



**Delft University of Technology**
**Faculty Electrical Engineering, Mathematics and Computer Science**
**Delft Institute of Applied Mathematics**

# The impact on the final delivery schedule of different procedures that handle arriving customers during route optimization in a real-time Dynamic Time Slot Management system

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfilment of the requirements
for the degree

**MASTER OF SCIENCE**
**in**
**APPLIED MATHEMATICS**

**by Quinten Cederhout**
**Delft, the Netherlands**
**July 8th, 2020**

Student number: 4382862

| | | |
|---|---|---|
| Thesis committee: | Dr. F. M. de Oliveira Filho, | TU Delft, supervisor |
| | Dr. D. C. Gijswijt, | TU Delft |
| | Dr. ir. M. Keijzer, | TU Delft |
| | drs. A. Rietveld, | ORTEC, supervisor |

# Abstract

Dynamic Time Slot Management (DTSM) is a system often used in online retail to manage the delivery of goods to customers. With DTSM customers arrive over time and place orders. They get presented with a set of time slots and the customer picks the time slot in which he wants the goods to be delivered to his home. A DTSM system creates a time slot offer for the customers based on the current delivery schedule, which is a solution to a Vehicle Routing Problem (VRP). Each accepted customer is added to this delivery schedule. This delivery schedule gets periodically optimized by an optimization algorithm. During this optimization, new customers may arrive and receive time slots based on the non-optimized schedule, which causes a discontinuity in the system. In this thesis, five different procedures have been created that deal with this problem. The impact of the different procedures is analyzed by simulating the DTSM system and comparing the final delivery schedules. Solving the problem by leaving out the optimization or insertion step of the DTSM system is outperformed by all procedures. Procedures that barely make use of the optimized schedule accept the least number of customers. The procedure that delays customers, which makes it similar to the theoretical setting, accepts the most customers, but this comes at the cost of poor customer service. The most promising results are found by the procedure that inserts new customers into the optimized schedule. Enhancing this procedure with a merge algorithm improves the performance slightly.

# Contents

# List of abbreviations

| | |
|---|---|
| CVRPTW | Capacitated Vehicle Routing Problem with Time Windows |
| CVRS | Comtec Vehicle Routing Service |
| DTSM | Dynamic Time Slot Management |
| LNS | Large Neighborhood Search |
| MILP | Mixed Integer Linear Program |
| OCS | ORTEC Cloud Services |
| STSM | Static Time Slot management |
| TSM | Time Slot Management |
| TSP | Traveling Salesman Problem |
| VRP | Vehicle Routing Problem |
| VRPTW | Vehicle Routing Problem with Time Windows |

# Preface

I like the theoretical component of mathematics, but since I started my master Applied Mathematics I have always been interested in its practical use. Doing my thesis at ORTEC was a great way to see what mathematics can do in the real world.

I would like to thank all ORTEC colleagues for collaborating with me and especially the team for letting me be part of it like a full member, even when we were not allowed into the office anymore.

I am grateful for the help of Fernando, my supervisor at the TU Delft, who answered all my questions and gave me critical feedback with which I could improve my research. I also would like to thank Dion and Marleen for being part of the committee in these weird times.

I want to thank Laura and Arjen, my supervisors at ORTEC, for the helpful brainstorm sessions and for the continuous support and confidence they gave me, which was necessary to finish this project.

Lastly, I thank my family for supporting me from the beginning to the end.

*Quinten Cederhout*
*Delft, July 2020*

# 1 Introduction

Internet shopping has become increasingly popular in the past years. Nowadays you can order products from retailers, like grocery stores, to be delivered to your front door. It becomes interesting when these deliveries involve orders of size, with groceries being the most prominent example. Larger retailers often have a delivery service to serve all their customers. This attended home delivery, called 'attended' because the customer needs to be at home when these deliveries are made, is an intricate process that comes with lots of logistical difficulties. To handle these problems, retailers can make use of Time Slot Management (TSM), a system focused on good customer service with the goal to produce a delivery schedule the retailer can use.

Dynamic Time Slot Management (DTSM) is a sub-category of TSM, characterized by being an ongoing and therefore dynamic system. In practice, some steps of this may happen simultaneously. The problems posed by this real time aspect are often neglected in the theoretical setting. In this dissertation, we will describe these problems and investigate possible procedures to deal with a particular problem caused by the real-time aspect.

This research was carried out at ORTEC, a company that has a lot of experience with all kinds of logistical problems. They have developed software that is broadly used by customers of different logistical areas. Some of these customers are retailers, for example supermarkets, who have or are starting to use attended home delivery services. To realize this, ORTEC makes use of a Dynamic Time Slot Management system.

## 1.1. Problem description

Retailers need to deliver all ordered products to the customers' homes. If handled badly, this can be very costly and decrease the profit of the retailer by a large amount. With attended home delivery one major issue is failing to make a delivery, usually, because a customer is not at home. In case of such a delivery failure, the order needs to be returned and restocked somewhere and the delivery needs to be rescheduled. For grocery retailers this becomes even more costly as groceries can spoil quickly and need to be replaced.

To mitigate the problem of the customer being away from home at the time of delivery a lot of online retailers make use of time slots on their websites. When the customer makes a request for an order the retailer provides the customer with a set of time slots, which are narrow time window usually with a length of one or two hours. Such a set of time slots is called a *time slot offer*. The customer picks one of these time slots and the retailer will then have to make sure to deliver the products to the customer within the chosen time slot.

Not every customer gets offered the same time slots. Deciding which time slots to offer to which customer and ensuring that a feasible delivery schedule can be created is done by a system called Time Slot Management. TSM exists in all kinds of ways and varieties, but more and more retailers try to actively manage their time slots during the ordering process. This is generally called Dynamic Time

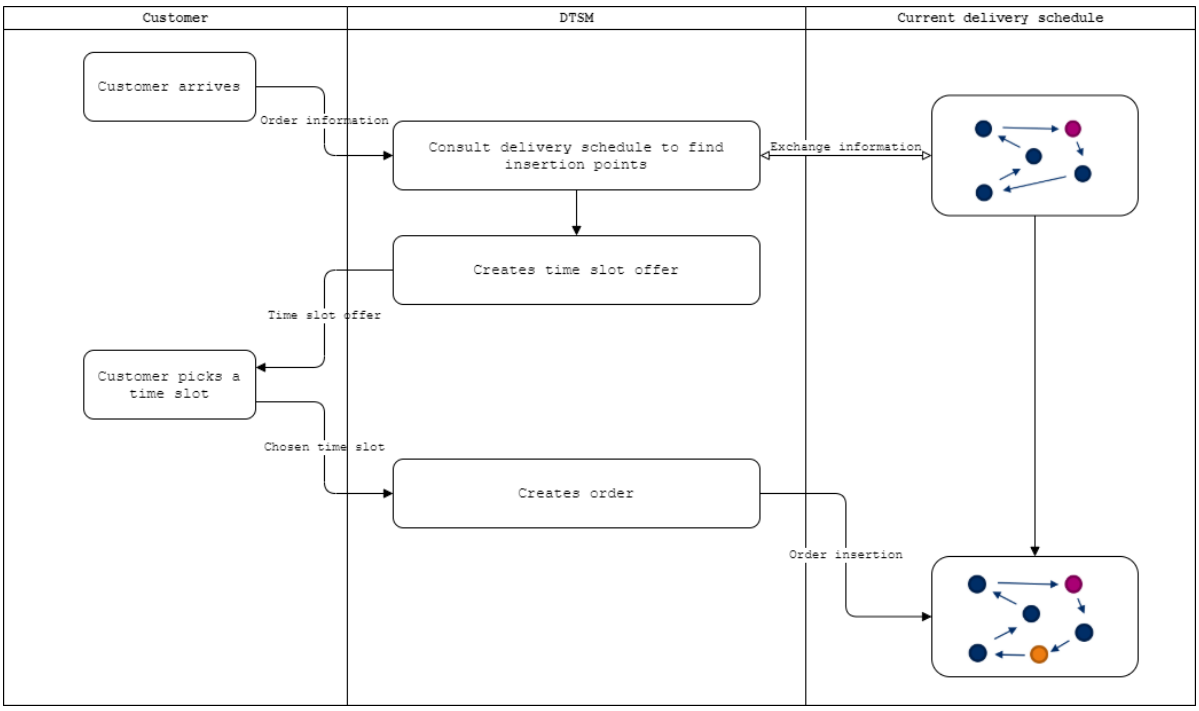



**Figure 1.1:** Handling of one customer in a DTSM system

Slot Management, which is also the focus of this thesis.

With DTSM you keep track of all orders that have been accepted so far. Based on these orders you create a delivery schedule, using the information of the available resources. When a new customer arrives and places an order, a time slot offer is created based upon the temporary delivery schedule. Only time slots which would create a feasible schedule, if they were to be picked, are offered to the new customer. After the customer has picked a time slot, the actual order is created and inserted in the current delivery schedule. This updated delivery schedule can then be used to create a time slot offer for the next customer. Figure 1.1 shows an overview of this part of the system.

Retailers can have multiple goals in mind with their Time Slot Management. Usually the main objective is to serve as many customers as possible, which is most of the time limited by the amount of resources the retailer has available. Secondly, they want to reduce planning costs, which can include a variety of things. Most commonly they want to minimize the needed amount of vehicles, the total driving distance and the total driving time.

Because of these objectives, there is often an additional component to a DTSM system. Periodically a route optimization algorithm is used to re-optimize the current delivery schedule, this is called an optimization run. This optimization algorithm tries to lower the planning costs by creating a new delivery schedule to serve all customers. This algorithm can be very intricate and can create a completely new delivery schedule or it can be more simple and only alter the current delivery schedule. Another benefit of a re-optimization is that it can free up space in the routes to allow for more customers to be served. This re-optimization will be the main focus of this thesis.

Whenever an optimization run is done in a DTSM system, a copy of the current delivery schedule is created. With this copy an optimization algorithm creates a new delivery schedule, after which this new delivery schedule replaces the delivery schedule that is used by the DTSM system. In theory, this works well. However, when we consider the real-time aspect of the DTSM system we run into a problem. Running the optimization algorithm takes time. In this time period customers can still arrive and request new time slot offers. These time slot offers will be created based upon the schedule currently present in the DTSM system, which is the schedule from before the optimization run. The customers can select a time slot like normal and their orders will be included in the current delivery schedule. When the optimization algorithm finishes it has created a new routing schedule. However,

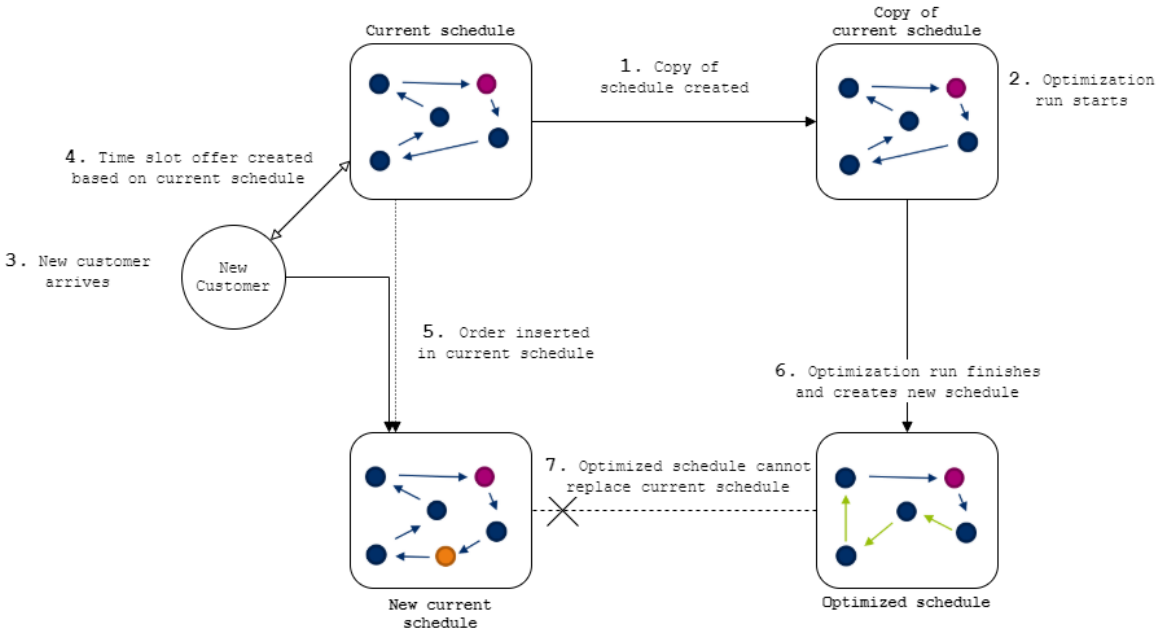



**Figure 1.2:** Example of a customer arriving during the optimization run in a DTSM system.

this new routing schedule does not include the new customers that arrived during the optimization run and their corresponding orders might not fit in the new routing schedule as their time slot offers were based on an older delivery schedule. There is no clear indication of what the best solution is to this problem. Figure 1.2 shows an overview of this problem.

## 1.2. Thesis objective

A lot of research has been done in regard to Dynamic Time Slot Management and Time Slot Management in general. Most literature supports the idea that DTSM is the most promising system and many aspects of this system have been looked at. However, after extensive literature research, we found that the problem with the real-time aspect of the optimization algorithm, which is a common problem in practice, has not been considered much and as far as we know is certainly not analyzed in detail. As the products of ORTEC provide a Dynamic Time Slot Management system it was a natural choice to look at this problem.

Therefore, the goal of this thesis is to give more insight into this problem and provide possible procedures to deal with it. Our main research question is:

*What procedures can be used to handle customers that arrive during the optimization phase of DTSM and what is the impact of these procedures on the final delivery schedule?*

To answer this question, we lay out a mathematical description for the DTSM system, which will include the real-time aspect. Furthermore, we will propose multiple different procedures that can be used to handle customers that arrive while the optimization algorithm is running. These procedures will range from simple solutions to more advanced novel methods. We will analyze the different advantages and disadvantages of each procedure for practical use.

To measure the impact of each procedure, we will run computational experiments using data from a large retailer, which is a customer of ORTEC. Using this data, we can simulate the full DTSM system using one of the described procedures to solve the real-time problem. We will analyze the different delivery schedules that will be produced, focusing on the objectives which are important to retailers. Because we only use the data from one retailer, we make sure to look at different cases with variety to make the results also interesting for other retailers.

## 1.3. Thesis outline

In Chapter 2, the relevant literature and mathematical background of the problem will be discussed. Here we also provide a mathematical description for the DTSM system, including the real-time aspect. After this, we discuss possible procedures that deal with the problem of customers arriving during the optimization phase of DTSM in Chapter 3. In Chapter 4, we look at the software created by ORTEC and how it can be used to set up a DTSM system. In Chapter 5, we describe the data we use for our computational experiments and we explain the setup of the simulations. We then show the results of the computational study. In the last chapter, Chapter 6, we summarize our results a give a conclusion. We also explain the limitations of this research. We end this thesis by giving recommendations for future research.

# 2

# Literature and mathematical background

Attended home delivery has many difficult challenges (Mkansi et al., 2018). All the logistical problems that come with it can be solved in many different ways. Our focus will be on Dynamic Time Slot Management, a practice that is becoming popular with the rise of retail through the internet. In this chapter, we will explore the most relevant literature on this topic. We first discuss the Time Slot Management system in Section 2.1, where we have a look at the different aspects and variations that exists for TSM. After this, we focus on DTSM in Section 2.2, for which we will provide a mathematical model. We describe the main problem discussed in this thesis in Section 2.3, where we look at the real-time aspect of DTSM. After this, we finish our mathematical model by regarding the three different aspects of DTSM separately, of which the first two will be the offering strategy and the insertion method found in sections 2.4 and 2.5 respectively. Before we consider the optimization algorithm of DTSM in section 2.8, we explore the well-known vehicle routing problem and solving methods for it in sections 2.6 and 2.7. We end this chapter in section 2.9 by describing our contribution to current literature.

## 2.1. Time Slot Management

Different customers request time slots and place orders until a given point in time. After this time, called the *cut-off time*, no new customers arrive. The demand amount and the location of these customers are known. When the customers make a request, they are provided with a set of short time windows, called *time slots*, from which they pick one or none. Such a set of time slots is called a *time slot offer*. This results in an order which needs to be delivered to the customer in the picked time slot. Important is that all these time slots are for a time after the cut-off time. After the cut-off time a full delivery schedule is created which includes all accepted orders. Although in literature it is known under many different names, we call this system Time Slot Management.

The main purpose of a TSM system is to provide good customer service. Evidently, we can make a delivery schedule that is a lot more efficient if orders were not restricted by their time slots, as using time slots will certainly increase the routing costs (Ramaekers et al., 2018). However, we still want to use time slots to provide better customer service and avoid all problems that come with delivering to customers that are away from the delivery address. From a retailer perspective the objective of a TSM system is clear: serving as many customers as possible. Serving more customers provides better customer service and more revenue for the retailer. The secondary objective is to make the cost of delivering all the orders as low as possible. We call this the *PlanCosts* of a delivery schedule.

### 2.1.1. Creating a time slot offer

Before the whole process of Time Slot Management starts, the set of possible time slots needs to be determined by the retailer. Note that this is not equal to the time slot offer that is shown to the customer, because a time slot offer is a subset of the set of all possible slots (but possible equal full set). The retailer has to choose what the initial set of possible time slots looks like. The length and the starting time of the time slots need to be determined, but also whether some time slots overlap or not. These choices can have a big impact on the final delivery schedule. For example, using one-hour time

slots instead of two-hour time slots would result in way higher costs on the routes, which makes sense as smaller time slots are much more restricting when creating feasible routes. Lin and Mahmassani (2002) describe this very well. This part of the system shows the trade-off between customer service and routing costs which is present in many facets of TSM. In this case customers prefer smaller time slots, but the smaller the time slots are, the higher the routing costs will be. Köhler et al. (2019) analyze the influence of customer behavior on the routing cost when differently sized time slots are used.

The most important thing a TSM system handles is deciding which time slots to offer to a customer. In all cases we need to make sure that all time slots given to a customer still ensure that we can create a feasible delivery schedule whenever the customer picks one of the offered time slots. We will also refer to such time slots as *feasible time slots* for a certain customer. With creating a time slot offer we also have to deal with the trade-off between customer service and the efficiency of the delivery schedule. For example, for a certain customer we could determine which time slot is most beneficial for us if the customer would pick that time slot and then only offer that single time slot to the customer. However, giving the customer only one option to pick from is considered bad customer service. There are many different ways in which a time slot offer can be created, this is also called the offering strategy.

We have two main categories in which TSM can be divided; Static Time Slot Management (STSM) and Dynamic Time Slot Management (Agatz et al., 2013). This division is largely characterized by the offering strategies used. Both systems try to handle all the different tasks that come with letting customers choose their own moment of delivery. With STSM most actions are predetermined and forecasting is an important aspect. DTSM systems on the other hand maintain an active delivery schedule that gets updated with each arriving customer. This delivery schedule is then used to create time slot offers for new customers. We will first give a general overview of STSM. After this we will dive deeper into the DTSM system, which is more relevant to this thesis.

### 2.1.2. Static Time Slot Management

One of the categories of TSM is Static Time Slot Management. With STSM most calculations are done before any customers arrive. The idea is that before any customers arrive it is already known what time slots will be offered to a certain customer, usually based on their location. For good results, this usually requires some amount of forecasting. After the cut-off time, only one route optimization algorithm is used to create the final delivery schedule. Different versions of Static Time Slot Management are described in Agatz et al. (2011) and Spliet and Gabor (2015). We will give two examples of common STSM strategies, both these methods try to take the knowledge from all previous delivery schedules to their advantage. Based on these old delivery schedules they try to predict the demand for a new day. Both methods use this information, but in a different way.

#### Tactical Time Slot Management

The first version, sometimes referred to as Tactical Time Slot Management, divides the area in which the customers are located into multiple sections. Each of these sections then gets their own set of available time slots, which can include multiples of the same time slot. These sets are based on the predicted demand and on previously used sets of available time slots. Whenever a customer requests a time slot offer, all the available time slots of the section the customer is located in gets returned. After the customer has picked a time slot, this time slot gets removed from the set of available time slots in that section.

#### Existing routes

The second strategy is broadly used by a popular retailer in the Netherlands. With this strategy the delivery schedule for every day is created in advance, based on the predicted orders. When a customer requests time slots, they generally get offered only one very strict time slot for every day. This is done by looking at which pre-made routes are the closest to the customer and then determining at what time the vehicle would arrive at the customer if they were inserted in that route. Only time slots that correspond with these arrival times are offered to the customer. Usually there are not many routes that come close to a certain customer, which means that a very limited number of time slots are offered. The main difference between this approach and that of a Dynamic Time Slot Management is that the routes are pre-made and are not updated or changed during the process.

**Advantages and disadvantages**
Both systems are based on returning customers, making it that previous schedules give a very good idea of what the new schedules should look like. An advantage of both methods is that time slot offer can be created very quickly, as they are predetermined. The advantage of the method with a predetermined route is that the resulting delivery schedule often is quite low on costs compared to other methods. The disadvantage of STSM systems compared to DTSM is that they often accept fewer customers and provide fewer time slots to customers, sometimes not more than one. Therefore, DTSM systems provide better customer service. STSM systems are also heavily dependent on how good their forecasting is. Whenever there is a day for which the demand is difficult to predict the STSM system might perform badly. A DTSM system is a lot more flexible and can, due to its dynamic nature, adept to unexpected customer behavior.

## 2.2. Dynamic Time Slot Management

A different approach than the static method is Dynamic Time Slot Management. As described in the introduction, with DTSM there is always a current delivery schedule kept in memory. New time slot offers are created based on this delivery schedule and every new accepted customer is inserted in the current delivery schedule. In most cases there is an optimization algorithm that is used to periodically optimize the current delivery schedule. After the cut-off time this optimization algorithm is used once more to find the final delivery schedule. In this section we will further discuss the different aspects of DTSM. Before we do that we will lay out a mathematical description of a full DTSM system.

### 2.2.1. Mathematical description

In this section, we will give a mathematical description of the TSM system. This includes a formulation of the underlying Vehicle Routing Problem. We use the term delivery schedule which in this context is the solution to a Vehicle Routing Problem. It is important to note that some specific parts of the description, like the content of location data, knowledge of quantities or certain VRP extensions, would be modeled slightly differently based upon different scenarios. The description we give is what is assumed to be correct for the problems we deal with in the rest of this thesis. For ease of explanation we will refer to our mathematical description of the DTSM system as our mathematical model.

**Problem input**
We have a collection $C$ of customers which arrive within a time period of $[0, T]$, where $T \in \mathbb{N}$ is the cut-off time in seconds after which no new customers are accepted and the final delivery schedule is made. At any point before the cut-off time, it is not known when and how many customers are yet to arrive. Every customer $i \in C$ has its own set of properties:

| | |
|---|---|
| $L_i$ | the location data, which are the coordinates of the delivery address of the customer; |
| $t_i \in \mathbb{N}$ | the arrival time in seconds at which the customer makes a request. Note that $t_i \leq T$; |
| $q_i > 0$ | the quantity of the desired delivery; |
| $u_i \in \mathbb{N}$ | the service duration in seconds, which is the minimum time a vehicle has to spend at this location to perform the delivery. |

We have a set of available time slots $\mathcal{T} = \{S_1, \ldots, S_n\}$. Each time slot $S_j = [a_j, b_j]$ in this set is an interval with a starting time $a_j$ and end time $b_j$, with $T < a_j < b_j$. If customer $i$ makes a delivery request, a time slot offer $\mathcal{T}_i \subseteq \mathcal{T}$ is created. The customer then chooses one or none of these slots from $\mathcal{T}_i$. To do this, we assume that the customer has an ordered set of preferred time slots $\mathcal{T}_i^{pref} \subseteq \mathcal{T}$. The customer then picks the first time slot from $\mathcal{T}_i^{pref}$ that is also in $\mathcal{T}_i$. If $\mathcal{T}_i^{pref} \cap \mathcal{T}_i = \emptyset$ the customer leaves and does not place a delivery order. Customers that pick a slot are added to the collection $C^{acc}$ of all accepted customers.

The objective is to maximize the number of accepted customers $|C^{acc}|$. However, we need to be able to create a feasible delivery schedule, which is the solution to a Vehicle Routing Problem. This places a constraint on the offering of time slots, as only time slots that allow for a feasible delivery schedule are offered to a particular customer. As fewer time slots are offered to a customer, changes become

higher that there the customer does not pick a time slot and leaves, which of course does not increase the number of accepted customers.

**Vehicle Routing Problem**
After customers have picked their time slots, the problem transforms into a Capacitated Vehicle Routing Problem with Time Windows (CVRPTW), where the time slots form the time windows of the different orders. Depending on the specific problem, this Vehicle Routing Problem (VRP) can have multiple additional extensions, see Sections 2.6.3 and 5.1.3. We will now give our model formulation.

We have the complete directed graph $G = (V, E)$, where $V = C^{acc} \cup \{D\}$ and $D$ is the depot. In the graph we have the set of edges $E$ which connect all customers and the depot. For each edge $(i, j)$ with $i \neq j$ we have the non-negative travel time $t_{i,j}$ and the travel distance $d_{i,j}$. We assume that this travel time describes the fastest route from one location to another, which means that the triangle-inequality holds. We call $K$ the collection of vehicles available at the depot. Each vehicle has to start and end at the depot and cannot return to the depot in between. The depot has an earliest departure time and latest arrival time $[\varepsilon^D, \lambda^D]$ which determine the time period in which a vehicle can leave and return to the depot. Each vehicle $k \in K$ has its own earliest departure time and latest arrival time $[\varepsilon^k, \lambda^k]$. Each vehicle also has a capacity $Q_k > 0$.

We define a route $\rho_k$ of a vehicle $k$ as a simple cycle in $G$ that starts at the depot. We denote $V(\rho_k)$ as the vertex sequence of this cycle. A delivery is successful if there is a vehicle which has a route that visits the customer location. There are multiple constraints for a vehicle:

- a vehicle has at most one route;
- for each vehicle $k \in K$ we have $\sum_{i \in C^{acc}} \mathbb{1}_{V(\rho_k)}(i) \cdot q_i \leq Q_k$, i.e. a vehicles' capacity cannot be exceeded by the orders it delivers;
- for each vehicle $k \in K$ we have that for each customer $i \in V(\rho_k)$, vehicle $k$ must arrive before the end time $b_i$ and if the vehicle arrives before start time $a_i$ it has to wait until $a_i$. A vehicle must stay at least for a duration of $u_i$ before departing from the customer;
- each vehicle must leave and return within $[\varepsilon^k, \lambda^k]$ and $[\varepsilon^D, \lambda^D]$, the time windows of depot and vehicle.

To summarize, a feasible delivery schedule is a set of at most $|K|$ routes which visit all customers while satisfying the time slot, time window, and capacity constraints.

Our primary objective is to maximize the number of accepted customers while creating a feasible delivery schedule. The secondary objective is to minimize the PlanCosts, which in our case will be determined by a cost function of the number of vehicles that serve at least one customer, the total travel time of all the routes and the total distance of all the routes. Section 2.6.1 explains how this can be done.

**DTSM specifics**
One of the first descriptions of Dynamic Time Slot Management is by Campbell and Savelsbergh (2005), which shows us that there are many different solution methods to handle DTSM. Our model so far is not yet complete, as we miss the aspects specific to a DTSM system. We can break this down into three parts:
1. Decide which time slots to offer when a request is made.
With DTSM, this is based on the *current delivery schedule*. At any point in the system, we create a feasible delivery schedule for all accepted customers up to that point present in $C^{acc}$. Based on this schedule we decide which time slots to offer to new customers. How we do this can be found in Section 2.4.

2. Decide how to insert an accepted customer into the current delivery schedule.
As soon as a customer arrives and picks a time slot, this customer gets added to $C^{acc}$, which means that a new current delivery schedule needs to be created. We do this by inserting the customer into the current schedule. We will discuss this more in Section 2.5.

3. Decide what the optimization algorithm looks like which can optimize a feasible delivery schedule.
Once after the cut-off time and periodically throughout the process the current delivery schedule gets

optimized. More details about the used optimization algorithm to do this can be found in Section 2.8.

Each of these parts poses a problem for which many solutions exist. Combining a solution for each part makes a full DTSM system, which can therefore be quite diverse. However, before we dive deeper into the workings of a DTSM system we will first describe the problem we are trying to solve. This problem comes from the real-time aspect, which is so far neglected in the model. The next section will explain this further.

## 2.3. Real-time DTSM

In the theoretical setting every aspect of a Dynamic Time Slot Management system can happen neatly one after the other. A customer arrives, picks a time slot, its order gets inserted into the current schedule, the schedule might be re-optimized and then the next customer arrives and can be processed in the same way. Reality, however, can be quite different. Recent research from Visser et al. (2019b) has shown that the real-time aspect of Dynamic Time Slot Management is an important factor to consider.

They have researched the consequences of multiple customers making a time slot request at the same time. This happens because every action described above requires some amount of time. For example, while one customer is still choosing which time slot they prefer, a new customer might arrive and demand a time slot offer. In the end, this can lead to an infeasible delivery schedule. Further information on how this can be modeled and best dealt with can be found in Visser et al. (2019a). In our case, we will neglect this aspect of the real-time aspect like most literature. We assume that the creation of time slots, the customer's choice and insertion into the current schedule all happen instantly. We do this because we want to explore a different aspect of the real-time scenario, which is the time it takes to run an optimization algorithm.

### 2.3.1. (Re-)optimization in a real-time scenario

As mentioned, most dynamic time slotting solutions use an algorithm that (re)-optimizes the current route plans at certain times in the process, for example once every fifteen minutes. We include this in our model by giving declaring the time $T_{opt}$ the time it takes for the optimization algorithm to create a new delivery schedule. Say the algorithm starts at time $t_s$. We then find if there are customers with an arrival time that falls within $[t_s, t_s+T_{opt}]$. These customers will be processed like normal, so they request a time slot offer and will receive this based on the 'old' not yet optimized delivery schedule. When the optimization algorithms finishes, we now have new orders with picked time slots based on old routes and a new route plan which does not include these orders yet. The new orders and the new schedule might not fit well together, which means that placing the new orders in the new schedule might result in an infeasible schedule. This incompatibility forms a new problem that needs to be solved.

This problem is the main topic of this thesis. In Chapter 3 we will further analyze this problem and we will also pose various procedures that could be used to avoid or solve the problem that the real-time aspect introduces. Before we do this, we discuss all the necessary background information in the rest of this chapter.

## 2.4. DTSM: Offering strategy

The defining aspect of Dynamic Time Slot Management is that a time slot offer is created based upon the current delivery schedule. Much research has been dedicated to how this is best approached.

In any case, it has to be determined what time slots from the potential time slots are available. For every potential time slot, it is checked if an order with this time slot could be inserted in the schedule while keeping a feasible schedule. This order is based on the customer's information.

### 2.4.1. Model: offering strategy

Before the DTSM system starts, a first feasible delivery schedule is made by creating a route for every available vehicle in $K$. Each of these routes only contains the depot $D$, as the starting vertex and as the ending vertex. This schedule will serve as the first current delivery schedule. All new arriving orders

will later be added to these routes. For each route, we keep track of specific values. For every edge $e = (C, D)$ in the route we store the *earliest departure time* $\varepsilon(e)$ and the *latest arrival time* $\lambda(e)$. The earliest departure time is the earliest time we can leave after serving the customer of vertex $C$. The latest arrival time is the latest time we can arrive at the customer of vertex $D$ and still be able to serve them.

Given a route $\rho = [(D, o_1), (o_1, o_2), \dots, (o_{n-1}, o_n), (o_n, D)]$ and the other input of our problem we can use a *Forward pass* algorithm (Algorithm 1) to determine the earliest departure time of each edge in the route. Similarly, we can use a *Backward pass* algorithm (Algorithm 2) to determine the latest arrival time of each edge in the route.

Recall:

| | |
|---|---|
| $\epsilon^D, \epsilon^k$ | earliest departure time for depot $D$ and vehicle $k$ respectively; |
| $\lambda^D, \lambda^k$ | latest arrival time for depot $D$ and vehicle $k$ respectively; |
| $a_o$ | start of time slot of order $o$; |
| $b_o$ | end of time slot of order $o$; |
| $t_{o_1,o_2}$ | travel time from order $o_1$ to order $o_2$; |
| $u_o$ | service duration at order $o$; |

**Algorithm 1:** Forward pass

**Input:** Depot $D$;
Vehicle $k$;
Route $\rho = [e_0, e_1, \dots, e_{n-1}, e_n] = [(D, o_1), (o_1, o_2), \dots, (o_{n-1}, o_n), (o_n, D)]$;

**Result:** Calculation of earliest departure times.

$\varepsilon(e_0) = \max(\varepsilon^D, \varepsilon^k)$;
**for** *i = 1 → n* **do**
  $\varepsilon(e_i) = \max(a_{o_i}, \varepsilon(e_{i-1}) + t_{o_{i-1},o_i}) + u_{o_i}$ ;
**end**

**Algorithm 2:** Backward pass

**Input:** Depot $D$;
Vehicle $k$;
Route $\rho_k = [e_0, e_1, \dots, e_{n-1}, e_n] = [(D, o_1), (o_1, o_2), \dots, (o_{n-1}, o_n), (o_n, D)]$;

**Result:** Calculation of latest arrival times.

$\lambda(e_n) = \min(\lambda^D, \lambda^k)$;
**for** *i = (n-1) → 0* **do**
  $\lambda(e_i) = \min(b_{o_i}, \lambda(e_{i+1}) - t_{o_{i-1},o_i}) - u_{o_i}$ ;
**end**

The earliest departure times and latest arrival times can be used to assess if a certain time slot can be offered to a new customer. We do this by checking if the customer could be inserted in any part of the existing routes from the current schedule. If so, we determine if this insertion point overlaps with any of the available time slots. Each time slot that overlaps with such an insertion point is added to the time slot offer for the customer. The pseudo-code for this can be found in Algorithm 3.

**Efficient Move Evaluations**

The proposed method of creating a time slot offer is heavily influenced by the research of Visser and Spliet (2017). They describe a method using so-called ready time functions that can evaluate the

**Algorithm 3:** Creating a time slot offer

**Input:** New customer $c$;
Set of accepted orders $C^{acc}$;
Set of available time slots $\mathcal{T} = \{S_1, \dots, S_n\}$;
Depot $D$;
Vehicles $K = \{k_1, \dots, k_m\}$;
Routes $R = \{\rho_{k_1}, \dots, \rho_{k_m}\}$;

**Output:** Time slot offer $\mathcal{T}_c$.

$\mathcal{T}_c = \{\}$;
**foreach** *Vehicle* $k \in K$ **do**
  **if** $\sum_{i \in C^{acc}} \mathbb{1}_{V(\rho_k)}(i) \cdot q_i + q_c \leq Q_k$ **then**
    **foreach** *edge* $e = (e_1, e_2) \in \rho_k$ **do**
      StartWindow = $\varepsilon(e) + t_{e_1,c} + u_c$;
      EndWindow = $\lambda(e) - t_{e_1,c}$;
      **if** StartWindow < EndWindow **then**
        **foreach** time slot $S \in \mathcal{T}$ **do**
          **if** $S$ overlaps with [StartWindow,Endwindow] **then**
            Add $S$ to $\mathcal{T}_c$;
          **end**
        **end**
      **end**
    **end**
  **end**
**end**

effect of small changes to a delivery schedule efficiently. In short, a ready time function is quite similar to storing the earliest departure time for a vertex of a route, except that it can also deal with time-dependent travel times. They showed that these ready time functions can be stored and updated quickly using a smart data structure. The structure therefore for quick evaluations when a simple change is made, which can be used for finding good insertion points efficiently. Doing this efficiently is important as any time spent on creating a time slot offer can result in an infeasible delivery schedule due to the real-time aspect.

## 2.4.2. Creating a smart time slot offer

Naturally, all time slots that result in an infeasible schedule if they were to be picked by the customer are not included in the time slot offer. However, this does not mean that all other possible time slots should be an option for the customer. For example, say that a new customer arrives and request time slots. It might be that his neighbor has already placed an order and is included in the schedule. The costs on the delivery schedule will be a lot lower if the new customer picks the same time slot as his neighbor did, as there already is a vehicle close to the delivery address of the new customer. A good example of the impact of smartly offering time slots is found in the work of Klein et al. (2018).

### Time slot costs

Sometimes we are also interested in the increase in costs of the delivery schedule if a specific time slot is chosen. We name this the costs of a time slot. These costs could entail a great many things, as it is fully dependent on the objectives of the underlying VRP it is based on.

There are multiple ways in which the costs of time slots can be involved in creating a time slot offer. First of all, based on the costs certain time slots can be removed for the offer, which means they will not be available for the customer. In such a case it is customary to remove a certain percentage of time slots and of course to remove the time slots with the highest costs.

Another common usage of time slot costs is found in pricing time slots (Yang et al., 2016; Klein et al., 2019). Retailers give the customer a price they have to pay in order to pick a certain time slot. This

price can be different for each offered time slot. Naturally, these prices can be based on the cost of offering a time slot to a certain customer. Pricing time slots is also often used to spread the demand evenly throughout the day, as popular time slots will get higher prices than unpopular ones. This idea can also be found in Campbell and Savelsbergh (2006).

**Forecasting**
When the first customers arrive all routes are still empty in the schedule that is kept in memory with the DTSM system. This means that these first customers often can pick from all available time slots, which can be bad for the costs of the final delivery schedule. It is possible to use a combination of Tactical Time Slot Management as described in Section 2.1.2 together with DTSM. In this case, the forecasted delivery schedule is used to create the time slot offer for the first customers, after which the delivery schedule from the DTSM system is used again. Yang et al. (2016) describe such a system.

### 2.4.3. Customer service

The idea of offering fewer time slots to a customer to be able to create a more efficient delivery schedule sounds nice. However, one important thing to consider is the customer service. In general, retailers want to deliver good customer service, which is one of the reasons to use a Time Slot Management system in the first place. This is reflected in the choice retailers make regarding the offering of time slots. Retailers can choose to leave out certain time slots in order to gain more efficient routes and lower the costs of the delivery schedule. However, this would reduce the customer service as the customer now has fewer options. Therefore, most retailers choose to not leave out any time slots and possibly use pricing strategies instead.

Another important factor is the computation time of creating a time slot offer. When a customer request time slots on the retailer's website they do not want to wait a long time before the available time slots show up. Depending on the retailer, they might want this response time to even be below half a second. In this time, the time slot offer needs to be created, which means that there needs to be a very efficient way of checking which time slots are available. In practice, it is often not feasible to do more extensive calculations to decide if a certain time slot could better be removed or not based on the costs of the schedule. This means that many sophisticated methods from the various literature for creating a time slot offer are not practically usable.

## 2.5. DTSM: Insertion method

After the customer has picked a time slot, the order can be created. This order now needs to be inserted into the current delivery schedule. A very easy way to do this is to remember what the insertion point was when creating a time slot offer, as at that stage we did already check for feasibility.

Another option is to use an insertion method, which is similar to the insertion methods used in a construction algorithm for the Vehicle Routing Problem, which we will also cover in Section 2.8.1. The insertion can be easy which keeps all current routes intact and inserts the new order in the sequence of a route. Insertion methods can also be quite complicated, for example by starting again with empty routes and building up the delivery schedule from scratch but now with the new order included. In such a case the insertion method is more similar to a VRP solving algorithm as described in Section 2.7. Using such a more sophisticated method can result in a better delivery schedule as the result of the algorithm.

However, the computation time of this step is an important consideration which is again often neglected in TSM literature. In our case, a new customer cannot be served before the order of the previous customer is inserted into the current schedule. Therefore, if the insertion method takes too long, new customers would have to wait or we would have to deal with infeasibility problems. As we chose to neglect the time it takes for an insertion to happen in our model, it does not make sense to use a very intricate insertion method which takes a lot of calculation time.

### 2.5.1. Model: Insertion method

We want to use an efficient insertion method to insert new orders into the current delivery schedule. To do this we will use a method similar to a Parallel Cheapest Insertion algorithm, which is a well-known algorithm to solve a VRP problem. An example of a Parallel Cheapest Insertion algorithm can be found

in Appendix A (Algorithm 12). However, by using the knowledge we get from the creation of a time slot offer using Algorithm 3 we do not have to execute a Cheapest Insertion algorithm on its own and we can efficiently insert customers into the schedule after they have picked a time slot.

Using Algorithm 3 we create a time slot offer. We note that using this algorithm we check every insertion point of every route. Whenever such an insertion point is checked for feasibility, the increase in PlanCosts of that insertion can also be easily determined. Assume we have an edge $e = (e_1, e_2)$ of a route in the current schedule in which we are considering to insert a new customer $c$. Given that the insertion would be feasible, the difference in traveled distance can be measured by $d_{e_1,c} + d_{c,e_2} - d_{e_1,e_2}$. In a similar way, the difference in travel time can be measured by $t_{e_1,c} + t_{c,e_2} - t_{e_1,e_2}$. By looking if the route already contained an order before the insertion, we can determine if the insertion would be the first order to be inserted in the route, which means that the vehicle has to leave the depot. With these three variables, we can calculate the increase in PlanCosts this specific insertion would give.

Every time slot that has an overlap with any feasible insertion point is included in the time slot offer. It is possible that multiple insertion points overlap with the time slot because we consider every edge of every route. Every insertion point has its own increase in PlanCosts associated with it. The cost of the time slot will be the minimum of these increases.

We store the minimum cost of each of time slots. For each time slot we also store the insertion point that corresponds to the minimum cost of that time slot. If the specific time slot then gets picked by the customer, we insert the order in the stored insertion point without any further calculation. This avoids the need of using a separate Cheapest Insertion algorithm.

## 2.6. Vehicle Routing Problem

Creating or optimizing a delivery schedule are all based around methods that deal with the Vehicle Routing Problem. In this section, we will give more detail about this mathematical problem and the methods that exist to solve it before we can describe the optimization algorithm we will use in our DTSM model.

One cannot talk about the Vehicle Routing Problem without first mentioning the Traveling Salesman Problem (TSP) (Flood, 1955). TSP is one of the oldest and most well-known optimization problems we know of. This problem knows many forms. The classic description starts with a ‘salesman’, who lives in his hometown. The salesman wants to visit all nearby towns to sell his wares and eventually come back to the hometown which was his starting point. His goal is to visit the towns in the most efficient order. Mathematically speaking, it is equivalent to finding the cheapest Hamiltonian cycle in a complete weighted graph.

The Traveling Salesman Problem belongs to the group of NP-hard problems, as proved by Papadimitriou (1977). This means that, unless P=NP, there exists no algorithm that runs in polynomial time which can solve this problem.

The Vehicle Routing Problem is a generalization of the Traveling Salesman Problem and it was first described by Dantzig and Ramser (1959). It essentially is the same problem except that there are multiple salesmen, in this case called vehicles. These vehicles all have the same starting point. We also have several other locations, which are the customers to which the vehicles need to deliver a certain demand. The goal is the find a set of routes for the vehicles such that all customers are visited once and the total cost of the routes is minimized. The vehicles have to end their route at the same location they started from. Furthermore, all vehicles usually have a fixed capacity of goods they can carry, which limits some possibilities. We have described the VRP as vehicles delivering goods to customers, as that is the setting of this thesis. However, the VRP can be found in many different settings. This difference is quickly seen when describing the route costs, which can be done in many different ways. For example, it can be the cost of the vehicle that is used, but also the cost of driving distance, travel time or fuel usage. Often we find that the costs are defined by multiple variables.

As VRP is a generalization of the TSP it is also an NP-hard problem. A consequence is that for larger instances it can be hard to find an exact solution for the problem within a reasonable time. Therefore, when solving a vehicle routing problem in practice most of the time heuristic methods are used. We will discuss this more in Section 2.7.

The general Vehicle Routing Problem can be easily formulated. However, in practice a VRP often has a lot of extra constraints or additions which significantly increase the complexity of the problem. There is a lot of literature about all the varieties of the VRP. In general, the extensions of the VRP do not make the problem easier to solve, they remain NP-hard problems, as supported by Lenstra and Kan (2006). In the following sections we will have a look at the most relevant extensions for our setting.

### 2.6.1. Multi-objective VRP and order planning

As mentioned before, the objective of a Vehicle Routing Problem can be many different things. To simulate real-world scenarios we often need to optimize over multiple different objectives, creating a Multi-objective VRP (Jozefowiez et al., 2008).

A common way a VRP objective is set up is with the use of a cost function. In the case of our model this consists of the number of vehicles used $UV$, the total distance $TD$ in kilometers and the total travel time $TT$ in minutes of all vehicles. We define a cost function, which is often a linear function. This cost function $cost$ would look something like

$$cost(UV, TD, TT) = a \cdot UV + b \cdot TD + c \cdot TT,$$

where $a, b, c > 0$. The parameters $a, b$ and $c$, often called 'weights', are used to scale the different variables which determine the importance of a certain variable relative to the other variables. Determining the right values for the weights can be very tricky and is something that gets adjusted for every problem.

If a solution does not visit all customers it is called infeasible in the classical VRP. However, in many models and heuristic algorithms this is not a hard constraint. In this case, the main objective becomes to serve as many customers as possible. In such a scenario a lexicographic ordering of the objective can be used. Such an ordering determines the importance of an objective. Consider an example where the first objective is to maximize the number of served customers and the second objective is minimizing a certain cost function. In this way, we prefer solutions with more served customers over a solution with less served customers, even when the costs of the second solution may be significantly lower. Clearly, the lexicographical ordering could be extended with even more objectives and it is not necessary to include a cost function as an objective.

### 2.6.2. Vehicle Routing Problem with Time Windows

A very common extended VRP is the Vehicle Routing Problem with Time Windows (VRPTW) (Desaulniers et al., 2014). In this problem, all customers have their own independent time window with a start time and an end time. The extra constraint is that deliveries can only be made within these time windows. The number of possible solutions can be greatly restricted by these time windows as we are adding more constraints. This can also have a big influence on the cost of the optimal solution (Lin and Mahmassani, 2002). However, in theory adding time windows to an existing VRP does not necessarily increase the cost of the optimal solution. We can always add time window restrictions to an existing VRP in such a way that all delivery moments corresponding to the optimal solution of the original problem are within the chosen time windows. In this way, the optimal solutions of the original VRP is also a feasible solution of the created VRPTW and is thus also the optimal solution with exactly the same costs. Using Dynamic Time Slot Management, customers pick the time slot in which they want the delivery to be made. These time slots then become the time windows associated with the different deliveries, therefore the resulting routing problem is some form of the VRPTW. These time windows can increase the costs of the optimal solution by a lot or not at all. This idea supports the importance of a good DTSM model, as picking the right time slots is very important. In practice this turns out to be quite difficult, as Jabali et al. (2015) show that even when the retailer themselves can decide upon the time windows the routing costs will still increase.

### 2.6.3. VRP Extensions

Time windows are a very common and important extension to the VRP, but there are many more. In this section, we will describe some of the possible extensions of the Vehicle Routing Problem. We will not discuss them all as there are way too many. The extensions we do talk about are chosen because they are relevant to the specific problem we will be looking at in this thesis. A good overview of most VRP extensions can be found in Irnich et al. (2014).

Because of the practical applications of the VRP, a lot of these extensions have been studied in various literature pieces. However, most of the time theory focuses on a specific extension that is extensively analyzed. For a real-world problem we often find that a lot of extensions are combined which makes for a very complicated problem (Braekers et al., 2016).

**Heterogeneous vehicles**
In the classical Vehicle Routing Problem all vehicles are the same. In reality, we often deal with a heterogeneous fleet of vehicles (Kritikos and Ioannou, 2013), which means that each vehicle is unique and can have different attributes compared to the other vehicles. This can apply in many different ways. For example, a vehicle might not be able to serve every customer because it does not have the necessary capabilities that other vehicles do. An easier example is the capacity of the vehicles. Smaller vehicles might have less capacity than larger vehicles, but the cost of using a large vehicle might be higher, which should be taken into account in the cost function of the problem.

**Resource availability and breaks**
Often tied in with a heterogeneous fleet of vehicles comes resource availability. A depot might have its own time window in which the vehicles have to return. The vehicles themselves can also have a time window in which they are available, which differs for every vehicle in the heterogeneous case.

Every vehicle needs to be driven by at least one driver. In real working conditions, the drivers often are required to take a break somewhere within their route. Such a break has a certain duration and a time window in which it has to take place. It can be seen as another order that needs to be included in the route, with the only difference being that a break has no fixed location.

## 2.7. VRP solving algorithms

A lot of different methods have been developed to solve the Vehicle Routing Problem. Which method works best depends on the specific scenario and type of VRP that is considered. Exact methods, which calculate the optimal solution of a problem, were the first solution methods. However, the problem is NP-hard, which means that for larger instances of the problem it becomes infeasible to use exact methods as the computation time will be too long for a practical purpose. Therefore a lot of methods make use of heuristics in order to approach the optimal solution as close as possible in a given amount of time. We will not discuss all existing algorithms for solving VRP's, because there are way too many, mainly due to the fact that there are so many variations on the Vehicle Routing Problem and a slight change in an algorithm already creates a new algorithm. In the rest of this section, we will discuss some more general algorithms and we will give more details about the algorithms which are related to those used by ORTEC.

### 2.7.1. Exact methods

Exact methods find the optimal solution of a Vehicle Routing Problem. The problem can be modeled in different ways, but usually takes the form of a Mixed Integer Linear Program (MILP). Such a model looks like the following:

$$\min_{x,y} \mathbf{cx} + \mathbf{hy} \tag{2.1}$$

$$\text{such that } \mathbf{Ax} + \mathbf{Gy} \leq \mathbf{b} \tag{2.2}$$

$$\mathbf{x} \in \mathbb{Z}_+^n \tag{2.3}$$

$$\mathbf{y} \in \mathbb{R}_+^p, \tag{2.4}$$

where $\mathbf{c} \in \mathbb{R}^n$ $\mathbf{h} \in \mathbb{R}^p$, $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{G} \in \mathbb{R}^{m \times p}$, $\mathbf{b} \in \mathbb{R}^m$ .

Depending on the solution method, the precise formulation of the model is adjusted. Good examples of this is described by van der Sar (2018), which includes examples for Branch and Bound and Branch and Price algorithms, which are very common algorithms used for solving a Mixed Integer Linear Program. An overview of other well known exact methods, such as Column Generation, can be found in Baldacci et al. (2012). Exact methods ensure that you find the optimal solution to a problem. However, when a Vehicle Routing Problem includes multiple different extensions it quickly becomes difficult to include this in the MILP. Furthermore, the algorithms are designed to find a solution quickly, but for large instances

with for example more than a thousand orders, the algorithms will take so much computation time that it is not useful in a practical setting.

### 2.7.2. Heuristics and hybrid methods

As said before, heuristic methods for solving the Vehicle Routing Problem try to find the best solution within a given amount of time or iterations. Different than with exact methods, if we use a heuristic to solve the problem we are not guaranteed that the solution found is the optimal solution. In this section, we will give a brief overview of some popular groups of heuristic methods. Most of these heuristics are further described in Breedam (1994).

The most basic heuristic methods are Construction Methods, which construct a delivery schedule using the input of the VRP. A simple and often used method is adding customers one by one to a route in the delivery schedule. This can be done randomly, resulting in a form of a Greedy Construction heuristic (Ehmke and Campbell, 2014) or by determining an order in which the customers are inserted, for example using a Cheapest Insert Search (Köhler et al., 2019). Similar to this are the so-called Savings heuristics (Paessens, 1988). Other construction methods try to cluster close orders together in the same routes before actually creating the routes, for example with the Sweep algorithm (Renaud and Boctor, 2002).

Most heuristics are only able to improve a given delivery schedule, construction methods are therefore often used to create the initial input that other heuristics need.

Another category of heuristic methods are Local Search methods (van der Sar, 2018). These methods start with a feasible solution. In each iteration of the method, all solutions in the neighborhood of the current solutions are explored. The neighboring solutions are found by slightly perturbing the current solution in some way. A better solution, compared to the current solution, is found within the neighborhood and this solution is the starting point of the next iteration. If no solution in the neighborhood is better than the current solution the algorithm terminates. With multiple constraints, it becomes hard to define a good neighborhood that leads to good solutions, as perturbation can easily lead to infeasible solutions. While local search methods are often quite fast and can have great results in practice, they do have a big disadvantage. When we use a local search method it is possible to end up in a local optimum. The current solution can be the optimal solution in its neighborhood, terminating the algorithm, but there might still be a better solution somewhere else in the solution space.

Variable Neighborhood Search methods are a variation on the local search methods. The principal is the same, except that in each iteration the way of perturbing the current solution can be changed. Often this is done when a local optimum is found in order to try to escape this local optimum and find a better solution in a different area of the solution space (Pisinger and Ropke, 2007).

More variations are Large Neighborhood Search (LNS) methods. In fact, they are identical to Local Search methods in the way they work, except that in this case the searched neighborhood is very large. Searching through such a neighborhood takes a lot of time so this is often done smartly, meaning that the way to perturb the current solution can be quite sophisticated. When we make use of a Large Neighborhood Search the chance of getting stuck in a local optimal solution is significantly lower than with normal Local Search methods (Ahuja et al., 2002). LNS methods are often used in optimization algorithms to escape a local optimum as shown in Figure 2.1.

Metaheuristics are higher-level procedures compared to previously explained methods, although in most cases the Variable Neighborhood Search and Large Neighborhood Search can also be seen as metaheuristics. Almost all metaheuristics include local search methods as part of their solution. Furthermore, each metaheuristic has a way in which it tries to avoid getting stuck in a local optimum. The well-known metaheuristic Simulated Annealing does this by accepting worse solutions under certain conditions during its iterative process. By decreasing the change of accepting worse solutions as the algorithm carries it eventually converges to a good solution. Tabu Search is another metaheuristic that can accept worse solutions in order to more thoroughly search the solution space. This method makes use of Tabu Lists which keep track of previously explored solutions that should not be visited anymore, increasing the broadness of the exploration of the solution space. Totally different kinds of metaheuristics are Genetic algorithms, which try to evolve, mutate and combine solutions to find increasingly better solutions. There are many more variations and categories, which we will not all describe. A

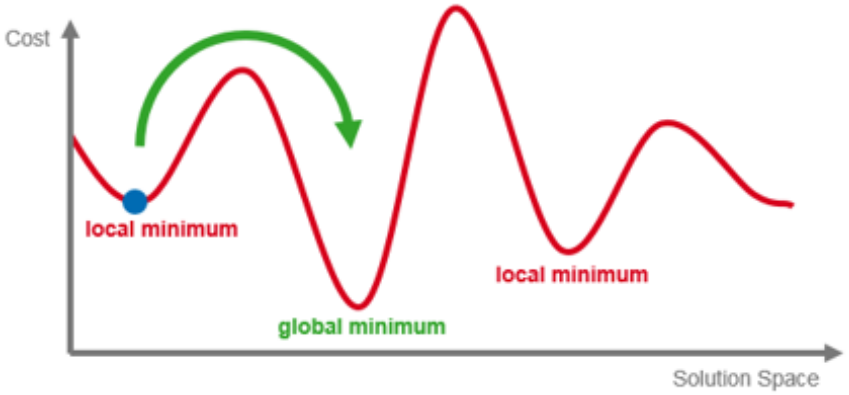



**Figure 2.1:** Example of a LNS which escapes a the local minimum of an one dimensional solutions space. ORTEC (2019)

good overview can be found in Labadie et al. (2016).

The main reason we cannot name all existing algorithms is that multiple heuristics can be combined to create so-called hybrid methods. Hybrid methods can be created in a variety of ways. The easiest example is to use the output of one heuristic as the input for the next heuristic, but it is also possible to combine parts of different heuristics. Every named metaheuristic essentially is a hybrid method, as they make use of construction methods and local search methods. Changing the local search methods used in such a metaheuristic already creates a new hybrid method. Hybrid methods have the advantage that they can use the strengths of multiple heuristics and avoid the flaws a single method might have.

## 2.8. DTSM: Optimization algorithm

The optimization algorithm in a DTSM system is used to occasionally (re)-optimizes the current delivery schedule. In Section 2.7 we have described many different algorithms to solve a Vehicle Routing Problem. All the described methods could be optimization methods used in DTSM, so we will not describe them again here. In this section, we will give a high-level description of the optimization algorithm used for our DTSM model. The pseudo-code for the full optimization algorithm will be detailed in Appendix A.

It is impractical to solve real-world problems with exact methods because there often is a limited time in which a VRP has to be solved. Therefore, the optimization algorithms used by ORTEC are based on multiple heuristics, creating a hybrid method, which is what we will use in our model. This hybrid method can be divided in three parts; Construction, Local Search and Large Neighborhood Search.

### 2.8.1. Construction

To perform Local Search steps and a Large Neighborhood Search method, an initial solution is needed. The construction method provides this solution. This construction method can be very advanced and produce a very good solution, but this also takes more computation time. Instead, the construction method could be used to quickly provide a solution so that the improvement methods can start to find better solutions. Which of these options is best mostly relies on the local search methods that are used and how much computation time is available.

Most of the time, construction starts with empty routes that have to be filled with the yet unplanned orders. The delivery schedule produced by the construction method can be greatly influenced by the first orders that are inserted. Therefore, it can be very beneficial to first pick some key orders to insert first. Such orders are called seed orders. Choosing what the seed orders should be, depends on the objectives and the problem. For example, it could be that some orders should not be in the same route, this could immediately be ensured by choosing these orders as seed orders for different routes. Other options include choosing a very hard task, for example with strict time windows and high demand, to be inserted first as a seed order to ensure it is included as it can be very hard to insert in a route at a

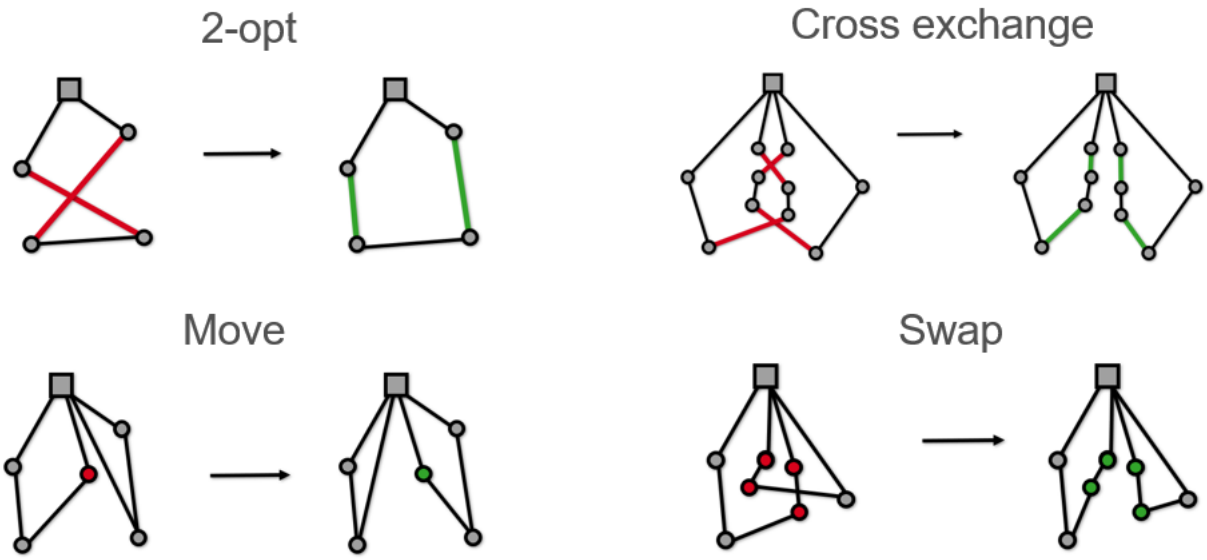



**Figure 2.2:** Examples of different local search methods. ORTEC (2019)

later time.

Routes can be build up in two main ways; sequential or parallel. With sequential insertion, one route is selected and orders are inserted one by one until no more orders fit in it. After which the next route is selected which can be filled with orders. In this case, the first order inserted in every route may be a specified seed order. With parallel insertion, the orders are still inserted one by one, but can immediately be inserted in every possible route. In this case, it is possible that first a seed order is inserted into every route.

After seed orders are inserted the other orders are inserted one by one. The orders can be sorted on different criteria to determine which order is to be inserted first, these criteria are again based on the objectives. An insertion method needs to be specified to determine where to insert the orders. A basic example is Cheapest Insertion, where the order gets inserted in the cheapest spot, meaning that it will increase the total costs the least, regardless of what those costs might be. A common sorting criterion follows from the Parallel Cheapest Insertion method, where for every order the cheapest insertion cost is determined and the order with the lowest insertion cost gets inserted first. Another common example is Parallel Regret insertion, where for every order it is determined what the lowest insertion cost and second-lowest insertion cost is. The difference between these two values is the regret value of the order. The idea is that the higher this value is, the more regret you are going to have if the order gets inserted in its second-best insertion point instead of the best insertion point. Therefore, with Parallel Regret insertion, you insert the order in sequence, starting with the order with the highest regret value. After each insertion, the regret value is recalculated.

### 2.8.2. Local search

After the construction phase, we try to improve the delivery schedule. The first step in this is with the use of local search methods. With a local search, the neighborhood of the current solution is explored to find a better solution. An iteration could end when the first solution is found that is better than the original solution. Another possibility is to check all solutions available in the neighborhood and pick the best one there is to continue with. ORTEC software uses a variation of the second approach, where it tries to check most options in the neighborhood, but first uses estimations and quick feasibility checks to speed up this process. To get a neighborhood of a solution it needs to be perturbed in a certain way. There are different local search methods that can do this. Below we describe the common improvement methods which are also used in the software of ORTEC. Examples of these four local search methods can be found in Figure 2.2.

The first method we discuss is 2-opt. The method makes changes to a single route or two different

routes. From this route, it deletes two edges, which are the connections between two locations in a route. It then creates a new route by adding two new edges, which can only be done in one way if we still want to get a feasible route that is different from the first one. The method is often used to eliminate crossings from a route. Some part of the route may reverse due to a 2-opt step, this is a big disadvantage of the method as this could easily mean that the resulting route is infeasible when (strict) time windows need to be met.

Cross-exchange is another common improvement method. Two sub-paths are selected from two routes or two independent parts of the same route. A sub-path is a sequence of consecutive orders on a route, usually limited to a certain size. These sub-paths are then interchanged between the routes. This is done by removing four edges, two from each route, and then adding four new edges. It can be seen as performing two 2-opt steps at the same time.

Similar to the Cross-exchange is the Swap improvement method. Again, two sets of consecutive orders are selected, this time they can be from the same route but they are of equal size and are independent. The two sets are then interchanged.

The Move improvement method is a variation on the Swap method. In this case, only one set of orders is selected, which is then moved to a different route or a different place within the same route.

### 2.8.3. Large Neighborhood Search

A solution found by the Local Search methods is always a local optimum, but not necessarily the global optimum. To try and escape this local optimum a Large Neighborhood Search is used. The algorithm used for this at ORTEC is based on the Ruin & Recreate method, as described in Schrimpf et al. (2000). Ruin & Recreate is an iterative algorithm. With each iteration, a part of the delivery schedule is 'ruined' by removing some of the orders. Then, the routes are 'recreated' by re-inserting these orders, resulting in a new schedule which could have lower costs than the previous schedule. If an improvement is found, the new schedule is used for the next iteration and otherwise, we stick to the old schedule. With the LNS further improvements are found which we were not able to find using only Local Search methods.

As Local Search methods are fast and able to provide great results they are often used in between some of the LNS steps and after the final iteration to intensify the search. This is only done when the delivery schedule found by the ruin and recreate method is not way worse than the original delivery schedule. A threshold is set for how much worse the found schedule can be before we perform the local search step.

Algorithm 4 describes the pseudo-code of a Ruin & Recreate algorithm.

**Algorithm 4:** Ruin & Recreate

**Input:** A delivery schedule

```
for A number of iterations do
    Copy the current schedule;
    Select a ruin method and a recreate method;
    Select a set of orders to remove from the copied schedule;
    Remove the orders from the copied schedule;
    Add the orders to the copied schedule using the recreate method to create a new schedule;
    if costs of new schedule are lower than a set threshold then
        Perform local search;
    end
    if cost are lower than original schedule then
        replace current schedule with new schedule;
    end
end
```

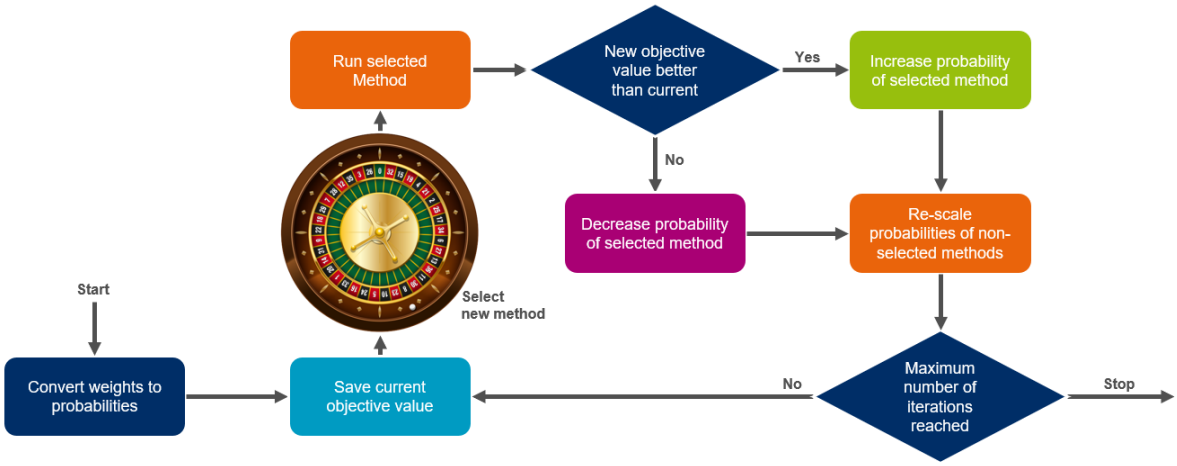



**Figure 2.3:** Ruin & Recreate using a roulette wheel. ORTEC (2019)

**Ruin methods**
There are many methods for removing orders. The simplest way is randomly selecting orders to remove from the delivery schedule. Other methods remove clusters of orders by randomly selecting an order and removing all the orders in its neighborhood or by removing all the orders of a single route. Yet other methods remove orders based on their characteristics, for example by removing orders that come with very high costs. With each method comes the decision of how much of the delivery schedule should be ruined, which is usually a certain percentage of all the orders.

**Recreate methods**
To recreate the delivery we use insertion methods similar to those described in section 2.8.1. Cheapest insertion, parallel cheapest insertion, sequential insertion and regret insertion methods are all valid options to recreate the ruined delivery schedule. After the delivery schedule is recreated, a local search method can be used to find some quick improvements.

**Roulette wheel**
A combination of a ruin method and a recreate method form a pair that can be used in an iteration of the Ruin & Recreate algorithm. The used pair can be different for each iteration. To decide which method to use the algorithm makes use of a so-called roulette wheel. This roulette wheel assigns a probability value to each pair of ruin and recreate methods. At first, each method has an equal chance to be used by the R&R algorithm. Whenever a certain combination is used we check how well this pair performed. If a delivery schedule with lower costs is found, we increase the probability of using this pair in one of the next iterations and as a consequence the probability of using one of the other pairs slightly decreases. In the same way, whenever a pair does not find a good improvement in the delivery schedule, we decrease the probability of using this pair of removal and recreate methods for the next iterations.

### 2.8.4. Optimization moments
We will periodically use the optimization algorithm to optimize the current delivery schedule. We could consider a DTSM system in which the optimization algorithm is immediately started again after the previous optimization run finishes, which would be beneficial as the optimization algorithm can only improve the current solution. However, in practice this is not a feasible procedure as this will cost a lot of CPU time. It would be a fine idea if only one DTSM system for one given day needed to be executed, but retailers often have a large number of divisions and offer time slots for multiple dates, possibly a month away from the current day. This means that DTSM systems are running for each division and each day, which makes saving some CPU time profitable.

Therefore, the frequency of optimization often increases once we get closer to the cutoff time. The idea is that there are so few new orders at the beginning of a DTSM system that running an optimization

algorithm is not that useful. Close to the cutoff time, there are a lot of new customers that want to place an order and running an optimization algorithm has a significant effect. Therefore there often is a point in time in which the frequency of using the optimization algorithm gets drastically increased. For example, the optimization algorithm could be used only every twelve hours after the first customer has placed an order and the DTSM system starts, but will run every hour in the last two days before the cutoff time.

## 2.9. Contribution of this thesis

In this chapter, we have shown the many aspects of Dynamic Time Slot Management and the accompanied route scheduling have been researched. Most of the existing literature is focused on creating good time slot offers, pricing time slots or on different kinds of insertion methods. Some of the research also considers an optimization algorithm that periodically optimizes the current delivery schedule.

To our knowledge, only one previous research Visser et al. (2019b) considered the real-time aspect of DTSM. They created a model to see what would happen if multiple customers arrive and request time slots at the same time. This research in some cases did include an optimization algorithm that ran periodically. However, in their simulations they made it so that when a customer would arrive while the optimization algorithm was running, the resulting optimized schedule would be ignored as it could be infeasible to include the new customer.

In this dissertation, we would like to explore possible procedures to deal with new customers that arrive while the optimization algorithm is running other than ‘throwing away’ the optimized schedule. We will explore this in detail in the next chapter. As there is, to our knowledge, no known literature about the effect of the optimizer in a real-time DTSM scenario, we hope to give more insight into the impact of the chosen methods with this research.

# 3

# Procedures for the real-time problem

In the previous chapters, we have described Dynamic Time Slot Management in detail. The main problem we will now focus on in this thesis is "How to deal with customers that arrived while the optimization algorithm is running?". In this chapter, we will describe the different procedures we created to handle this problem.

## 3.1. Different procedures

With Dynamic Time Slot Management we keep track of a delivery schedule throughout the process. We will call this delivery schedule $A$. Schedule $A$ consists of several routes in which all customer orders that were accepted up until that time are present. If a new customer arrives, they get a time slot offer based on schedule $A$ and after they picked a time slot their order gets inserted, updating schedule $A$ to include this order.

Periodically, a copy of schedule $A$ gets created so that an optimization algorithm can use this to improve the schedule. This algorithm produces a new delivery schedule, which we will call schedule $B$. Running the optimization algorithm to produce such a new schedule is called an *optimization run*. Ideally, schedule $A$ would be replaced by schedule $B$ in the system as soon as schedule $B$ is available because the goal of the optimization run is to provide a more efficient schedule to be used in the rest of the DTSM process. However, new customers can arrive during the optimization run. These customers get time slot offers which are based on schedule $A$. The resulting order after they choose a time slot is also inserted in schedule $A$, which means that schedule $A$ always contains all current accepted customers. However, when the copy of the schedule was created for the optimization run, the new customers were not yet present, which means that these new customers are not included in schedule $B$. As a consequence, we cannot simply replace schedule $A$ with schedule $B$, because schedule $A$ has more planned customers than schedule $B$.

In this section, we explore possible procedures to deal with this problem. The input for such a procedure consists of schedule A, schedule B and all orders that arrived during the optimization run (which are also included in schedule A). The output of such a procedure is a routing schedule $C$, which is the schedule used to carry on with the DTSM process. All procedures are aimed at creating a feasible delivery schedule, which means that the orders of all accepted customers need to be included in schedule $C$.

As the real-time aspect often gets ignored in a theoretical setting, existing literature does not immediately provide procedures we can use. We have derived multiple procedures based on ideas that come up in TSM literature. Each procedure had different advantages and disadvantages and it is not known in advance which procedure will provide the best results, which is why we will investigate multiple procedures instead of only one. With our computational experiments, we will compare the different procedures to find which work best in which scenario. Next to that, the proposed procedures are not the only solutions to the problem. We think that the procedures we choose are most intuitive and cover the most different advantages and disadvantages one could think of. The procedures also range from

simple changes to using more complicated algorithms. Furthermore, it is interesting to see if there is a great variance in results when comparing these procedures, as it is not known how much the impact is of using a different procedure to deal with the real-time optimization problem.

An overview of all procedures can be found in Table 3.1.

### 3.1.1. Procedure 1: Delay Customer

In the theoretical DTSM setting, without the real-time aspect, no parts of the system happen simultaneously but neatly one after the other. This first procedure tries to mimic that setting. This can be done by not allowing customers to request a time slot offer while the optimization algorithm is running. Instead, the arrival times of those customers are delayed until after the optimization run has finished and schedule $A$ has been replaced with the new optimized schedule $B$. This means that the customers have to wait on the optimization run before they can request a time slot offer and place an order. After this, each of the waiting customers is one by one inserted like normal. This happens in the order they arrived previously.

This procedure benefits from the fact that it can fully use the optimized schedule and then uses the information of each previous customer. A disadvantage of this procedure is that it provides bad customer service as it makes customers wait during optimizations. This makes this procedure unusable for retailers that want a hundred percent up-time on their websites.

### 3.1.2. Procedure 2: Delete schedule $B$

One of the main problems to solve is that the optimized schedule $B$ does not include all orders when customers arrived during the optimization run. A solution to avoid this problem is to never use schedule $B$ whenever this happens. So whenever no customers arrived during optimization schedule $B$ can be used and otherwise we discard schedule $B$ continue the process with schedule $A$.

This procedure was used by Visser et al. (2019a) in their research about real-time DTSM.

The advantage of this method is that there are no drastic changes to the original system. Quiet periods, in which few or no customers arrive, are used to perform the optimization and the system runs as normal in the busy periods without optimization. This does mean that, when a lot of customers keep arriving in close intervals, it is possible that schedule $B$ is never used, which is the main disadvantage of this procedure.

### 3.1.3. Procedure 3: Insert new customers in schedule $B$

A more sophisticated procedure is to try to insert all orders of the customers that arrived during the optimization run into schedule $B$. We do this by using the insertion method from the DTSM system. These orders may not fit, as the time slot offers for these customers were created based on schedule $A$. So we have two scenarios. If all new orders fit into schedule $B$, we use this newly created schedule if it has lower plan costs than schedule $A$. If not all orders fit in schedule $B$ or the plan costs after inserting are higher, we discard it and keep using schedule $A$. It is possible that inserting the new orders in the optimized schedule results in a worse schedule than the original schedule $A$ because the time slots that were available for the new customers were based on schedule $A$ and not on schedule $B$.

Not just discarding the optimized schedule when customers arrived during optimization is the main advantage of this procedure. A slight disadvantage is that this procedure takes more calculation time than others to perform the additional insertion methods and whenever the insertion does not yield a better schedule, this calculation time was unnecessary.

### 3.1.4. Procedure 4: Merge schedules

Schedule $A$ has all orders included in the schedule, but schedule $B$ has more efficient routes as it is optimized. This procedure tries to take the best from both worlds by merging these two schedules. All schedules contain the same number of routes, as this is equal to the number of available vehicles. We create a new schedule $C$ by looking at each individual vehicle. For each vehicle, we decide if we want to use the corresponding route of schedule $A$ or the corresponding route of schedule $B$. To decide which routes to use we use a merging algorithm.

**Merge Algorithm**

Merging the schedules in this way needs to be done in a smart way. We will illustrate this with a few examples. It is important to remember what the priority of the goals is:
1. Ensure that all orders in the resulting schedule $C$.
2. If possible, use routes of schedule $B$.

This is not trivial, as orders have been moved or swapped between vehicles by the optimization algorithm. This means that some orders might be in different routes if we compare schedules $A$ and $B$. For our examples, we will assume that there are nine orders, illustrated by the numbers 1 through 9. We also assume that the last order, order 9, was from a customer that arrived during the optimization run, which means it is not included in schedule $B$. We now consider the following input:

| Schedule $A$ | | Schedule $B$ | | Schedule $C$ |
|---|---|---|---|---|
| $1 \to 2 \to 3 \to 4$ | Vehicle 1 | $2 \to 1 \to 4 \to 3$ | $\Rightarrow$ | $2 \to 1 \to 4 \to 3$ |
| $5 \to 6 \to 7 \to 8 \to 9$ | Vehicle 2 | $8 \to 5 \to 7 \to 6$ | | $5 \to 6 \to 7 \to 8 \to 9$ |

In this case, it is easy to decide which routes to use for the new schedule. For vehicle 2 we have to pick the route from schedule $A$, as this is the only route that contains order 9. For vehicle 1 we have no such restriction, which means that we pick the route from schedule $B$, as this is an optimized route.

We have to pick every route that contains a new order from schedule $A$, as seen in the previous example. Sadly, we cannot just pick the routes from schedule $B$ for the remaining vehicles. We will illustrate this with the next input:

| Schedule $A$ | | Schedule $B$ | | Schedule $C$ |
|---|---|---|---|---|
| $1 \to 2 \to 3$ | Vehicle 1 | $2 \to 1 \to 3$ | $\Rightarrow$ | $2 \to 1 \to 3$ |
| $4 \to 5 \to 6$ | Vehicle 2 | $5 \to 6$ | | $4 \to 5 \to 6$ |
| $7 \to 8 \to 9$ | Vehicle 3 | $8 \to 4 \to 7$ | | $7 \to 8 \to 9$ |

We see that in schedule $A$ order 9 was included in vehicle 3, so we have to choose the route of schedule $A$ for vehicle 3. We would like to use the routes from schedule $B$ for vehicles 1 and 2. However, this is not possible as order 4 would then not be included in the new schedule. Order 4 got switched from vehicle 2 to vehicle 3 by the optimization algorithm. Because we picked schedule $A$ for vehicle 3, we also need to pick schedule $A$ for vehicle 2 to ensure that the merged delivery schedule includes all orders. Vehicle 1 has no orders in common with either vehicle 2 or 3 in both schedules in this case, which means that we can safely use the route of schedule $B$ for vehicle 1 to complete the new schedule.

In the last case, we call vehicles 2 and 3 'connected'. Two vehicles are connected if one ore more orders got moved from one vehicle to the other by the optimization algorithm. In other words, vehicle $x$ and $y$ are connected if there is an order that was assigned to vehicle $x$ in schedule $A$ and is now assigned to vehicle $y$ in schedule $B$. When we are merging two schedules we have to consider connected vehicles.

Assume we have two connected vehicles $1$ and $2$ and we create the final schedule by picking the route from schedule $A$ for vehicle 1 and vehicle $2$ the route from schedule $B$. Because the vehicles are connected we have two scenarios:
An order got moved from vehicle 1 in schedule $A$ to vehicle 2 in schedule $B$. In this case, the resulting schedule will include the moved order twice.
An order got moved from vehicle 2 in schedule $A$ to vehicle 1 in schedule $B$. In this case the resulting schedule will not include the moved order.
It is even possible that both scenarios happen at the same time when multiple orders got moved between vehicles during optimization. Both scenarios can be seen in the example below.

| $A$ | | $B$ | | $C$ |
|---|---|---|---|---|
| $1 \to 2$ | Vehicle 1 | 1 | $\Rightarrow$ | $1 \to 2$ |
| 3 | Vehicle 2 | $3 \to 2$ | | $3 \to 2$ |

| $A$ | | $B$ | | $C$ |
|---|---|---|---|---|
| 1 | Vehicle 1 | $1 \to 2$ | $\Rightarrow$ | 1 |
| $3 \to 2$ | Vehicle 2 | 3 | | 3 |

We want to ensure that the resulting schedule $C$ is feasible, which means that every order is included exactly once. In general, this means we need to pick the routes from the same delivery schedule for a

pair of vehicles that is connected because we would fall into at least one of the scenarios depicted out above otherwise, which means we would have an infeasible schedule.

To summarize, we have two main facts which we use to determine how to merge schedule $A$ and $B$:

- Vehicles that include a new order must use the route from schedule $A$
- Vehicles that are connected must use routes from the same schedule

Algorithm 5 shows the pseudo-code that can be used to determine which routes should be used when merging two schedules.

**Algorithm 5:** Merge Algorithm

**Result:** Schedule $C$

Create a graph $G = (V, E)$, where $V$ is the set of vehicles;
For two vehicles $v_1, v_2 \in V$ we have: $(v_1, v_2) \in E \iff v_1$ and $v_2$ are connected;
Find the connected components in $G$ ;
**foreach** connected component in $G$ **do**
    **if** there is a new order present in any vehicle of the connected component **then**
        use the routes from schedule $A$ for all vehicles in this component;
    **end**
    **else**
        use the routes from schedule $B$ for all vehicles in this component;
    **end**
**end**

The main advantage of this procedure is the fact that you still use the optimized schedule as much as possible while making sure that the new orders are included. These new orders remain in routes for which they were intended to be in by using the routes schedule $A$ for those, which should allow the routes to stay efficient. However, it is possible that all vehicles or most vehicles are connected, which would mean that the new merged schedule is almost equal to the old schedule $A$. Another slight disadvantage of this procedure is that it takes extra calculation time to perform the merging algorithm.

### 3.1.5. Procedure 5: Insert new customers in schedule $B$ and Merge

The last procedure is a combination between Procedures 3 and 4. Similar to Procedure 3 we try to insert all new orders in schedule $B$ which creates a new schedule $B'$. However, when not all new orders fit in this schedule we do not immediately discard it this time. Instead, we use the merging algorithm described in Procedure 4 to merge schedule $A$ and $B'$ to find a new feasible schedule.

The advantage of this method is that it uses the full potential of the optimized schedule by inserting the new customers. Whenever this fails we still try to make the best use out of the optimized schedule using the merging algorithm. This comes however with both disadvantages that inserting new orders could lead to a bad schedule, because time slots are based on an older schedule, and that if there are a lot of connected vehicles we still do not use the optimized schedule. It also increases the calculation time the most across all procedures.

## 3.2. Alternative procedures

To analyze the performance of the procedures we will compare them against each other. Additionally, we will use two alternative procedures for additional information. These procedures are thinned down versions of the DTSM system. By leaving out parts of the DTSM system the procedures avoid the real-time problem. However, this also means that these methods should probably not be used in practice as they will perform worse or do not provide the desired system. We will explain this further for each individual procedure.

**Table 3.1:** Overview of all procedures.

| Procedure | Explanation | Advantages | Disadvantages |
|---|---|---|---|
| 1 | Delay customers | All information used. | Poor customer service. |
| 2 | Delete optimized schedule | No drastic changes, quiet periods used efficiently. | Possibility optimizer is never used. |
| 3 | Insert or delete | More use of optimized schedule. | Time slots of inserted orders are not based on optimized schedule. |
| 4 | Merge schedules | Orders remain in routes they fit well in. | Does nothing when most vehicles are connected. |
| 5 | Insert or merge | Both advantages of 3 and 4. | Both disadvantages of 3 and 4. |

### 3.2.1. Procedure NoOpt: No optimization step is used

Another very simple procedure is provided by not using an optimization algorithm during the DTSM system. This means that other than inserting customers into the current schedule $A$ no changes are made to the delivery schedule. As there is no optimizing done, we automatically avoid the problem of new customers arriving during the optimization. After the cutoff time, one optimization run is done to find the final delivery schedule. This procedure allows us to see how useful it is to use an optimization algorithm in the first place. As we are not sure of the impact of the optimization algorithm, this method does not function as a real bound on the problem. However, we would expect that any other procedure should outperform this one.

### 3.2.2. Procedure NoIns: No insertion step is used

Another possibility is to not use an insertion method, which means that the schedule $A$ does not get updated when a new customer arrives. Instead, the order of the customer is stored until the next optimization run starts. Then all new orders are directly given together with schedule $A$ as input for the optimization algorithm. After the optimization run finishes, we update schedule $A$ with the new schedule, regardless of any customers that arrived during the optimization run. These customers, together with any more that will arrive, will be given as input for the next optimization run and will until that time not be in the current delivery schedule.

With this method, you do not have to solve any discrepancies between the current and optimized schedule. A big disadvantage of this method is the possibility of orders becoming impossible to fit in the delivery schedule, as new request might be made when not all previous orders are in the schedule. These orders will remain unplanned and will not be included in the final delivery schedule. This possible for an infeasible schedule makes this method not a valid solution to the real-time problem.

# 4

# ORTEC Services

ORTEC as a company has many customers. Their products support various customers, of which a lot are in the logistics branch. Among these customers are retailers who perform some form of attended home delivery. As of today, ORTEC is in the process of combining two of their products to deliver a Dynamic Time Slot Management system to these retailers. Both of these products are cloud services, a service provided through the internet. The first is a time slot management service and the second is a routing service. Together they form a DTSM solution. In this chapter, we discuss the details of the ORTEC Cloud Services.

## 4.1. ORTEC Cloud Services

*The information in this section is confidential and is not included in the public version of this thesis.*

## 4.2. TimeSlotting Service

*The information in this section is confidential and is not included in the public version of this thesis. It explains how the TimeSlotting service of ORTEC works. This service handles most parts of a Dynamic Time Slot Management system.*

## 4.3. COMTEC Vehicle Routing Service Cloud

*The information in this section is confidential and is not included in the public version of this thesis. It explains how the COMTEC Vehicle Routing Service (CVRS) of ORTEC works. This service handles the routing algorithms of ORTEC. Some ORTEC-specific terms remain unexplained in this version. The most important one is the command template.*

## 4.4. TimeSlotting and CVRS combined

The two cloud services can together form a Dynamic Time Slot Management system. The TimeSlotting service is used to store the delivery schedule and to create time slot offers for new arriving customers. After the customer picks a time slot an order is created which can then be inserted in the current delivery schedule. The DTSM system can include an optimization algorithm using CVRS, all orders that need to be planned along with the appropriate command template are sent to the CVRS service. The resulting schedule from CVRS can then be sent to the TimeSlotting service to update the stored delivery schedule.

# 5

# Computational experiments

To answer the main question of this thesis we perform various simulations of a DTSM system to test the five different procedures we have described in Chapter 3. For this, we make use of the data of a customer of ORTEC. We will first give more details about the data and how we use it to set up the simulations in Section 5.1. After that, we will show the results of our computational experiments in Section 5.2.

## 5.1. Simulation setup

In a simulation, we want to create a delivery schedule for one single day. To do this we have to simulate the timeline leading up to that day including all aspects of a Dynamic Time Slot Management system.

Before we start simulating the timeline, a delivery schedule with empty routes will be created. The simulation then starts with the first customer that arrives to place an order for the selected day. This customer will request a time slot offer and will then select a time slot or leave. If the customer picked a time slot the resulting order is be inserted into the delivery schedule. This process then repeats for every arriving customer. In set intervals we will use the optimizer to optimize the current delivery schedule. This optimizer runs for a set amount of time. We check if there are any customers that have an arrival time during the optimization and if that is the case, we will use one of the different procedures, as described in Chapter 3, to handle these customers. This process continues until the set cut-off time. Then, the simulation ends after one more optimization run that creates the final delivery schedule.

To do such a simulation we need input data.

### 5.1.1. Data description

For our computational experiments, we will use the data of a large retailer, that we will call Retailer A for the rest of this thesis. Retailer A has been a customer of ORTEC for some time. They already perform attended home delivery using a Dynamic Time Slot Management system set up by ORTEC. This provides us with suitable data to perform our computational experiments with. Our goal is to find new conclusions about Dynamic Time Slot Management in a general setting, but we will only use the data from Retailer A, which could skew our results. For this reason, we have extensively discussed among the ORTEC consultants what data was best suited and we came to the conclusion that the data of Retailer A is representative for many general Time Slot Management problems, due to its variety in variables and problem sizes. This makes us think that our results will also be applicable to other scenarios.

Retailer A has split its operating area into multiple parts, called divisions. Each division is fully independent from another which means that on beforehand, we know for each location to which division it would belong. We use this fact to our advantage, as there are many differences between the divisions, giving us a good variety of data to consider and compare. Each division has a single depot in which

**Table 5.1:** Cases used for simulations.

| Case | Division | Date | Number of customers | Number of vehicles |
|---|---|---|---|---|
| 1 | 39 | August 29th | 104 | 7 |
| 2 | 23 | November 7th | 178 | 27 |
| 3 | 24 | August 27th | 1295 | 114 |
| 4 | 24 | August 29th | 1114 | 99 |
| 5 | 24 | November 4th | 1633 | 141 |

every available vehicle starts and finishes. These vehicles are also all assigned to there own division beforehand.

Using the data set of Retailer A we will create a couple of smaller data sets that can be used to do a simulation of the DTSM system to create a delivery schedule for one day. From now on we will call such a data set a *case*.

**Cases**
The data is divided into 157 divisions. For each division, we have access to the full data of in total ten days, from two periods of five consecutive days. Each day is considered completely independent of any other day, as each day has its own set of orders and vehicles. This means that we have information for in total have 1570 potential days to make a simulation out of. The data includes all accepted orders which were carried out on the given day and all vehicles used to deliver these orders. To determine which divisions were interesting we considered the number of orders delivered and the number of vehicles used on average on a given day. The real values never deviate much from this average so we can use this information to decide which divisions should be used.

From this, we found that most divisions have not more than 100 orders to handle on a given day on average, which is very little. The Vehicle Routing Problem will be relatively easy to solve if we use the data of these divisions and the results will therefore be not very interesting. We do find that there is one division, number 24, that has on average more than a thousand orders. This division is much more interesting as the resulting VRPs will become quite difficult. According to ORTEC consultants and what we have seen in literature, this number of orders is also more in line with the numbers that other retailers have to deal with in practice and will therefore provide the most interesting results.

To still have some diversity, we will also use the data from divisions 23 and 39. These divisions both have days on which they handle more than a hundred orders. We have decided on precisely these divisions because of the vehicle/order ratio, where division 39 has a very high ratio of about 15 orders per vehicle and division 23 a low number of about 6 orders per vehicle. In combination with the 11 orders per vehicle that is often seen with division 24, we have a good variety of different data sets.

We created five different cases in total. One day from both divisions 23 and 39 and three days from the large division 24. An overview of the different cases we have created can be found in Table 5.1.

**Case data and VRP**
A single case contains all data needed for the simulation of DTSM system which creates the delivery schedule of a single day of a single division. This division has only one depot of which we know the location. The depot is always available from $[06:00 - 23:59]$. Furthermore, the case data contains all orders and used vehicles for that day. For each vehicle we know:

- Start and end time
- Start and end location, which is the location of the depot
- Capacities
- Break duration
- Break interval, which is the time window in which the break needs to take place

For each order we know:

- Customer location
- Service duration
- Original picked time slot
- Quantities
- Customer arrival time, which is the time at which order was placed by the customer.

With the available data, we have all information needed to create a Vehicle Routing Problem as described in Section 2.2.1. Because we want to keep the simulation close to the real setting of Retailer A we include the VRP extensions they use for their delivery schedules. We have start and end times for each route, which determines the time window in which the vehicle must leave and return to the depot. Every vehicle has three forms of capacity, one for the number of crates, one for the number of special crates and one for the total weight. Each order also has these three quantities. These individual capacities cannot be exceeded by the total corresponding quantity of the orders inside of the vehicle. Vehicles also have a specified break, which consists of a time duration and a time interval in which this break takes place. This break can be seen as an additional mandatory order on the route which does not have a specific location. These VRP extensions are also described in Section 2.6.3.

We do not know which orders were assigned to which vehicle and what the exact routes of these vehicles looked like. We only know the number of vehicles used and all characteristics of the vehicles like starting times and capacity. This means that we do not have access to the original delivery schedule for comparison.

**Only accepted customers**
The data only includes customers for which an order has been planned and was delivered. This means that we do not have data on any customers that requested a time slot offer and then decided to not pick any of the time slots and leave. This could mean that problem cases made from this data become very easy to solve, as it is all data from an actually executed delivery schedule. We can avoid this problem by modeling the customers' preferences with a random choice. We will discuss this more in the next section.

### 5.1.2. Time slots

Retailer A provides its customers with time slots that span almost the entire day. The length of each time slot is one hour. The earliest time slot starts at 07:00 and the latest time slot starts at 22:00. Time slot offers will be created using the current delivery schedule. Every time slot that would result in a feasible delivery schedule is offered to the customers, so deciding which time slots to present to the customers is purely based on the feasibility of the solution.

**Customer choice**
An arriving customer requests a time slot offer. This gets created and presented to the customer. The customer then picks one of these time slots or chooses to leave. As described in the mathematical description in Section 2.2.1, we will model this by giving each customer an ordered set of preferred time slots, which is a subset of the full set of time slots. This means that we need to create these ordered sets of preferred time slots before the simulation starts. In DTSM literature studies, these subsets are often uniformly randomly assigned. This is based on the fact that retailers often use pricing to spread out the demand evenly over all possible time slots (Klein et al., 2019). However, in our data set, we have seen that the distribution of picked time slots in the delivery schedules of Retailer A is not uniform.

From the data we know which time slot the customers picked originally after they got a time slot offer. In Figure 5.1 we have summarized this using the data of all cases. We can clearly see that it does not represent a uniform distribution as some time slots are chosen a lot more often than others. We will need to take this into account with our computational experiments.

We generate a set of preferred time slots for each customer by drawing from the set of available time slots according to the distribution we have in the data. For each case, we use the distribution of time slots as shown in the whole division that corresponds with that case. By assigning the preferred time

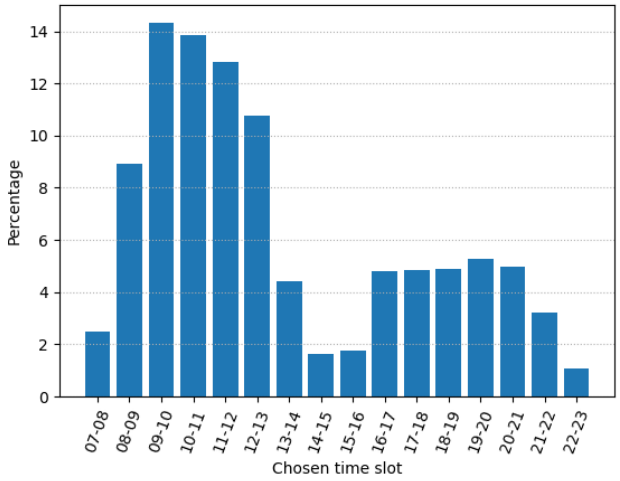



**Figure 5.1:** The distribution of the time slots chosen by the customers of division 24.

**Table 5.2:** The cost set used for the cost function.

| Variable | Value |
|---|---|
| Cost per used vehicle $UV$ | 200 |
| Cost per hour $TT$ | 30 |
| Cost per meter $TD$ | 0.000621373 |

slots to customers in this way, it is likely that the simulated customers prefer a different time slot than the time slot that was originally assigned to their order in the data, but the distribution of all preferences should be similar to the distribution of the originally picked time slots. An added benefit of assigning the preferred time slots according to the distribution to simulated customers is that the resulting Vehicle Routing Problems will not be trivial as it is not likely not possible to produce the same delivery schedule the data was based on.

### 5.1.3. Optimization algorithm

For the simulations, we have decided to use the optimization algorithm that best fits Retailer A. The command template for this algorithm (Appendix A) was specifically tailored with the help of ORTEC consultants to work well for this retailer. It is important to know the duration of this algorithm for the timeline of our simulations. This command template was created to perform well with a run time of fifteen minutes, so this will be the duration of our optimization algorithm as well.

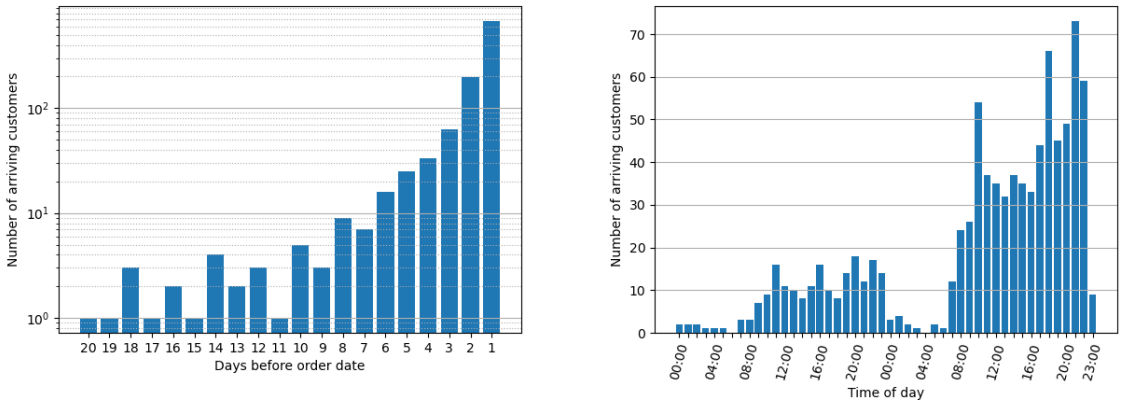
#### Objective and cost set

The optimization algorithm will consider multiple objectives, as explained in Section **??**. The main objective is the number of planned customers. The secondary objective is the PlanCosts. The PlanCosts are calculated using a cost function. These objectives will be considered in lexicographical order.
We calculate the PlanCosts with a cost function. This cost function makes use of a cost set, found in Table 5.2. This cost set is based on previous work done by ORTEC consultants in collaboration with Retailer A. The used cost function is linear:

$$cost(UV, TT, TD) = 200 \cdot UV + 30 \cdot TT + 0.000621373 \cdot TD. \tag{5.1}$$

### 5.1.4. Simulating the timeline

To simulate the timeline properly we need to know at which point in time customers arrive and when the optimization algorithm starts and ends. As we already know that the optimization algorithm will run




**(a)** Arriving customers during the DTSM process. **(b)** Arriving customers in the last two days

**Figure 5.2:** Arriving customers during the DTSM process for customers of case 3.

for fifteen minutes we only need to determine the start times of the optimization algorithm. We will base the customer arrival times and the optimization start times on the available data we have.

For each order, and thus customer, we know at which time they requested a time slot offer in the original DTSM system. This time is somewhere in the days leading up to the date of the order they placed, as the cut-off time for placing orders is at 00:00 of the day the orders should be delivered in. An example of the arrival times of a full case with approximately a thousand orders can be found in Figure 5.2a. The figure shows only the data of one case, but this distribution is representative of all different cases. Note that this figure has a logarithmic scale, which indicates that most customers arrive within the last two days before the cut-off time. The hourly arrivals of the last two days can be found in Figure 5.2b. Even here we see that the closer we get to the cut-off time the more customers arrive. We will use this knowledge to determine the frequency in which the optimization algorithm is used.

**Optimization frequency**
In Section 2.8.4 we addressed the fact that it is not beneficial to run an optimization algorithm at all times and most definitely not when there are not many new customers arriving. This is supported by the data as seen in Figure 5.2. Most customers arrive at the end of the timeline, near the cut-off time. To save unnecessary calculation time we will apply the practice of starting with a low frequency of optimization runs, which increases to a high frequency near the cut-off time.

In our case, at the start of the simulation, the optimization algorithm will be executed every twelve hours of the timeline, starting twelve hours after the arrival of the first customer. Then, when we are in the last two days before the cutoff time this frequency is changed from running the algorithm every twelve hours to running it every hour. The run time for the algorithm will be fifteen minutes. We have decided on these settings together with ORTEC consultants and experience from ORTEC indicates that these settings are representative of different retailers that use a DTSM system.

**Customers arriving during optimization**
Whenever a customer arrived during an optimization run we use one of the procedures from Chapter 3 to solve the resulting problem. Our goal is to see what the differences in the final delivery schedule are when a different procedure is used. This means that the situation that a procedure is needed must occur. Now that we have determined when the optimization algorithm will be executed we know what the simulation timeline for each case will look like. Table 5.3 shows that about twenty percent of the customers arrive during an optimization run, which should be an adequate amount to see a different result when a different procedure is used.

**Multiple simulations**
Each case will be simulated with all different procedures. To get more reliable results, some cases are simulated multiple times with the same procedures. As repeating the same simulation would yield the

**Table 5.3:** Customers arriving during optimization

| Case | Number of customers | Percentage arriving during optimization |
|---|---|---|
| 1 | 104 | 17.3% |
| 2 | 178 | 16.9% |
| 3 | 1295 | 18.5% |
| 4 | 1114 | 21.8% |
| 5 | 1633 | 21.6% |

same results a different set of preferred time slots is generated according to our defined distribution each time a simulation with a certain case is done. This will create a new problem while keeping as many similarities between the different simulations as we can. For example, for case 1 we will generate five sets of preferred time slots for each customer and for each procedure we did five simulations using each set once.

As the run time of the simulations is very large, especially for the larger cases, we will only run multiple simulations for the two smaller cases. The three larger cases are from the same division, so this is similar to doing three simulations for one large case, which makes up for the fact that only one simulation per large case is done.

## 5.2. Results

In this section, we will discuss the results of our computational experiments. We will discuss the small and large cases separately. For each case we will show:

- Planned customers, which is the number of customers that are included in the final delivery schedule. This is the main objective we want to maximize;
- Unplanned customers, which is the number of customers that picked a time slot but could not be included in the final delivery schedule by the optimization algorithm. If this is more than zero the delivery schedule is considered infeasible and all procedures are designed to prevent this;
- Customers left, which is the number of customers that did not pick a time slot and left, because they did not get offered a time slot that was part of their set of preferred time slots;
- PlanCosts, which are the costs of the final delivery schedule according to the cost function as found in Equation 5.1, minimizing this is our secondary objective after the number of planned customers is maximized.

Note that the PlanCosts are made up of the three variables: Number of used vehicles, the total time needed to execute the delivery schedule and the total distance of the delivery schedule. When analyzing the results we have considered the individual variables instead of only at the PlanCosts. However, it turns out that looking at the individual variables gives no new insight compared to only considering the PlanCosts. For this reason, the values of the individual variables are not shown in the results in this chapter.

Each case has been simulated with all procedures as described in Chapter 3. An overview of what each procedure entails can be found in Table 3.1.

### 5.2.1. Small cases

The first two cases have a relatively little number of customers. Each case has been simulated several times, each time with a different set of time slot preferences for each customer. Tables 5.4 and 5.5 show the average results over all these simulations.

#### Case 1

The first thing we notice is that the number of planned orders is similar across all procedures. Procedure 1 performs the best with 97 planned orders on average. This can be expected as this procedure mimics the theoretical setting in which the real-time aspect of arriving customers is neglected. Procedure 5 has the second-highest number of planned orders with an average of 96. This is slightly higher than

**Table 5.4:** Case 1: 104 total orders, 7 available vehicles, average results over 5 simulations.

| Procedure | Planned orders | Unplanned orders | Customers left | PlanCosts |
|---|---|---|---|---|
| 1 | 97 | 0 | 7 | 2678.46 |
| 2 | 94.8 | 0 | 9.2 | 2665.66 |
| 3 | 95.6 | 0 | 8.4 | 2681.43 |
| 4 | 94.8 | 0 | 9.2 | 2682.83 |
| 5 | 96 | 0 | 8 | 2681.88 |
| NoOpt | 88.8 | 0 | 15.2 | 2627.02 |
| NoIns | 98.2 | 0.2 | 5.6 | 2687.43 |

the average of Procedure 3 with an average of 95.6. This is also expected as these procedures are similar in the fact that they try to insert customers that arrived during the optimization in the optimized delivery schedule. When this fails Procedure 5 makes use of the merge algorithm which Procedure 3 does not. The merge algorithm can only improve the solution which gives Procedure 5 the slight edge over Procedure 3.

Procedure 2 and 4 have the lowest number of planned orders. Procedure 2 uses the non-optimized schedule $A$ whenever a customer arrived during the optimization run. As a consequence, the delivery schedule is not optimized often which makes the solution worse. Procedure 4 uses the merge algorithm, if a lot of vehicles are connected, as defined in Section 3.1.4, this procedure will also use a schedule that is very similar to the non-optimized schedule $A$.

None of the procedures have any unplanned orders which would result in an infeasible delivery schedule. This is expected as each procedure was created in such a way that unplanned orders would be prevented.

Furthermore, the average PlanCosts of the procedures are quite similar. Schedules with fewer planned orders should have lower PlanCosts, as each extra delivery usually will increase the distance and time needed for the whole delivery schedule. Besides that, it becomes easier to create an efficient final delivery schedule after the cut-off time when fewer customers are accepted. This causes the PlanCosts to be lower than the PlanCosts of procedures which have more planned customers in most cases. A good example of this is Procedure 2. On the other hand, we also see that Procedure 1, which has the most planned orders, also has one of the lowest average PlanCosts. Procedure 1 does not have to deal with customers that arrive while the optimization algorithm is running and it can therefore use the full potential of the optimized schedule every time an optimization run is done. All the other procedures have to compromise in some way on the quality of the delivery schedule to make sure that customers that arrive during optimization are dealt with, making it so that the overall PlanCosts increase.

Interestingly enough, the Procedure NoIns has the highest number of planned customers. Because the schedule in memory does not get updated with an insertion after every accepted customer, more customers are accepted by the system as there is more room left within the routes of the schedule. Even though it is not guaranteed that these accepted customers can be planned, it happens so that the optimization algorithm is often able to include the customers at someplace in the delivery schedule, even though this might be a whole different insertion point than the offered time slot was based on. It is however possible that the optimization algorithm is not able to plan all those customers, which we see in the average count of 0.2 unplanned customers. This number is very low, as with a small case it is easier for the optimization algorithm to create a feasible delivery schedule even with a not so careful time slot offering process. However, this still means that for one of the five simulations we ended up with a final delivery schedule that is infeasible, which we want to avoid at all costs.

The Procedure NoOpt, which uses no optimization algorithm performs, significantly worse than each procedure, as we see that the worst Procedures 2 and 5 still plans about 6 customers more than NoOpt on average. This result is in line with the idea that the use of an optimization algorithm is beneficial to a DTSM system when simple insertion methods are used. Because the delivery schedules are not optimized, fewer time slots are offered to customers and more customers leave without placing an order. We note that this procedure has the lowest average PlanCosts of all results, but this is expected as there is a low number of orders present in the delivery schedule.

**Table 5.5:** Case 2: 178 total orders, 27 available vehicles, average results over 3 simulations.

| Procedure | Planned orders | Unplanned orders | Customers left | PlanCosts |
|---|---|---|---|---|
| 1 | 178 | 0 | 0 | 6477.77 |
| 2 | 178 | 0 | 0 | 6574.41 |
| 3 | 178 | 0 | 0 | 6464.74 |
| 4 | 178 | 0 | 0 | 6545.99 |
| 5 | 178 | 0 | 0 | 6464.74 |
| NoOpt | 177.33 | 0 | 0.67 | 6453.14 |
| NoIns | 178 | 0 | 0 | 6720.36 |

**Case 2**
The results from Case 2 are very different compared to Case 1. In the results (Table 5.5) we see that all customers are accepted and all orders planned in each procedure. The only exception is the Procedure NoOpt, which further lets us believe that no optimization can indeed be seen as a lower bound to the objective of maximizing planned orders. Without an optimization algorithm, the PlanCosts are never lower than the PlanCosts of the same system that does use an optimization algorithm periodically, therefore the PlanCosts without the use of an optimization algorithm tend to be higher on average, which means the delivery schedule is less efficient and a less efficient delivery schedule has less room to plan extra customers.

The PlanCosts show a larger gap between the various procedures. Procedures 1, 3 and 5 produce delivery schedules with the lowest average PlanCosts which are quite a bit below the other procedures. Even more interesting to see is that Procedure 3 performs better than Procedure 1. This is unexpected, as explained, Procedure 1 can take full advantage of the optimization algorithm, where Procedure 3 has to deal with customers arriving during the optimization run. In Section 5.2.2 we will find a possible explanation in the shifting of arriving customers. Next to that, Procedure 3 deals with new customers this by inserting the new orders into the optimized schedule. As all orders can be planned quite easily in this case, this insertion never fails. This also explains why Procedure 5 produces the same results, as there is no difference between 3 and 5 if the insertion step never fails.

### 5.2.2. Large cases
Cases 3, 4 and 5 have a relatively large set of customers, these cases are all from the same division but for different days. The results for these cases can be found in Tables 5.6, 5.7 and 5.8.

Looking at the larger instances, the difference between the procedures becomes more clear. Procedures 1, 3 and 5 have more planned orders in the final delivery schedule compared to the other procedures, where Procedure 1 always has the most. Whenever each of these three procedures reaches the highest number of planned customers, which happened in Case 2 and now also in Case 4, we do see that Procedures 3 and 5 have lower PlanCosts than Procedure 1.

Procedure NoIns also reaches a high number of planned orders, but this comes with the cost of a large number of unplanned orders, which makes the delivery schedules created by this procedure infeasible and thus not usable in practice.

**Table 5.6:** Case 3: 1295 total orders, 114 available vehicles, results from 1 simulation.

| Procedure | Planned orders | Unplanned orders | Customers left | PlanCosts |
|---|---|---|---|---|
| 1 | 1233 | 0 | 62 | 47583.40 |
| 2 | 1009 | 0 | 286 | 42338.80 |
| 3 | 1223 | 0 | 72 | 47429.27 |
| 4 | 1001 | 0 | 294 | 40916.82 |
| 5 | 1223 | 0 | 72 | 47429.27 |
| NoOpt | 920 | 0 | 375 | 38086.38 |
| NoIns | 1230 | 33 | 32 | 47477.68 |

**Table 5.7:** Case 4: 1114 total orders, 99 available vehicles, results from 1 simulation.

| Procedure | Planned customers | Unplanned customers | Customers left | PlanCosts |
|---|---|---|---|---|
| 1 | 1049 | 1 | 64 | 41221.20 |
| 2 | 911 | 0 | 203 | 38726.65 |
| 3 | 1049 | 1 | 64 | 41084.51 |
| 4 | 895 | 0 | 219 | 37076.62 |
| 5 | 1049 | 1 | 64 | 41084.51 |
| NoOpt | 806 | 0 | 308 | 34302.67 |
| NoIns | 1046 | 22 | 46 | 41090.34 |

**Table 5.8:** Case 5: 1633 total orders, 141 available vehicles, results from 1 simulation.

| Procedure | Planned customers | Unplanned customers | Customers left | PlanCosts |
|---|---|---|---|---|
| 1 | 1592 | 1 | 40 | 58617.44 |
| 2 | 1366 | 0 | 267 | 52689.61 |
| 3 | 1590 | 0 | 43 | 58800.27 |
| 4 | 1390 | 0 | 243 | 53369.52 |
| 5 | 1590 | 0 | 43 | 58800.27 |

The results of Procedures 3 and 5 are identical in each of the three large cases. However, this does not mean that there was no difference between the procedures. Recall that these are the procedures that try to insert new orders into the optimized schedule $B$. If this fails Procedure 3 uses the old schedule $A$ and Procedure 5 uses the merge algorithm. There are occasions in which the insertion step failed and Procedure 5 produced a slightly better intermediate delivery schedule with lower PlanCosts than Procedure 3. However, this improvement is so marginal, that it does not affect the final delivery schedule in any way. An example of this can be found in Table 5.12 which we will discuss in the next section.

Some procedures have unplanned orders in the final delivery schedule. We see this happening for the three best procedures in Case 4 and in Procedure 1 for case 5. In theory, this should not be possible as the procedures ware made to prevent this. However, the cause of these unplanned orders lies within the used systems. When a customer requests a time slot offer the Timeslotting service determines all insertion points of the customer into the current routing schedule and creates a time slot offer based on these insertion points. However, the routing schedule after inserting the customer gets created by the CVRS service. Due to small differences in the way these services calculate things, like travel distance, it is possible that CVRS finds that the order of the customer can not be planned in the place the Timeslotting service found when giving out the time slot offer. This causes the order to be unplanned in the delivery schedule when using CVRS. The optimization algorithm may fix this problem by finding a different spot for the order to be planned in, but when this is not the case the final delivery schedule can have several unplanned orders.

**Intermediate results**

To get a better understanding of the behavior of all procedures we look at the intermediate results. Figure 5.3 shows an example of the timeline of Case 3, where each line represents a different procedure. To indicate the timeline we use the number of arrived customers. Note that not all procedures plan have the same number of planned orders, which means that this graph does not show directly how good or bad a certain procedure is in comparison to another.

The plot gives a good idea of the different behaviors of the various procedures. Whenever a customer gets accepted and inserted into the delivery schedule the PlanCosts will always increase, which explains the steady climbing trend in the graph. Every time the optimization algorithm is executed we might find a schedule with lower PlanCosts, which is shown as a decline in the graph. For example, we can see that this happens only at the very end with Procedure NoOpt, as no optimization algorithm is used during the process with this method.

We see that Procedure 2 has very few declines. Often during an optimization run, there is at least

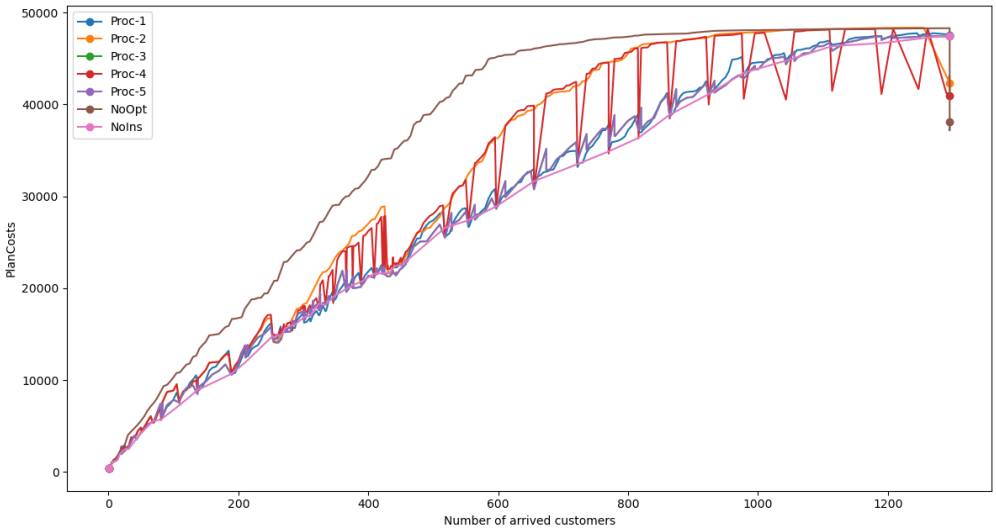



**Figure 5.3:** The PlanCosts progression of the delivery schedule in DTSM of case 3.

one customer which arrived during this optimization. When this happens, the optimized schedule is not used by Procedure 2. The times that no customer arrived during the optimization and after the cut-off time, we see that there is a drastic decrease in PlanCosts as the system now continues with an optimized schedule. We found that Procedure 2 accepts fewer customers than the other procedures, a possible cause can be that the PlanCosts of the current routing schedule is often high in comparison to other procedures, which means the delivery schedule is less efficient. As a consequence, there is less room to plan extra orders and thus accept new customers.

Procedure 4 shows interesting behavior. The optimization algorithm often finds solutions with low PlanCosts, which is shown by the steep declines in the graph. However, after the merge algorithm is used the PlanCosts immediately shoot back up again in most cases, which shows that the merged schedule is often not that efficient. In fact, the merged schedule is often very similar to the not optimized schedule, which can be seen by the fact that it follows the line of Procedure 2 which often uses no optimizer. The merge algorithm performs in this way when a lot of the vehicles are connected, as explained in Section 3.1.4.

At the end of the simulation using Procedures 2 and 4, the optimization algorithm finds schedules with PlanCosts that are a lot lower than with the other procedures, this is because at this point the DTSM system has fewer accepted customers using these procedures compared to others, which makes it easier to create an efficient delivery schedule.

For Procedures 1, 3 and 5 we see the same trend. PlanCosts increase with the insertion of customers, they decrease when the optimization algorithm finishes and go quickly back up again as the procedures deal with customers that arrived during the optimization. Procedure 3 and 5 are seemingly equal in the graph, as the improvements over Procedure 3 found by using the merge algorithm in Procedure 5 are so small that they are not visible. It is interesting to see the difference between Procedures 1 and 3. Procedure 1 delays customers whenever they arrive during optimization, which means that after the optimization finishes more customers need to be handled than with the other procedures. Procedure 3 does not delay the customers, which we see in the fact that the PlanCosts rise higher before an optimization algorithm is executed. Both procedures produce good delivery schedules after an optimization, but because the peaks in Procedure 1 get delayed a bit the final PlanCosts end up being a little higher than with Procedure 3.

Tables 5.9 and 5.10 show delivery schedules of about halfway through the DTSM system. We can clearly see that the procedures which have accepted the most customers in the final delivery schedule

**Table 5.9:** Case 3: Intermediate results after 630 customers.

| Procedure | Planned customers | Unplanned customers | Customers left | PlanCosts |
|---|---|---|---|---|
| 1 | 630 | 0 | 0 | 31459.71 |
| 2 | 629 | 1 | 0 | 38709.17 |
| 3 | 630 | 0 | 0 | 31476.17 |
| 4 | 630 | 0 | 0 | 39268.32 |
| 5 | 630 | 0 | 0 | 31476.17 |

**Table 5.10:** Case 4: Intermediate results after 520 customers.

| Procedure | Planned customers | Unplanned customers | Customers left | PlanCosts |
|---|---|---|---|---|
| 1 | 520 | 0 | 0 | 27639.13 |
| 2 | 519 | 1 | 0 | 28250.04 |
| 3 | 520 | 0 | 0 | 25962.86 |
| 4 | 520 | 0 | 0 | 27841.88 |
| 5 | 520 | 0 | 0 | 25962.86 |

now have the lowest PlanCosts in the intermediate results. This supports the idea that using the optimization algorithm efficiently to create a better delivery schedule helps for finding better time slot offers for customers which leads to more planned orders in the end.

**Comparison between Procedures 3 and 5**
Procedures 3 and 5 often produce the same final results, but during the simulation they do perform differently. Recall that both procedures try to insert the new customers into the optimized schedule $B$ to create a new schedule $B'$. If this fails Procedure 3 will just use the old schedule $A$ and Procedure 5 will use the merge algorithm (Algorithm 5) to merge schedules $A$ and $B'$.

A good example of these differences can be found in the simulation of case 3. Before customer 1115 arrives the optimization algorithm starts. The optimization finishes and produces the delivery schedule with the results as found in Table 5.11.

**Table 5.11:** Case 3: Output of an optimization run with 1114 customers.

| Delivery schedule | Planned customers | Unplanned customers | Customers left | PlanCosts |
|---|---|---|---|---|
| $B$ | 1106 | 0 | 8 | 45954.33 |

During this optimization run, twenty more customers arrived and requested a time slot offer. Three of these customers did not pick a time slot and place an order, the others did. Each time, the customers were inserted in the current delivery schedule $A$. This produced the current delivery schedule $A$ of which the associated results are shown in Table 5.12. At this point, the different procedures would be used to find the delivery schedule with which we can continue the DTSM process.

In the case of Procedures 3 and 5, the seventeen new customers are inserted in the optimized schedule $B$ to create the new schedule $B'$, which can also be found in Table 5.12. As we can see, one order did not fit and remains unplanned. Procedure 5 would now execute the merge algorithm to merge schedules $A$ and $B'$ while Procedure 3 would continue using delivery schedule $A$. Both delivery schedules $A$ and the merged schedule are very similar, but the merged schedule has slightly lower PlanCosts and is a bit more efficient. This means that you would expect that Procedure 5 performs better overall. However, we have seen in the final results of case 3 (Table 5.6) that this is not the case. The increase in efficiency of the merged schedule is so small that it makes no difference in the offering of time slots for the next customers, which results in the same accepted customers with the same chosen time slots and thus the same final delivery schedule. So even though using the merge algorithm in this case creates more efficient routes, it does not have an impact.

**Table 5.12:** Case 3: Results after 1134 arrived customers.

| Delivery schedule | Planned customers | Unplanned customers | Customers left | PlanCosts |
|---|---|---|---|---|
| $A$ | 1123 | 0 | 11 | 47030.25 |
| $B'$ | 1122 | 1 | 11 | 46202.24 |
| Merged | 1123 | 0 | 11 | 47006.93 |

**Table 5.13:** Percentage of choices made by all accepted customers using the data of all simulations.

| Procedure | 1st time slot chosen | 2nd time slot chosen | 3th time slot chosen |
|---|---|---|---|
| 1 | 83.6% | 12.2% | 2.6% |
| 2 | 81.1% | 12.7% | 6.1% |
| 3 | 83.6% | 12.3% | 4.1% |
| 4 | 81.8% | 13.2% | 5.0% |
| 5 | 84.2% | 12.9% | 2.9% |
| NoOpt | 76.1% | 16.0% | 7.9% |
| NoIns | 85.3% | 12.1% | 2.6% |

### 5.2.3. Customers choices

Each customer has an ordered set of three preferred time slots from which they pick the first that is available in a time slot offer. Table 5.13 shows the time slots the customers picked from the time slot offer they received from the DTSM system. This excludes all customers that did not pick a time slot and left. The results are as expected. The Procedures 1, 3 and 5 that accept the most customers offer more time slots, thus the customers are more often able to pick the first choice from their list of preferred time slots. Otherwise, the table does not show a very significant difference between the different procedures.

### 5.2.4. Merge algorithm in Procedure 4

The merge algorithm (Algorithm 5), that is used in Procedures 4 and 5 combines an 'old' delivery schedule $A$ and an optimized delivery schedule $B$. The algorithm tries to use as many routes from the optimized schedule $B$, but to create a feasible delivery schedule it can be restricted to use some of the routes from schedule $A$. When a lot of vehicles are connected in the two delivery schedules these restrictions could mean that we completely use delivery schedule $A$. On the other hand, when no or very few vehicles are connected, almost all routes from schedule $B$ can be used. Whenever there were no orders during the optimization run, all routes from the optimized schedule $B$ will be used.

Procedure 4 uses this algorithm after each optimization run. Procedure 5 does this only when it failed to insert the new customers into the optimized delivery schedule $B$ and we have seen does not happen often.

We mapped the behavior of the merge algorithm in Procedure 4 of the simulations in Table 5.14. For this we look at the overall number of routes that are used from schedule $A$ and $B$, this is not including empty routes. We also look at how many times a complete schedule $A$ or $B$ is used and how many times a mix of the two schedules was made.

It seems that the more customers get accepted, the harder it is for the merge algorithm to find a good delivery schedule. In Case 1 there are no more than 104 customers at most. It happens that with those amounts, schedules $A$ and $B$ have relatively few vehicles that are connected, which makes it possible to use a lot of the routes from vehicle $B$. As the number of customers increases, more vehicles are connected between schedules $A$ and $B$. Furthermore, the frequency of arriving customers increases, which means that more customers are arriving during an optimization run. These customers can be placed in different routes in schedule $A$. The merge algorithm at least needs to use all these routes from schedule $A$ to create a feasible delivery schedule.

This means that for cases 3, 4 and 5 we do have moments where all vehicles are connected to a route which includes a new order and all routes from schedule $A$ need to be used. Note that for these cases the routes from schedule $A$ are used more often than the routes from schedule $B$. At the same

**Table 5.14:** The results of the merge algorithm for each case. Routes schedule $A$ and Routes schedule $B$ detail the percentage of routes that was used from schedule $A$ or $B$ to form the merged schedule. Complete $A$ and Complete $B$ detail the percentage of times the merged schedule was equal to respectively schedule $A$ or $B$ after merging and Mixed schedules is the percentage of times the merged schedule was a mix of both $A$ and $B$.

| Case | Routes schedule $A$ | Routes schedule $B$ | Complete $A$ | Mixed schedules | Complete $B$ |
|---|---|---|---|---|---|
| 1 | 12% | 88% | 0% | 33% | 67% |
| 2 | 39% | 61% | 3% | 43% | 54% |
| 3 | 73% | 27% | 19% | 33% | 46% |
| 4 | 71% | 29% | 17% | 32% | 51% |
| 5 | 80% | 20% | 30% | 35% | 35% |

time we still see that a high percentage consists of merged schedules that completely uses the routes from schedule $B$, which seems contradictory. However, this happens because in the beginning of a simulation not all routes are filled with orders and the full schedule $B$ is often used when there is a small number of orders. This means that a relatively low number of routes from schedule $B$ get used if we look at the end of the simulation where all routes are in use.

# 6

# Conclusions and discussion

In this chapter, we will summarize our computational experiments and give an answer to the research question: What procedures can be used to handle customers that arrive during the optimization phase of DTSM and what is the impact of these procedures on the final delivery schedule? Afterward, we will discuss the limitations of the research and we give recommendations for further research.

## 6.1. Conclusions

In this thesis, we introduced a problem regarding the real-time aspect of Dynamic Time Slot Management. Customers that request a time slot offer while the optimization algorithm is running produce orders with time slots that are based on the delivery schedule in memory, which is not the optimized delivery schedule. After the optimization algorithm finishes, these new orders are not included in the optimized delivery schedule.

We have proposed five different procedures to deal with the problem. The goal of these procedures is to create a delivery schedule that can be used to carry on the DTSM process. The procedures were created in such a way that they always produce a feasible delivery schedule with no unplanned orders. With our computational experiments, we tried to demonstrate the impact of using a certain procedure. The main objective for this was the number of accepted customers in the DTSM system, the secondary objective the costs of the delivery schedule.

First of all, we have seen that using an optimization algorithm in a DTSM system with a simple insertion method drastically improves the final delivery schedule. All procedures perform significantly better than a simple DTSM system which uses no optimization algorithm. We also compared the procedures against a DTSM system that does not use an insertion method. Even though this provided a delivery schedule with a high number of planned customers and decent costs, we found that it also accepts customers which remain unplanned, resulting in an infeasible delivery schedule.

Procedures that try to use the optimization algorithm as much as possible provide a better final delivery schedule than procedures that do not. Procedure 2 does not use the optimized schedule when customers arrived during its creation. As a consequence, the PlanCosts of the schedule that is kept in memory during the DTSM process is often higher than with other procedures. We have seen that this causes fewer time slots to be offered which eventually leads to fewer accepted customers.

The merge algorithm of Procedure 4 showed the same problem. When the number of orders in the delivery schedule increases the number of customers that arrive during optimization increases and vehicles become more connected when comparing the old schedule $A$ to the optimized schedule $B$. Together these two things causes the merge algorithm to use mostly routes from schedule $A$ and this is similar to not using an optimization algorithm at all.

Procedure 1 provides the best results of all the tested procedures. It does this by mimicking the theoretical setting. Customers have to wait before they get receive a time slot offer if the optimization

algorithm is running. This procedure has the major downside that it provides poor customer service and most retailers would not want to use such a system.

Procedure 3 does not suffer from this downside and provides similar results to Procedure 1. By trying to insert customers that arrived during an optimization run into the optimized schedule, Procedure 1 finds efficient delivery schedules throughout the DTSM process. By using the delays on customers, Procedure 1 also shifts the peaks of arriving customers slightly over the timeline, which causes Procedure 3 to find even lower costs for the delivery schedule in cases where the same number of customers are accepted.

Procedure 5 expands on Procedure 3 using the merge algorithm whenever the insertion of new customers into the optimized delivery schedule fails. We see that this procedure does provide slightly better results than Procedure 3 in small cases. However, for the larger cases the results are equal. We found that during the DTSM process, the insertion rarely fails, which means that the merge algorithm is rarely utilized. Whenever insertion does fail in the large cases, the improvements that the merge algorithm found over using the old delivery schedule were only marginal, which causes the final delivery schedule to be equal to the one produced by Procedure 3, using no merge algorithm.

## 6.2. Limitations

We based our conclusions on the computational experiments done in this thesis. To perform these computational experiments we have done assumptions. These assumptions and other limitations will be discussed in this section.

The optimization algorithms used in this study are mostly based on heuristics. As is the nature of a heuristic, we cannot be certain that a found solution is also the optimal solution. We use the found solutions to analyze which procedures performed the best. The optimization algorithm may find find worse solutions for a certain procedure than another, without the influence of the procedure itself. However, each heuristic is used multiple times within one simulation for a certain procedure, which means that it is less likely that one less optimal optimization run results in a bad final solution. To further mitigate this problem we also did some simulations multiple times.

The created mathematical model and the used cases for the simulation tailor to a very specific problem and we used the data and specific inputs that apply to Retailer A. While we do think that the conclusions found in this thesis apply to a broader set of problems, we cannot be certain that this is the case.

There is a discontinuity between the cloud services of ORTEC. This results in orders being unplanned in a simulation while this in theory should not be possible. This could be an influence on the final results.

We have not considered the calculation time that each procedure will take. This calculation time could become an issue in practice as customers could arrive during the execution of a certain procedure if this takes a long time. However, we do believe that the computation time is negligible and it will not have a heavy impact on the final results if it were to be considered.

## 6.3. Recommendations for future research

Dynamic Time Slot Management is a fairly new subject and many aspects can still be researched. This thesis also raises new questions that could be answered in future research. We will list suggestions for further research that could allow for even more efficient DTSM systems in the future.

- Test how the different procedures interact with different optimization algorithms and different frequencies of use of the optimization algorithm
- Test new procedures that deal with customers that arrive during the optimization algorithm
- Perform the simulations in this thesis with a data set of a different retailer
- Explore the computation time of the different procedures and the impact that may have
- Allow for simultaneous arriving customers and customers arriving during optimization at the same time

# Bibliography


Agatz, N., Campbell, A. M., Fleischmann, M., and Savelsbergh, M. W. P. (2011). Time Slot Management in Attended Home Delivery. *Transportation Science*, 45(3):435–449.

Agatz, N., Campbell, A. M., Fleischmann, M., van Nunen, J., and Savelsbergh, M. W. P. (2013). Revenue management opportunities for Internet retailers. *Journal of Revenue and Pricing Management*, 12(2):128–138.

Ahuja, R. K., Ergun, z., Orlin, J. B., and Punnen, A. P. (2002). A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102.

Baldacci, R., Mingozzi, A., and Roberti, R. (2012). Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, 218(1):1–6.

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

Breedam, A. V. (1994). *An Analysis of the Behavior of Heuristics for the Vehicle Routing Problem for a selection of problems with Vehicle-related, Customer-related, and Time-related Constraints*. PhD thesis, Antwerp Management School.

Campbell, A. M. and Savelsbergh, M. W. P. (2005). Decision Support for Consumer Direct Grocery Initiatives. *Transportation Science*, 39(3):313–327.

Campbell, A. M. and Savelsbergh, M. W. P. (2006). Incentive Schemes for Attended Home Delivery Services. *Transportation Science*, 40(3):327–341.

Dantzig, G. B. and Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science*, 6(1):80–91.

Desaulniers, G., Madsen, O., and Røpke, S. (2014). *The Vehicle Routing Problem with Time Windows*, pages 119–160. Society for Industrial and Applied Mathematics, 2 edition.

Ehmke, J. F. and Campbell, A. M. (2014). Customer acceptance mechanisms for home deliveries in metropolitan areas. *European Journal of Operational Research*, 233(1):193–207.

Flood, M. M. (1955). The Traveling-Salesman Problem. *Operations Research*, 4(1):61–75.

Irnich, S., Toth, P., and Vigo, D. (2014). *The Family of Vehicle Routing Problems*, pages 1–23. Number 18 in MOS-SIAM Series on Optimization. SIAM.

Jabali, O., Leus, R., Van Woensel, T., and de Kok, T. (2015). Self-imposed time windows in vehicle routing problems. *OR Spectrum*, 37(2):331–352.

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

Köhler, C., Ehmke, J. F., and Campbell, A. M. (2019). Flexible time window management for attended home deliveries. *Omega*, 91:102023.

Klein, R., Mackert, J., Neugebauer, M., and Steinhardt, C. (2018). A model-based approximation of opportunity cost for dynamic pricing in attended home delivery. *OR Spectrum*, 40(4):969–996.

Klein, R., Neugebauer, M., Ratkovitch, D., and Steinhardt, C. (2019). Differentiated Time Slot Pricing Under Routing Considerations in Attended Home Delivery. *Transportation Science*, 53(1):236–255.

Kritikos, M. N. and Ioannou, G. (2013). The heterogeneous fleet vehicle routing problem with overloads and time windows. *International Journal of Production Economics*, 144(1):68–75.

Labadie, N., Prins, C., and Prodhon, C. (2016). *Metaheuristics for Vehicle Routing Problems*.

Lenstra, J. and Kan, A. (2006). Complexity of vehicle routing and scheduling problems. *Networks*, 11:221 – 227.

Lin, I. and Mahmassani, H. (2002). Can online grocers deliver?: Some logistics considerations. *Transportation Research Record Journal of the Transportation Research Board*, 1817:17–24.

Masse, M. (2011). *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. O'Reilly Media.

Mkansi, M., Eresia-Eke, C., and Emmanuel-Ebikake, O. (2018). E-grocery challenges and remedies: Global market leaders perspective. *Cogent Business & Management*, 5(1).

ORTEC (2019). Introduction training cvrs 2019.

Paessens, H. (1988). The savings algorithm for the vehicle routing problem. *European Journal of Operational Research*, 34(3):336–344.

Papadimitriou, C. H. (1977). The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237 – 244.

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

Ramaekers, K., Caris, A., Moons, S., and van Gils, T. (2018). Using an integrated order picking-vehicle routing problem to study the impact of delivery time windows in e-commerce. *European Transport Research Review*, 10(2):56.

Renaud, J. and Boctor, F. F. (2002). A sweep-based algorithm for the fleet size and mix vehicle routing problem. *European Journal of Operational Research*, 140(3):618–628.

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.

Spliet, R. and Gabor, A. F. (2015). The Time Window Assignment Vehicle Routing Problem. *Transportation Science*, 49(4):721–731.

van der Sar, E. (2018). Multi-attribute vehicle routing problems: Exploring the limits of exact methods in practice. Master's thesis, Delft University of Technology.

Visser, T. R., Agatz, N., and Spliet, R. (2019a). Simultaneous Customer Interaction in Online Booking Systems for Attended Home Delivery. *SSRN Electronic Journal*.

Visser, T. R., Agatz, N., and Spliet, R. (2019b). *When microseconds add up: On the real-time performance of Dynamic Time Slot Management*. PhD thesis, Erasmus University Rotterdam.

Visser, T. R. and Spliet, R. (2017). Efficient Move Evaluations for Time-Dependent Vehicle Routing Problems. page 25.

Yang, X., Strauss, A. K., Currie, C. S. M., and Eglese, R. (2016). Choice-Based Demand Management and Vehicle Routing in E-Fulfillment. *Transportation Science*, 50(2):473–488.

# A

# Pseudo code for the optimization algorithm

In this appendix, we detail the optimization algorithm that is used in the Dynamic Time Slot Management system in our computational experiments. To make it understandable the algorithm is subdivided in many different parts. This appendix contains the pseudo-code for all of these parts.

The starting input consists of a set of all orders that need to be planned and a delivery schedule with all the routes, these routes can already be filled with some of the orders. We assume that we have the information for all the orders and vehicles of the routes as described in Section 2.2.1.

## A.1. Main algorithms

The first algorithm describes the full optimization algorithm. The objectives of the optimization are defined and it uses the schedule as an input to generate a new and possible better delivery schedule. To do this it calls upon multiple different algorithms.

**Algorithm 6:** Optimization algorithm

**Input:** The delivery schedule $S$, the set of unplanned orders $O$

**Objectives:** The objectives are in lexigraphical order, starting with the most important one:
1. Maximize number of planned orders
2. Minimize PlanCosts (Contains Total travel time, Total distance and number of used routes)
3. Minimize total route durations
4. Minimize total travel time
5. Minimize waiting time
6. Minimize total distance
7. Minimize number of used routes

**Algorithm** *7* `Construction(`*S,O*`)`

**Algorithm** *8* `LocalSearch(`*S, i = 200 iterations, WithinTrip = false*`)`

**Algorithm** *12* `ParallelCheapestInsertion(`*S,O*`)`

**for** *4* iterations **do**
  **Algorithm** *9* `RuinAndRecreate(`*S, P=10%*`)`
  **Algorithm** *9* `RuinAndRecreate(`*S, P=15%*`)`
**end**

**Algorithm** *11* `MultiTripRemoval(`*S*`)`

**Algorithm** *8* `LocalSearch(`*S, i = 200 iterations, WithinTrip = true*`)`

The construction algorithm describes the construction phase of the optimization. In this phase all orders that are not yet included in a route are added to the schedule if possible. If routes are still empty, seed tasks are selected first to insert to populate the routes after which the orders are inserted.

**Algorithm 7:** Construction

**Input:** The delivery schedule $S$, the set of unplanned orders $O$

$R$ = all routes of $S$ that contain at least one order
**Algorithm** *12* `ParallelCheapestInsertion(`*S,R,O*`)`

$R$ = all routes of $S$ that contain no orders
**Algorithm** *13* `SequentialInsertion(`*S,R,O*`)`

In multiple phases of the optimization a local search step is executed. In this local search step multiple different local search algorithms are used to improve the solution, by finding better solutions in a nearby neighborhood of the current solution. This can be done by replacing orders over the whole solution or only within the routes.

**Algorithm 8:** LocalSearch

**Input:** The delivery schedule $S$, number of iterations $i$, boolean $WithinTrip$

Estimators = $\{Drivingtime, Distance\}$
$N$ = Neigborhood size = 50 closest orders
**foreach** $E \in$ *Estimators* **do**
    **Algorithm** *15* `2-Opt(`$S, N, i, E$`)`
    **Algorithm** *16* `CrossExchange(`$S, N, i, E$`)`
    **Algorithm** *17* `Move(`$S, N, i, E, n = 1, WithinTrip$`)`
    **Algorithm** *18* `Swap(`$S, N, i, E, n = 1, WithinTrip$`)`
    $n = 0.05$ times the amount of orders in $S$
    **Algorithm** *17* `Move(`$S, N, i, E, n, WithinTrip$`)`
**end**

## A.2. Ruin and recreate algorithms

The Ruin and Recreate algorithm is describes the large neighborhood search phase of the optimization. A roulette wheel selects a ruin and a recreate method to form a pair each time a step in the algorithm is executed. In this way a part of the solution is destroyed and rebuilt to form new solutions, with the goal to find an improved solution. Dependent on the success of pair of ruin and recreate methods the pair is used more or less often by the roulette wheel.

**Algorithm 9:** RuinAndRecreate

**Input:** The delivery schedule $S$, the unplanned orders $O$, the percentage of orders to remove $P$

**RouletteWheel:** A combination of one Ruin and one Recreate method is picked.
- Ruin methods:
  1. `RandomRemoval()`
  2. `RelatedRemoval()`
  3. `WorstRemoval()`
  4. `RandomClusterRemoval()`
  5. `WorstClusterRemoval()`
  6. `TripRemoval()`
- Recreate methods:
  1. `ParallelRegretInsertion()`
  2. `SequentialInsertion()`

Set the probabilities of the RouletteWheel such that each combination has equal change to be chosen

**for** *50 iterations* **do**
- Create a copy $S'$ of $S$
- Use RouletteWheel to find a combination of a `Ruin()` and `Recreate()` method
- **Algorithm** *10* `RuinAndRecreateStep(`$S'$`,`$P$`, Ruin(), Recreate(),`$O$`)`
- **if** $C(S') < C(S)$ **then**
  - Increase the probability of this combination of methods being chosen by the RouletteWheel
  - Adjust the probabilities of the other combination of methods
  - Replace $S$ with $S'$
- **end**
- **else**
  - Decrease the probability of this combination of methods being chosen by the RouletteWheel
  - Adjust the probabilities of the other combination of methods
- **end**

**end**

In one Ruin and Recreate step the solution is destroyed by the selected ruin method and rebuilt by the recreate method. After that a quick local search step is included to intensify the search for a better solution.

**Algorithm 10:** RuinAndRecreateStep

**Input:** The delivery schedule $S$, a percentage of orders to remove $P$, a ruin method $Ruin(S,P)$, a recreate method $Recreate(S,O)$, the set of unplanned orders $O$

Use the method `Ruin(`$S,P$`)` to remove orders from $S$
Use the method `Recreate(`$S,O$`)` to insert orders in $S$
**Algorithm** *8* `LocalSearch(`$S, i = 50$ *iterations*, $WithinTrip = false$`)`

The Multi Trip Removal algorithm is a Ruin and Recreate algorithm which is always executed with the same set pairs of ruin and recreate methods, which were not part of the roulette wheel. The ruin methods in this version remove a complete route or multiple routes, after which the solution is rebuilt again using a recreate method.

**Algorithm 11:** MultiTripRemoval

**Input:** The delivery schedule $S$, the unplanned orders $O$

Create a copy $S'$ of $S$
Select a random route of $S$ and remove all orders from this route
**Algorithm** *12* `ParallelCheapestInsertion`$(S', O)$
**if** $C(S') < 1.05 \cdot C(S)$ **then**
  **Algorithm** *8* `LocalSearch`(*S, i = 200 iterations, WithinTrip = false*)
  **if** $C(S') < 1.05 \cdot C(S)$ **then**
    Replace $S$ with $S'$
  **end**
**end**
Create a copy $S'$ of $S$
Select a random route of $S$ and remove all orders from this route
**Algorithm** *14* `ParallelRegretInsertion`$(S', O)$
**if** $C(S') < 1.05 \cdot C(S)$ **then**
  **Algorithm** *8* `LocalSearch`(*S, i = 200 iterations, WithinTrip = false*)
  **if** $C(S') < 1.05 \cdot C(S)$ **then**
    Replace $S$ with $S'$
  **end**
**end**
Create a copy $S'$ of $S$
Select 2 random routes of $S$ and remove all orders from these routes
**Algorithm** *12* `ParallelCheapestInsertion`$(S', O)$
**if** $C(S') < 1.05 \cdot C(S)$ **then**
  **Algorithm** *8* `LocalSearch`(*S, i = 200 iterations, WithinTrip = false*)
  **if** $C(S') < 1.05 \cdot C(S)$ **then**
    Replace $S$ with $S'$
  **end**
**end**
Create a copy $S'$ of $S$
Select 2 random routes of $S$ and remove all orders from these routes
**Algorithm** *14* `ParallelRegretInsertion`$(S', O)$
**if** $C(S') < 1.05 \cdot C(S)$ **then**
  **Algorithm** *8* `LocalSearch`(*S, i = 200 iterations, WithinTrip = false*)
  **if** $C(S') < 1.05 \cdot C(S)$ **then**
    Replace $S$ with $S'$
  **end**
**end**

## A.3. Insertion Algorithms

The Parallel Cheapest Insertion algorithm is a well known route optimization algorithm which is used to assign unplanned orders to different vehicles. The orders are inserted into the routes according to the increase in plan costs this insertion causes, starting with the order that increases the plan costs the least.

**Algorithm 12:** ParallelCheapestInsertion

**Input:** The delivery schedule $S$, the set of routes to consider $R$, the set of unplanned orders $O$

```
while O is not empty do
    foreach order o ∈ O do
        foreach route ρ in schedule R do
            Check if order o can be inserted in the sequence of orders of ρ
            Calculate the increase in objective value of inserting the order in each possible insertion
             point
        end
    end
    if an order has a feasible insertion point then
        Insert the order which has the lowest increase in objective value in S
    end
    else
        break
    end
end
```

The Sequential Insertion algorithm is a construction method which inserts orders into different routes in a specific sequence. First the routes are sorted, after which the first route is selected. Then a seed order is selected and this order is added to the selected route. Based on this seed order, as much orders as possible are added to the route, after which we move on to the next route. The variables used in the description of this algorithm are all detailed in Chapter 2.

**Algorithm 13:** SequentialInsertion

**Input:** The delivery schedule $S$, the set of routes to consider $R$, the set of unplanned orders $O$

**Remarks:** All sorting criteria are considered in lexicographical order.

**Sort** $R$ **by the criteria:**
1. increasing order: Earliest start time of the route $\varepsilon^k$
2. decreasing order: Time window duration $\lambda^k - \varepsilon^k$
3. decreasing order: Resource capacity $Q_k$

**end**

**foreach** *route* $\rho$ *in* $R$ **do**
- **Sort** $O$ **by the criteria:**
  1. increasing order: End of time slot $b_i$
  2. decreasing order: Distance from depot to order $d_{D,i}$
  3. decreasing order: Quantity of the order $q_i$
- **end**
- Insert the first order $o_{seed} \in O$ into route $\rho$ as the seed order
- **Sort** $O$ **by the criteria:**
  1. increasing order: Distance to the seed order in steps of 5 kilometers
  2. decreasing order: End of time slot $b_i$
  3. decreasing order: Distance to the seed order $d_{i,o_{seed}}$
  4. decreasing order: Quantity of the order $q_i$
- **end**
- **foreach** *order* $o \in O$ **do**
  - Insert order $o$ in $\rho$
  - **if** *Insertion failed* **then**
    - Break
  - **end**
- **end**

**end**

The Parallel Regret Insertion algorithm is a construction method similar to the parallel cheapest insertion algorithm. Only in this case we consider the best insertion point and second best insertion point for each order. The difference between this is the regret value. Orders are added in sequence, starting with the order with the highest regret value.

**Algorithm 14:** ParallelRegretInsertion

**Input:** The delivery schedule $S$, the set of unplanned orders $O$

```
while O is not empty do
    foreach order o ∈ O do
        foreach route ρ in schedule R do
            Check if order o can be inserted in the sequence of orders of ρ
            Calculate the increase in objective value of inserting the order in each possible insertion
             point
        end
        Regretvalue = second lowest increase - lowest increase
    end
    Insert the order which has the highest regret value in S
    if Insertion failed then
        Break
    end
end
```

## A.4. Local Search and Removal Methods

The 2-opt algorithm is a local search method which swaps two edges with two different edges while maintaining feasible routes. This algorithm can quickly eliminate crossings in a solution, which are often not optimal.

**Algorithm 15:** 2-Opt

**Input:** The delivery schedule $S$, neigborhood size $N$, estimation variable $E$, iterations $i$

**Remarks:** We denote $C(x)$ as the function that determines the objective value of schedule $x$ or route $x$. We denote $E(x)$ as the function that determines the estimation value of schedule $x$ or route $x$ using the estimation variable.

**repeat**
- Only edges within the same neigborhood $N$ are selected
- Select a route $\rho$ from $S$ with order sequence $V(\rho) = \{0, 1, 2, \dots, n, 0\}$
- Select two edges $\{x, x+1\}, \{y, y+1\} \in \rho$ with $x < y$ and $x \neq y+1 \neq 0$
- Only edges within the same neigborhood $N$ are selected
- $r_{new} = \{0, 1, \dots, x, \quad y, y-1, y-2, \dots, x+1, \quad y+1, y+2, \dots, n, 0\}$
- create route $\rho_{new}$ with order sequence $r_{new}$
- **if** $E(\rho_{new}) < E(\rho)$ **then**
  - **if** $C(\rho_{new}) < C(\rho)$ **then**
    - replace $\rho$ with $\rho_{new}$
  - **end**
- **end**

**until** no improvement is found for $i$ iterations

The Cross Exchange algorithm selects two edges from a route and two edges from a different part of the route or a different route and exchanges the edges.

**Algorithm 16:** CrossExchange

**Input:** The delivery schedule $S$, neigborhood size $N$, estimation variable $E$, iterations $i$

**Remarks:** We denote $C(x)$ as the function that determines the objective value of schedule $x$ or route $x$. We denote $E(x)$ as the function that determines the estimation value of schedule $x$ or route $x$ using the estimation variable.

**repeat**
- select 2 different routes $\rho_1, \rho_2$ from $S$ with order sequence $V(\rho_1) = \{0, 1, 2, \dots, n, 0\}$ and $V(\rho_2) = \{0, n+1, n+2, \dots, n+m, 0\}$
- Only edges within the same neigborhood $N$ are selected
- select two edges $\{x_1, x_1+1\}, \{y_1, y_1+1\} \in \rho_1$ with $x_1 < y_1$ and $x_1 \neq y_1 + 1 \neq 0$
- select two edges $\{x_2, x_2+1\}, \{y_2, y_2+1\} \in \rho_2$ with $x_2 < y_2$ and $x_2 \neq y_2 + 1 \neq 0$
- $r_{new1} = \{0, 1, \dots, x_1, \quad x_2+1, x_2+2 \dots, y_2, \quad y_1+1, y_1+2, \dots, n, 0\}$
- $r_{new2} = \{0, 1, \dots, x_2, \quad x_1+1, x_1+2 \dots, y_1, \quad y_2+1, y_2+2, \dots, n, 0\}$
- create route $\rho_{new1}$ with order sequence $r_{new1}$ and route $\rho_{new2}$ with order sequence $r_{new2}$
- **if** $E(\rho_{new1}) + E(\rho_{new2}) < E(\rho_1) + E(\rho_2)$ **then**
  - **if** $C(\rho_{new1}) + C(\rho_{new2}) < C(\rho_1) + C(\rho_2)$ **then**
    - replace $\rho$ with $\rho_{new}$
  - **end**
- **end**

**until** no improvement is found for $i$ iterations

The Move algorithm simply moves a selection of orders which are in sequence somewhere in the delivery schedule to a different point in the delivery schedule. This can be in the same or in a different route.

**Algorithm 17:** Move

**Input:** The delivery schedule $S$, neigborhood size $N$, estimation variable $E$, iterations $i$, boolean $OnlyWithinTrip$, number of selected orders $n$

**Remarks:** We denote $C(x)$ as the function that determines the objective value of schedule $x$ or route $x$. We denote $E(x)$ as the function that determines the estimation value of schedule $x$ or route $x$ using the estimation variable. If the boolean $OnlyWithinTrip$ is set to 'true', the method will select in such ways that each orders stays in the same route.

**repeat**
- create a copy $S'$ of $S$
- randomly select a route $\rho$ from $S$ with order sequence $V(\rho) = \{0, 1, 2, \dots, n, 0\}$
- randomly select a maximum of $n$ orders $O$ from $\rho$ which are in sequence
- remove the orders in $O$ from $S'$
- randomly select a route $\rho_2$ in $S$
- insert the orders in $O$ in the same sequence in a random insertion point in $\rho_2$
- **if** $E(S') < E(S)$ **then**
  - **if** $C(S') < C(S)$ **then**
    - replace $S$ with $S'$
  - **end**
- **end**

**until** no improvement is found for $i$ iterations

The Swap algorithm exchanges two sections of routes in order to find a new solution. It is possible that these sections are from the same route.

**Algorithm 18:** Swap

**Input:** The delivery schedule $S$, neigborhood size $N$, estimation variable $E$, iterations $i$, boolean $OnlyWithinTrip$, number of selected orders $n$

**Remarks:** We denote $C(x)$ as the function that determines the objective value of schedule $x$ or route $x$. We denote $E(x)$ as the function that determines the estimation value of schedule $x$ or route $x$ using the estimation variable. If the boolean $OnlyWithinTrip$ is set to 'true', the method will select in such ways that each orders stays in the same route.

**repeat**
- create a copy $S'$ of $S$
- select route $\rho_1$ from $S'$
- select a maximum of $n$ orders $O_1$ from $\rho_1$ which are in sequence
- select a route $\rho_2$ from $S'$
- select a maximum of $n$ orders $O_2$ from $\rho_2$ which are in sequence and such that $O_1 \cap O_2 = \emptyset$
- swap sequences $O_1$ and $O_2$
- insert the orders in $O$ in the same sequence in a random insertion point in $\rho_2$
- **if** $E(S') < E(S)$ **then**
  - **if** $C(S') < C(S)$ **then**
    - replace $S$ with $S'$
  - **end**
- **end**

**until** no improvement is found for $i$ iterations

This following overview details the different ruin methods that can be used by the Ruin and Recreate algorithm. Each method selects an amount of orders in some way and removes these from the current solution.

**Algorithm 19:** Ruin methods

**Input:** A delivery schedule $S$, a percentage of orders to remove $P$, the amount of clusters to remove $r$.

**Notation:** $O$ represents all orders in schedule $S$ and $O_r$ the orders that have been removed. $O_l$ contains all orders that are not in $O_r$. $P_n = \lceil P|O| \rceil$ is the number of orders that need to be removed.

**Method** `RandomRemoval(S, P)`**:**
- $O_r = \{\}$
- **while** $|O_r| < P_n$ **do**
  - Select an order $o \in O_l$ at random
  - $O_r = O_r \cup \{o\}$
- **end**
- Remove all orders in $O_r$ from schedule $S$.

**End**

**Method** `RelatedRemoval(S, P)`**:**
- Select an order $o \in O$ at random
- $O_r = \{o\}$
- **while** $|O_r| < P_n$ **do**
  - Select an order $o \in O_r$ at random
  - Sort $O_l$ according to the distance to $o$. Select order $i$ that is closest to $o$. $O_r = O_r \cup \{i\}$
- **end**
- Remove all orders in $O_r$ from schedule $S$.

**End**

**Method** `WorstRemoval(S, P)`**:**
- **foreach** *order* $o$ *in* $O$ **do**
  - Calculate the decrease in objective value if $o$ is removed from $S$
- **end**
- Sort $O$ in descending order according to the calculated decrease in objective value
- Remove the first $P_n$ orders in $O$ from schedule $S$

**End**

**Method** `RandomClusterRemoval(S, P, r)`**:**
- $O_r = \{\}$
- **for** $r$ *times* **do**
  - Select an order $o \in O_l$ at random
  - Sort $O_l$ according to the distance to $o$
  - Select the first $\lceil P_n/r \rceil$ orders $I$ that are closest to $o$
  - $O_r = O_r \cup I$
- **end**
- Remove all orders in $O_r$ from schedule $S$.

**End**

**Method** `WorstClusterRemoval(S, P, r)`**:**
- **foreach** *order* $o$ *in* $O$ **do**
  - Calculate the decrease in objective value if $o$ is removed from $S$
- **end**
- Sort $O$ in descending order according to the calculated decrease in objective value
- $O_r = \{\}$
- **for** $r$ *times* **do**
  - Select the first order $o \in O$ that is also in $O_l$
  - Sort $O_l$ according to the distance to $o$
  - Select the first $\lceil P_n/r \rceil$ orders $I$ that are closest to $o$
  - $O_r = O_r \cup I$
- **end**
- Remove all orders in $O_r$ from schedule $S$.

**End**

**Method** `TripRemoval(S, P)`**:**
- Randomly select a percentage $P$ of the routes in $S$
- Remove all orders from those routes from $S$

**End**