automatically chose *MinPts* and *Eps* the most suitable to a problem instance. They evaluated that the change in *MinPts* was not impactful for two-dimensional data above a given threshold. Following their work, but also the one from Steinhaus (2015), the value of *MinPts* should be fixed to 2*dimension. In our case, that is latitude and longitude; thus, we have a value of *MinPts* of 4. Going above this threshold does not yield significant differences except for the increase in computation time. However, going

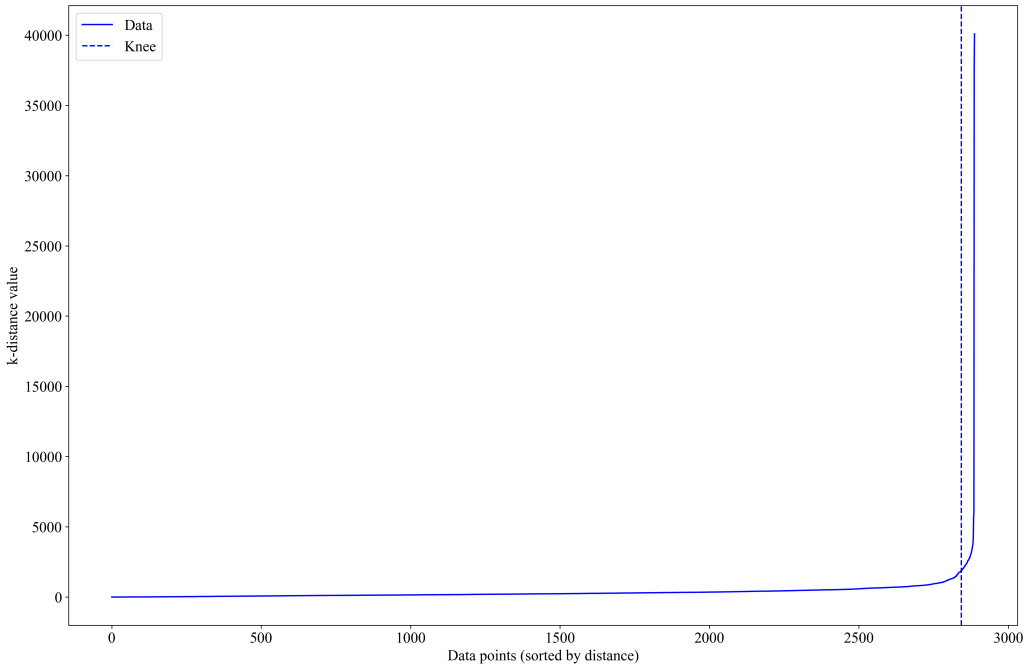

Figure 3.7: 4-distance plot of an example instance with the elbow located in dotted line

for *MinPts* = 1 is not advised as it would mean that each point can become a cluster and thus boil down to extracting the number of customers. The automatic choice of *Eps* is a bit more intricate as it relies on the k-nearest neighbour analysis prior to the DBSCAN algorithm. Here, Steinhaus (2015) and Sander et al. (1998) do not agree on the k to take in the nearest neighbour analysis. The first advises going for k = *MinPts* while the second advises going for k = *MinPts*-1. Sander et al. (1998) evaluated that the selection of k was not crucial for the proper operation of the algorithm. We chose the method of Steinhaus (2015) because it is the most intuitive, recent and represented one in literature (Starczewski et al., 2020; Akbari and Unland, 2016). The next step is constructing an increasing k-distance plot with the k-distance as the y-axis and the customer number as the x-axis. On this graph, an elbow usually appears, separating close points from noise points. Our goal is to find the y-value of the elbow and fix the *Eps* parameter as this value, as every point with a k-nearest distance lower than this value would be considered a core point. The k-nearest distance above this threshold would lead to a border point that might be classified as an outlier.

Both Steinhaus (2015) and Sander et al. (1998) have proposed an interactive approach to choose the threshold based on this graph, where the user has to select it by himself/herself. However, as we plan to automate it completely, we used the Python package *Kneed* based on the *Kneedle* algorithm proposed by Satopää et al. (2011) to determine the knee/elbow in that type of graph automatically, as shown in Figure 3.7. Thus, we can retrieve the y-value of the elbow and fix it as the *Eps* parameter as proposed by Steinhaus (2015); Starczewski et al. (2020); Sander et al. (1998). The DBSCAN algorithm is then ready to run with two parameters adapted to the case we have at hand. We can note that another method was proposed by Sawant (2014) where they first fix the *Eps* parameter and then the *MinPts* one. As fixing the *Eps* parameter was more computationally intensive, we chose to stick with the previously explained method.

The features that could be extracted after clustering are:

- centroids of the clusters (Kerschke et al., 2018);
- the cluster ratio (Smith-Miles and van Hemert, 2011; Hutter et al., 2014; Rasku et al., 2016);

  It is the number of clusters divided by the total number of customers. Clusters are found using the DBSCAN algorithm.
- the outlier ratio (Smith-Miles and van Hemert, 2011; Hutter et al., 2014; Rasku et al., 2016);

  It is the number of outliers divided by the total number of customers. An outlier is a customer not assigned to a cluster during the clustering algorithm.
- the variance of the number of cities in each cluster (Pihera and Musliu, 2014; Hutter et al., 2014; Rasku et al., 2016);
- mean distance to the cluster centroids (Mean radius of clusters)(Mersmann et al., 2012; Steinhaus, 2015).

3.2.3 Graph-based features

The third group is based on graph instances. The concept of Minimum Spanning Tree (MST) appears to be crucial for this group as most features rely on the calculation of such *MST* as introduced in Hutter et al. (2014) and Mersmann et al. (2012). A *MST* is a connected subgraph connecting all nodes with minimal edge weights in between and without any cycles. One of the main reasons to look into this type of feature is because it can provide an upper bound for the optimal tour (i.e. solution of MST is within a factor two of optimal) (Mersmann et al., 2012). The features that are extracted are, in general:

- Summary statistics on the depth of the MST (Kerschke et al., 2018; Mersmann et al., 2012; Rasku et al., 2016);
- Summary statistics on the edge cost of the MST (Kerschke et al., 2018; Mersmann et al., 2012; Hutter et al., 2014; Rasku et al., 2016);
- Sum of the distances in the MST, normalized by the sum of all pairwise distances (Mersmann et al., 2012);
- Maximum degree of nodes (Kanda et al., 2016);
- Node degree (Hutter et al., 2014; Rasku et al., 2016);
- Minimum Spanning Tree from the depot (Rasku et al., 2016).

Again, this is not an exhaustive list of all features tested in the literature, but it gives an idea of the type of features it allows to extract. However, these graph-based features are more challenging to obtain than most location-based features because they first require the computation of the *MST*, which takes a logarithmic time. They are thus less prevalent in the literature compared to location-based features.

3.2.4 Demand, capacity and time window features

Finally, we have a group of features specific to the trucks' capacity, the customers' demand and the time window associated with both the trucks and the client.

This part has been less extensively researched as these features depend highly on the problem characteristics and the data available for each problem instance. We will thus summarise a list of what is usually done in the literature and discuss our features in a later chapter.

- Summary statistics on the demand structure (Steinhaus, 2015; Rasku et al., 2016);
- Ratio of total demand over total capacity (Steinhaus, 2015; Rasku et al., 2016);
- Ratio of maximum cluster demand to vehicle capacity (Rasku et al., 2016; Steinhaus, 2015);
- Ratio of cluster outlier to overall demand (Rasku et al., 2016; Steinhaus, 2015);
- Ratio of largest demand over capacity (Rasku et al., 2016; Steinhaus, 2015);
- Ratio of the average number of clients per vehicle (Rasku et al., 2016; Steinhaus, 2015);
- Minimum number of trucks (Rasku et al., 2016; Steinhaus, 2015);
- Capacity of vehicles (Kolen et al., 1987);
- Total demand (Kolen et al., 1987);
- Number of vehicles (Kolen et al., 1987);
- Description of the delivery time window (Gan et al., 2012);
- Description of the time window width (Cordeau et al., 2002).

Overall, the use of time-window features is scarce in the literature on CVRPTW, probably because it is hard to present these characteristics concisely and because they are unavailable and not used by every algorithm.

This section highlighted quite an extensive list of features, and we have decided not to work with all of them. Instead, we have retained a subset to examine in detail and assess their impact on our target variables to identify the most meaningful ones. These will be introduced in Chapter 5.

3.3 Solution features

The previous section was dedicated to the computation of features before the optimization. However, the solutions we get after optimization can also be summarised. The main characteristics we are looking at in the sense of optimization are the percentage of the tasks delivered for the incomplete solutions and the time it takes to reach the plateau in the optimization process of the OHD algorithm for the complete solutions. However, there are other features that we could extract and predict for, which would maybe better describe the complexity of the optimization.

A list of solution features has been introduced by Arnold and Sörensen (2019) and is reported here:

- Average number of intersections per customer
- Longest distance between 2 connected customers, per route
- Average distance between depot to directly-connected customers
- Average distance between routes (the center of gravity of the route)
- Average width per route
- Average span per route
- Average compactness per route, by width
- Average compactness per route, by radian
- Average depth per route
- Standard deviation of the number of customers/route
- Degree of capacity utilization

These solution features could be interesting to look at later in the analysis to determine the hardness of some instances in a better way.

Chapter 4

OHD algorithm

This chapter will explain the OHD algorithm in more detail without discussing the specifics for confidentiality reasons. Nevertheless, the overview of the steps we will provide should be enough to get a general idea of how OHD works.

The OHD algorithms consist of four stages: **Construction, Local Search, Plan Remaining, Ruin and Recreate**. In the following sections, we will go into more detail on these stages' steps.

Construction

Similar to the *Contruction* stage introduced in Chapter 3, this step's primary goal is to construct an initial solution (possibly incomplete).

Route Sorting

In this step, the instance routes are sorted according to specific hierarchical criteria. Then, following the order of routes, we run the following step of *Plan Seed Task* for each of them.

Plan Seed task

Here, one task is selected as a seed to add to the route according to some closeness criteria. Then, it is inserted by Cheapest (Sequential) Insertion, where the "cheapest" is also defined by some criteria.

Plan Tasks Close to Seed

This step works by sorting the remaining tasks and making batches of k tasks, which are then added to initialized routes by Parallel Insertion. We refer the reader to the Chapter 3 to understand the difference between Cheapest Sequential and Parallel insertions.

Local Search Within Trips

In this step, intra-route Local Search steps are tried with a limited number of recursions. Although we will not give further details, the steps consist of some introduced in Chapter 3.

Plan Remaining Tasks

Here the algorithm makes its last attempt to add more tasks into routes by Cheapest Parallel Insertion with all remaining tasks considered.

By the end of this stage, we are not guaranteed a complete solution, as some of the tasks will not be planned due to their infeasibility during the insertion steps.

Local Search

In this step, the solution is improved iteratively by searching its neighborhood for better solutions. This neighborhood is defined by a number of configurable operators, like the ones discussed in Chapter 3. This step concludes once no more improvements can be found or a maximum number of iterations is reached.

Plan Remaining

Hopefully, after the intra-route Local Search steps, the solution has changed and improved. So, the Cheapest Parallel insertion is tried again to ensure that some tasks that could be inserted are not skipped before proceeding to the next stage. After the Cheapest Parallel Insertion, the algorithm also runs the **Local Search** Stage once more in case the algorithm can insert new tasks.

Ruin and Recreate

To escape from the local optimum reached during local search, more drastic operators have to be used. In this step, a (small) part of the solution is ruined using a variety of strategies (for examples see Section 3.1.2) and then rebuilt. Many combinations of ruin and recreate strategies are attempted in order to diversify the set of solutions considered and explore more of the solution space.

Chapter 5

Main analysis

So far we have explained what the CVRPTWs faced by ORTEC's clients look like and how OHD solves them. We have also introduced the features that might be interesting to consider in building our predictive model for the performance measures we have selected. To remind the reader, we are going to build a model that predicts if OHD is going to output a complete solution, the percentage of the tasks delivered for the incomplete solutions, and the time it takes to reach the plateau in the optimization process of the OHD algorithm for the complete solutions.

We must note that building a predictive model given only a very limited number of real-life CVRPTW instances is almost impossible. Especially in complex data like CVRPTW instances, none of the predictive models will work well. Thus, we are obliged to simulate our own real-life like CVRPTW instances. This has been done before by, for example, Solomon (1987). However, the new idea in our thesis is the possibility of fixing the values of some of the features before we simulate the instances. It allows us to control the feature values we want the simulated instances to have, thus offering the possibility to capture their effect anywhere in our feature value space. This also leads to more robust statistical conclusions and models.

Another critical point in simulating instances is that we cannot simulate millions of instances randomly. That is because we cannot afford millions of runs financially and in terms of the time it takes. Thus, we have to carefully select the feature values for which we want to simulate instances. For example, if we wanted to run OHD for all the combinations of 5 features at 10 different points, this would result in 100,000 instances. Thus, we need to find the balance between the number of instances we simulate and the ability of those instances to capture the features' impacts on the OHD performance.

In statistics, this is commonly done by the Design of Experiments (DoE). This clever way of selecting different combinations of feature values has been done by Arnold and Sörensen (2019), even if they did not describe it clearly as an experimental design. Overall the use of experimental design helps us answer, more robustly, questions like: "What has an impact on the target variable, and how important is that impact?" or, in statistical terms, find the factors that have a significant impact on the response variable (Box et al., 2005). The whole experimental design will be designed using concepts from the book of Goos and Jones (2011) and the JMP software.

Before explaining our simulation process and the Design of Experiments, this chapter will introduce selected features and report their values on several CVRPTW instances from ORTEC's clients.

A summary of the results is first shown to have quicker access to the core results of this

thesis; the methodology and precise discussion of the results will then be explained for each section.

5.1 Main Results

This section gives a brief summary of the main results of the analyses made in this chapter. For more details, we refer the reader to the following sections in this chapter.

5.1.1 Features

Table 5.1 summarizes the results gathered on the 12 extracted features. The first two columns highlight if the given feature was present in literature and/or was of primary interest to ORTEC. ≈ indicates that the feature was not very common in the literature. The remaining three columns show, for each model created, the importance of those features, ranked by magnitude (1 being the most important). ✗ is given for the features which are not significant, and if the features come in the form of interaction with another feature, it is denoted as "interaction".

Table 5.1: Main results for the features

Features	Literature	ORTEC	Solution completeness classification	Percentage of planned tasks regression	Plateau time regression
Number of customers	✓	✓	3	2	1
Number of vehicles	✓	✓	2	1	5
Distance matrix mean	✓	✗	✗	interaction	✗
Distance matrix standard deviation	✓	✗	✗	interaction	interaction
Mean distance between depots and centroid	✓	✓	✗	✗	✗
Number of clusters	✓	✓	7	✗	interaction
Customer time window length mean	≈	✓	4	3	4
Customer time window length standard deviation	≈	✓	6	interaction	✗
Vehicle time window length mean	≈	✓	5	4	3
Vehicle time window length standard deviation	≈	✓	8	✗	6
Customer peak demand location	✗	✓	1	5	2
Vehicle peak availability location	✗	✓	interaction	interaction	✗

5.1.2 Feature intervals

Table 5.2 summarizes the ranges of values that were tested during the *DoE* step.

Table 5.2: Summary of ranges of values for each feature

Features	Lower Limit	Upper Limit
Number of customers	100	2000
Number of vehicles	24	364
Customer time window length mean (minutes)	60	280
Customer time window length standard deviation (minutes)	0	125
Vehicle time window length mean (minutes)	220	480
Vehicle time window length standard deviation (minutes)	0	100
Customer peak demand location	0.05	0.95
Vehicle peak availability location	0.05	0.95

The Customer peak demand location and the Vehicle peak availability location are defined as $\frac{peak\ time}{end\ time}$, where peak/end time is measured in minutes passed after the start of overall delivery period,

5.1.3 Simulation errors

Table 5.3 summarizes the errors in the feature values of the simulated instances compared to the target feature values defined in the *DoE*. Absolute errors were reported for the features of Customer peak demand location and Vehicle peak availability location due to the problem with their scale. As the values of both features are between 0 and 1, the error rate of 50% for 0.05 vs 0.1 does not give the correct intuition on the error. In our case, simulated values of 0.05 instead of 0.1 is considered good as the absolute error is minimal, and the simulation around 0.1 is not easy.

Table 5.3: Summary of error rates of simulation

Feature	Errors rate in %
Number of customers	0%
Number of vehicles	1.8%
Customer time window length mean	0%
Customer time window length standard deviation	0.5%
Vehicle time window length mean	0.2%
Vehicle time window length standard deviation	4%
Feature	**Absolute errors**
Customer peak demand location	0.11
Vehicle peak availability location	0.25

5.1.4 Modelling

Table Table 5.4 summarizes the performances of the best-performing models on the independent test set. For the performances of other models, the reader can look at Subsection 5.7.2.

Table 5.4: Performance of the best model for each predictive model

Target variable	Model	Accuracy	MAE%
Solution completeness classification	RBF SVM Classification	0.96	NA
Percentage of planned tasks regression	Random Forest Regression	NA	6.46%
Plateau time regression	RBF SVM Regression	NA	30.52%

Accuracy is defined as the proportion of the number of correctly classified instances to the total number of instances. $MAE\%$ is defined as:

$$MAE\% = \sum_{i=1}^{k} \frac{|y_i - \hat{y}_i|}{\bar{y}}$$

where $\bar{y}$ is the mean of the target outputs of the regression problem. So, this measure takes into account the scale of the target output. That is important as the MAE of 10 is not the same for 100 as for 1 when we do not divide the sum by the target outputs' mean.

To summarize the table, our models are good at the solution completeness classification and the percentage of planned tasks regression but not on the plateau time regression. That is because the $MAE\%$ of 30.52% is not low.

5.2 Features

As explained in Chapter 3, many features exist in the literature. In this section, we will detail the subset chosen for this analysis. We also chose to add some features not detailed in the literature as they are specific to ORTEC's data.

5.2.1 Input features computation

Number of customers and vehicles

These are the most straightforward features to extract. They was calculated by counting the number of tasks in an instance and counting the number of routes in that instance. From now on, the number of customers will be referred to as n, and the number of vehicles will be symbolized by m. Here we do not consider the number of depots as for most of the data we have used it stays the same at the value of 4.

Distance Matrix

The most important feature that was known from the literature to describe costs (second/third objective in OHD) is the distance matrix. We are less interested in predicting the final cost, but it might be an interesting feature that could explain the future target variables.

We built a Python function that can calculate the distance matrix either using *Geodesic* distance or an internal ORTEC tool to calculate ground distances between two locations, both of them expressed in meters. Geodesic distance is the shortest distance between two points at the surface of the Earth, taking into account the curvature of the Earth. The ground distance of ORTEC's API is the effective distance on the ground between two locations that are accessible by road.

The distance matrix is then reported as:

$$\begin{bmatrix} 0 & d_{12} & d_{13} & d_{14} & ... \\ d_{21} & 0 & d_{23} & d_{24} & ... \\ d_{31} & d_{32} & 0 & d_{34} & ... \\ d_{41} & d_{42} & d_{43} & 0 & ... \\ ... & ... & ... & ... & ... \end{bmatrix}$$

Here, d_{ij} is the distance between the customers i and j. Note that the matrix is non-symmetric. This is expectable as the driving distance between customers i and j is not always the same as the one between j and i.

On this distance matrix, we calculated two characteristics: the mean (μ_D) and standard deviation (s_D).

Centroid Depot Mean Distance

Another feature described in the literature was the distance between the centroid of the customer's locations and the depot. Some clients of ORTEC have several depots available. We thus extended the calculation to the mean of the distances between the centroid of the tasks and each depot. In the rest of this analysis, the centroid depot mean distance will be referred to as μ_{cd}. The mathematic formula for it is:

$$\mu_{cd} = \frac{\sum_i^{N_d} d_{ci}}{N_d}$$

with N_d being the number of depots and d_{ci} being the *geodesic* distance between the centroid of the customer's locations and the depot i. As the centroid is not placed on a real address on the map, ORTEC's API to calculate ground distances does not work. Also, calculating ground distances between an imaginary point (centroid) and a true location does not make much sense. We thus decided to continue with *Geodesic* distances to describe these distances.

A Python function was thus constructed to compute this value based on the depots' and tasks' locations. This distance is reported in kilometres.

Number of Clusters

As explained in Subsection 3.2.2, the literature already had quite an extensive description of how to retrieve the number of clusters. The only problem in the calculation pipeline is that the elbow point had to be decided on the k-nearest neighbour graph by hand, which is unsuitable for automatic feature calculation. Our work was to find a robust way to detect the elbow point for each instance and then use this distance directly in the DBSCAN algorithm. This has been done with the Python package *Kneed*. An example instance is shown in Figure 5.1 before running any clustering algorithm. On the other hand, Figure 5.2 shows an example of the same instance after running the algorithm.

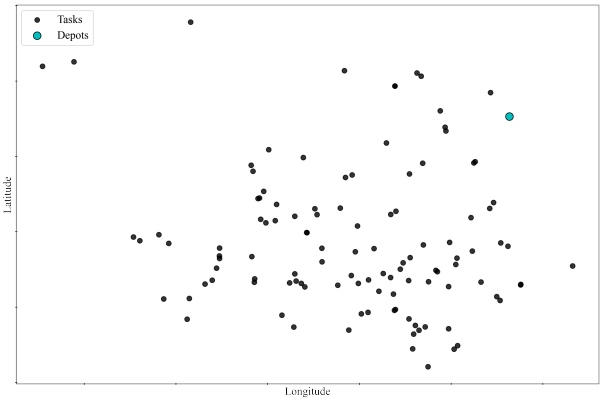

Figure 5.1: Example of unclustered instance

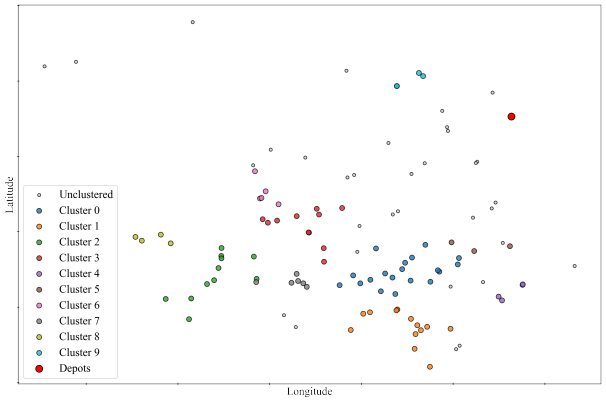

Figure 5.2: Example of clustered instance

This complete automated pipeline build can retrieve the number of clusters from any given instance. This number of clusters will be denoted as c in the rest of our work.

Customer Time Window

Even if the literature did not bring much information about which feature to compute when discussing time windows, a discussion with ORTEC's employees that work on a day-to-day

basis with this algorithm highlighted that it is a feature that could be meaningful to describe the target variables. We first decided to look at the time windows chosen by the customers. To be precise, we are interested in the duration of the time window rather than its proper starting and end time. However, only *finish time* and a *start time* are given, so we subtract them to get the final time window length value. We then take the average of each customer's time window length to compute the Customer Time Window Mean (Customer TW Mean) symbolized as μ_{CTW} in the remainder of this work. We also computed the standard deviation of these time window lengths to get the Customer Time Window Std (Customer TW Std) symbolized as s_{CTW}. Both μ_{CTW} and s_{CTW} are expressed in minutes.

Vehicle Time Window

We also decided to compute the time window length of each vehicle. Here again, we are interested in the proper duration of the time window rather than its starting end ending point. This could help us understand how tight time windows are and how hard it could be to deliver to the customers. In this case, we can extract the duration value directly from a column called *MaxDuration*. However, for some clients of ORTEC, this column does not exist, and we thus take the difference between the latest finishing time and the earliest starting time of the vehicle. This is done for each vehicle in an instance. Here again, we describe these values by their mean (Vehicle TW Mean) symbolized as μ_{VTW} and standard deviation (Vehicle TW Std) symbolized as s_{VTW}, and they are expressed in minutes.

Customer Demand Location

When analyzing the time windows of the customer, giving the duration was not telling the whole story. Describing where the demand lies the most could also be of great interest in describing the instance more precisely. To do so, we developed a new feature based on calculating the number of customers for each period of time. This feature is developed in this work and is not, to our knowledge, present in literature. A deeper analysis of the time window distribution throughout the day is available in Subsection 5.4.4.

For each instance, we divide the whole domain of delivery period into k small intervals $[t_i, t_{i+1}]$ $(i{=}0, ..., k-1)$. Then, for each interval, we count the number of customer time windows that cross this interval. This could let us know which interval is crossed more and, thus, where the demand lies more. To be more precise in calculating the overall demand, we assign a weight to each customer time window such that the weight is 1 over the extent of the time window of the customer. That means that we always fix the area of a time window to be 1, 1 unit of demand. With this weight, we ensure that a customer with a bigger time window has less impact on the overall demand in one bin than a customer with a tight time window.

Mathematically, the function of demand for interval $[t_i, t_{i+1}]$ is:

$$Demand([t_i, t_{i+1}]) = \sum_{j=1}^{\#customers} \frac{I_{TW_j}([t_i, t_{i+1}])}{|TW_j|}$$

Here TW_j is the time window interval with length $|TW_j|$, and $I_{TW_j}([t_i, t_{i+1}])$=1 of the TW_j completely contains the interval $[t_i, t_{i+1}]$.

A graph representing this calculation is shown in Figure 5.3. As we can see, customer 1 has a tighter time window than customer 3. Its impact on bin one is thus greater than the impact of customer three on bin one. For better comparison purposes, we scale the whole domain

of time windows. The x-axis is thus the time elapsed since the earliest time window started, scaled from 0 to 1.

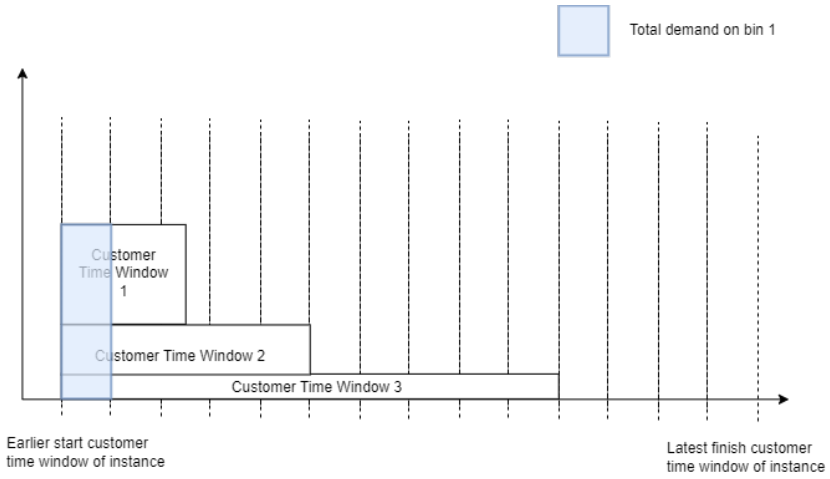

Figure 5.3: Visualization of the computation of customer demand

An example of such computation applied to an instance is shown in Figure 5.4. The parameter we extract from this graph is the time the peak demand is reached, called Customer Demand Location, symbolized by l_c.

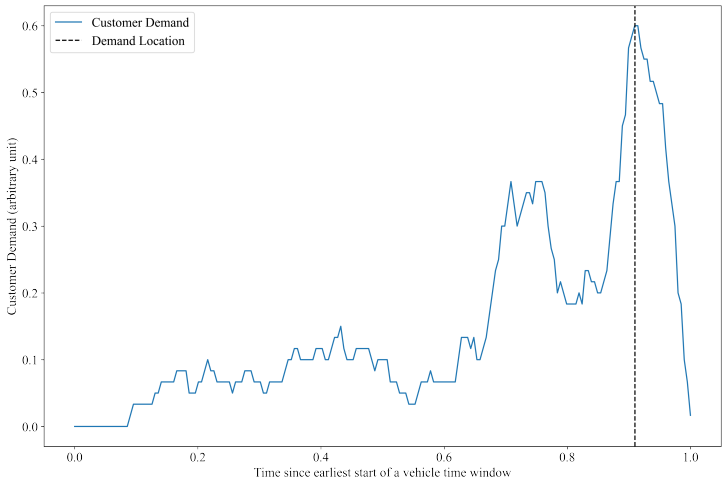

Figure 5.4: Example of customer demand plot

As the time windows are often given as discrete options for customers in real-life applications, we usually witness a staircase graph instead of a smoothed graph as shown in Figure 5.4. It thus happens that the maximum is a plateau instead of a peak. We then take the Demand Location equal to the time in the middle of the plateau.

A last remark on this feature is that since ORTEC has several clients, they have different ways of working. Some clients work by half days where the instances are at most 11 hours spread. Other clients work for a full day, and instances are thus spread over 16 hours. Thus, we see two peaks instead of one. Usually, one peak is because of morning demand, and one

is because of evening demand. To better represent this behaviour when extracting l_c, we split instances spread over the whole day in half and calculate the peak location for each part. Then we average the peak locations to get to one final value of l_c. An example of a bimodal demand graph is shown in Figure 5.5.

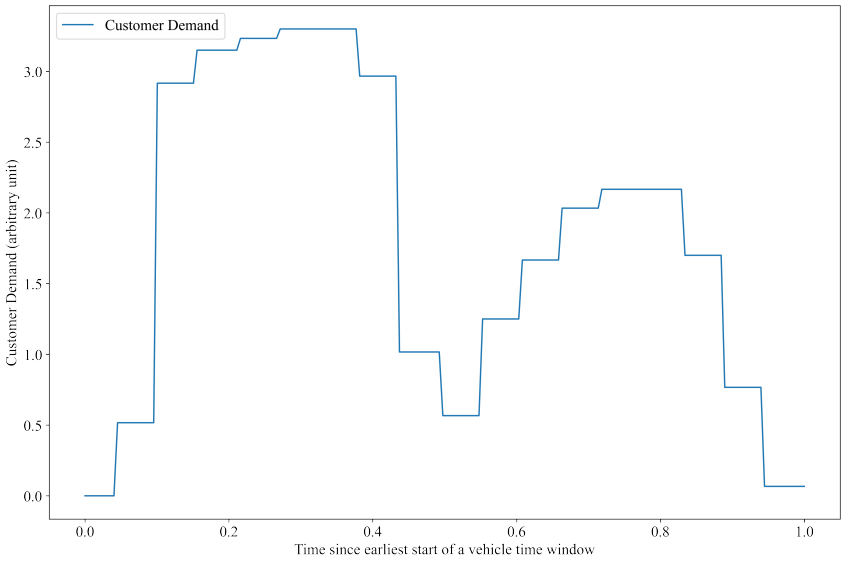

Figure 5.5: Example of bimodal customer demand

One could argue that using capacity units to determine the weights might have been better. Still, as each client has its unit type, we would not be able to compare the graphs between several clients and, thus, not automatically detect a demand location. Also, that would have made the extraction less robust to new clients coming at ORTEC, as a new definition of the weights should be rewritten for each new client.

Vehicle Availability Location

This parameter is the same concept as the Demand Location previously introduced, except that we define it for the availability of the vehicles throughout the instance. We wanted to describe when most vehicles were available and when most were not. The same graph is computed with the same weight idea, except the area represents one unit of availability here. A graph showing the computation on an example instance is shown in Figure 5.6.

Again, we define the location as the time peak availability is reached. This location is the only parameter we extract from the graph and is called Vehicle Availability Location, symbolized as l_v.

In real-life applications such as ORTEC, vehicle time windows are usually discrete and not continuous. It thus happens, such as for the demand parameter, that the graph exhibits a staircase behaviour. In that case, we also take the mean time of the maximal plateau to be the location. When instances spread over more than 11 hours, the same principle is adopted for customer demand location.

5.2.2 Output features extraction

As explained in Section 2.2, two files are reported after optimization. One contains the logs, and the other contains the output routes and KPIs. Information extracted in this

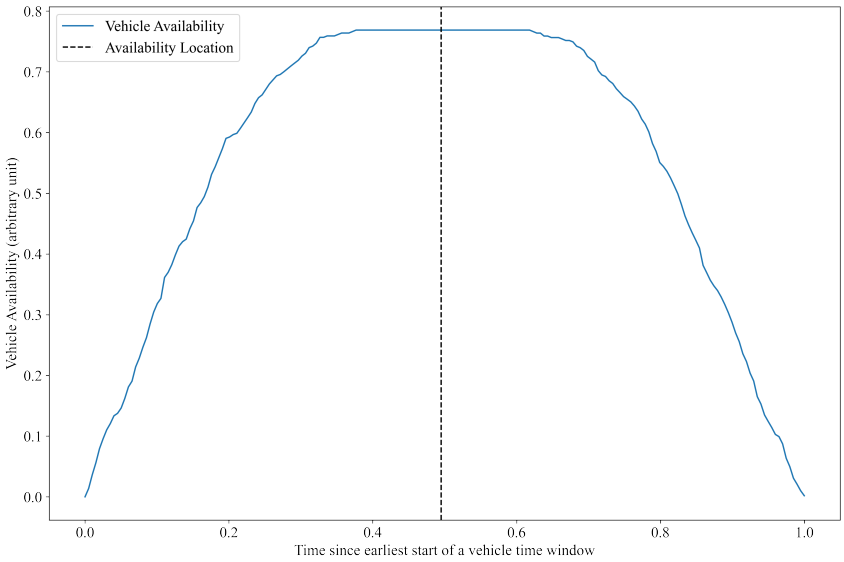

Figure 5.6: Example of vehicle availability plot

section comes from either one or the other.

Number of planned tasks

As highlighted in Section 2.2, the first objective of the algorithm is to plan all tasks. The first feature we want to retrieve is thus the number of tasks planned, as it will help us understand whether or not the algorithm was able to find a complete solution. Also, for incomplete instances, we will predict the percentage of tasks planned from all requested. Retrieving the number of planned tasks is thus of great importance. This can be retrieved from the KPIs in the output file.

Final Cost

Another important variable, as it is the second/third objective of the algorithm, is the final cost after optimization. We can retrieve it directly from the KPIs of each optimization. We might want to predict this variable later in this work to better understand to what point the algorithm can optimize. One important thing is that for an incomplete solution output by the algorithm, it makes little sense to look at the cost, as it means that the algorithm was not even capable of planning all the tasks. The cost becomes then irrelevant.

Cost Array

Along with the final cost, following the optimization process and seeing how the cost evolves across the optimization could be of interest. To do so, we can look at the log file which gives the cost of the new best solution encountered until the current step. The best solution is defined in terms of the objectives defined earlier, so first of all, the number of planned tasks. This is an important remark to understand better the cost pattern, which will be shown later.

Final Time

Understanding the cost behaviour is interesting but gets even more interesting when we

look at it with the optimization time. We thus wanted to also retrieve the final time of optimization. This is done by looking at the log file and extracting the time of the last solution found by the algorithm. This variable could also be predicted to know how much time the algorithm could take on any given instance. This could allow for business-related insights for ORTEC such as the price of one run of the OHD algorithm.

Time Array

Understanding the pattern in the optimization behaviour is taken to another level when we look at both the time and the cost together. To do so and to follow their evolution throughout the optimization process, we want to, analogously to the cost array, retrieve all the times at which the algorithm found the new best solutions. We could then plot the cost as a function of time, and that could already highlight some optimization patterns. Those will be detailed later in this chapter.

In other words, we can get a sequence of $(time, cost)$ pairs, each of which shows the time and the cost of every new best solution (defined in terms of the hierarchical objectives). $time$ comes from the *Time Array* and $cost$ comes from the *Cost Array*

5.3 Exploratory Data Analysis on ORTEC instances

In this section, we will report the exploratory data analysis results we have gathered while looking at data from 5 different clients of ORTEC. A total of 97 instances were used, and we extracted all of the input features previously introduced. A big role of this exploratory data analysis was understanding the domain's span of each feature to design our simulation better. As the clients used for this analysis have very different needs and locations, we best describe the type of values that could arise from any instance. The intervals have been summarized earlier in Table 5.2.

Number of Customers

We first wanted to see how the number of customers behaved across the different ORTEC clients. This is shown in Figure 5.7 where counts are given instead of the density. This is to ensure we have some ideas about the counts for each client type. Density would not be able to provide that. From this plot, two observations can be made. The first one is that some clients, such as client 5 and client 1, have the same type of instances in terms of the number of customers as the histograms are peaked around one value. Other clients have instances that vary more in size, as shown by client 4. The second observation is that the domain of values spreads from around 100 customers in one instance to slightly more than 3000 for the biggest instances. As the number of examples was not the same for each client, we cannot interpret the difference in the height of bars from different clients.

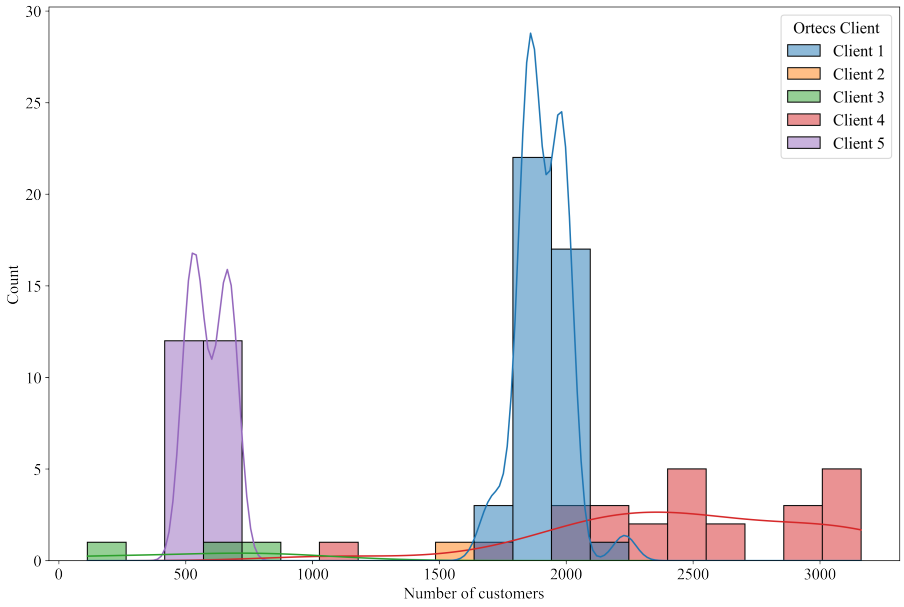

Figure 5.7: Distribution of the number of customers across 97 instances

Number of Vehicles

A second interesting feature was the number of vehicles available in each instance. This is reported in Figure 5.8, where the bars are colored depending on the client. Here again, we can quickly see that some clients of ORTEC always have the same amount of vehicles available for distribution, such as client 1,3,5. On the other hand, client 4 provides a different number of vehicles for distribution. Overall, the domain of values witnessed by ORTEC ranges from around 20 to 370.

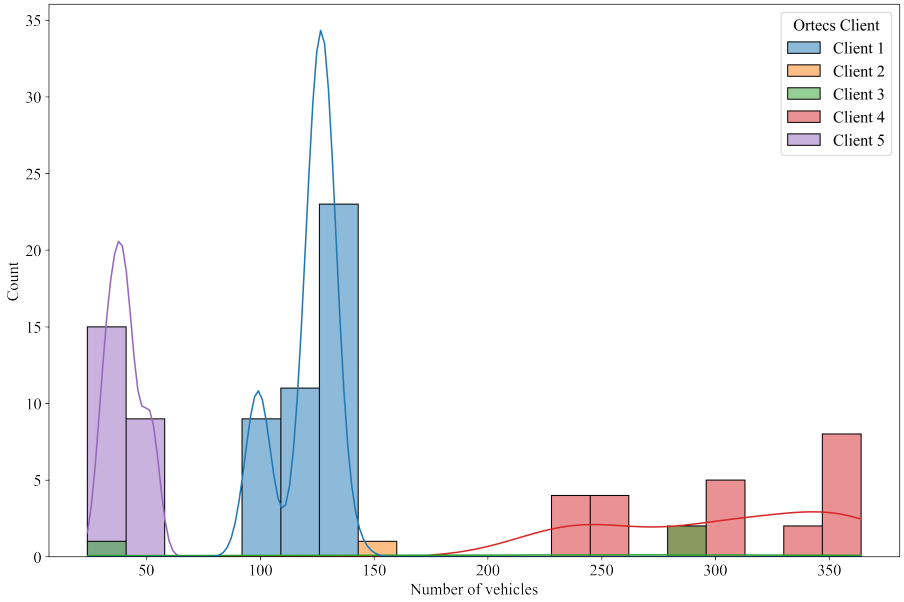

Figure 5.8: Distribution of the number of vehicles across 97 instances

Number of Clusters

The story told by the graph representing the distribution of the number of clusters in Figure 5.9 is slightly different. As we can see, the number of clusters seems way more variable across several instances of the same client. An example is given by client 4, which has instances spreading across the whole domain of values. We see that client 5 has more instances with a low number of clusters, while client 1 has two number of clusters that are dominant compared to the rest of its instances. Overall, cluster number ranges from around 5 to more than 100.

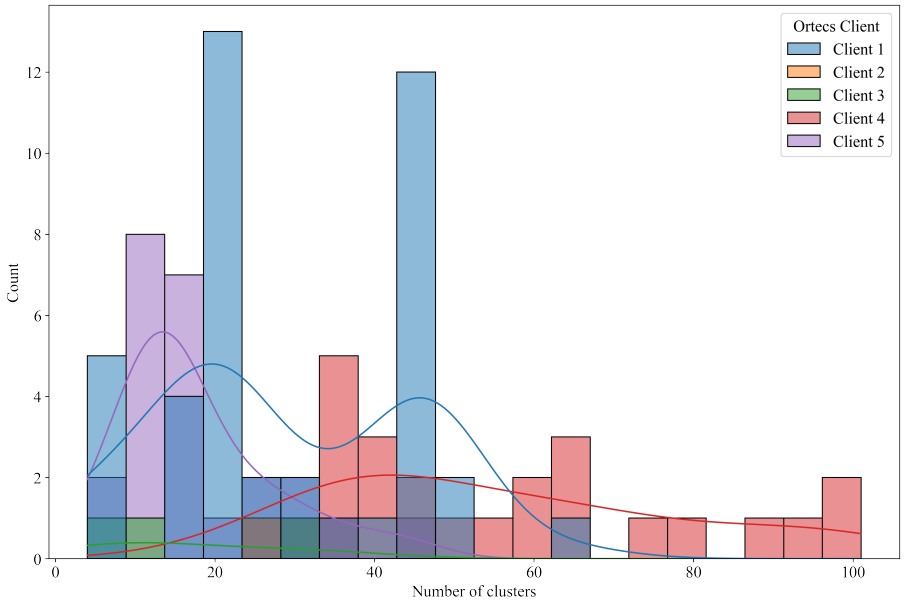

Figure 5.9: Distribution of the number of clusters across 97 instances

Distance Matrix Mean

The distribution of the means of the distance matrix is shown in Figure 5.10, coloured by the clients. We can see that the domain of values bounds between 5000 and slightly less than 90000. Client 1, again, shows a bimodal behaviour with a peak at 12500 and another at 20000. They do not have big values reported among their instances. However, client 4 shows a different behaviour as it spreads across the whole domain of values.

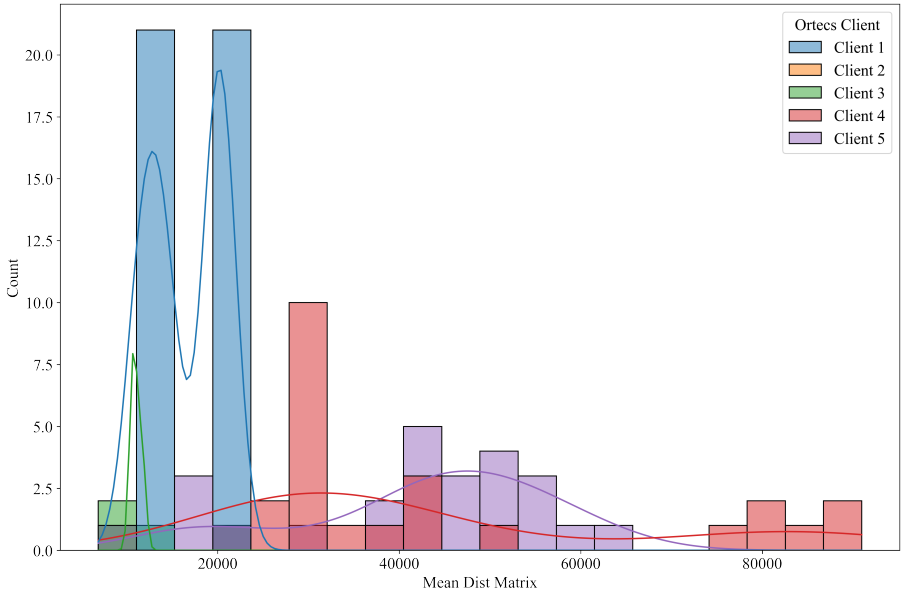

Figure 5.10: Distribution of the mean of distance matrix across 97 instances

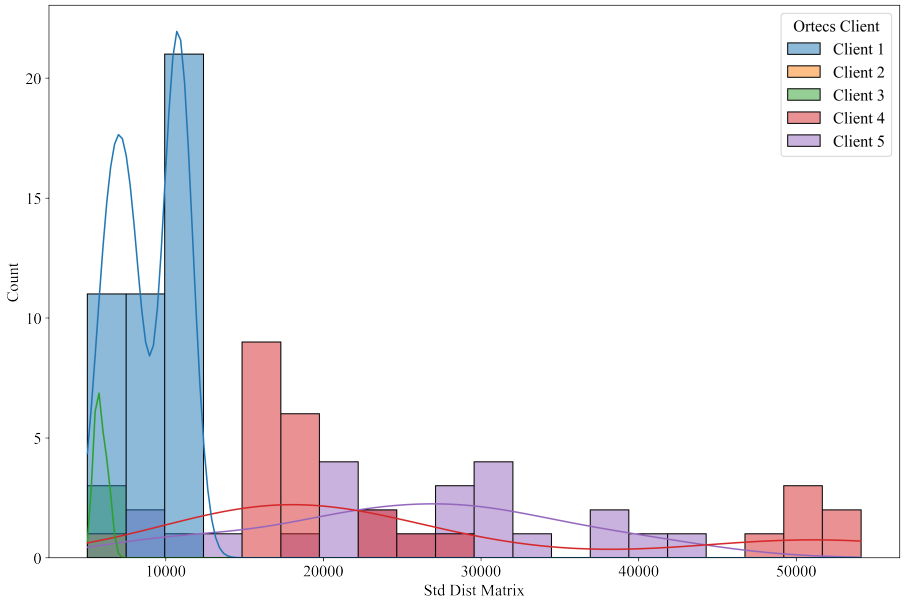

Figure 5.11: Distribution of the standard deviation of distance matrix across 97 instances

Distance Matrix Std

The distribution of the standard deviation of the distance matrix is reported in Figure 5.11. Here again, client 1 shows a bimodal behaviour of both values around 10000 meters. Clients 4 and 5 are quite spread across the whole domain of values we witness. The values range from around 3000 to more than 50000 meters.

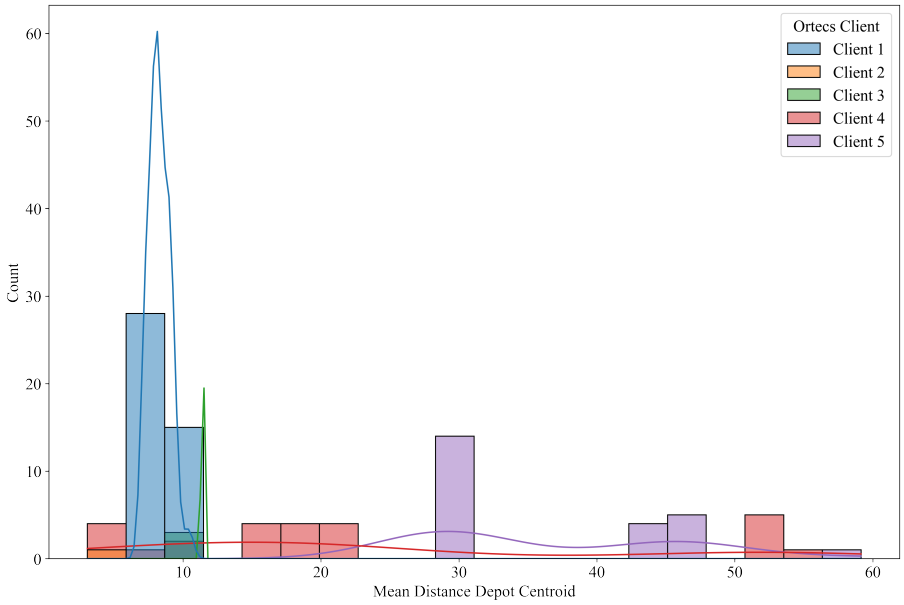

Figure 5.12: Distribution of the mean of the distances between depot and centroid across 97 instances

Mean Distance Centroid Depot

The distribution of the mean distance between depots and the centroid is shown in Figure 5.12. From that plot, two conclusions can be drawn. Firstly, the mean distance between the centroid of the points and the depots ranges between 2 km and 59 km. Secondly, client 1 has a small distance from the centroid to the depot, probably because the depot lies in the centre of the points and the instance is small. On the other hand, client 4 shows more hectic behaviour, surely because of more variety in the instances it has. The big values are reached for instances where the depot is off-centered or when several depots are quite spread across the instance.

Customer Time Window Mean

The distribution of the mean of the customer time window lengths is shown in Figure 5.13. We can see different behaviours between the different clients of ORTEC. Client 1 has only one time window duration offered for all the clients in its instance, which is 60 minutes. Thus, they propose tight time windows compared to clients 4 or 5, with time windows lengths between 150 and 280 minutes. The whole domain of values ranges from 60 to 280 minutes for the customer time window length mean.

Customer Time Window Standard Deviation

The distribution of the customer time window length standard deviation is shown in Figure 5.14. We can see that clients 1 and 2 have 0 as the standard deviation. This means

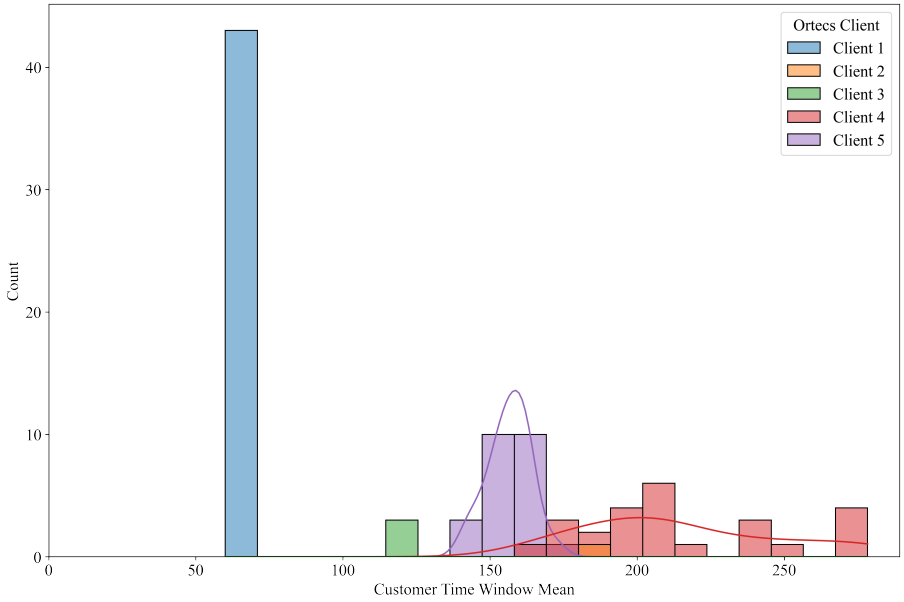

Figure 5.13: Distribution of the mean of the duration of the time windows of the customers across 97 instances

that they always provide the same type of time windows as was expected from bars shown for the means in Figure 5.13. The rest of the clients are between 60 and 120 minutes of standard deviation. This is also consistent with the mean distributions we witnessed in the previous plot. The domain of values ranges from 0 to around 120 minutes.

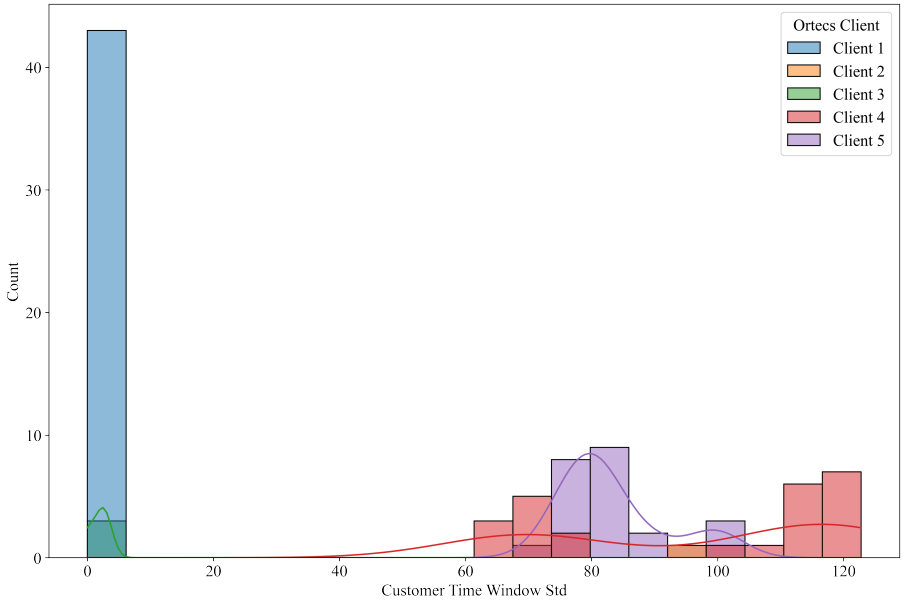

Figure 5.14: Distribution of the standard deviation of the duration of the time windows of the customers across 97 instances

Vehicle Time Window Mean

A similar analysis of the time window lengths was carried out for the vehicles. The means

of the time window lengths of each instance are reported in Figure 5.15. We see that, probably for convenience, each client has only one time window for its vehicles, except for clients 1 and 3, who allow for different types of vehicle time windows between several instances. Clients 4 and 5 use wide time windows for vehicles as they are up to 480 minutes (8h). Client 3, however, uses quite tight time windows for vehicles. The domain is spread from 220 to 480 minutes.

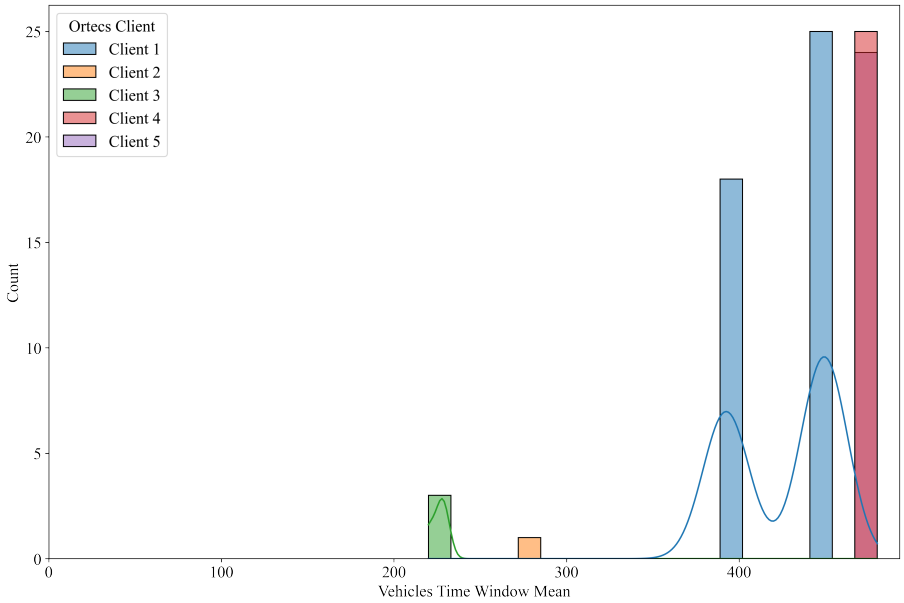

Figure 5.15: Distribution of the mean of the duration of the time window lengths of the vehicles across 97 instances

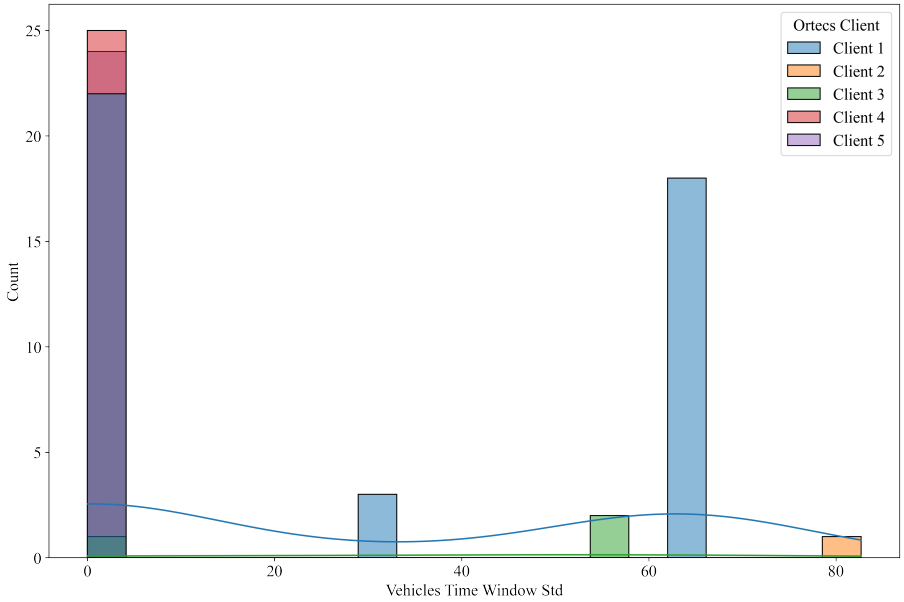

Figure 5.16: Distribution of the standard deviation of the duration of the time windows of the vehicles across 97 instances

Vehicle Time Window Standard Deviation

To finalize the description of the time window lengths of the vehicles, we wanted to look at the standard deviation of these time window lengths in each instance. Their distribution is reported in Figure 5.16. As expected, clients proposing only one type of time window duration have a 0 standard deviation. Client 3 has instances where several time windows are available within the instance; hence the peak is around 60 minutes. Client 1 displays a more hectic behaviour as some instances have 0 standard deviations meaning that all time windows duration are the same. At the same time, it also shows peaks at 30, 65 and 80, meaning that we have more or less diversity of the time window lengths for the vehicles available in an instance. The general domain of this value lies between 0 and 80.

Customer Demand Location

The location of the peak has been extracted for each instance as explained in Subsection 5.2.1. We have shown the distribution of these locations in Figure 5.17. We can see that some clients have peak demand always located at the start, such as client 5. On the other hand, client 4 has more diverse instances where the peak is in the middle or at the end of a working day. The customer demand is overall spread across the whole domain between 0 and 1.

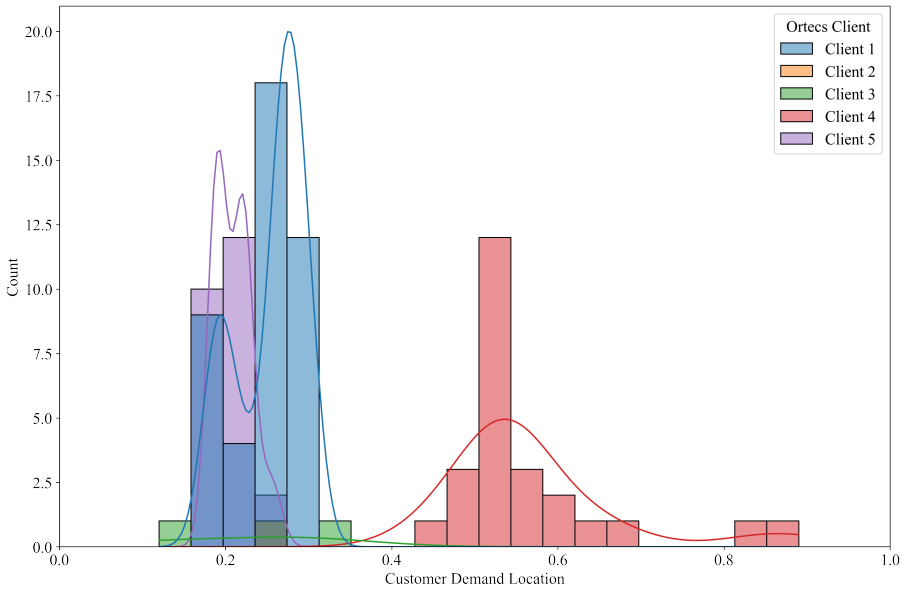

Figure 5.17: Distribution of the customer demand location across 97 instances

Vehicle Availability Location

The location of the peak of the vehicle availability has been extracted following the same process as previously explained. The distribution is shown in Figure 5.18. We can see that client 1 has more availability at the start of the day/start of the half day because location peaks lie at 0.15, 0.2 and 0.25. Client 5 has its peak demand around 0.25. The peak is at 0.5, not because the peak availability is around noon but because vehicles are available throughout the day. Taking the mean of this availability thus leads to a peak around noon. We can also

see that no peak was found at more than 0.5. This means that no distributor asks for a peak availability of its vehicle at the end of the working day. The domain of values is thus between 0.15 and 0.5.

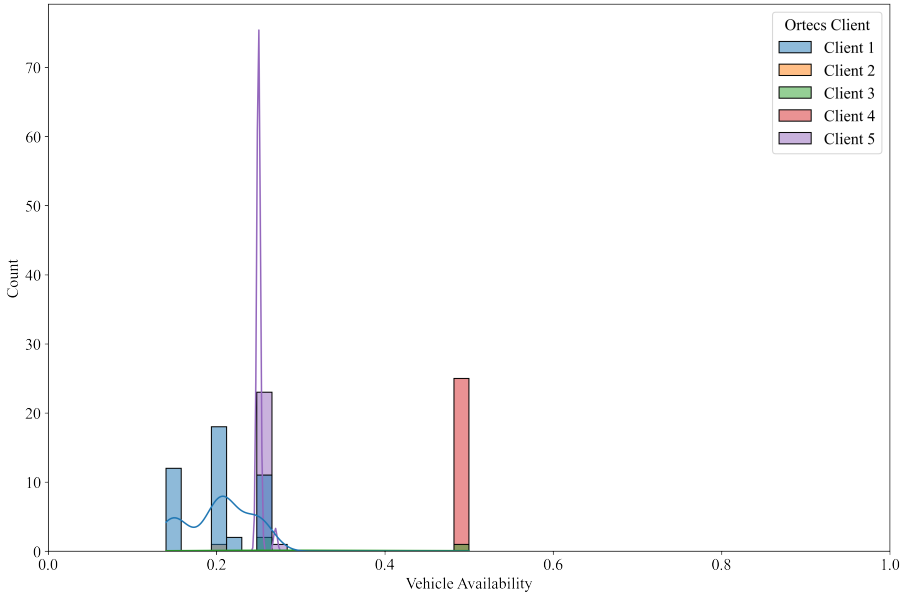

Figure 5.18: Distribution of the vehicle availability location across 97 instances

5.4 Simulation

There are, in total, 12 features introduced in the previous sections. In our simulation process, we controlled 8 of them and developed a theoretical basis for how 3 other (out of the 4) could be potentially controlled. Even for the 8 controlled features, the values are never precisely the same as the desired ones due to randomness in our simulation process. So, depending on how well we can control their values, we divide the 8 features into four classes: *perfectly controlled* $\{n,\ m\}$, *well controlled* $\{\mu_{CTW},\ s_{CTW},\ \mu_{VTW},\ s_{VTW}\}$, *semi controlled* $\{l_c,\ l_v\}$. In the simulation process, we start by fixing the values of the *perfectly controlled* features (Number of Customers and Number of Vehicles). Next, we assign TWs for both the customers and the vehicles. Finally, we assign coordinates for the customer locations. Depot locations are fixed over all the examples and simulated instances.

5.4.1 Perfectly controlled features

As explained above, we start by selecting the values of n and m, which fall into this class of perfectly controlled features. That is because after specifying in the beginning, they are not changed due to randomness during the simulation.

5.4.2 Well and semi controlled features

The next step is to assign TWs for the n customers and the m vehicles in the interval $[0, 660]$. Following the examples of real-life instances of ORTEC clients, we decided to take 11 hours or 660 minutes to be the overall delivery period where the Customer TWs and Vehicle TWs can fall.

Customer TW related features

First, we generate the lengths of the TWs T from the uniform distribution $U[a, b]$. Mathematically speaking if we want $E(T) = \mu_{CTW}$ and $Var(T) = s^2_{CTW}$, where $T \sim U[a, b]$, then $a = \mu_{CTW} - \sqrt{3}s_{CTW}$ and $b = \mu_{CTW} + \sqrt{3}s_{CTW}$. Thus, in fact we have two constraints that $\mu_{CTW} - \sqrt{3}s_{CTW} \geq 0$ and $\mu_{CTW} + \sqrt{3}s_{CTW} \leq 660$ for the values of μ_{CTW} and s_{CTW} we can generate. That is because if the lower bound of the uniform distribution is negative, we might get $T < 0$, which is unacceptable. Similarly, $T > 660$ is not acceptable either.

Next, we generate the midpoints of the TWs M in the interval $[0, 1]$ for each T we have generated so that in the end, we can take the TWs to be $[660M - T/2, 660M + T/2]$. Here the main objective is to ensure that $E(M) = l_c$ when generating M. This, in practice, does help with steering the generation in the right direction, and the computed Customer Demand Locations usually get close to l_c. There are several choices on what distribution to choose for the generation of M. An obvious choice would be, of course, a Gaussian distribution. However, one big reason the Gaussian distribution is unsuitable is that, by nature, it is a distribution with its support being the interval $(-\infty, +\infty)$. So, we would need to take the extra step of bounding the values it can give. There are many ways to see why this would be a problem. For example, assume that we want l_c to be 0.9, and if we sample from Gaussian distribution with mean 0.9, we are equally likely to get samples from either side of 0.9. Thus there is a high chance that many of the values $660M + T/2$ would be larger than 660, making the sample unacceptable. Considering this limitations of the Gaussian distribution in generating M, the Beta distribution is more suitable, which has the the support of $[0, 1]$. (Figure 5.19).

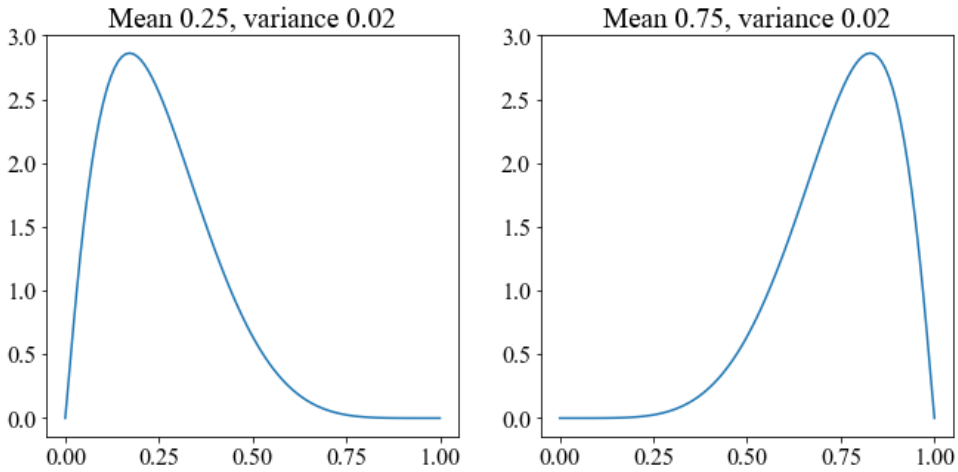

Figure 5.19: Skewness by the mean

Beta distribution needs two parameters to be specified: α and β. As mentioned, we only need to ensure that the $E(M) = l_c$, which is insufficient to identify the Beta distribution fully. We specified the $Var(M) = 0.02$, through which we will now have a fully specified Beta distribution. Here the choice of 0.02 is not based on any mathematical argumentation but rather taken at the value that will result in a more stable simulation process. This leads to the parameters $\alpha = (\frac{1-l_c}{0.02} - \frac{1}{l_c})l_c^2$ and $\beta = (\frac{1-l_c}{0.02} - \frac{1}{l_c})(\frac{1}{l_c} - 1)l_c^2$.

Now, even if M were generated between $[0, 1]$ we might still end up having $660M + T/2 > 660$ or $660M - T/2 < 0$. To that end, we reject such combinations of (T, M). In doing so, it is important to note that as smaller Customer TWs are given the advantage to be kept, our samples' μ_{CTW} and s_{CTW} are also changed. Let us call the obtained feature values after the rejection step $\tilde{\mu}_{CTW}$ and $\tilde{s}_{CTW}$. Thus, to obtain μ_{CTW} and s_{CTW} from a generated sample with $\tilde{\mu}_{CTW}$ and $\tilde{s}_{CTW}$, we will be changing our samples t_i of time window lengths to $(t_i - \tilde{\mu}_{CTW})\frac{s_{CTW}}{\tilde{s}_{CTW}} + \mu_{CTW}$. By making this transformation, we obtain the original mean and the standard deviation μ_{CTW} and s_{CTW}, respectively.

Mathematically this can be seen by the following:

$$E((T - \tilde{\mu}_{CTW})\frac{s_{CTW}}{\tilde{s}_{CTW}} + \mu_{CTW}) = (E(T) - \tilde{\mu}_{CTW})\frac{s_{CTW}}{\tilde{s}_{CTW}} + \mu_{CTW} = \mu_{CTW}$$

$$Std((T - \tilde{\mu}_{CTW})\frac{s_{CTW}}{\tilde{s}_{CTW}} + \mu_{CTW}) = Std(T)\frac{s_{CTW}}{\tilde{s}_{CTW}} = s_{CTW}$$

In our implementation, we check if $\delta_i = (t_i - \tilde{\mu}_{CTW})\frac{s_{CTW}}{\tilde{s}_{CTW}} + \mu_{CTW} - t_i$ can be added to any of the sides of the original TWs that were generated. Given that it is possible, we add δ_i to either of the sides. If it is possible to add to both sides, we will add half of δ_i to each side. Although one can argue that it is not always possible to add the δ_i thus, we cannot guarantee theoretical μ_{CTW} and s_{CTW}, in practice, it works quite well in getting very close to the theoretical values of μ_{CTW} and s_{CTW}. In general, the Law of Large Numbers also partially explains some of this closeness as we have taken samples from distributions for many customers.

Vehicle TW related features

These features are simulated the same way it they were simulated for the customers.

5.4.3 Uncontrolled features

The values of the features μ_D, s_D, c, μ_{cd} are never controlled in the simulation process due to the lack of a mechanism that can guarantee the values of these features. However, we have done some theoretical work on guiding the simulation to move towards the values of the μ_D, s_D, c we want.

First, the customers' locations we generate cannot be random points on the map. In real-life cases, the locations have fundamental restrictions in that they have to be located at specific addresses with street names and numbers so that delivery is possible. To resolve this issue, we have pooled the coordinates of all the customers from all the real-life client instances we have from ORTEC. This gives us around 30000 customer coordinates, from which we must simultaneously select n customers and guide μ_D, s_D, c.

First, we cluster all our pooled coordinates using the DBSCAN with *Eps*=0.002 and *MinSample*=4 as introduced in Subsection 3.2.2. Here the value *MinSample*=4 is calculated from 2*dimension as explained in Chapter 3 and the value of *Eps*=0.002 is taken based on the visual sensibility of clusters obtained. As a result, DBSCAN outputs 385 clusters with their centroids and some noise coordinates.

We start our coordinate generation process by randomly choosing a seed customer coordinate, which must be a noise customer obtained from the DBSCAN. Then we rank the clusters based on the distances of their centroids to the seed customers. (Figure 5.20).

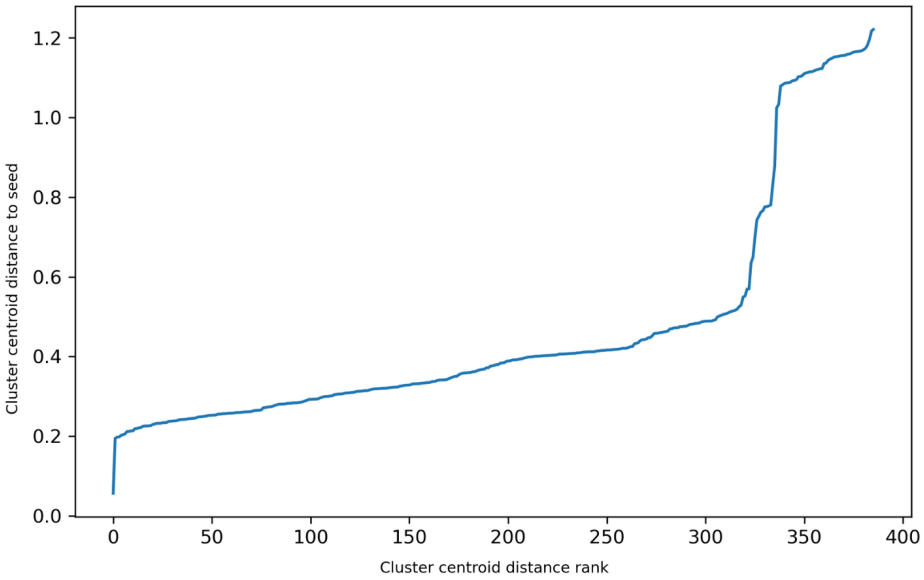

Figure 5.20: Distance between cluster centroids and seed by rank

With that information, we can guide μ_D by selecting coordinates only from clusters with centroids located far or close to the seed, depending on how big we want μ_D to be. For example, to get high μ_D, we can take coordinates from the clusters of around rank 350. The number of clusters from which we select the customers could guide c. Finally, to guide s_D, we specify how varied we want our selected clusters to be around the one we selected of a specific rank. To extend our previous example, we can take clusters {340, 321, 350, 353, 352} (low variance of ranks) to guide for small s_D, or we could take {100, 332, 2, 354, 351} (high variance of ranks) to guide for high s_D.

Finally, we must mention that when we select customer coordinates from 30000 customers, we keep the same Service Time at the customer's location. This is to limit the number of features we explore and keep them consistent with academic articles' suggestions.

Regarding vehicles, preparation duration is always 15 minutes before and after the driving period. Break rules are the same for all vehicles in all instances with 15 minutes twice during the delivery, so we keep them the same. For other characteristics such as priority, capacity, costs, and travel time/service time factors, we keep the diversity/proportions of different values/classes the same as in real-life instances.

5.4.4 Notes on Time Window simulation

TW generation techniques from academic articles

One might be interested if there are benchmark CVRPTW instances that were simulated before in academic articles, and if yes, how the TWs were generated for those instances. To answer that question, we will give the TW generation mechanisms given in two different articles.

Besides the heuristic steps introduced in Chapter 3, Solomon (1987) has also simulated a set of CVRPTW instances. In this paper, similar to our steps, the TWs were generated by their midpoints and lengths. However, the midpoint was generated using a Uniform distribution, and the length was generated using a Normal distribution. We have to note that unlike us, the author did not have restrictions on the values of μ_{CTW} and s_{CTW} which is why this TW generation mechanism is enough if the only objective is to get diversity in feature values.

Another example of TW generation is given in the paper of Sartori and Buriol (2020) where the TW lengths are only taken to be either 240 minutes or 480 minutes, and the midpoints are taken from the Uniform distribution. If we wanted the TW lengths to be from a discrete set, controlling the feature values of μ_{CTW}, s_{CTW}, μ_{VTW}, and s_{VTW} would have been much more challenging. For example, selecting 10 customer TW lengths from {240,480} such that μ_{CTW}=300 and s_{CTW}=50 might not even be possible.

However, given this disagreement between the academic way of generating TWs (especially our new way) which involves at least one continuous probability distribution and the real-life TWs where the customers are given a set of "discrete" TWs, several questions might arise. One of them that we will try to answer is: How does the performance of the OHD algorithm on the real-life instances change if we keep the μ_{CTW} and s_{CTW} the same but change TWs such that they are not of discrete lengths?

Difference between academic (continuous) and real-life (discrete) generations

For all ORTEC clients' instances, customers are given discrete TWs, such as 12:00-14:00 or 18:00-19:30 of discrete midpoints and lengths. To analyse the performance of the OHD algorithm when μ_{CTW} and s_{CTW} are kept the same but the TWs are "smoothed", we will use 30 CVRPTW instances from one of the ORTEC's clients.

The smoothing step involves checking the number of minutes Δt we can add to both sides of all TWs of customers. This is found to be always larger than 0. Then, we take n equally spaced points Δt_i in between $[-\Delta t, \Delta t]$. By adding this value to one of the sides of the TW and subtracting it from the other side, we will not change the values of the features μ_{CTW} and s_{CTW}. We will not be changing the lengths of individual TWs, for that matter. To see the effect of this "smoothing" process, we can use the plot of demands that was introduced

earlier in Subsection 5.2.2. In Figure 5.21, we can see that originally there were staircase-alike demands as the TWs were chosen from a discrete set. However, the staircase-like structure is not there anymore after the smoothing step we introduced.

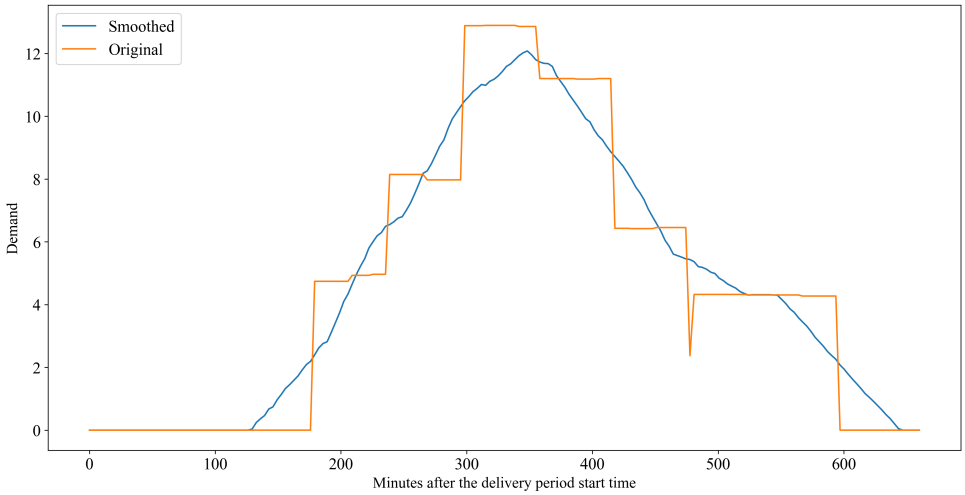

Figure 5.21: Original and Smoothed demands

In terms of performance, the first objective value, which is the number of tasks delivered, is the same for both the original and smoothed instances for all 30 instances. The costs are also very similar, as seen from the Figure 5.22. For a sound statistical conclusion, the Wilcoxon signed-rank test and paired t-test give non-significant results with p-values of 0.44 and 0.5, respectively. So, we can conclude that there is no significant difference in OHD's performance between the original and smoothed instances for 30 CVRPTW instances.

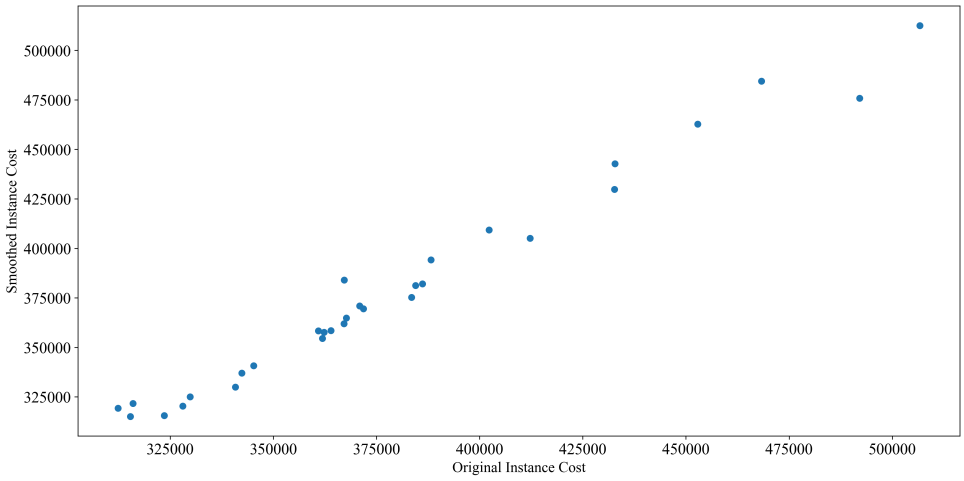

Figure 5.22: Original and Smoothed Instance Costs

5.5 Design of Experiments

There are two ways to gather data in the realm of experimentation and research. The first way is when the process of interest already took place. This is when the experimenter can only witness what happened and try to understand the process by measuring precisely different variables that he/she might think are explanatory of the target variable. He/She then hopes that the process and measurements went well enough such that he/she can correctly assess the impact of each variable on the desired output. This is the usual way of treating the data as, most of the time, the data has already been collected, and statisticians and data analysts do not have to gather them alone.

The second way to gather the data appears when the process still needs to occur. Although quite similar, this approach asks for more anticipation because more options are left open to the experimenter. This approach relies on the fact that we can still decide with which parameters to run the process of interest. Two main assumptions exist to dive into this world: the process did not yet take place and the fact that we can somehow control the value of the parameters tested. If both assumptions hold, we can decide at which value we want to test different parameters instead of only passively measuring them when the process occurred. This allows for a much better repartition of the tests throughout the parameter space, allowing for a better estimation of their impact on the target variable. This second way of conducting research is not always accessible, but when it is, it can bring much more power to the data gathered. This way of conducting research is called the Design of Experiments (DoE).

Another significant advantage of the DoE approach is that we need fewer runs/tests to achieve usable results since estimating each parameter impact is more precise. This is an important advantage when the time of testing is long or when the cost of a test is really expensive. In both situations, the number of runs one can afford is scarce, and it is important to minimize it. However, with this approach, we can still come to conclusive results with a few well-chosen runs. In our case, running the algorithm takes some time but not to the point that it makes the number of runs a limited resource. As we were in a place where we could simulate instances with controlled parameters and where these instances were still to be simulated, we decided to apply principles of design of experiments. That is because of its power and the insurance that the whole parameter space has been visited.

In our work, we have decided to go for a surface design experiment type. This type of design is specifically used when a few factors are present and are known to be significant to predict the output variable. In our case, we expect, from the literature, that the features chosen are significant, which is why we do not carry a screening design before this surface design. The surface response design ensures that the variable space is visited such that we can easily estimate the quadratic effect of each factor along with two-factor interaction and main effects. It is mainly used when the goal is to predict, thanks to a good visualization of the response surface. As our future goal is to predict a chosen target variable, we decided that this design was the way to go.

Thanks to the analysis in Section 5.3, we defined the value domain of each input parameter, also called a factor. Because not all parameters are controllable during simulation, we only developed a design for the eight controllable factors cited in Subsection 5.4.1 and Subsection 5.4.2. The domain of these eight factors is essential to specify the space that could be searched. The domain of the demand location and vehicle availability location has been shrunk between 0.05 and 0.95 because of the increase in simulation time brought when simulating values lower than 0.05 and higher than 0.95.

Because we developed the simulation process, we understood which parts of the design space could not be reached because of internal constraints. These constraints have been specified in the experimental design program and are listed here. The disallowed combinations are:

- $\mu_{CTW} - \sqrt{3} * s_{CTW} < 0$
- $\mu_{CTW} + \sqrt{3} * s_{CTW} > 660$
- $\mu_{VTW} - \sqrt{3} * s_{VTW} < 0$
- $\mu_{VTW} + \sqrt{3} * s_{VTW} > 660$

After specifying all constraints and all domain values, we had to specify the optimality criterion needed for the design creation. There are two types of optimality usually used in that type of design. The first one is D-optimality, while the second one is I-optimality. One is designed to maximize the determinant of the information matrix, and the other is designed to minimize the average prediction variance over the design space. One can look at Figure 5.23, which shows the fraction of the design space plot. This summarizes the proportion of the design where the relative prediction variance lies under a given value. The aim is to have a large proportion of the design space with a low relative prediction variance. In our case, we compared both optimality criteria. The ideal for our predictive purpose is thus the I-optimal design, as expected.

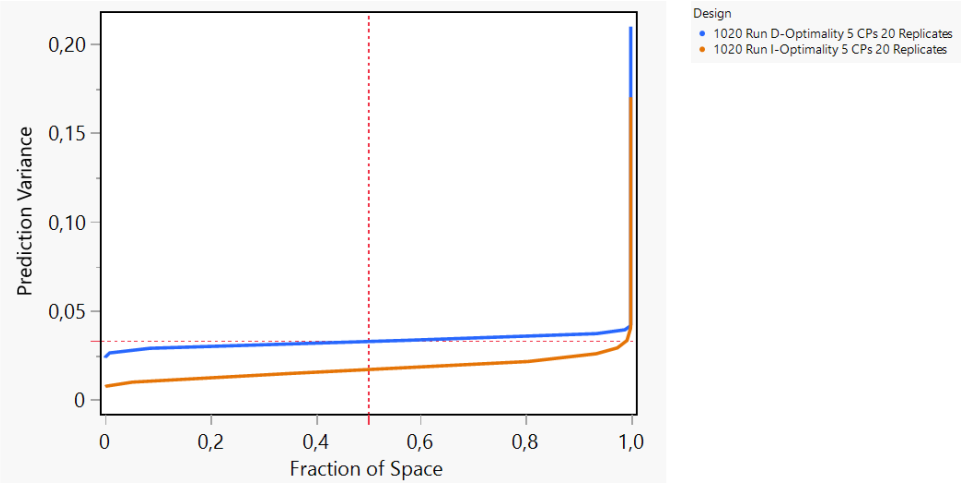

Figure 5.23: Fraction of Design Space plot

Based on a quick estimation of our process flow time, we specified that we could afford around 1000 runs for this DoE and let the algorithm of design selection work. We specified that we wanted 1020 runs with 5 centre points and 20 replicates. Centre points are added to better evaluate the error variance, and replicates help increase the precision of some estimates. Both should help in the end to better determine significant factors. After 5 different runs of the algorithm design creation, the best one was selected. With that amount of runs, the model we could estimate is much bigger than only a model with main effects, two-factor interaction and quadratic effects. However, the sparsity principle states that most of the variability in a measured output is due to variation in a small number of input factors. This is the same principle as the Pareto principle but applied to the design of experiments. We will thus start

with a response surface model and increase the parameters if we need three-factor interaction or cubic terms.

We have exported some design evaluation plots to describe it better. Figure 5.24 shows the expected correlation map of the factors based on the different values tested. We see that the design allows for almost no correlation between the parameters that we want to estimate. The existing correlation appears because of the constraints we set on the domain values.

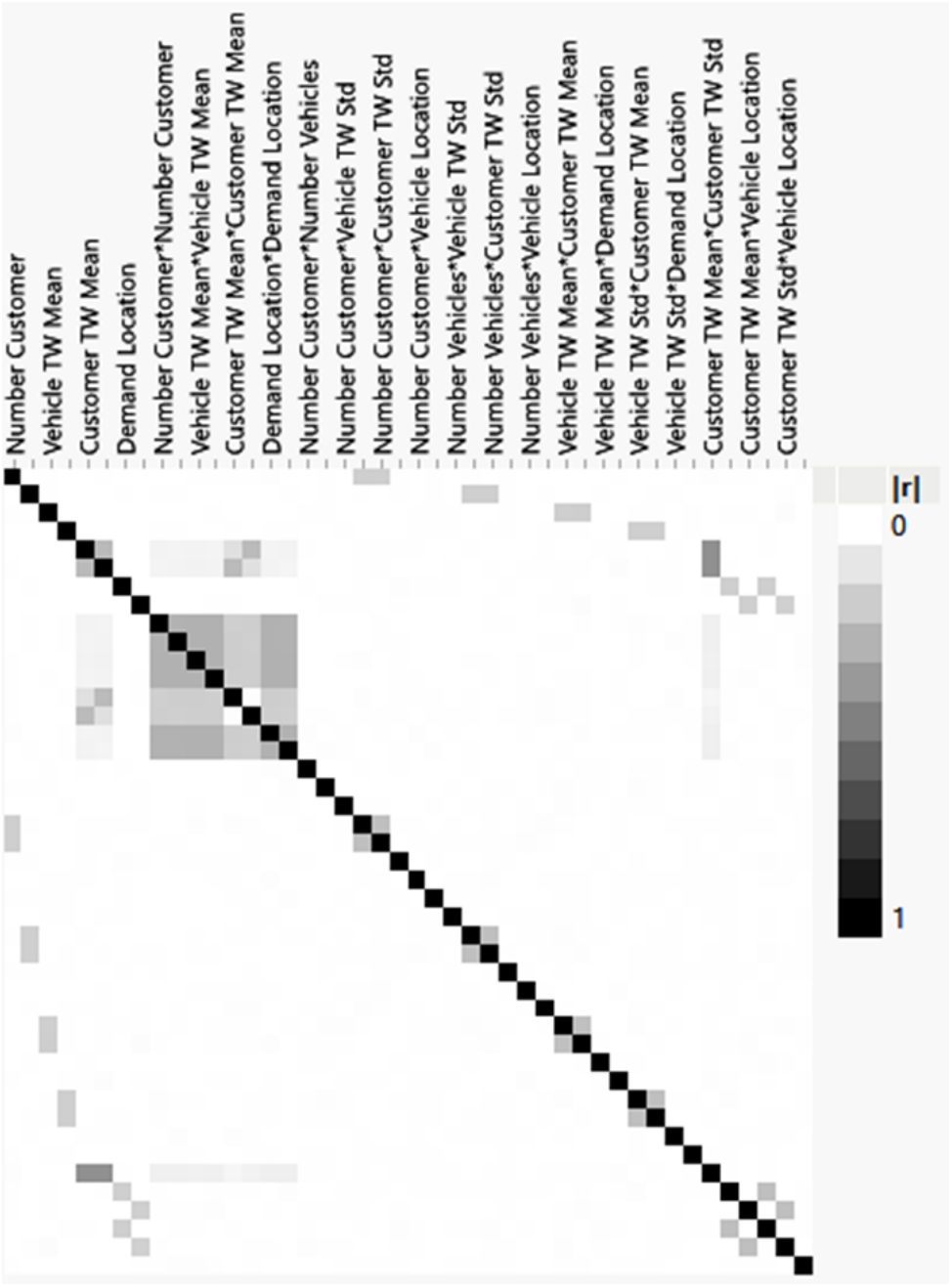

Figure 5.24: Correlation Map of features

In Figure 5.25, we see a 2D map of the points tested over the whole domain of n and m. As there are 6 more features which vary other than these two, one point on the plot represents

multiple points in a larger dimension where all 8 features are included. As one can see, points are scattered over the whole domain, and a centre point is also present.

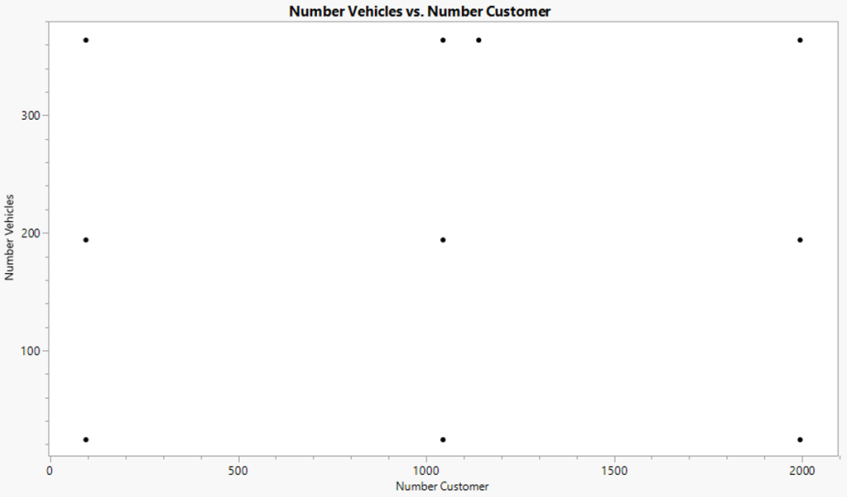

Figure 5.25: Distribution of the number of customers and number of vehicles

The Figure 5.26, represents the distribution of μ_{CTW} and s_{CTW} values that have been chosen for the design. We see that the upper left corner is not visited because of the constraint $\mu_{CTW} - \sqrt{3} * s_{CTW} < 0$, and that points tend to lie on the edge of that constraint border. However, the constraint $\mu_{CTW} + \sqrt{3} * s_{CTW} > 660$ is never active as it is never violated for any values of μ_{CTW} and s_{CTW} in their domains of values that we are interested in. That is why we do not see a lower linear bound on the plot.

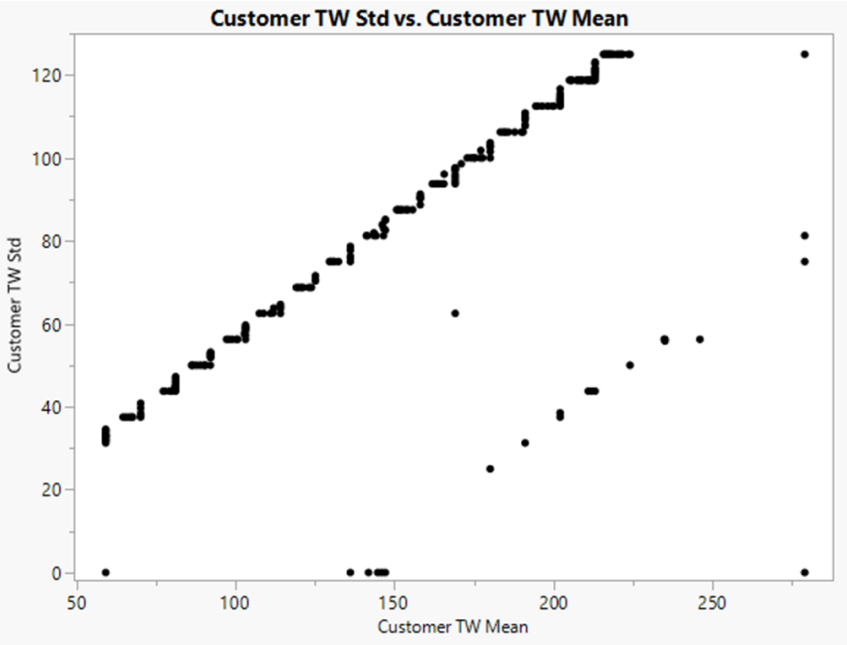

Figure 5.26: Distribution of customer time window lengths' mean and customer time window lengths' standard deviation

In Figure 5.27, the same scatterplot is shown for the distribution μ_{VTW} and s_{VTW}. Here, the constraints are never met, so the whole domain is tested. We see that we test 3 to 4

different values for each factor. That is expected as we only intend to assess the significance of the quadratic effects, not the higher degrees.

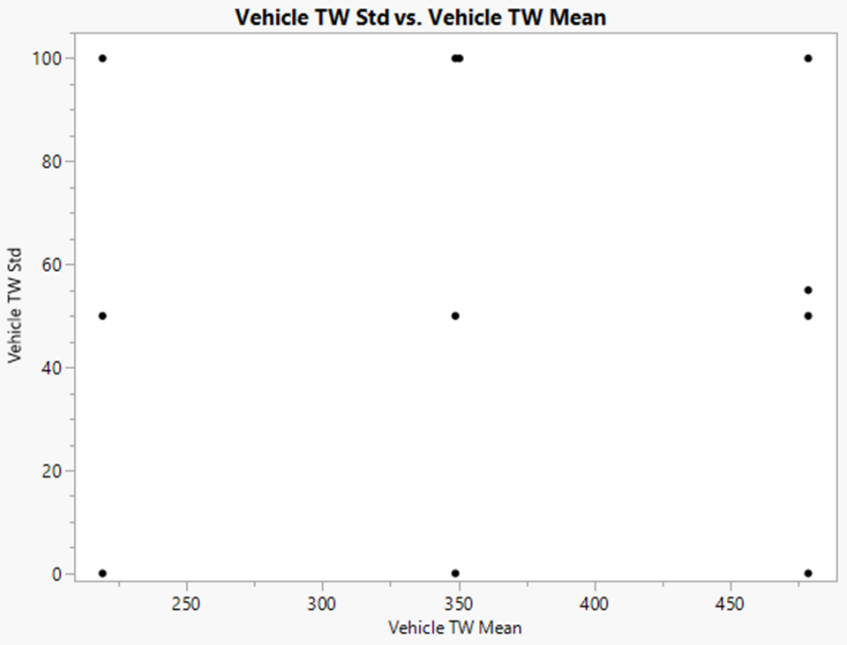

Figure 5.27: Distribution of vehicle time window lengths' mean and vehicle time window lengths' standard deviation

Finally, in Figure 5.28, we show the distribution of l_c and l_v values that are supposed to be tested. As no constraint lies on them, they also span to cover the whole domain.

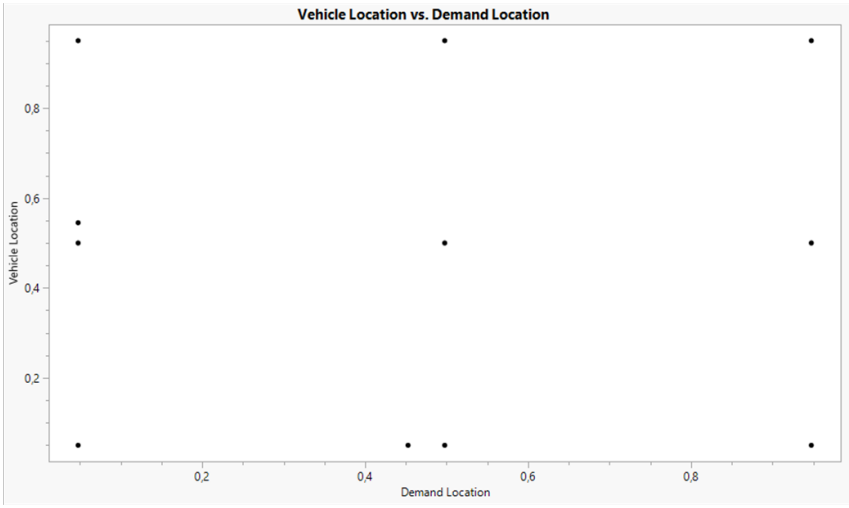

Figure 5.28: Distribution of customer demand location and vehicle availability location

The table shown in Table 5.5 summarizes how efficient is the DoE that we have built. In the DoE setting, efficiency is defined in terms of the standard error of the parameter estimates of the model, Linear Regression/ANOVA in our case. Theoretically, standard errors of the coefficients are bounded below by $\frac{\sigma}{\sqrt{1020}} = 0.031\sigma$ for an experiment with 1020 runs. Here σ is the variance of the error terms in the Linear Regression. *Relative Std Error of Estimate* is the

standard error of the coefficient estimate divided by the error standard deviation (σ). So as we can see from the table, all *Relative Std Error of Estimates* are more than 0.031, and some are very close to it. The coefficient estimates for which *Relative Std Error of Estimates* are much more than 0.031 (in bold) are exactly the features for which we had constraints (Time Window related ones). That is not surprising as the feature value combinations we could test were limited for them. Relating to this, *Fractional Increase in CI Length* is the fractional increase in standard errors/CI lengths of the coefficient estimates compared to the theoretical minimum standard errors/CI lengths. So, this fractional is larger for the features with constraints than for the unconstrained ones.

Table 5.5: Estimated efficiency of each coefficient in the response surface model

Term	Fractional Increase in CI Length	Relative Std Error of Estimate
Intercept	1.754	0.086
Number Customer	0.259	0.039
Number Vehicles	0.259	0.039
Vehicle TW Mean	0.259	0.039
Vehicle TW Std	0.258	0.039
Customer TW Mean	**1.247**	**0.07**
Customer TW Std	**1.242**	**0.07**
Demand Location	0.258	0.039
Vehicle Location	0.258	0.039
Number Customer*Number Customer	**1.462**	**0.077**
Number Vehicles*Number Vehicles	**1.462**	**0.077**
Vehicle TW Mean*Vehicle TW Mean	**1.468**	**0.077**
Vehicle TW Std*Vehicle TW Std	**1.467**	**0.077**
Customer TW Mean*Customer TW Mean	**1.77**	**0.087**
Customer TW Std*Customer TW Std	**1.783**	**0.087**
Demand Location*Demand Location	**1.468**	**0.077**
Vehicle Location*Vehicle Location	**1.461**	**0.077**
Number Customer*Number Vehicles	0.349	0.042
Number Customer*Vehicle TW Mean	0.347	0.042
Number Customer*Vehicle TW Std	0.347	0.042
Number Customer*Customer TW Mean	**0.66**	**0.052**
Number Customer*Customer TW Std	**0.656**	**0.052**
Number Customer*Demand Location	0.348	0.042
Number Customer*Vehicle Location	0.349	0.042
Number Vehicles*Vehicle TW Mean	0.347	0.042
Number Vehicles*Vehicle TW Std	0.347	0.042
Number Vehicles*Customer TW Mean	**0.657**	**0.052**
Number Vehicles*Customer TW Std	**0.658**	**0.052**
Number Vehicles*Demand Location	0.347	0.042
Number Vehicles*Vehicle Location	0.349	0.042
Vehicle TW Mean*Vehicle TW Std	0.346	0.042
Vehicle TW Mean*Customer TW Mean	**0.655**	**0.052**
Vehicle TW Mean*Customer TW Std	**0.656**	**0.052**
Vehicle TW Mean*Demand Location	0.346	0.042
Vehicle TW Mean*Vehicle Location	0.347	0.042
Vehicle TW Std*Customer TW Mean	**0.658**	**0.052**
Vehicle TW Std*Customer TW Std	**0.655**	**0.052**
Vehicle TW Std*Demand Location	0.346	0.042
Vehicle TW Std*Vehicle Location	0.347	0.042
Customer TW Mean*Customer TW Std	**2.089**	**0.097**
Customer TW Mean*Demand Location	**0.657**	**0.052**
Customer TW Mean*Vehicle Location	**0.66**	**0.052**
Customer TW Std*Demand Location	**0.657**	**0.052**
Customer TW Std*Vehicle Location	**0.658**	**0.052**
Demand Location*Vehicle Location	0.347	0.042

5.6 Simulation quality check

With our simulation framework and our DoE ready, we could simulate 1020 instances. After the simulation, we extracted the features as we did previously for ORTEC clients in Subsection 5.2.1. In this section, we will assess the simulation's efficiency by comparing our simulation's results with what was expected from the Design of Experiments.

Figure 5.29 shows the correlation map calculated from the extracted features. We can see that most of the effects we want to understand are not correlated. However, to further assess this, we can compare this figure to Figure 5.24. We can see from this comparison that the simulation was pretty good, as the effects that were supposed to be mildly correlated are mildly correlated, and most effects are not. However, we also see that some correlation arises that was not expected, for example, with the vehicle availability location factor. This could be due to a lack of strong control of this factor during simulation.

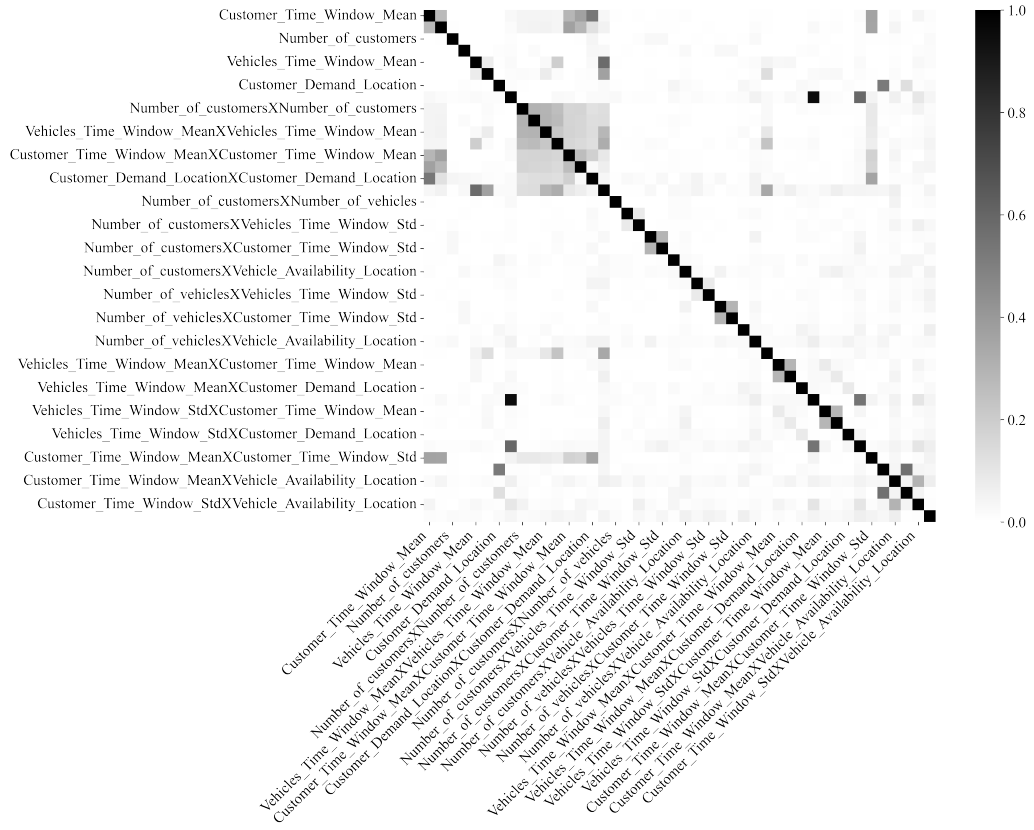

Figure 5.29: Correlation map of simulated instances

Figure 5.30 can be compared with Figure 5.25. We see that the values simulated are quite similar to the ones planned. However, that is not the case for the customer demand and vehicle availability shown in Figure 5.31. We expected a plot such as the one shown in Figure 5.28, but it is clear that we do not control exactly the simulation of the demand location and vehicle availability location.

The distribution of customer time window length mean, and the standard deviation is shown in Figure 5.32. We can see that we obtain the same behaviour as the one expected from Figure 5.26. Our simulation went reasonably well for this factor. Lastly, the distribution of the vehicle time window length mean and the standard deviation is shown in Figure 5.33. Most of

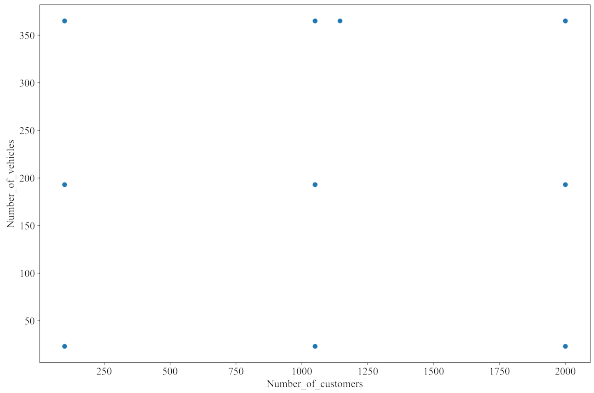

Figure 5.30: Distribution of the number of customers and number of vehicles in the simulated instances

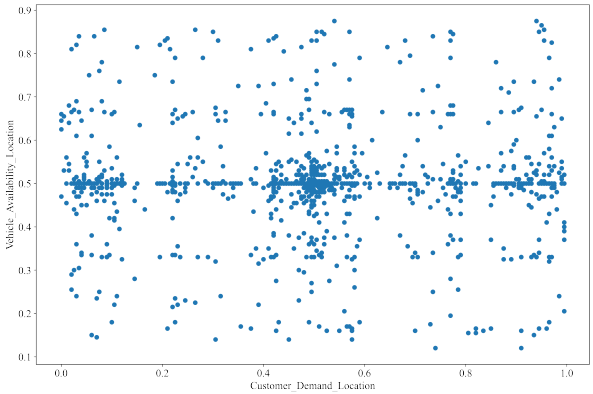

Figure 5.31: Distribution of the customer demand location and vehicle availability location in the simulated instances

the domain is correctly simulated except for a small drift in the upper right corresponding to the simulation of high means and high standard deviations. This comes from the fact that it is hard to simulate instances that tend to exceed the limit of the entire working window (660 minutes). For example, simulating an instance with a time window length mean of 480 and a time window length standard deviation of 100 has more chance to generate working windows higher than 660 than generating time window lengths of mean 240 and a standard deviation of 60. As explained in Subsection 5.4.2, a rejection step occurs when simulating these time windows. If we generate a lot of time windows going beyond the bounds, they are rejected and influence the new distribution of selected time window lengths. This is why we see this drift in generated vehicle time windows when they are supposed to be high. The rejection happens less with lower time windows, and we do not see a drift in the graph at these places.

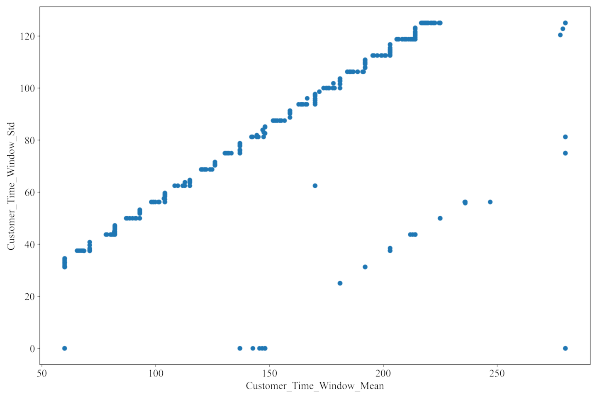

Figure 5.32: Distribution of the customer time window length mean and standard deviation in the simulated instances

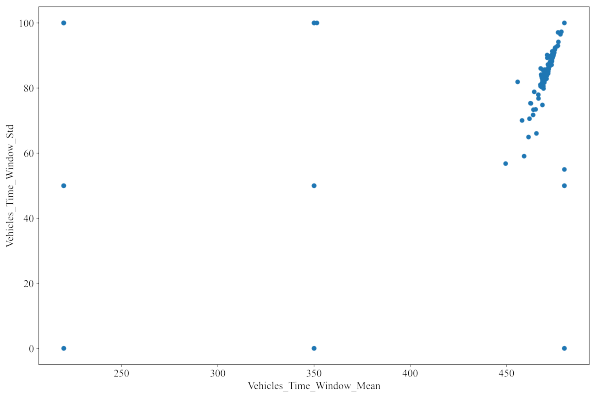

Figure 5.33: Distribution of the vehicle time window length mean and standard deviation in the simulated instances

To have a more precise idea of the efficiency of the simulation, we decided to compute the mean difference between the expected value and the simulated value for each feature. This is reported in Table 5.6 and Table 5.7. As expected, n is perfectly controlled. However, it has been mentioned that the feature m is not controlled perfectly; in practice, the error rate is slightly more than 0%. That is because we want to keep the proportions of the different vehicle

types with different priorities and travel time/service time factors the same as in the original instances. So, when selecting vehicles and respecting the proportions, there are expected to be some rounding errors, e.g. 99 vehicles chosen instead of 100.

Table 5.6: Error rates of perfectly and well controlled features

Feature	Errors rate in %
n	0%
m	1.8%
μ_{CTW}	0%
s_{CTW}	0.5%
μ_{VTW}	0.2%
s_{VTW}	4%

Table 5.7: Error rates of semi-controlled features

Feature	Absolute errors
l_c	0.11
l_v	0.25

We also see that the s_{VTW} is slightly higher than the other features errors, as expected from Figure 5.33 and from the explanation given here above.

We reported absolute errors due to their scale for the *semi controlled* features. For example, when 0.05 is generated for 0.1, we consider it a well-simulated instance as it is hard to get very close to the exact value. It is especially true for the l_c and l_v, which are close to 0 and 1. The absolute errors are up to 0.25, as seen from Figure 5.31.

5.7 Classification on completeness

Now that we have 1020 instances shown to be of a reasonable quality, we are ready for the modelling part of our thesis. The first step we want to take in applying ML to understand the performance of the OHD is to build a classification model that can identify if the algorithm can plan all the tasks. We remind the reader that the output of OHD, where all customer orders were delivered, is called a "complete solution" in our thesis. In contrast, any other output of the algorithm is referred to as an "incomplete solution", although it is not a "solution" in its feasibility sense.

5.7.1 EDA on the input features

Out of the 12 features introduced at the beginning of the chapter, we have controlled 8 features when simulating instances. However, as we are still interested in the impact of 4 other features in the classification model, we will explore all the 12 features $n, m, \mu_{CTW}, s_{CTW}, \mu_{VTW}, s_{VTW}, l_c, l_v, \mu_D, s_D, c, \mu_{cd}$ in our further analysis.

As it can be seen visually from the Figure 5.34, for most of the features except the 3 of 4 features we did not control in our simulation, there is a clear difference in distributions between the complete and incomplete solutions.

If we run the nonparametric Kolmogorov-Smirnov test, which tests for the similarity in the distributions in a statistically robust manner, we get the following results:

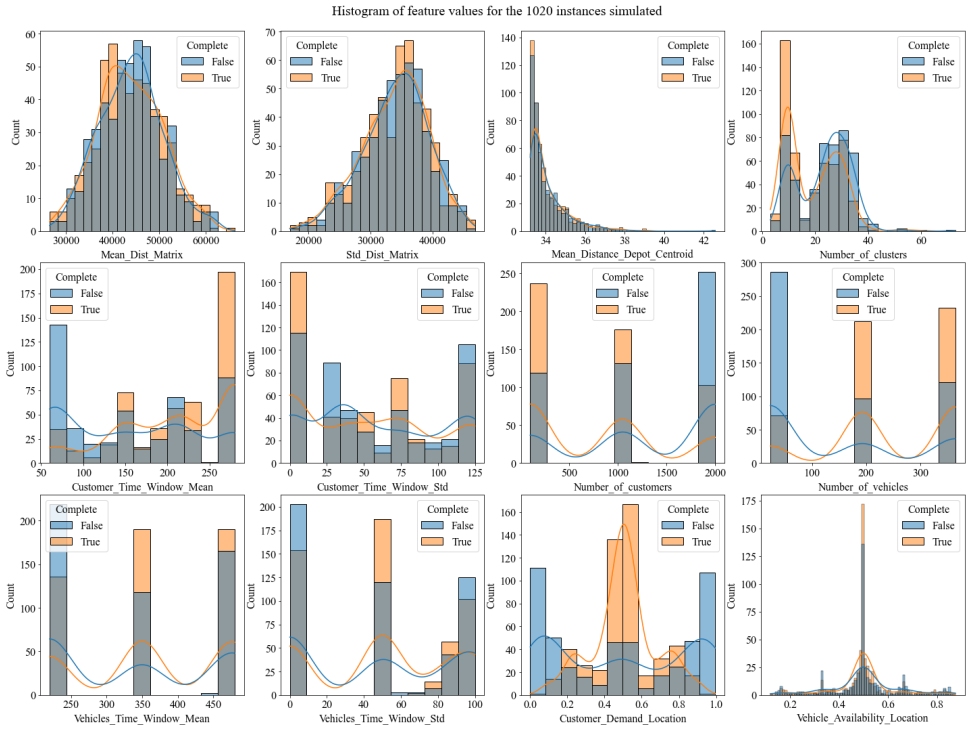

Figure 5.34: Distribution of Complete and Incomplete Solutions

Thus, we proceed to the modelling part using the 9 significant features at the 0.05 level.

Table 5.8: KS test output

Feature	Test Statistic	p-value
Number of Customers	0.30	**3.51e-21**
Number of Vehicles	0.43	**2.57e-43**
Vehicle TW Mean	0.19	**2.60e-08**
Vehicle TW Std	0.12	**0.00095**
Customer TW Mean	0.30	**3.85e-22**
Customer TW Std	0.12	**0.00083**
Customer Demand Location	0.30	**3.31e-21**
Vehicle Availability Location	0.11	**0.00348**
Distance Matrix Mean	0.0	0.37306
Distance Matrix Std	0.04	0.85780
Mean Distance Centroid Depot	0.04	0.69522
Number of Clusters	0.22	**4.888e-11**

5.7.2 Modelling

In this part of the section, we will give the performances of the well-known classification models: Regularized Logistic Regression, Linear Discriminant Analysis, Quadratic Discriminant Analysis, KNN Classification, Polynomial SVM Classification, RBF SVM Classification, and Random Forest Classification.

As some models are sensitive to the scale of the data, they need to be transformed into a common scale. To that end, we will standardize our data by subtracting the mean and dividing it by the standard deviation of each feature.

When it comes to the performance assessment criteria of the models, many choices are possible. Accuracy, precision, recall, F1-score and many others. However, as the dataset is almost perfectly balanced with 516 complete and 504 incomplete solutions, it is reasonable to use accuracy. We have also split the data into training and test sets with 70% and 30% ratios as it is commonly done. This is unnecessary for parametric models like Linear Discriminant Analysis or Quadratic Discriminant Analysis, where the chances of overfitting are small. However, to be consistent in comparing with other non-parametric models, we check their performances on an independent test set.

One also has to note that some models are families of different models with specific parameter values. So whenever we needed to specify the values of some of the parameters, 10-fold cross-validation was carried out. This ensures that we select the best-performing model with the right parameters, and it has another advantage of ensuring that the model does not overfit.

As a first step, we consider only the main effects of the features in our models. After training the models and, we get the following performances given in Table 5.9 where all models except Regularized Logistic Regression, Linear Discriminant Analysis, and Quadratic Discriminant Analysis perform similarly well for both the training and test set, generalizing well.

It is evident that while solving the problem of CVRPTW heuristically using OHD, the features do not just impact the output independently. For example, given that the Customer Demand Location being close to 0.1 does not mean a lot by itself, but combined with the

Table 5.9: Performance of the models with the main effects only

Model	Training set Accuracy	Test set Accuracy
Regularized Logistic Regression	0.79	0.77
Linear Discriminant Analysis	0.79	0.78
Quadratic Discriminant Analysis	0.90	0.87
KNN Classification	1	0.90
Polynomial SVM Classification	0.96	0.90
RBF SVM Classification	0.96	0.95
Random Forest Classification	1	0.9

Vehicle Availability Location being close to 0.9 might result in a more challenging CVRPTW instance as the time when vehicles can deliver and time when orders need to be delivered, are very different. So motivated by that, we ran the same models with all interaction effects added.

The performances are given in Table 5.10, where almost all models, including the three mentioned before, have improved in their performances without overfitting. The Regularized Logistic Regression and Linear Discriminant Analysis perform slightly worse but generalize much better to the test set.

Table 5.10: Performance of the models with the interactions added

Model	Training set Accuracy	Test set Accuracy
Regularized Logistic Regression	0.84	0.82
Linear Discriminant Analysis	0.82	0.80
Quadratic Discriminant Analysis	0.94	0.87
KNN Classification	1	0.86
Polynomial SVM Classification	0.98	0.91
RBF SVM Classification	0.99	0.90
Random Forest Classification	1	0.91

One can notice that the best-performing models Quadratic Discriminant Analysis, KNN Classification, Polynomial SVM Classification, RBF SVM Classification, and Random Forest Classification, have non-linear decision boundaries compared to the worst-performing Linear Discriminant Analysis and the Regularized Logistic Regression, for example. Thus, we tried to add a quadratic effect into the model to add some "curvature". Hopefully, this will help improve the performances of our models, especially those with only linear decision boundaries.

As expected, the Linear Discriminant Analysis and the Regularized Logistic Regression increased in their performances, passing the accuracy of 0.90 for the training set (Table 5.11). Overall, all the models seem to perform well when including all the main effects, interactions, and square effects.

As a next step in our analysis, we would like to provide some explanations on what are the variables which are impacting the completeness of the solution. For that, we select one of the models we have run. As one can notice, the marginal differences between the models'

Table 5.11: Performance of the models with the interactions and square effects added

Model	Training set Accuracy	Test set Accuracy
Regularized Logistic Regression	0.96	0.93
Linear Discriminant Analysis	0.94	0.89
Quadratic Discriminant Analysis	0.96	0.90
KNN Classification	1	0.87
Polynomial SVM Classification	0.96	0.94
RBF SVM Classification	0.96	0.92
Random Forest Classification	1	0.91

performances are minimal. Therefore, we decided to proceed with Regularized Logistic Regression because of its advantage of interpretability.

Regularized Logistic Regression

Regularized Logistic Regression is a more general version of Logistic Regression where we add a penalty term to the cost function, which needs to be minimized. Having this penalty term helps avoid overfitting when we have many features, which is the case in our application (54 features). There are two types of penalty terms:

- L_1 for which the cost function includes $C\|\beta\|_1$
- L_2 for which the cost function includes $\frac{C}{2}\|\beta\|_2^2$

where β is the vector of coefficients in Logistic Regression.

In building the model, we must select which penalty term we want to use and the strength of the penalty C. The 10-fold cross-validation has given the optimal parameters of L_1-penalty with the regularization parameter of $C = 0.46$. Theoretically and in practice, using L_1-penalty means that some of the features in the model are given the coefficient of 0. In other words, they are ignored and do not impact the model's output. As for the value of C, smaller values mean stronger regularization, leading to models that do not overfit.

In this specific model, there are 37 features with non-zero coefficients out of 54 features. As it is not an easy task to list them all and also interpret them, we decided to report the coefficients of 7 most impactful ones with the largest coefficients in magnitude in Table 5.12

The model can be summarized in the following way:

$$ln(\frac{P(complete)}{P(incomplete)}) = 4.1 - 3.13l_c^2 + 2.73m - 1.96n - 1.48m^2 + 1.22\mu_{CTW} + 0.84\mu_{VTW} + 0.77l_c l_v + ...$$

In other words, in (regularized) logistic regression, *the coefficient of a specific feature is interpreted as the log increase in odds when we increase the value of the feature by 1 when other features are kept constant.*

Following this rule, there are several possible interpretations we could infer from the coefficients in the table:

$\underline{l_c^2}$

Table 5.12: Coefficients of the features for the Regularized Logistic Regression

Feature	Coefficient
Customer Demand Location^2	-3.13
Number of Vehicles	2.73
Number of Customers	-1.96
Number of Vehicles^2	-1.48
Customer TW Mean	1.22
Vehicle TW Mean	0.84
Customer Demand Location X Vehicle Availability Location	0.77

For this feature, the original values of l_c are in between [0, 0.995], and after the standardization, it is in between [-1.75, 1.79]. So the values of l_c^2 are between [0, 3.20]. In our model, increasing l_c^2 decreases the odds of the solution being complete as the feature's coefficient is negative. Large values in the standardized data translate to the original l_c being close to 0 or 1. That means the customer demand locations close to the beginning or the end of the delivery period make it harder for OHD to achieve a complete solution.

$\underline{m, m^2}$

For this feature, the original values of m are in between [23, 365], and after the standardization, it is in [-1.18, 1.20]. In the model, m comes as 2.73m-1.48m^2, which increases if m increases for almost all m in [-1.18, 1.20]. In the original scale, having more vehicles increases the odds of reaching a complete solution. This is unsurprising as having more vehicles should help deliver all the orders.

$\underline{n}$

For this feature, the original values of n are in between [100, 2000], and after the standardization, it is in [-1.21, 1.20]. Increasing n decreases the odds of the solution being complete, as the feature's coefficient is negative. Obviously, increasing the number of customers should decrease the odds of reaching a complete solution.

$\underline{\mu_{CTW}}$

For this feature, the original values of μ_{CTW} are in between [60, 280], and after the standardization, it is in [-1.54, 1.23]. Increasing μ_{CTW} increases the odds of the solution being complete as the feature's coefficient is positive. Instances with customers having larger Customer TW Mean have more freedom in delivery time. So the instance should have higher odds of having a complete solution.

$\underline{\mu_{VTW}}$

For this feature, the original values of μ_{VTW} are in between [220, 480], and after the standardization, it is in [-1.22, 1.20]. Increasing μ_{VTW} increases the odds that the solution is complete as the coefficient of the Vehicle TW Mean feature is positive. Instances with larger Vehicle TW Mean, thus more time for delivery, should have higher odds of complete solutions.

$\underline{l_c l_v}$

For l_c, the original values are in between [0, 0.995], and after the standardization, it is in [-1.75, 1.79]. For l_v, the original values are in between [0.12, 0.875], and after the standardization, it is in [-2.92, 2.84]. The interaction $l_c l_v$ most likely explains the effect of the difference between the times when vehicles are available and the most customer demand.

When l_c and l_v are of the same sign (both being on the same side of 0.5 in the original scale), this adds a positive effect to the log odds ratio as the coefficient is positive. When they are of different signs, then this adds a negative effect to the log odds ratio.

5.8 Regression on the percentage of planned tasks

After this first classification model, we are left with two types of instances that behave differently in the OHD algorithm. The first group is the "complete solution" group, where all orders are delivered. The second group is the "incomplete solution" group, where not all orders are delivered, although not considered a solution in the CVRPTW feasibility sense. Nevertheless, it can bring insight into how close the algorithm was to assigning all orders and what features are essential for this algorithm to assign more orders.

The target variable will thus be the ratio of orders planned by the algorithm to the total amount of tasks in the instance, expressed as a percentage. Usually, the analysis of data gathered by Design of Experiments is done by linear regression and ANOVA to assess the importance of each feature in predicting the output variable. We are thus going to analyze the 8 controlled features as well as their interactions and quadratic effects to complete the Design of Experiments approach. However, we will conduct an exploratory analysis of all extracted features (including 4 uncontrolled features) and develop further models containing them. By doing that, we can assess the marginal increase in the models' performances by including 4 more extracted features.

5.8.1 EDA on the input features

As seen visually from the Figure 5.35, a few factors seem to have a relationship with the percentage of tasks planned, mainly the number of customers and vehicles. This should be investigated with an ANOVA analysis and proper statistical tests.

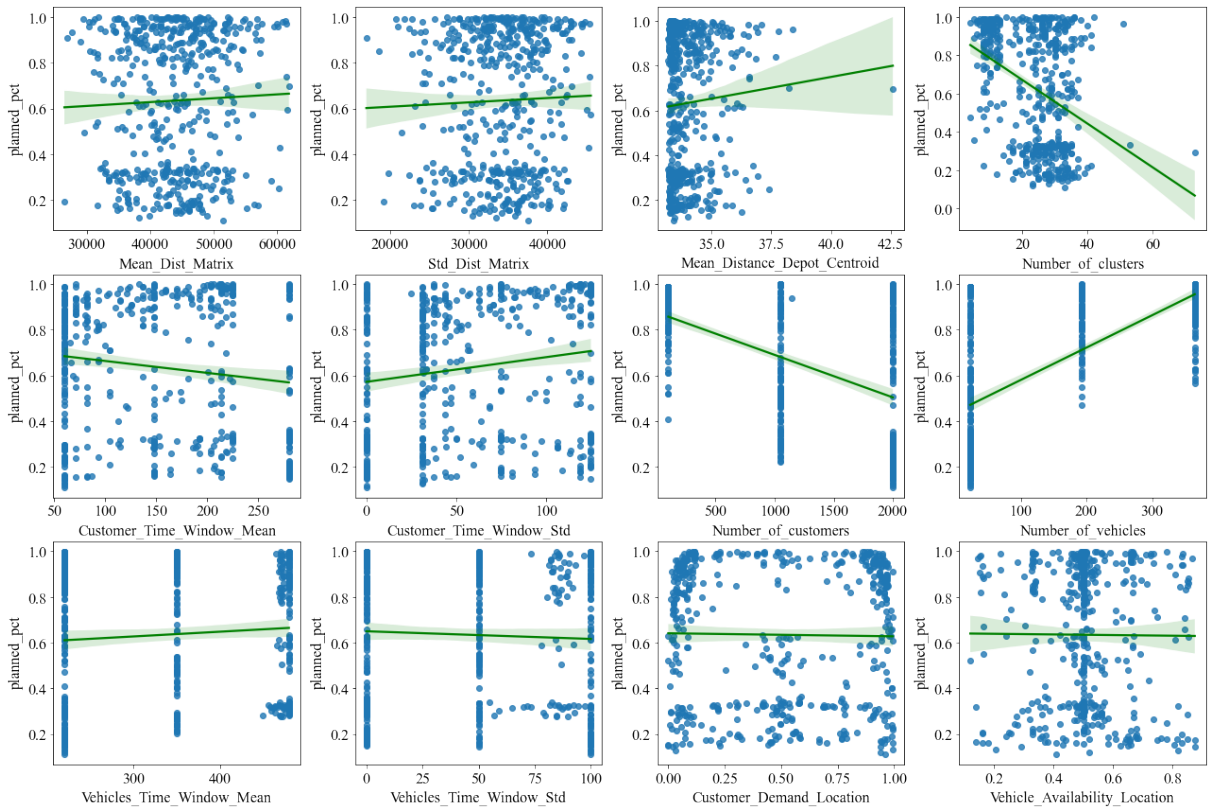

Figure 5.35: Feature values vs the percentage of planned tasks, along with a regression

Figure 5.36 shows a scatter plot of two factors (number of vehicles and customers) coloured by the percentage of planned tasks. We can see by the colouring that the effect of the number of customers is different depending on the level of the number of vehicles. This is a sign that there might be an interaction between those two factors. But, again, this needs to be adequately stated with statistical tests.

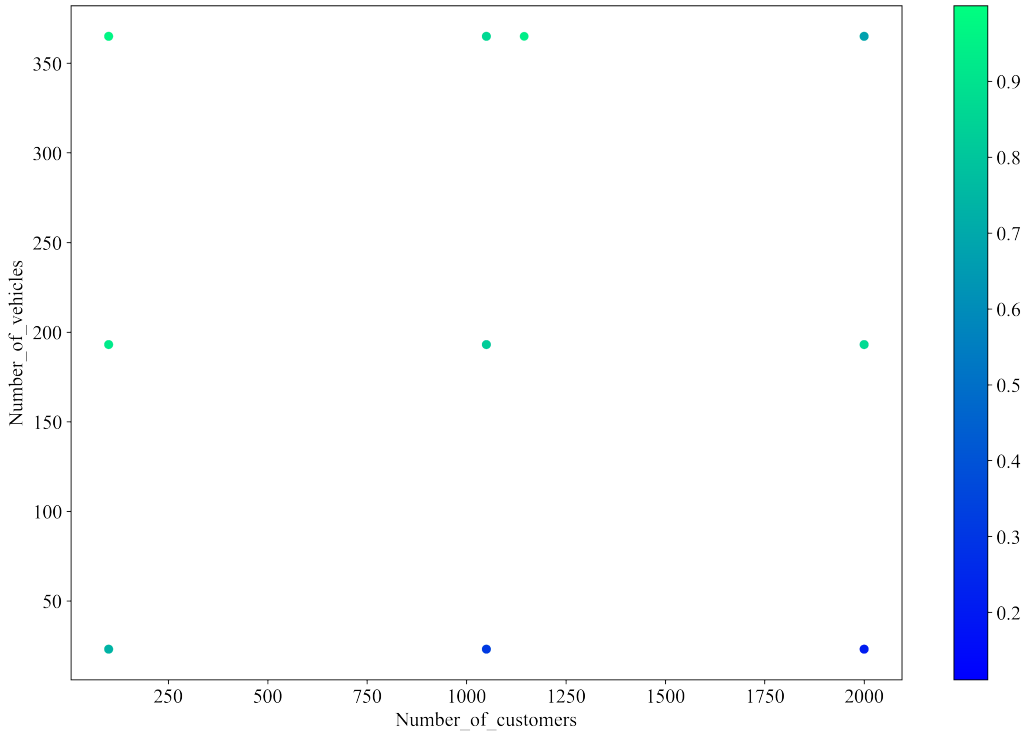

Figure 5.36: Number of customers vs number of vehicles coloured by percentage of planned tasks

5.8.2 Response surface model

In this part, we have conducted a linear regression on the model containing all main, interaction, and quadratic effects. We have then refined the effects by forward and backward selection to achieve a model with only significant parameters. An ANOVA analysis has also been carried out and outputs the same significant effects as the one selected from the forward/backward selection. In this case, we only consider the eight controlled factors in finalizing the Design of Experiments approach. Out of the 44 model parameters, 15 were significant. The list and their coefficient are reported in Table 5.13.

$\underline{m}$

One can see that the number of vehicles is the most important feature in describing the percentage of tasks that will be planned. That could come from the fact that there is a significant capacity constraint that arises when we decrease the number of vehicles. As the coefficient is positive, it means that by increasing the number of vehicles by 1 standardized unit, the percentage of tasks planned increases by 0.33%.

Table 5.13: Significant parameter in the linear response surface model of planned percentage

Parameter	Coef	P-value	F value in ANOVA	P-value
Number_of_vehicles	0.3323	0.000	3730.537727	1.044746e-230
Number_of_customers	-0.1409	0.000	2114.268157	1.701909e-179
Number_of_customersXNumber_of_vehicles	0.0830	0.000	812.313109	6.395552e-106
Number_of_vehiclesXNumber_of_vehicles	-0.1412	0.000	497.094831	1.866191e-76
Customer_Time_Window_Mean	0.0532	0.000	207.997009	1.570236e-39
Vehicles_Time_Window_Mean	0.0528	0.000	220.345616	2.107376e-41
Number_of_customersXNumber_of_customers	0.0661	0.000	110.694733	1.805233e-23
Customer_Demand_LocationXVehicle_Availability_Location	0.0299	0.000	64.737931	6.553603e-15
Customer_Time_Window_MeanXNumber_of_vehicles	0.0360	0.000	60.038208	5.437045e-14
Customer_Demand_LocationXCustomer_Demand_Location	-0.0519	0.000	53.031083	1.325145e-12
Customer_Time_Window_MeanXNumber_of_customers	-0.0257	0.000	24.675494	9.399531e-07
Number_of_vehiclesXVehicles_Time_Window_Mean	-0.0157	0.000	23.014057	2.140160e-06
Customer_Time_Window_StdXNumber_of_customers	0.0142	0.001	10.620110	1.196682e-03
Customer_Time_Window_StdXNumber_of_vehicles	-0.0111	0.011	8.173533	4.432516e-03
Customer_Time_Window_MeanXCustomer_Time_Window_Mean	-0.0098	0.042	4.177476	4.150099e-02

$\underline{mn}$

The coefficient of this interaction is positive. This means that increasing the value of one of both features also increases the impact of the other features on the target variable. For example, at value 0 of the number of customers, $y = 0.3323m$, while at value 1 of the number of customers, $y = 0.3323m + 0.083m = 0.4153m$ (if we only look at the effect of the number of customers and neglect other effects).

$\underline{\mu_{CTW} \text{ and } \mu_{VTW}}$

They both have positive coefficients. This means that increasing the time window length of customer and vehicle increases the percentage of tasks planned. That is quite expected as it relaxes some delivery constraints and allows for more possible solutions in the delivery scheme.

$\underline{l_c l_v}$

As in the classification model, we see an interaction between the location of the peak demand of the customers and the peak availability of the vehicles. This interaction is negative, as expected, because we expect that the further apart these peaks are, the more difficult it gets to plan all tasks.

$\underline{l_c^2}$

This quadratic effect in the customer demand location states that the effect has a parabolic behaviour. As the coefficient is negative, extreme values of customer demand location in the domain decrease the percentage of planned tasks. In contrast, values in the middle of the domain do not negatively impact the percentage of planned tasks.

Not all effects have a clear concrete explanation, but the most explanatory ones have been discussed here.

5.8.3 Modelling

In this part, we will give the performances of the well-known regression models: Linear Regression, Nearest Neighbor Regression, Polynomial Kernel SVM Regression, RBF Kernel SVM Regression, and Random Forest Regression. The purpose here is more about achieving

the best predictive performances than having great explanatory power.

All models were developed on standardized data to remove multi-collinearity and fix the scale problem for some models. When comparing models, we will use several criteria to assess the models' performances well. The AIC will not be computable for all our models, so we will not consider it. However, we'll look at R^2_{adj}, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) and *MAE%*. Comparing R^2_{adj} between training and test set can help us understand the model's generalization. Looking at values of *MAE* and *RMSE* allows us to understand better the nature of the errors made. Finally, computing *MAE%* by dividing the *MAE* by the mean of the true values, one can know the percentage of error we make when predicting a certain value.

One also has to note that some models are families of different models with specific parameter values. So whenever we needed to specify the values of some of the parameters, 10-fold cross-validation was carried out. This ensures that we select the best-performing model with the right parameters, and it has another advantage of ensuring that the model does not overfit.

As a first step, we consider models with all main, interaction and quadratic effects of our 8 controlled parameters. As a result, we get the following performances (Table 5.14):

Table 5.14: Modelling with response surface model of 8 factors

Model	Training R^2 adj	R^2 adj	MAE	RMSE	MAE%
Linear Regression	0.9427	0.8915	0.066	0.086	10.73%
KNN Regression	0.9999	0.7293	0.091	0.136	14.68%
Polynomial SVM Regression	0.9295	0.8794	0.075	0.091	12.04%
RBF SVM Regression	0.9402	0.8754	0.076	0.092	12.25%
Random Forest Regression	**0.9923**	**0.9444**	**0.042**	**0.061**	**6.95%**
Beta Regression	0.9626	0.9285	0.053	0.069	8.51%

Table 5.14 shows that all models are pretty good, even though they lack a bit of generalization between training and test set. The best model here is the random forest model that yields a 94% R^2_{adj}. Overall, it seems to work well.

We wanted to see if we could achieve the same performance with fewer parameters. Thus we tried building models with the parameters found significant by our ANOVA analysis. Those models are reported in Table 5.15. Here, we achieve almost the same *MAE* and *RMSE* but with fewer parameters. That leads to a general improvement of the R^2_{adj}. The best model is still the Random Forest regression.

Table 5.15: Modelling with response surface model of limited factors

Model	Training R^2 adj	R^2 adj	MAE	RMSE	MAE%
Linear Regression	0.9396	0.8973	0.066	0.084	10.63%
KNN Regression	0.9999	0.9571	0.040	0.061	6.48%
Polynomial SVM Regression	0.9363	0.9029	0.075	0.092	12.15%
RBF SVM Regression	0.9588	0.9376	0.063	0.073	10.16%
Random Forest Regression	**0.9950**	**0.9632**	**0.040**	**0.057**	**6.46%**
Beta Regression	0.9634	0.9506	0.050	0.065	7.99%

Finally, we wanted to gather insights about the 4 other features we did not control but extracted. From our exploratory data analysis, they could impact the percentage of planned tasks, and we wanted to look at that from a more statistical point of view. We conducted the same ANOVA analysis as before to detect the significant effects. We selected only significant effects to build models with (20 in total), and these results are shown in Table 5.16. Overall, almost none of the 4 uncontrolled factors had a significant effect. The only effects barely detected were one interaction with the mean of the distance matrix and one with the standard deviation of the distance matrix. We see a slight decrease in performance compared to the limited models with only 8 factors. Thus, these two interactions seem to have only a tiny impact that explains barely anything.

Table 5.16: Modelling with response surface model of 12 limited factors

Model	Training R^2 adj	R^2 adj	MAE	RMSE	MAE%
Linear Regression	0.9480	0.9169	0.065	0.084	10.47%
KNN Regression	0.9999	0.9381	0.048	0.072	7.75%
Polynomial SVM Regression	0.9410	0.9074	0.071	0.089	11.47%
RBF SVM Regression	0.9516	0.9235	0.064	0.080	10.40%
Random Forest Regression	**0.9933**	**0.9595**	**0.041**	**0.058**	**6.9%**
Beta Regression	0.9641	0.9463	0.051	0.067	8.15%

5.9 Regression on the time to reach the plateau

As explained in Chapter 2, OHD is a heuristic algorithm with a limited number of steps. At every step, OHD tries to improve the solution by optimizing the hierarchical objective values. When the algorithm is terminated, we obtain a solution with great details. However, besides the final solution output, the OHD algorithm also outputs information about the (possibly incomplete) solutions obtained at every optimization step. That includes the time passed since the beginning of the optimization, tasks scheduled, and cost.

In the algorithm, the number of tasks delivered must be maximized first, followed by other objectives. This means that, in fact, for all complete solutions that we obtain from the OHD, there is a time when the algorithm finds a feasible solution. And this is followed by the steps taken to optimize other objective values. Our goal in this part of the section is to predict the time (in seconds) it takes for the OHD algorithm to reach a feasible solution of "good enough" quality. Here we define the "good enough" solution as the one with a slightly higher cost but saves a reasonable amount of time by being accepted before we wait for the algorithm to terminate.

To better explain how exactly we define a "good enough" solution for the OHD algorithm, it is useful to look at the following graph (Figure 5.37):

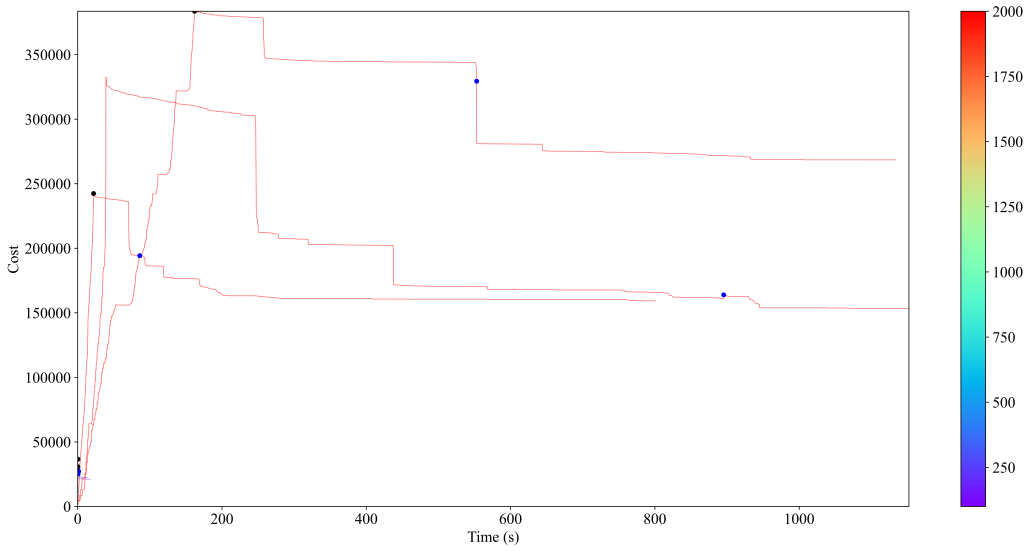

Figure 5.37: Cost decrease of instances coloured by the number of customers in instance

For this figure, we have selected a fraction of all complete solutions that OHD achieved and plotted the cost decrease over the optimization process time. We can see that the costs are not monotonically decreasing. That is because the number of tasks delivered is the first objective (not the cost). However, as seen, the cost decreases after the OHD finds the first feasible solution (given as black points). We coloured the lines depending on the number of customers in the instance. This is purely for exploratory purposes, and any other variable of interest could replace the number of customers.

The blue points are the "good enough solutions" and are defined as follows:

1. For every complete solution, find when the OHD was able to find a feasible solution (black point).

2. Find the point at which the cost of optimization was 22% more than the final cost.

3. Take the latest of two points. This way, we can guarantee that a good enough solution is still feasible. That is why we see only one blue point on one of the line plots where the black and the blue points match. This happens when the solution with 22% more cost is encountered before the feasible solution.

One can potentially replace the 22% with any other percentage he/she might be interested in and consider it a "good enough" solution. However, as we do not have any preference for any percentage, we have analyzed the percentage that gives a good balance between the time saved and the percentage. First, for each complete instance, we calculate the percentage of time saved when we accept the solution with p% more cost for p between 1 and 100 with the increment of 1. Then, we average the time saved in percentages for each cost percent level over all the complete instances. Thus we get the following plot which shows the average time saved in percentages for p between 1 and 100 (Figure 5.38).

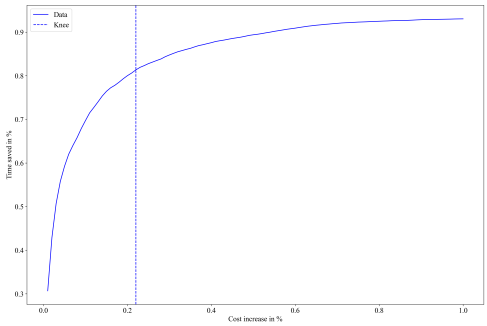

Figure 5.38: Average time saved in percents

Then, as we can see, 22% is selected as the knee point that identifies the point at which the trade-off between the p and the time saved in percent is optimal.

In the modelling part, we will refer to the "good enough" solution/blue point as the plateau of the optimization. Time it takes to reach the plateau will be called "plateau time".

5.9.1 EDA on the input features

As with the previous analysis, we wanted to see how each variable behaved compared to the target variable. This is shown in Figure 5.39. From that, we can directly see that number of vehicles and customers will have an effect on the plateau time. Other effects are harder to see on the graph and will be defined by linear regression and ANOVA.

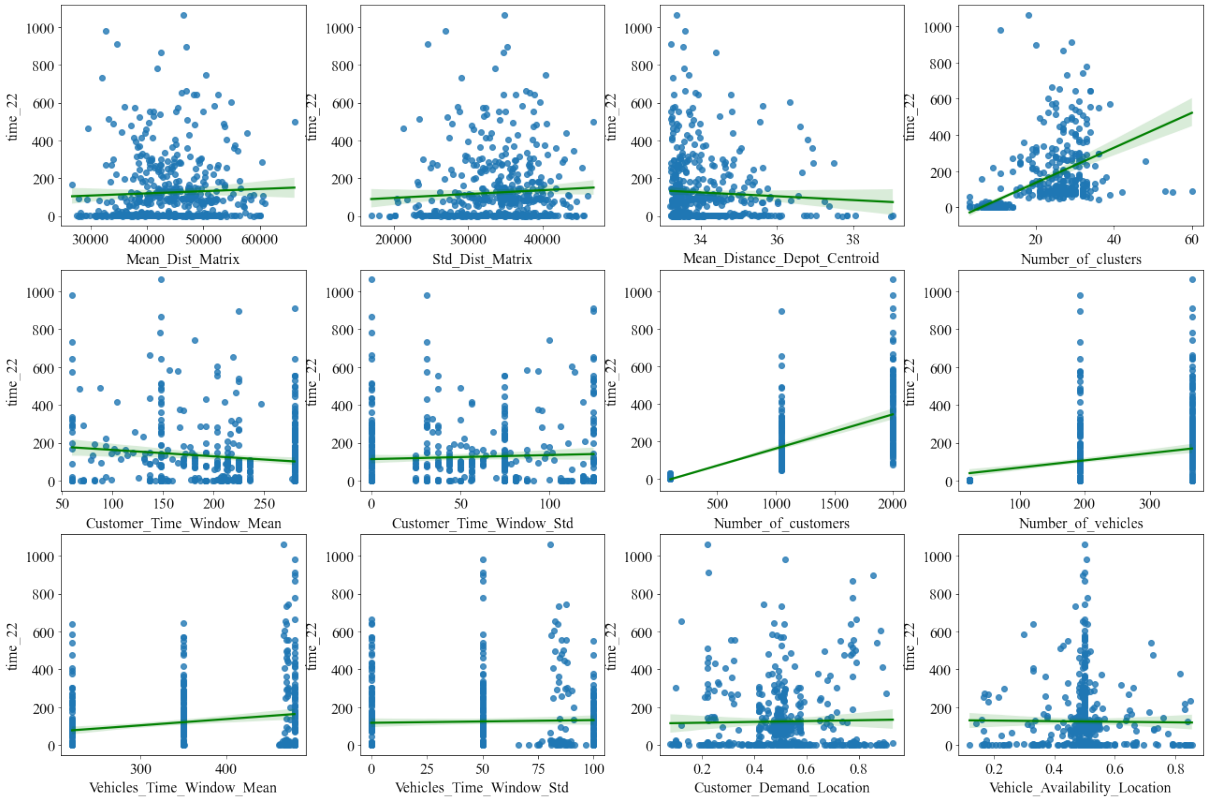

Figure 5.39: Feature values vs the plateau time, along with a regression

5.9.2 Response Surface Model

As for the percentage of planned tasks, we will first delve into the linear regression/ANOVA analysis to end the Design of Experiments flow process and have significant explanatory power. The linear model has been refined by forward and backward selection to get to a model with only significant terms. The table of those values is shown in Table 5.17. Features given in the list largely overlap with the significant features from the ANOVA tests.

$\underline{n}$

One can see that the number of customers is the first significant variable to predict the time we reach the plateau. That is not a surprise as it clearly appeared from Figure 5.37 that the algorithm took more time for instances with more customers.

$\underline{l_c^2}$

The customer demand location squared is also significant. As for the percentage of planned tasks, having a demand that lies in the extreme seems to decrease the algorithm's efficiency. In this case, it increases the time of reaching the plateau, as the coefficient is positive.

$\underline{\mu_{CTW}}$

The effect of the customer time window length mean is negative, meaning that if we increase the time window length, the algorithm takes less time to reach the plateau. That is expected as a wider time window of the customer relaxes time constraints. Thus, a quicker optimal solution can be found.

$\underline{\mu_{CTW} n}$

The interaction between the mean of the customer time window length and the number

Table 5.17: Significant parameter in the linear response surface model of plateau time

Parameter	Coef	P-value	F value in ANOVA	P-value
Number_of_customers	136.7895	0.000	1227.075859	2.439712e-136
Customer_Demand_LocationXCustomer_Demand_Location	22.1975	0.000	84.126407	1.234611e-18
Customer_Time_Window_MeanXNumber_of_customers	-39.1342	0.000	70.346080	5.178711e-16
Customer_Time_Window_MeanXCustomer_Time_Window_Mean	10.7200	0.012	45.856936	3.601276e-11
Customer_Time_Window_StdXNumber_of_customers	24.6297	0.000	37.230064	2.113994e-09
Vehicles_Time_Window_MeanXVehicles_Time_Window_Std	21.7506	0.000	26.157717	4.499974e-07
Vehicles_Time_Window_Mean	31.2797	0.000	24.588741	9.755443e-07
Number_of_customersXVehicles_Time_Window_Mean	21.4028	0.000	28.778539	1.246090e-07
Vehicles_Time_Window_MeanXVehicles_Time_Window_Mean	19.7869	0.001	15.605860	8.929977e-05
Customer_Time_Window_Std	16.7074	0.000	13.488130	2.661155e-04
Customer_Time_Window_Mean	-22.3779	0.000	12.705196	3.998430e-04
Number_of_customersXNumber_of_vehicles	21.8989	0.000	8.941469	2.925930e-03
Number_of_vehicles	17.7617	0.000	10.574888	1.224205e-03
Vehicles_Time_Window_Std	11.5404	0.003	9.078718	2.717938e-03
Customer_Time_Window_MeanXVehicles_Time_Window_Mean	-9.4381	0.014	7.045395	8.201680e-03
Customer_Demand_Location	13.1926	0.001	5.761495	1.674790e-02
Customer_Demand_LocationXNumber_of_customers	11.7679	0.002	9.311161	2.399416e-03
Number_of_vehiclesXVehicles_Time_Window_Std	8.9289	0.014	6.056039	1.419771e-02

of customers is also significant, with a negative coefficient. That means that if we increase by one standardized unit the number of customers, the effect of the customer time window length mean is even more negative than it alone. In the opposite sense, if we increase by one standardized unit the customer time window length mean, the effect of the number of customers is decreased by 39.13. That means that increasing the number of customers has a less negative effect when the customer time window length mean is high than when it is low.

$\underline{\mu_{VTW}}$

The vehicle time window mean is significant with a positive coefficient. This means that increasing the time window mean of the vehicle increases the time we reach the plateau. That might be a bit surprising as we relax time constraints if the vehicle service length is higher, but it does not help the algorithm. That could be because increasing the time window length allows the algorithm to try to insert a lot of customers to the vehicles taking more time.

$\underline{\mu_{VTW}n}$

The interaction between number of customers and vehicle time window length mean is also significant. As the coefficient is positive, increasing by one standardized unit the number of customers increases the effect of vehicle time window length mean by 21.40. So the effect of vehicle time window length mean is even greater on the time we reach the plateau. That also means that the effect of the number of customers on the time to reach the plateau is greater when the vehicle time window length mean is greater. This can also be explained by the same hypothesis as the one stated above.

$\underline{\mu_{VTW}^2}$

The vehicle time window length mean also has a quadratic effect with a positive coefficient. That means, as for the customer demand location effect, that a time window length around the center of the interval (0 mean) is the best setting to minimize the plateau time and that going more towards the edges of the domain increases the time the plateau is reached. This highlights the fact that there are several effects of the time window length mean on the plateau time. Although choosing a low time window length mean is favoured from a linear point of view, the quadratic effect also shows that it is not ideal. The ideal point to minimize the plateau time would lie somewhere in the middle but depends on the values of the other factors

that interact with vehicle time window length mean.

$\underline{m}$

The number of vehicles significantly affects the time it takes for the algorithm to reach the plateau. To be precise, the effect is such that if we increase the number of vehicles, the time the plateau is reached is greater. That could come from the fact that the algorithm has a bigger group of vehicles to try and select from.

$\underline{mn}$

A last interesting effect is the interaction between the number of customers and the number of vehicles. The coefficient is positive, signifying that the effect of one of those is increased when the other one is increased. The logical hypothesis here is that increasing the value of one or the other induces more possible combinations overall, thus, more tests to do by the algorithm, meaning more time.

Other factor are also significant but not all of them do have an intuitive explanation. It is important to note that the explanations we have provided are by no means experimentally proven. They are rather hypotheses that we think could be the possible explanations.

5.9.3 Modelling

Now that we have a good explanatory power of all the effects that are significant to describe the time to reach the plateau, we wanted to improve the modelling to provide better predictive performance, might it be at the expense of explanatory power.

As previously, we used adjusted R^2 as a first criterion to grasp our model's performance better. As many effects are introduced, it is important to have a criterion that balances the performance with the number of parameters in the model. We also looked at *MAE* and *RMSE* and *MAE%*. Some models need specific parameter values that can be optimized. Whenever we needed to specify the values of some of the parameters, 10-fold cross-validation was carried out. This ensures that we select the best-performing model with the right parameters, and it has another advantage of ensuring that the model does not overfit. We separated our data in a training and a test set with a 70/30 ratio.

A first batch of models we created is a regression model with all main, interaction and quadratic of the eight controlled factors. Their performances are reported in Table 5.18. The first column shows the training adjusted R^2 while the three others show the performances on test set. We have emphasized in bold the model that achieves the best performance, in this case, random forest regression. However, we can see that the model overfits a bit as there is a significant difference between training and test R^2_{adj}. Also, *MAE* and *RMSE* say that, on average we make an error of 43/74.5 seconds.

Table 5.18: Modelling with response surface model of 8 factors

Model	Training R^2 adj	R^2 adj	MAE	RMSE	MAE%
Linear Regression	0.7504	0.6503	58.859	82.613	45.55%
KNN Regression	0.9999	0.6557	51.741	81.969	42.58%
Polynomial SVM Regression	0.8872	0.5954	49.574	88.854	39.74%
RBF SVM Regression	0.7504	0.6503	58.859	82.613	45.56%
Random Forest Regression	**0.9567**	**0.7051**	**43.264**	**75.857**	**32.48%**

The second group of models developed were models with parameters found significant by the linear regression analysis. So we only have 18 factors instead of 44, hoping to capture the majority of the variation of the data while decreasing the number of parameters. The results are shown in Table 5.19. We see that almost all models increased in performance as they have more or less the same precision with fewer parameters. Also, the overfitting behaviour from previously is a bit decreased. The best model now is an SVM model using an RBF kernel function.

Table 5.19: Modelling with limited parameters of 8 factors

Model	Training R^2 adj	R^2 adj	MAE	RMSE	MAE%
Linear Regression	0.7530	0.7170	56.793	82.632	44.04%
KNN Regression	0.9999	0.7024	41.727	84.733	37.10%
Polynomial SVM Regression	0.8941	0.6063	47.921	97.463	39.35%
RBF SVM Regression	**0.8788**	**0.780**	**37.770**	**72.839**	**30.52%**
Random Forest Regression	0.9621	0.7682	42.329	74.783	32.29%

Finally, as we extracted 4 other features that we did not control, we wanted to look at what their impact could be on the time to reach the plateau. We conducted the same ANOVA analysis as previously, as we wanted to detect the significant effects. Overall, almost none of the 4 uncontrolled factors had a significant effect, the only effects barely detected were one interaction with the number of clusters and number of vehicles and one with the standard deviation of the distance matrix and the vehicle time window length standard deviation. They were in total 18 parameters specified for the third batch of models, and results are shown in Table 5.20. We see a small increase in performance compared to the limited models with only 8 factors. Thus, these two interactions seem to have a small but significant effect to describe the time of reaching the plateau.

Table 5.20: Modelling with limited parameters of 12 factors

Model	Training R^2 adj	R^2 adj	MAE	RMSE	MAE%
Linear Regression	0.7548	0.7199	57.243	82.208	44.47%
KNN Regression	0.9999	0.6891	44.829	86.605	38.99%
Polynomial SVM Regression	0.8959	0.6212	52.320	95.601	42.02%
RBF SVM Regression	0.8735	0.7763	41.793	73.472	35.01%
Random Forest Regression	**0.9624**	**0.7842**	**43.355**	**72.149**	**33.13%**

The modelling of the time to reach the plateau is not as good as one might expect. Two main hypotheses exist to explain why that could be the case. First, we have worked on only complete solutions for this model and divided that batch into training and test set. We thus have a low amount of examples to train a model on, and it could be of great interest to increase the number of samples for this model to be trained on. The second reason why these models do not perform tremendously well could be that we are missing an explanatory variable for this target variable. Finally, a last explanation could be that the definition of plateau time is not the most appropriate for further modelling purposes.

Chapter 6

Conclusions

To conclude, we were able to build predictive models for three different performance measures of the OHD algorithm:

- Solution completeness
- Percentage of tasks delivered for the incomplete solutions
- The time it takes to reach the plateau for the complete solutions

Two models performed very well with an accuracy of 0.96 and MAE% of 6.46%, respectively, with the third model having MAE% of 30.52%, which is not that high. We have also provided a list of important features in predicting those performance measures and explained why they might influence performance.

Besides the performance algorithm, we made a new simulation framework through which we have simulated 1020 CVRPTW instances used in the models. This simulation framework is new because we can specify the values of the features we want our simulated instances to have. Although imperfect, the obtained feature values are very close to the pre-specified ones.

We also incorporated the concept of Design of Experiments to generate meaningful CVRPTW instances. Those instances can capture the features' main, interaction, and square effects.

Everything mentioned above has also been summarized in Section 5.1 before.

We believe that our work could potentially be extended in many directions. For example, the same framework can be used for many other variations of VRPs. Individual parts of our work can also be used for other purposes. For example, the Design of Experiments and the specific simulation techniques we suggest can be used to generate new benchmark instances. This is especially true if one wants to generate real-life inspired instances with real customer location coordinates. On a larger scale, predicting the performances of operations research algorithms could be an interesting problem.

There are several improvements that our work could benefit from:

- The code we wrote for the simulation and feature extraction could be optimized even further. This would allow us to generate even more CVRPTW instances without spending too much time. More data could open the possibility of using more sophisticated models like Neural Networks.

- The errors made in terms of the Time Window generation for the vehicles could be avoided by introducing a method that works without the rejection step which was introduced.
- We focused more on setting up the framework and getting the first results with limited features. If there is more data, we could try out even more features than we have used and possibly develop new meaningful features. This could improve the performances of the three models even further.
- Our side on the academic (continuous) and real-life (discrete) TWs can be researched even further. For example, instead of smoothing in a deterministic way as we did, there is a potential to add randomness.
- The cost tolerance of 22% in our plateau time definition could be changed, and researched how that change impacts the regression model's performance.

Bibliography

Akbari, Z. and Unland, R. (2016). Automated determination of the input parameter of dbscan based on outlier detection. *Artificial Intelligence Applications and Innovations. AIAI 2016. IFIP Advances in Information and Communication Technology*, 475:280–291.

Arnold, F. and Sörensen, K. (2019). What makes a vrp solution good? the generation of problem-specific knowledge for heuristics. *Computers & Operations Research*, 106:280–288.

Box, G. E. P., Hunter, J. S., and Hunter, W. G. (2005). Statistics for experimenters: Design, innovation, and discovery. *Wiley Series in Probability and Statistics*.

Bräysy, O. and Gendreau, M. (2005). Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation science*, 39(1):104–118.

Chris Wernst (2017). Dbscan in python. [Online; accessed May 31, 2023].

Christiaens, J. and Vanden Berghe, G. (2016). A fresh ruin & recreate implementation for the capacitated vehicle routing problem. Technical report, KU Leuven, Ghent.

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

Cordeau, J.-F., Desaulniers, G., Desrosiers, J., Solomon, M. M., and Soumis, F. (2002). *7. VRP with Time Windows*, pages 157–193. Society for Industrial and Applied Mathematics.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812.

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

Dees, W. A. and Smith, R. J. (1981). Performance of interconnection rip-up and reroute strategies. In *18th Design Automation Conference*, pages 382–390. IEEE.

Desrochers, M., Desrosiers, J., and Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations research*, 40(2):342–354.

Fisher, M. L., Jörnsten, K. O., and Madsen, O. B. (1997). Vehicle routing with time windows: Two optimization algorithms. *Operations research*, 45(3):488–492.

Gan, X., Wang, Y., Li, S., and Niu, B. (2012). Vehicle routing problem with time windows and simultaneous delivery and pick-up service based on mcpso. *Mathematical Problems in Engineering*, 2012.

Goos, P. and Jones, B. (2011). *Optimal Design of Experiments: A Case-Study Approach*. John Wiley & Sons Inc., United States.

Hutter, F., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2014). Hutter2014-algorithm runtime prediction methods & evaluation. *Artificial Intelligence*, 206:79–111.

Ioannou, G., Kritikos, M., and Prastacos, G. (2001). A greedy look-ahead heuristic for the vehicle routing problem with time windows. *Journal of the Operational Research Society*, 52(5):523–537.

Jiang, H., Wang, Y., Tian, Y., Zhang, X., and Xiao, J. (2021). Feature construction for meta-heuristic algorithm recommendation of capacitated vehicle routing problems. *ACM Transactions on Evolutionary Learning and Optimization*, 1:1–28.

Kanda, J., Carvalho, A., Hruschka, E., and Soares, C. (2016). Selection of algorithms to solve traveling salesman problems using meta-learning1. *International Journal of Hybrid Intelligent Systems*, 8:117–128.

Kerschke, P., Kotthoff, L., Bossek, J., Hoos, H. H., and Trautmann, H. (2018). Leveraging tsp solver complementarity through machine learning. *Evolutionary Computation*, 26:597–620.

Kohl, N. and Madsen, O. B. (1997). An optimization algorithm for the vehicle routing problem with time windows based on lagrangian relaxation. *Operations research*, 45(3):395–406.

Kolen, A. W., Rinnooy Kan, A., and Trienekens, H. W. (1987). Vehicle routing with time windows. *Operations Research*, 35(2):266–273.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269.

Lucas, F., Billot, R., and Sevaux, M. (2019). A comment on "what makes a vrp solution good? the generation of problem-specific knowledge for heuristics". *Computers & Operations Research*, 110:130–134.

Mersmann, O., Bischl, B., Trautmann, H., Wagner, M., and Neumann, F. (2012). A novel feature-based approach to characterize algorithm performance for the traveling salesman problem. *Annals of mathematics and artificial intelligence*, 69:151–182.

Or, I. (1976). *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking, Ph.D. thesis*. Northwestern University.

Pihera, J. and Musliu, N. (2014). Application of machine learning to algorithm selection for tsp. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, 2014-December:47–54.

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

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

Potvin, J.-Y. and Rousseau, J.-M. (1995). An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society*, 46(12):1433–1446.

Prosser, P. and Shaw, P. (1996). Study of greedy search with multiple improvement heuristics for vehicle routing problems. Technical Report 96/201, Department of Computer Science, University of Strathclyde, Glasgow.

Rasku, J., Kärkkäinen, T., and Musliu, N. (2016). Feature extractors for describing vehicle routing problem instances. volume 50, pages 7.1–7.13. Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing.

Sander, J., Ester, M., Kriegel, H.-P., and Xu, X. (1998). Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Mining and Knowledge Discovery*, 2:169–194.

Sartori, C. S. and Buriol, L. S. (2020). A study on the pickup and delivery problem with time windows: Matheuristics and new instances. *Computers & Operations Research*, 124:105065.

Satopää, V., Albrecht, J., Irwin, D., and Raghavan, B. (2011). Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *31st International Conference on Distributed Computing Systems Workshops*, pages 166–171. IEEE.

Sawant, K. (2014). Adaptive methods for determining dbscan parameters. *IJISET-International Journal of Innovative Science, Engineering & Technology*, 1.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., and Dueck, G. (2000). Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming—CP98: 4th International Conference, CP98 Pisa, Italy, October 26–30, 1998 Proceedings 4*, pages 417–431. Springer.

Smith-Miles, K. and van Hemert, J. (2011). Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence*, 61:87–104.

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

Starczewski, A., Goetzen, P., and Er, M. J. (2020). A new method for automatic determining of the dbscan parameters. *Journal of Artificial Intelligence and Soft Computing Research*, 10:209–221.

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

Taillard, É., Badeau, P., Gendreau, M., Guertin, F., and Potvin, J.-Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation science*, 31(2):170–186.

Vidal, T., Laporte, G., and Matl, P. (2020). A concise guide to existing and emerging vehicle routing problem variants. *European Journal of Operational Research*, 286(2):401–416.

FACULTY OF SCIENCE
Geel Huis, Kasteelpark Arenberg 11 - box 2100
3001 Leuven (Heverlee)
tel. + 32 16 32 14 01
https://wet.kuleuven.be/english