Optimizing the pool allocation of multi-orders in an e-fulfilment centre

Master thesis

Author:
K. Poortema, s1733850

Industrial Engineering and Management
Faculty of Behavioural, Management and Social sciences

Supervisors University of Twente:
Dr. M.C. van der Heijden
Dr. D. Demirtas

Supervisors bol.com:
B. Thielen
L.A. Botman

November 2021

bol.com

UNIVERSITY OF TWENTE.

This version screens off confidential information related to the company involved in this research. Some numbers are replaced by **'XXX'** and figures are shielded. Furthermore, the results are given in ratios instead of absolute numbers.

Abstract

With the continuous transformation of digitalization in the past decades, e-commerce is becoming an indispensable part of our society. In the e-fulfilment centres that drive the e-commerce industry, efficient order picking systems are required to competitively handle a large extent of customer orders, selected from a wide product variety spread over an immense warehouse area, in a very limited time period. By pooling allocation, multi-ordered items are assigned to clustered zones within a warehouse. This pooling allocation stimulates the picking performance such that the average time required to pick an item is minimized, as this shortens the picking routes the operators process. As a result, this turns into minimization of labour costs. Simultaneously, fulfilment needs to be managed to ensure timely delivery.

In a case study performed at bol.com, a set of constructive heuristics is designed to improve the current pooling allocation method. By a simulation approach, it is shown that the seconds per pick for multi-orders could significantly be reduced, which result in significant annual cost savings for the e-retailer. Accompanied by this reduction in average time per pick, the fraction of orders that are not automatically fulfilled on time increases slightly.

Management summary

This report studies a unique assignment problem of multi-orders in an e-fulfilment centre of a market leader within the e-commerce in the Netherlands and Belgium: bol.com. Given many-and-small customer orders, wider product varieties, immense storage areas and limited delivery times, an efficient order picking system is required in e-fulfilment centres. In this problem, the 'pooling' method bol.com currently applies is introduced, which forms the basis for their order picking system for multi-orders. The goal of this study is, by means of practical and theoretical research, to come up with solutions to improve the picking productivity. By a simulation approach, the performance of a set of solution methods is tested against the current pooling method.

Problem statement

The storage area of the warehouse considered in this research is divided over tens of equally boxed zones wherein the items could be picked. To pick efficiently, the stock availability of all items is dynamically distributed over one or multiple zones by a randomized storage strategy bol.com is applying. By clustering items -needed to be picked- to a zone, distance will be saved as more items are assigned to that zone, which will save picking time. For multi-orders, orders consisting of multiple unique items or an item with a multiplicity of more than one, the assignment of its items are more complex than for mono-orders because they need to be compiled at the packing station. By the lack in picking performance of multi-orders, these types of orders are considered in this research.

Before an order becomes available for picking, it is preassigned to a packing station based on the shape dimensions and weight of its items. The multi-order picking process consists of three main steps. First, bol.com determines which of the available orders should be released into the picking process. Secondly, the multi-orders will be selected and added to pools, which are groups of orders (assigned to one or multiple zones) to ensure their items can be compiled efficiently after they are picked. As a third step, picking routes will be created within each zone that is selected by all created pools, which will be executed by the operators of bol.com. This research focuses on the second step, wherein items of multi-orders are assigned to zones. As new orders arrive continuously and the release of new orders into the picking process occurs every 15 minutes, the goal of this research is to reduce the average time per pick of all processed items over an entire operating day. This, while ensuring every item is fulfilled on time by taking into account the deadlines of all distribution carriers. A reduced time per pick should enable bol.com to handle the increased number of customer orders with limited additional investments in capital assets and people. A boundary condition is the calculation time of the pooling algorithm, which can not exceed four minutes for the orders of all packing stations sequentially. The main research question is formulated as follows:

How can the time per pick for multi-orders in the BFC be reduced by improving the current pooling algorithm, while maintaining the fulfilment criteria?

Approach and methods

To evaluate the performance of the pooling algorithm, an approximation function is derived which estimates the average time taken per pick based on the number of items that are assigned to a zone. Based on historical data, accurate parameters are learned for this estimation function. Since we are addressing a large-scale instance that includes usually more than a million non-binary decision variables which are measured by a non-linear defined objective function, an exact solution cannot be obtained within

the allowed computation time. Instead, constructive heuristics are considered as the most appropriate methods to use for this problem, which built up solutions from scratch by a set of decision rules. These decision rules are based on self-designed logic that effectively creates solutions within a limited time period.
The current algorithm is focusing solely on the cut-off time of orders, while this is irrelevant for more than 95% of the orders since their cut-off time is past an operating day ends. Besides, the current algorithm regularly ends up with small pools since its maximum capacity remains unused. In this research, it is exhibited the maximization of the size of a pool reduces the average time per pick.
In a created simulation environment, a variant on the current method and three newly designed methods are tested against the current method, which is set as the benchmark heuristic. One of the newly designed methods has its decision rules aligned to two main principles: first, the number of zones in a pool are minimized while it still creates full pools in terms of orders and secondly, when more zones are used in a pool the assignment of items to one zone should be maximized to the highest possible extent. All methods are tested for two different packing stations over two scenarios based on the total stock levels that were available at the beginning of a day.

Results

The results of the two scenarios regarding the stock levels (either high or low) do not show significant differences in comparison to each other. In terms of the average time per pick, the orders for one packing station are performing way better than the other since the maximum pool size of this packing station is higher and the picked items of these multi-orders are usually smaller and less heavy. The best performing algorithm for both packing stations, in terms of average time per pick, is the method that minimizes its zones for all pools constantly. This method is on average 1.8 seconds per pick faster than the current algorithm. As a result of the significant savings in time, the fulfilment score of automatically processed orders dropped down by 1% in comparison to the current algorithm. However, this difference is not considered as an issue by bol.com, as the control room is currently already prioritizing orders manually into their picking system. The calculation time measured in the testing environment indicates this method will take around three times the current method, which should be further investigated since the absolute calculation times of the simulation environment are not comparable to the software bol.com is using in practice. Therefore, we cannot yet conclude whether the calculation time of this method would fit in the allowed time period of four minutes.

Conclusions

- The picking performance can be improved by maximizing the pools along with its maximum pool size. A variant of the current method showed on average 0.033 X seconds per pick could be saved by changing the decision rules.
- Further on, the current pooling allocation method can even be more improved while the impact on automatic fulfilment is acceptable according to bol.com. By applying the method minimizing zones an average reduction per pick of 0.075 X seconds is expected, which will in terms of labour costs lead to a reduction of at least € XXX,XXX annually.

Recommendations

- The advice to the company is to experiment further with the best performing method that minimizes the zones used per pool. A new experiment performed in the services of bol.com should point out whether the calculation time is acceptable or not. If not, combinations of methods might be an alternative solution. As a second alternative, the variant of the benchmark algorithm should be impalement, since this leads to a decrease while the calculation time remains constant.
- Next to that, the characteristics of multi-orders should be monitored. As parameters of multi-orders might change over time, the company should investigate which strategy performs best in other situations by additional research.

Preface

The thesis in front of you marks the end of my master Industrial Engineering and Management at the University of Twente. During more than six unforgettable years of studying, I learned a lot, had many accomplishments and had great experiences. I am grateful that I had the opportunity to study at the University of Twente, which provided me with a lot of knowledge and inspiration.

I would like to thank Matthieu van der Heijden for all his valuable and accurate feedback as my first supervisor, which has helped me towards the research that is done. His structured and clear input during our discussions improved the quality of this research to a significant extent. I also would like to thank Derya Demirtas for providing valuable input as my second supervisor.

Moreover, I would like to express my gratitude towards my supervisors at bol.com. At first, I want to thank Bram Thielen for his continuous support and guidance throughout the entire project. I also want to thank Loek Botman for his valuable knowledge and his commitment to help when I needed it. The drive and enthusiasm of both men brought this research to a higher level. I am proud to have worked together with them and the rest of team 5P at bol.com, which really was an incredible adventure. Above that, it is great to see this team will make work of this research in the upcoming period.

Finally, I would like to thank my family and friends for supporting me during this research period.

I hope you enjoy reading this report and that this thesis may serve as a source of inspiration.

Koen Poortema

Amsterdam, November 2021

List of Figures

List of Tables

Glossary

BFC1 Bol.com Fulfilment Centre 1, the largest operating warehouse of Bol.com located in Waalwijk.

cut-off time The moment an order must be picked in order to be delivered to the customer in time.

CVRP Capacitated vehicle routing problem.

fulfilment score The fraction of orders that are processed in time and thus delivered to the assigned transport carrier for shipment.

items Single physical products customers can order.

mono-order Customer order consisting of only one item.

multi-order Customer order consisting of >1 items.

order pick Single ordered item that needs to be picked.

outbound line A particular packaging line in the outbound process of the BFC. Orders are planned based on the capacity per outbound line.

pick batch A group of customer orders that are picked in one single pick tour by an order picker. For bol.com, the pick batch is limited to the dimensions of one tote. Furthermore, every pick batch is dedicated to one, and only one, zone.

pick run A set of specific numbers of customer orders per outbound line that will be released into the picking process, based on the request by the operating control room in BFC1. This request is made approximately every 15 minutes during a working day.

picking route The route through one zone of the BFC in which the items of one pick batch are picked.

pool A subset of customer orders that are part of the same pick run. This subset of customer orders within a pool is distributed over one or multiple pick batches, which all are assigned to the same outbound line in BFC1.

poolzone The subset of a pool that consists of all the items (belonging to various multi-orders) that are assigned to one specific zone.

poolzone size The total number of items that are assigned to a zone.

spur Ingoing and outgoing points for totes to the conveyor, which are located at every zone.

stingray Tower shaped buffer station (connected to the conveyor) that stores totes temporarily before they are sorted and packed. Totes of pools are waiting in the stingray till the last tote of the pool arrives.

sweep run The last pick run of an operating day, used to bundle the last amount of orders of an outbound line. When this sweep run is called by the control room, all leftover orders of an outbound line must be assigned to pools in that pick run..

tote Blue bin that is used by an order picker to temporarily store items of one pick batch, while picking all the orders assigned to one pick tour. Totes are used to transport items within the warehouse.

tote fill The number of items placed in a tote.

tote fill rate The fraction of the tote filled by all its picked items, measured in volume.

zone A subset of the picking area of the warehouse where all items are stored. The picking area of the BFC is divided into 53 zones. Picks assigned to one pick batch must be retrieved from the same zone.

Contents

Chapter 1

Introduction

This report is a master thesis of the study Industrial Engineering and Management at the University of Twente, that describes a performed research for the e-retailing company bol.com. The first chapter introduces the company and the approach to finding the core problem of the assignment that was requested, including the motivation to come up with a conforming solution. In addition, a problem-solving approach is proposed to achieve the research goal. Next to this, the framework of the research is defined. Thereafter, the main research question is formulated to achieve this goal. The main research question is supported by sub research questions that each contribute to the solution for the research problem in this report.

1.1 Introduction bol.com

With the continuous transformation of digitalization in the past decades, online business becomes more and more important. Especially when it comes to day-to-day products, customers tend to make use of e-retailing as an indispensable part of our society. Bol.com is the largest online retailing company in the Benelux, which nowadays deliver over more than **XXX** million packages per year to their customers. In last year peak moment, they were even able to distribute **XXX** packages per day from only one warehouse.

Bol.com (which originally stands for 'Bertelsmann On-Line') was founded in 1999 as an online bookstore in Europe by the German media group Bertelsmann AG. In 2002, Bol only continued their business in the Netherlands since it was taken over by a joint venture of a group of German stakeholders. From 2004 and on, bol.com incrementally extended its range of products, starting with electronic devices, games and toys up to their tremendously large and various assortment as it is today; millions of different **items** are currently sold by bol.com (Bol.com, 2021).

The rapid growth of bol.com was accompanied by a few important milestones. From 2010, bol.com expand its customer focus by serving customers in the Flemish part of Belgium. In 2012, the Dutch international retailer Ahold (which is commonly known as Ahold Delhaize since they merged with the Belgium food retailer Delhaize in 2016) took over Bol.com. Furthermore, in 2015 bol.com made a big change in the regular e-commerce market by offering entrepreneurs and companies to sell their products via bol.com. From this moment on, bol.com not only acts as an online retailer anymore but as an online platform. In 2017, bol.com launches its first fully-owned Bol Fulfilment Center (further on in this research called **BFC1**. By the willingness to make every day a better one than the day before the company pursues this successful growth every year resulting in an increment in turnover every year.

1.2 Logistic processes at bol.com

To maintain the growth bol.com is expecting, sophisticated logistical support is of essential importance for the company. One of the biggest challenges is the optimization of e-fulfilment for this market leader in e-commerce. E-fulfilment can be defined as all necessary arrangements for business to sell their products on services on the internet (University, 2021). It is a commonly used term for a segment of the logistics in e-commerce, which includes the picking, sorting, packing and shipping processes of online customer orders.

1.2.1 Warehousing activities

In the field of e-fulfilment, a lot of research is focused on optimizing the order picking process, as this is the most labour-intensive process in the warehouse (Kulak, Sahin, & Taner, 2012). Bol.com is also focusing on this aspect of e-fulfilment, where multiple optimization projects have been performed already and several teams of employees are continuously striving for further improvement. The extending range of products leads to efficient utilization of warehouse capacity. Nowadays, on average approximately 200.000 items per day are distributed from the warehouses to customers in the Benelux. Currently, bol.com operates from five different warehouses. Every warehouse is used for specific types of items, either size or product-related. Detailed characteristics can be found in Appendix A. The largest warehouse is Bol.com Fulfillment Centre 1 (BFC1) and is used for the majority of all small to medium-sized packages.

BFC1 is located in Waalwijk and distributes **XX%** of the distributed items. This includes that on average around **XXX** items are distributed from BFC1. BFC1 provides a stock capacity of **XXX** items. Bol.com is currently building a second bol.com fulfilment centre (BFC2), which will be located next to BFC1. The expectation is that BFC2 will start operating in September 2021. In the warehouses, every item follows a structured process. Figure 1.1 shows the logistic processes that take place in BFC1. As depicted, the warehouse can be divided into three categorical segments: inbound, storage and outbound. The inbound activities cover the unloading and receiving of all items that are delivered by the suppliers. Next, the items need to be stored in the warehouse. This is manually done by operators in the warehouse following a put-away heuristic to locate all items efficiently. Further on, the outbound activities can be interpreted as the process between the moment an item is ordered by a customer till the item is distributed to that corresponding customer.

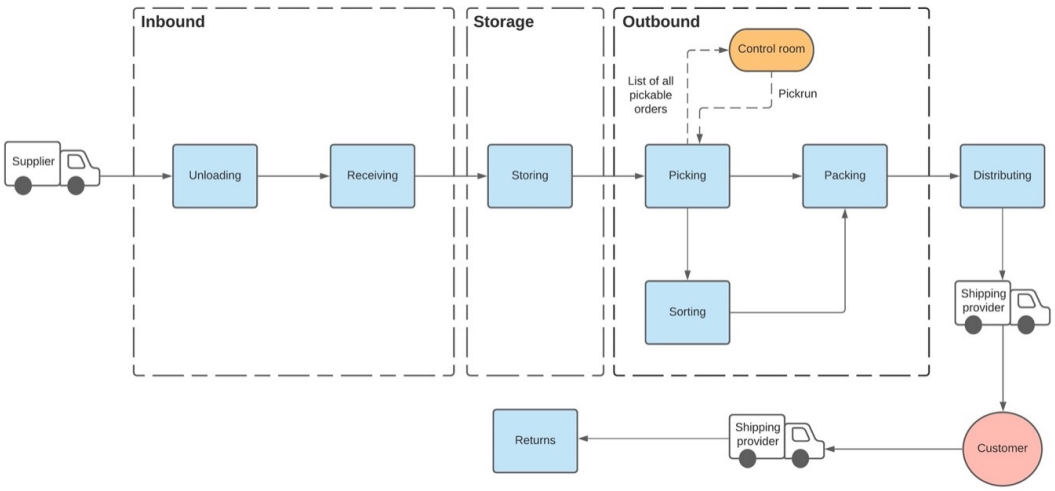

FIGURE 1.1: Flowchart of warehouses processes

The warehouse management system of bol.com (WMS) is operating as an ERP system. In each warehouse, the WMS keeps track of all placed orders, picking orders that are in process and the available stock. Currently, the picking process is handled by a control room of employees, who create approximately every 15 minutes a new pick run with pick batches that are sent to the order pickers. This **pick run** starts with a set of specific order numbers per outbound line that is released into the pickings process. These numbers are determined by the control room, whereafter bol.com applies a series of sorting heuristics and algorithms to determine which orders should be picked specifically. Thus, a pick run consists of a set of selected ordered items that are sent to the order pickers in BFC1 via the WMS.

The logic bol.com uses to decide which orders should be picked, and in which order, is very specific and dependent on many factors since the list of unassigned orders is usually very large. An important step bol.com performs to obtain efficient routing strategies for the operating order pickers is the creation of clustered batches, which are sub-groups within a pick run. A **pick batch** is a group of customer orders that are picked in one single picking route by an order picker, which is named consistently during this report.

In BFC1, the stored items are allocated to one of the 57 different zones. Each **zone** contains an ingoing and outgoing **spur** which are part of a large network of conveyors. The incoming spurs are used to store the items, while the outgoing spur is used to carry picked batches on the conveyor. In total, approximately seven kilometres of conveyor has been installed in BFC1 to transport all items within the standardized blue totes. In Figure 1.2, an outgoing spur including the conveyor with a few carried totes is shown.

A **tote** is a commonly used term at bol.com and is consistently used in this report, which is a blue shaped bin that is being used for one, and only one, pick batch of ordered items. The dimensions of a tote are 75 x 55 x 41 in centimetres. When an order picker starts a new picking batch, an empty tote is picked and stacked on a picking trolly whereafter all items of that batch could be collected. The pick batch ends when the order picker has succeeded in the picking of all items and the tote is placed back on the outgoing spur of the corresponding zone. The route the order picker walks to collect one pick batch of items is called the **picking route**. Thus, one pick batch is in other words one picking route performed by one order picker in one zone.

FIGURE 1.2: An outgoing spur

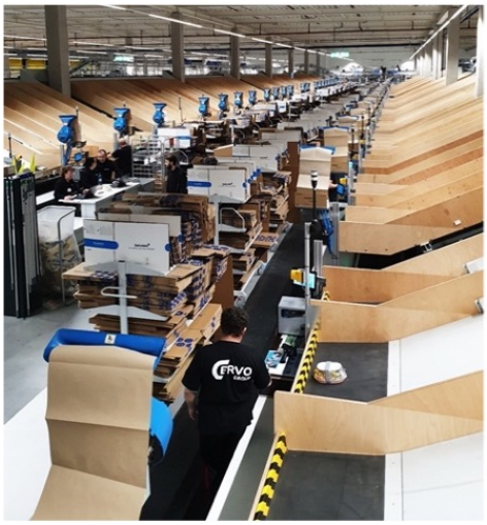

FIGURE 1.3: Outbound line 109

After the orders are collected in the tote by the order picker, the tote is shipped to the (by the control room determined) **outbound line**. An outbound line is an area where all items are manually packed, for which an impression is shown in Figure 1.3. Currently, BFC1 contains nine outbound lines. In these outbound lines, there is made an important difference in mono-orders and multi-orders. A **mono-order** is one single item that is ordered by a customer. In contrast, a **multi-order** is a customer order that contains either of the following characteristics:

1. An order that contains multiple products.
2. An order that contains a product that is ordered multiple times.

In Table 1.1, an overview of the picking performance of both types of orders over the year 2020 is given. An important difference is the average picking time per second, which differ around approximately **0.0633 X** seconds between mono-ordered items and multi-ordered items. As this seems not to

be a significant difference on one single pick, it certainly has an impact when multiplying the annual quantity of items that are processed by the warehouse. For instance in 2020, when more than **XXX** million multi-ordered items were picked, it results in an extra picking time for multi-items of around **XXX** million seconds, which can be rounded to 349 days. So, the complexity of multi-items causes currently an additional picking time of almost an entire year, which is annually more than **XXX** euro computed in labour costs.

Order type	**Average time per pick (sec)**	**Fraction of orders** (%)
Mono-order items	**XXX**	**XXX**
Multi-order items	**XXX**	**XXX**

TABLE 1.1: Overview of both order types in BFC1

1.2.2 Picking process

According to experts in the field at bol.com, the order picking process has the most influence on warehouse performance. Order picking, defined as the process of retrieving items from their locations in a warehouse, generally leads up to approximately 60% of all labour activities (Gademann & van de Velde, 2007). And since these labour activities are the most costly operations in a warehouse, order picking is one of the most crucial parts of the supply chain. In section 1.2, it was already noted that the control room leads up the process streams by creating pick runs approximately every 15 minutes. Hence, these pick runs only contain the order numbers per outbound line that should be picked. To determine which orders should be picked, bol.com applies a series of algorithms. These algorithms are designed to achieve effective picking productivity; all items should be picked on time and preferably are located close to each other, so the distance between picks can be minimized. In this sub-section, more insights into this picking process are explained.

Mono-orders and multi-orders

The different types of orders (mono and multi) are processed differently in BFC1, as multi-orders need to be packed together in the same box before they can be shipped to the customer. Thus, multi-order items are depending on each other. Before multi-orders arrive at the outbound line, they need to be compiled out of all the pick batches per zone corresponding to the same customer order. Then they can be packed. Logically, mono-orders are packed individually and do not have to be compiled. Therefore, the optimization of the picking productivity expressed by the average time per individual pick is a very clear goal. However, for multi-orders, more complexity is involved.

Multi-order picking process

Frequently, not all items of one order are available in one zone. To tackle this complexity, bol.com creates pools for multi-ordered items. A **pool** can be characterized as a group of customer orders that is part of the same pick run, which contain multiple pick batches divided over one or multiple zones. On average, the number of pools is between 5 and 6 pools per pick run. For each zone that is used in a pool, a list of items is created that belong to that specific zone, under the condition that all items are available for picking in that zone. This list of items is called a **poolzone**. Thereafter, for each poolzone, a specific capacitated vehicle routing problem (abbreviated **CVRP**) is solved. The CVRP creates the batches and the routes that result in a sequence of order picks that is followed by the order pickers.

By this procedure, different pick batches within a pool are able to contain all items that belong to the complete corresponding multi-order. Since the pick batches of a pool are created in the pick run, the orders are picked approximately at the same time (in other zones though). When a pick batch in a pool is completed and placed on the conveyor via the spur, the totes containing the items of a batch are temporarily stored in a general buffer, called the **stingray**. The stingray is a tower in BFC1 that is

accessible by the conveyor, which can store a limited capacity of totes. When the last pick batch of a pool arrives at the stingray, all totes of the pool are simultaneously sent to the sorting station of the belonging outbound line. Thereafter, the items are compiled at a sorting station of the outbound line and are packed successively.

In equation 1.1 below the relationship between the terminological terms of an **order pick**, pick batch, pool and a pick run is given, which are subsets of each other. These terms are frequently used during this research since they are relevant to the multi-order picking process.

$$Order \quad pick \subseteq Pick \quad batch \subseteq Pool \subseteq Pick \quad run \tag{1.1}$$

In Figure 1.4, a schematic view of all processes for the multi-order picking process is shown. As can be seen, available warehouse orders are divided into a set of pools. The size of a pool is based on the maximum allowed number of orders in a pool, which depends on the buffer capacity of the stingray and the sorting station of the corresponding outbound line. When the pool is created, the multi-orders are split into partial orders per zone. There is no fixed minimum or maximum of zones that can be used per pool. Furthermore, based on the poolzones the CVRP creates the pick batches including the route the order pickers will follow.

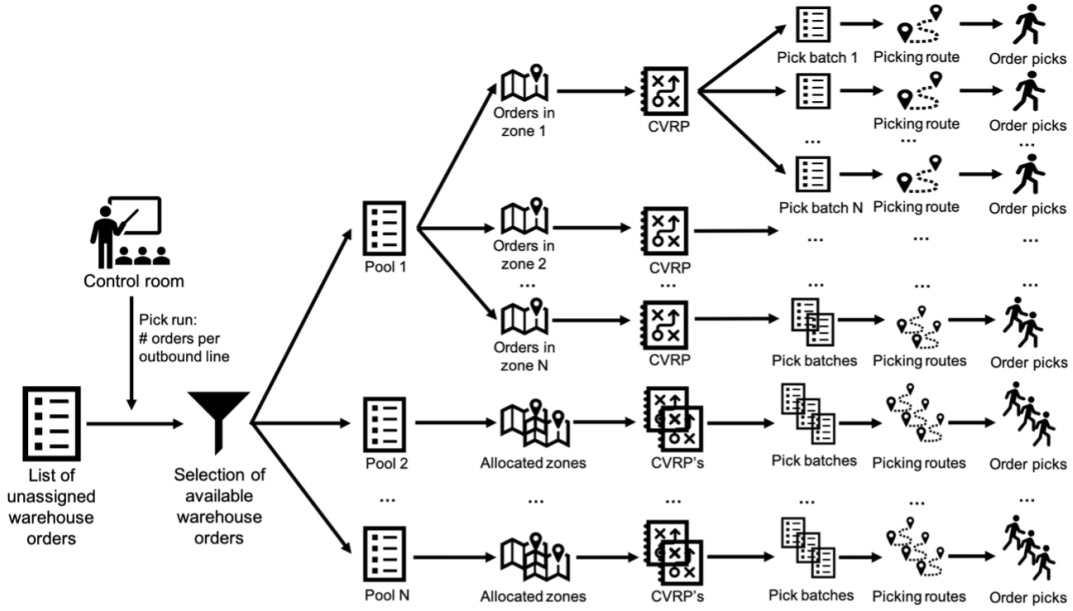

FIGURE 1.4: Process steps in the multi-order picking process

1.3 Problem statement

As mentioned, the product variety of bol.com and the number of customer orders increase every year. This means an increasing number of customer orders must be picked from a broader product assortment, within the same warehouse capacity bol.com currently provides. To accomplish this, the picking productivity constantly needs to be improved. This requires a critical view of the current process. The current way of assigning customer orders to the picking schedule of bol.com leads to a series of problems, which is explained in this section.

1.3.1 Research goal

The picking productivity, expressed in the average time that is taken for one single item by the operator, needs to be increased to meet the expected growth in the future. When this increment will not be met, the warehousing department of bol.com will slow down the rest of the logistic services. To avoid the current picking process resulting in (unnecessary) higher labour costs or delivery issues, bol.com is

seeking for continuous improvement. Listening to experts in the field, the average picking time could be reduced significantly by thoroughly evaluating all the current performed decisions in the picking process. The average picking time per item over the year 2020 is approximately **XXX** seconds per item. Besides, there is a gap in the performance of mono-orders and multi-orders (see Table 1.1). According to bol.com, the order picking system of multi-orders did not change over the last couple of years, while the amount of items that are being picked is currently significantly higher. Above that, a precise overview of all performed steps in the picking logic, including its impact, is lost.
By this knowledge and the expansion targets of bol.com, it can be stated that the average time per pick of multi-orders is currently too high. This can be defined as the knowledge problem which should be solved by this research. Thus, the goal of this research is to increase the picking productivity in BFC1, which is the variable that needs to be improved according to bol.com (Heerkens, 2012).

1.3.2 Tote capacity

As a cause of the defined knowledge problem, we note the tote capacity is not used optimally. The tote capacity can be measured by two different parameters: the tote fill rate and the tote fill. The **tote fill rate** measures the fulfilment of the summed volume of all picked items in the tote as a fraction of the total volume of a tote. The average tote fill rate over 2020 was around 27%, which is remarkably low. The second parameter is the **tote fill**, which is an indicator of the absolute number of items that are picked for one tote. This number of items is related to the picking productivity, which can be seen in Figure 1.5. In this graph, the relation between the picking productivity (measured by the average time taken for a pick per item) and the tote fill is shown. As can be observed, the average time per pick decreases by an exponential curve for totes containing more items. As the obvious solution might seem to maximize these tote fill, a contrary pattern is observed analyzing the data of bol.com. In Figure 1.6, the frequencies of the tote fill sizes of last year are shown. Interestingly, the majority totes consist of a low number of items. As Figure 1.5 is showing, this has resulted in low average picking times.
Note that by capacity limitations reaching large tote fills may become hard. The average tote fill over the last year yields 16.5 items. However, by the mentioned average tote fill rate of 27%, it is evident a lot of potential capacity is currently unfulfilled.

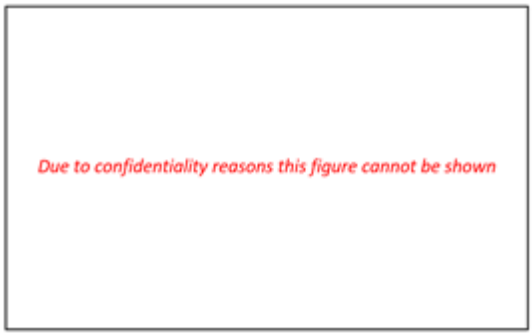

FIGURE 1.5: Average picking time over the tote fill size

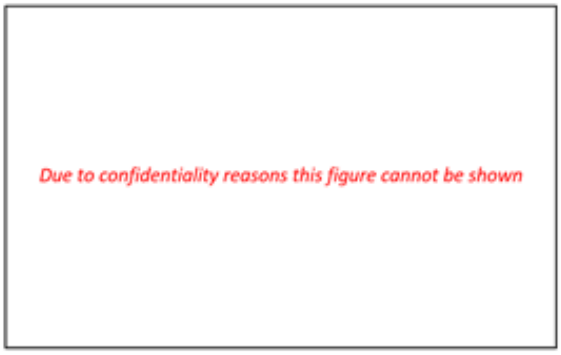

FIGURE 1.6: Frequency of the tote fill sizes

The inefficient use of tote capacity can be related to the fact that the created pick batches are too small. The creation of these inefficient pick batches is performed by the earlier mentioned CVRP, which optimizes the picking routes for the given list of items per zone considering the weight and capacity limitations of a tote. Though, the solution space of all CVRP's per zone is limited to the input they receive from the pooling method, which was carried out one stage before. This input is the number of items allocated to a zone, defined as a poolzone in this report. In addition, the **poolzone size** is defined as the total number of items that are assigned to a zone in a specific pool.

Analyzing the poolzone sizes in Figure 1.7, the same pattern regarding the average picking time is obtained as in 1.5. The average time per pick for low poolzone sizes are way higher, while again these sizes are most frequent. In Figure 1.8, the cumulative fraction of poolzone sizes are represented, where can be observed that the low number of poolzone sizes are dominant. For instance, 54% of all poolzone size consist of less than 20 items. Thus, the ineffective output of the pooling method disables the CVRP to come up with efficient solutions, since the list of items in that zone is fixed. Therefore, the pooling method has a crucial impact on the picking process.

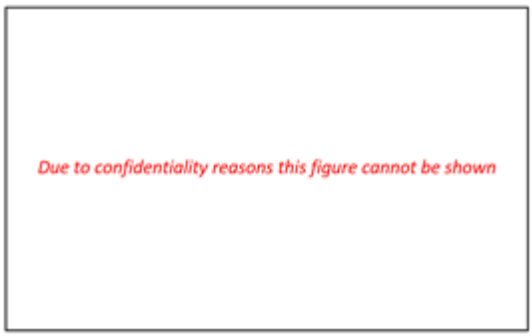

FIGURE 1.7: Average picking time of the poolzone sizes

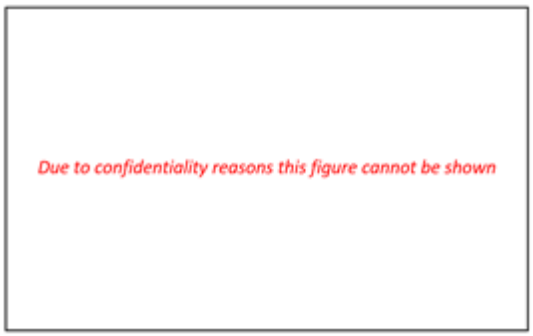

FIGURE 1.8: Cumulative fraction poolzone sizes

1.3.3 Core problem

In general, higher average picking times are related to a higher average distance between picks. Since bol.com is applying its zoning strategy, the average distance that needs to be travelled is dependent on the tote fill sizes of each pick batch assigned to the zones. On average, multi-order items require 44% more travelling distance than mono-orders, which is a serious concern. The non-optimal pooling allocation for multi-orders can be pointed out as the cause of this relative high distance and, thus, higher picking times.

To state once more, the current pooling method plays a fundamental part in the multi-order picking process, as it currently limits the batching method (CVRP) performed afterwards. Namely, the outcome of the CVRP is dependent on the input it receives by the pooling method. For that reason, the pooling method is set to be the most relevant issue within the picking process. Therefore, the core problem of this research is:

The pooling method in the order picking process of bol.com leads to inefficient picking of items belonging to multi-orders that can be picked per time.

1.4 Research framework

To derive solutions for the core problem addressed in the previous section, assumptions and limitations regarding the scope of this research have to be determined. In this section, we discuss the main assumptions that support a research environment towards feasible solutions for this problem.
This research is purely focused on the process of pooling generation, which is performed after the control room has determined the number of orders per outbound line. This implies the multi-order items (and thus the corresponding outbound lines of multi-orders) have full priority in this research, since pooling does not apply to mono-orders. In 2020, **XXX%** of all performed picks by order pickers were related to multi-orders. So, improving the picking productivity for multi-orders will have a significant impact on the total picking productivity in general.

In Figure 1.9, the process step of this research is shown explicitly (for the entire picking process, see Figure 1.4). This step consists of assigning items of orders to zone within pools. The successive

performed CVRP is thus excluded from the scope of this research and will remain unchanged. Listening to experts in the field at bol.com, thoroughly analyzing this algorithm will not result in a significant direct improvement.
Further on, the solution that is designed by this research is applied to BFC1 specifically. By the knowledge that this largest warehouse is still not utilized according to its maximum capacity, improving the current situation here is most effective. Besides, the results of this research may be valuable for upcoming warehouses of bol.com in the future, as bol.com is working on expansion plans regarding a BFC2 and even a BFC3 in the long term. However, the primary focus is on optimizing BFC1. In this warehouse, the layout and the current manpower of order pickers are assumed to be fixed. Further on, the stock locations and the large conveyor distribution network in BFC1 can also be considered as unassailable. Above that, decisions performed in the inbound department of bol.com are assumed to be constant, such as receiving and put-away strategies.

Another important factor in this research is the fulfilment criteria of bol.com. Each order has a **cut-off time**, which is defined as the ultimate moment in time all items of the order should be picked to be delivered on time (while taking into account remaining activities like sorting, packing and shipping). This cut-off time should be met under all circumstances to maintain customer satisfaction, which is measured by the fulfilment score. The **fulfilment score** can be interpreted as the percentage of orders that is delivered on time to the shipping carrier. Currently, the fulfilment score that is met is 99% of all orders. Practically, this implies every order must be fulfilled on time.

The goal of this research is to improve picking productivity. Improvements in large-scaled logistic operations like this proportionally correlate with investments in capital assets and labour. However, the goal of bol.com has always been to keep these additional costs minimized and still be more productive. The focus of this research is to attain the best possible solution regarding minimal additional costs.

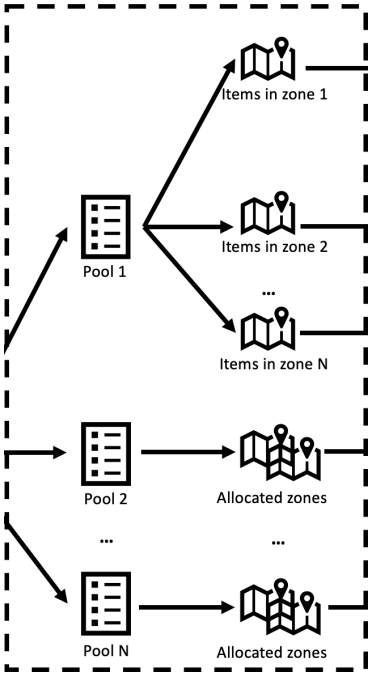

FIGURE 1.9: Schematic scope

1.5 Research questions

Related to the stated core problem, the research objective is to find a solution for the pooling method that will increase the overall picking productivity. This picking productivity is expressed in terms of the average time that is taken for one pick per minute per single order picker. Accordingly, this research investigates the following main research question:

How can the time per pick for multi-orders in the BFC be reduced by improving the current pooling algorithm, while maintaining the fulfilment criteria?

This question will be answered by the following sub-questions:

1. What performances in the current pooling method of bol.com are critical?
2. What improvement methods can be found in the literature?
3. Which methods are most suitable according to the situation of bol.com?

4. How can it be determined which method provides the best results for bol.com?

5. Based on the validated results, which method and implementation steps can we recommend to bol.com?

By thoroughly analyzing the effect the current pooling method has on the performance of bol.com, the bottlenecks of this pooling process are discovered. Access to relevant real-time data is permitted by bol.com. Lots of historical, qualitative data is available; all ordered items are marked by registered timestamps, including valid records like the belonging pick run/pool/batch/customer order. This large useful data extent enables performance measurement on the effect of pooling to picking productivity. Interesting parameters to analyse are complex characteristics of multi-orders in terms of size in the number of total items, quantity and availability along the zones. Furthermore, the flow of the supply chain in the warehouse is also considered.
To identify all steps and decisions that are made in the current pooling logic, a descriptive explanation will be supported by a flowchart and pseudocode of the process. These attributes provide insights into the current way of working of bol.com. In combination with the performance analysis, the research environment is inventoried.

For sub-question 2, an extensive literature study is performed. Similar -already investigated by other researchers- problems are considered along with their appropriateness and contribution to the research environment of this problem. Most likely, inspiration from multiple perspectives is needed since the situation of bol.com is very specific. When identifying related problems, solution methods related to these problems are addressed. Furthermore, a study on techniques to measure the output of the pooling method is performed. Since we are focusing on one specific step within an algorithmic series of multiple problems, an effective way for the pooling measurement should be defined.
By evaluating the appropriateness of methods discussed in the literature study, a set of potential useful methods is expected. Though, assumptions and parameter definitions should be set to construct a concrete solution approach. The decision which method(s) will have the highest potential to outperform the current algorithm (conducted by a weighted analysis of all potential methods) will come to light by answering sub-question 3.
The effectiveness of this analysis is measured on the expected outcome related to the solubility of the core problem (to what extent does the picking productivity improve?), regarding stated restrictions as the fulfilment criteria and the allowed calculation time. Thereafter, an experimental environment is designed, wherein assumptions are made in alignment with the stakeholders at bol.com. By recreating the pooling process in a representative simulation, a set of solution methods can be tested. Furthermore, selectively chosen data sets of different scenario's are sampled in order to validate the significance of the method(s).

After the experimental approach is set up and the results are obtained, the final sub-question of this research can be answered. In a concluding chapter, we elaborate on all findings during this research. The results of the selected methods are evaluated, where the impact on picking productivity are measured and weighed off against its feasibility. As a result, well-considered advice is given to bol.com on which method should be implemented. Additionally, the steps for implementing this into BFC1 are set out precisely.
Besides a conclusion, discussion points by the decisions made in this research are mentioned. Furthermore, recommendations for further research are given, which elaborate how this research could be extended and be of value to other warehouses.

Chapter 2

Current situation

In this section, the research question 'what performances in the current pooling method of bol.com are critical?' is answered. First, all steps of the current pooling method are explained in detail. Secondly, we address the environment wherein pools are created, along with all its limitations. Furthermore, indicators influencing the average picking time are exploited and measured on its performance. The characteristic of multi-orders are analyzed, whereof the most critical parameters are addressed. In a concluding section, the most important findings are summarized and converted to valuable input for the literature review that is addressed in the next chapter.

2.1 The current pooling method

In BFC1, the control room regulates the flow of the picking process by requesting pick runs. Just to state once more for clarity, the control room only requests a specific number of orders for every outbound line separately. So, it is not possible for the control room to request specific warehouse orders. Nevertheless, the control room optionally can prioritize orders manually. This might occur for orders having a near approaching cut-off time, where the control room wants to pick these orders as soon as possible. Next to this manual prioritization, orders are mainly based on two categories; orders that must be picked today to meet their cut-off time and orders that could be picked today but still could be in time if not. When the control requests these numbers per outbound line, the assignation of the pooling process starts.

In this chapter, we only elaborate on the most important steps that are performed. In the flowchart represented in Appendix B, all decision steps are given in more detail. To structurize, the algorithm can be split into two different phases and a preparation phase:

0. Sorting the orders
1. Adding orders to pools
    (a) Adding orders to pools based on zone rule
    (b) Optimize pools based on volume
2. Combining created pools

Since step 1 and step 2 include some fundamental decisions on how a pool is built, pseudocodes of both steps are given in Appendix C. These pseudocodes provide a more order perspective in comparison to the general flowchart in Appendix B. For a clear understanding of the process in general, a descriptive analysis of each step is given, including the input and output of each step.

0. Sorting the orders

Prior to the creating of pools, the available orders are sorted. All orders are checked on whether all ordered items of the multi-order are currently available in BFC1. If not, the order is ignored. When all available orders are leftover, the orders are sorted based on five consecutive criteria:

1. The priority[1] of the order (first must go, then could go)
2. The cut-off time of the order (ascending)

[1]The priority and cut-off time are related. The difference is that a priority value is given manually by the control room.

3. The delivery date[2] of the order (ascending)
4. The difficulty[3] of the order (descending)
5. The entry time of the order (ascending)

The first three sorting criteria are all based on the same hard time restriction, while the fourth criterion measures the 'difficulty' of orders to pick. However, this difficulty is based on a calculation of an outdated ABC classification on the popularity of items combined with the quantity of each item of the multi-order. For relatively new items, we note an empty value.

1a. Adding orders to pools based on zone rule
Input: Requested number of orders, sorted list of available orders and the stock availability.
After the sorting procedure, pools are be created. Bol.com currently applies a seed algorithm. According to the literature, a seed algorithm is defined as a method where a group is created by selecting first one appropriate order as a seed order for the batch, and then other orders are assigned to this group based on an adding rule (R. de Koster, Le-Duc, & Roodbergen, 2007).
At bol.com, the highest-ranked available order is chosen as the seed order of a new pool. Based on this order, efficient zones are determined that have all items of the order available. Then, the next step of the seed algorithm is to check for other available orders if they could be picked in the same zones as determined by the zones for the seed order. If so, the order is added to the pool. This process continues till the maximum pool size is reached, which is expressed in terms of absolute orders and differs per outbound line (mainly depending on the sorting station of the corresponding outbound line).

When the maximum pool size is reached, the pool is closed and a new one is created by a new seed order. Nevertheless, it mostly occurs all orders are checked but the maximum capacity is not reached (yet). In this situation, the next step is determined by the size of the pool at that moment. When the total number of orders in a pool is above a certain minimum (again, different per outbound line), the algorithm will go to step 1b. The minimum and maximum values for pools per outbound line can be found in Appendix D. When all orders are checked but the minimum is not reached, bol.com applies a relaxation rule regarding the zones and checks every order again. This relaxation rule holds that except for the already determined zones, one extra zone is allowed if this may fulfil the complete order. So, when an order contains three items, wherefore two of them could be picked in the zones of the seed order and one in another zone, the order will now be added to the pool. After this, the algorithm continues by searching for orders within the updated selection of zones; the determined zones are updated and the relaxation is ruled out again. When all items are checked for the second time or when the minimum is reached, the algorithm will go to step 1b.
Result: List of items assigned over the selected zones of the pool, where the number of orders that are assigned to the pool is between the minimum and maximum pool size.

1b. Optimize pools based on volume
Input: List of items assigned over the selected zones of the pool, where the number of orders that are assigned to the pool is between the minimum and maximum pool size.
In step 1a, the zones for the pool are determined by the seed order and by the zone relaxation rule (if necessary). In this phase, the algorithm optimizes the number of orders by one additional parameter; the fill rate in the total volume of all items is measured as a fraction of the total volume of one tote. While the pool has not reached its maximum capacity, more orders will be added if they can be fulfilled by the determined zones. However, all items assigned to the zones should be alignment by a fill rate improvement per zone (the total fill rate per zone should increase and may not exceed 100%.
This phase stops when all fill rates per zone are 90%, the allowed volume or the maximum pool size is reached, or all available orders are checked. Then, the pool is closed. If the request of the control is

[2]More or less related to the cut-off time, could vary per shipping carrier
[3]Bol.com applies a self-constructed rule of thumb to indicate the difficulty of items that is based on a simple ABC classification on the popularity of items and the current stock quantity.

not exceeded yet, a new pool will open and begin in step 1a. Logically, the just chosen orders for the previous pools and the corresponding stocking locations are not available anymore.

Result: List of items assigned over the selected zones of one pool, where the number of orders that are assigned to the pool is between the minimum and maximum pool size.

2. Combine created pools

Input: List of items assigned over the selected zones over all the pools, where all the number of orders are assigned to the pools ranges between the minimum and maximum pool size.

When the pick run request is met, the algorithm performs its last step where there will be checked if pools could be combined. Each pool will be checked among the other pools if zones are used in both pools. If overlay exists and the merge of both pools will not exceed the maximum pool size or fill rate per zone, the combination is considered feasible and the pools are merged. This process repeats until no feasible combinations are possible. At this point, the pooling process is finished for one outbound line. The algorithm will perform the same process to meet the request of other outbound lines. Eventually, the items assigned per zone will be sent to the next algorithm (a CVRP) of bol.com that determines the pick batches.

Result: List of items assigned per zone within the pools of the pick run, where the number of pools may be reduced by merge (if possible only).

2.2 Pooling environment

In the previous section, the specific steps of the algorithm itself are explained. In this section, we analyze a broader perspective; the setting wherein the algorithm is performed. Five important factors have impact on the current picking productivity, which are worth discussing.

2.2.1 Warehouse layout

The existing warehouse layout in BFC1 plays a fundamental role in the current way of working at bol.com. The chosen layout puts constraints and requirements on the operational decisions and therefore influences the efficiency of the picking process. In total, the BFC1 has a ground surface of 50.000 m^2. The building consists of four floors, rectangular shaped. Each floor is separated systematically into two parts, referred to as tower 'A' or 'B'. In Figure 2.1, the floor map of the ground floor is shown. The numbers on the map, including the red and green arrows, do not give any relevant information to this research.

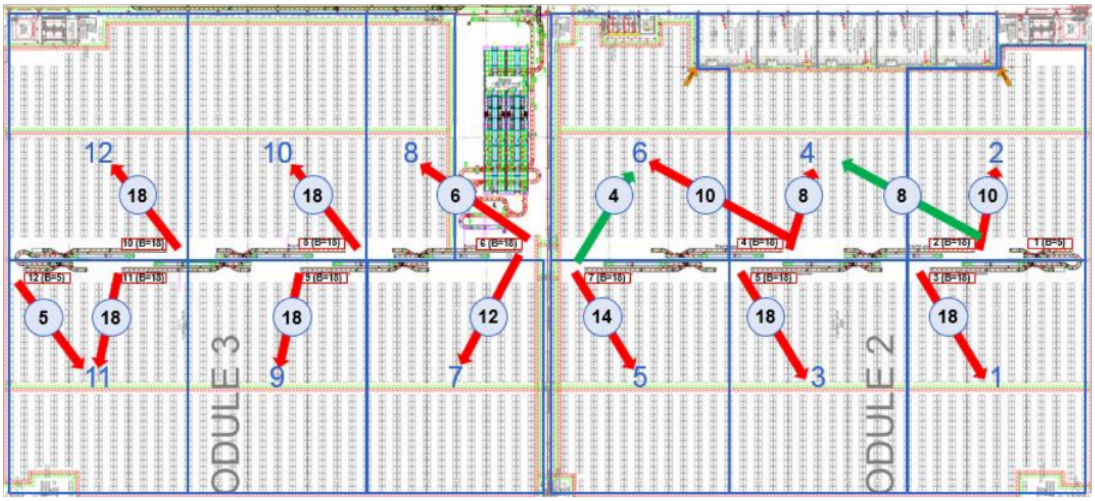

FIGURE 2.1: Floor map of the ground floor of BFC1

As mentioned before, bol.com applies a zoning technique in the BFC1 for picking its items, where the picking area is divided over multiple 57 zones. These zones can be distinguished by 52 equal 'normal' zones and five separated fireproof zones, where the flammable type of items are stored to reduce the risk of fire in the warehouse. These safety zones are located on the ground floor, located top right of the floor map of Figure 2.1.
Depending on the area space of each zone, the number of vertical aisles varies from 6 to 14. Each aisle contains available racks of different sizes to store items of corresponding sizes. In Figure 2.2, an impression of such an aisle is given. All storage locations, including the ones on the highest shelves, are accessible by the order pickers without any additional support of machines. The types of items that can be stored in the BFC1 can be at most 32 kilograms and have a total volume of 140 litres. Every aisle in the zone is split up into two parts by the addition of a cross-aisle, which enable the order pickers to switch to another aisle halfway into each zone. These cross aisles are located perpendicular to the vertical aisles from which the storage locations can be visited.

FIGURE 2.2: Aisle containing storage locations

2.2.2 Storage policies

Storage assignment is defined as the assignment of items to storage, which is proven to have a major impact on the picking productivity (Gil-Borras, Paardo, Alono-Ayuso, & Duarte, 2020) (van Gils, Ramaekers, Caris, & de Koster, 2017). The main used policies in scholarly literature are dedicated, class-based and randomized storage policies (Nieuwenhuyse, de Koster, & Colpaert, 2007). In BFC1, a randomized storage policy is applied, meaning the receiving items are allocated randomly over the available storage locations.

Besides the distinction of storing flammable items in special zones, no other separations over the zones are considered when storing the items. In the receiving stage, operators can fill totes (the same as used for picking) by all possible items. Though, the put-away advice of bol.com is to distribute the same items equally over the available totes, so they end up in multiple zones. When a tote is full, the tote is sent to the zone that has the lowest level of stock of these sets of specific items. Thereafter, an operator is assigned to pick up the tote from the incoming spur at the assigned zone. The order picker stores the items randomly within the zone, as long as the size of an item corresponds to the available size of the storage location.

By this storage strategy, the items could thus be available in multiple zones simultaneously. Even within a zone, a product can have multiple alternative storage locations. This variety of options increases the picking possibilities and thereby enables more efficient picking routes. For this reason, it provides essential information for the pooling algorithm.

2.2.3 Online order arrival

The third factor of impact is the online order arrival system bol.com is applying. The list of available orders changes dynamically over time, which can be explained by two factors: the addition of order arrivals and the removal of processed orders (when orders are picked they are removed from the list of available orders). New orders arriving may have a cut-off time of only a few hours later since bol.com ensures same-day delivery for a certain type of items. Also, the size of the total set of orders is not known in front, which (thus) varies per day. Because of this variety and consistently changing order set, computing pools and batches in front of an operating day is ineffective. To regulate the flow within the

warehouse, the control room of bol.com is manually making decisions when to request pick runs and how large these pick runs should be.
Every time a pick run is requested, the list of available orders is updated. At that moment, productive pools should be created. Next to that, the set of orders that are leftover are also important. When selecting the relative 'easier' orders, this might affect the performance at a later stage. Despite that, analysis of what makes an order easy have to be performed. Nevertheless, the objective should be focused on creating efficient pools over the long run, instead of just one efficient pool.

2.2.4 Sweep run

Picking operators in bol.com are approximately working between 7 AM till midnight. Thus, at night the picking production stands still. When a low number of orders is leftover for an outbound line at the end of an operating day, bol.com applies a **sweep run** to collect the last set of orders. This sweep run is considered as the last pick run of the day, where all leftover orders must be assigned into pools. As holds for all pick runs, this sweep run is requested by the control room; they decide when to pick all orders that are leftover for an outbound line. The main difference in this last pick run is that every order has to be included in one of the pools in the pick run since no order can be leftover. Because of this difference, the logic of this sweep run is slightly different than the heuristic bol.com is applying for the regular pick runs. Since this sweep run concerns just one pick run per day, and it involves a relatively minor part of the total orders that are processed each day. By the manual decision making the control room is performing, the impact of a sweep run has to be considered in this research problem.

2.2.5 Calculation time

A last important performance measure for bol.com is the calculation time of the pooling (and batching) algorithm. After a request has been placed by the control room, pick batches that meet the requirements of the control room should be generated within five minutes. By this allowed timespan, the system remains able to anticipate approaching cut-off times of orders. As Figure 2.3 indicates, two different steps have to be performed within these five minutes.

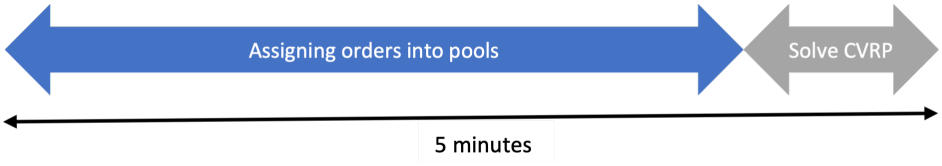

FIGURE 2.3: Schematic timespan of allowed calculation time

If selections of available orders are assigned to zones within the pools, solving the CVRP takes up to one minute. The IT systems of bol.com allow solving many batching problems in parallel. This means that in total one minute is spent on the formation of pick batches and the calculation of routes for one pick run. Therefore, the pooling method has to be performed in front of the remaining four minutes. The pooling method is not able to operate simultaneously for all outbound lines, which means it has to be performed sequentially when more orders are requested at multiple outbound lines.

2.3 Pooling performance

By the defined core problem in Section 1.3.3, it was observed a low poolzone size is the main indicator for high average picking times. Paradoxically, low poolzone sizes are most dominant in the current picking performance of pools. In this section, we discuss the possible causes of these frequently occurring low poolzone sizes. First, statistical pooling data is provided. Secondly, an analysis of the zone distribution over pools is shown. In the last subsection of this report, complex characteristics of multi-orders are exposed, since multi-orders can vary a lot.

2.3.1 General statistics

To give some indications on the average sizes of the different terms of orders and sets of items, a general overview is given in Table 2.1. Over the multi-ordered items of 2020, the average pool size is rounded 86.6 items. Regarding the poolzone sizes, we observe the number of items assigned to a zone is 22.8 items on average. Approximately 1.4 batches are produced by this number of items. Furthermore, it is noted that on average 3.8 zones are used within a pool.

Pooling statistics multi-orders 2020	
Total items	**XXX**
Total batches	**XXX**
Total pools	**XXX**
Average pool size	86.6 items
Average number of batches within pools	5.3 batches
Average tote fill per batch	16.5 items
Average number of zones used per pool	3.8 zones
Average number of items assigned per zone	22.8 items
Average number of batches per poolzone	1.4 batches

TABLE 2.1: Statistical numbers on the pooling performance over 2020

As mentioned in section 2.1, bol.com utilizes multiple outbound lines for sorting and packaging all multi-orders. In Figure 2.4 and 2.5, an overview of the performance of all four multi-order outbound lines is given. Outbound line 109 consists of an automatic sorter that compiles the multi-items in a pool automatically, which is, therefore, the most preferable outbound line to use. Nevertheless, not all multi-orders can be assigned to outbound line 109 because of weight and shape restrictions of orders for the automatic sorter. Besides, on busy days other outbound lines -having manual sorting stations- anyway have to be utilized to meet the daily demand. Outbound line 107 and outbound line 108 can be considered more or less the same since the type of orders that can be assigned to these outbound lines is the same. Generally, either outbound line 107 is in production or outbound line 108 is. The utilization for outbound line 106 is remarkably low, as it is used for specific cases only (mainly related to manual cut-off decisions). Since outbound lines 108 and 109 include almost all of the multi-orders, the rest of this research focuses entirely on these outbound lines.

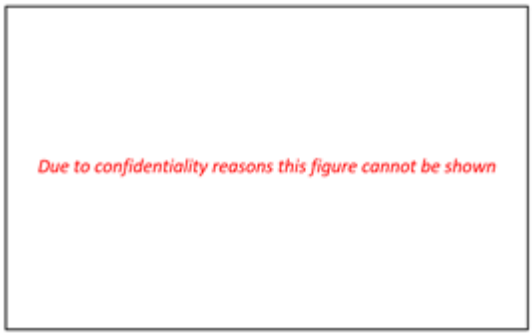

FIGURE 2.4: Utilization of outbound lines

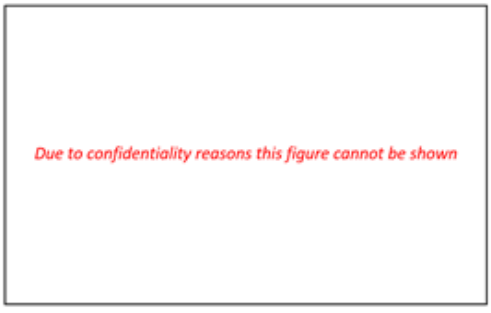

FIGURE 2.5: Average picking times outbound lines

As already mentioned (see Section 2.2), the pooling algorithm processes each outbound line separately; orders are assigned to an outbound line in a preliminary stage. This utilization (the number of orders that are processed) has impact on the average picking time of bol.com, which can be seen in the

average picking times per outbound line in Figure 2.4. The average picking times are significantly higher as the utilization decreases. So, when more orders are processed by an outbound line, the picking productivity increases. This is related to the size of the solution space of the available orders per outbound line; when there are more options are available the algorithm is more likely to come up with more efficient solutions.

2.3.2 Zone distribution

By the pooling method, selections of zones are determined which can fulfil all items of the selected orders. The number of zones that are (needed to be) used influences the picking performance. And since bol.com applies randomized storage policies, unique items could be stored in multiple zones. This results in very unique order possibilities for all the multi-orders. For many orders, at least more than just one zone is needed when assigning orders into pools. On average, 3.8 zones are used per pool. In Figure 2.6, the distribution of total used zones per pool is shown. Cumulatively, 91% of the pools contain less than 10 zones. In Figure 2.7, the average picking time of an average item is plotted along with the number of zones it uses for all the items the pool contains.

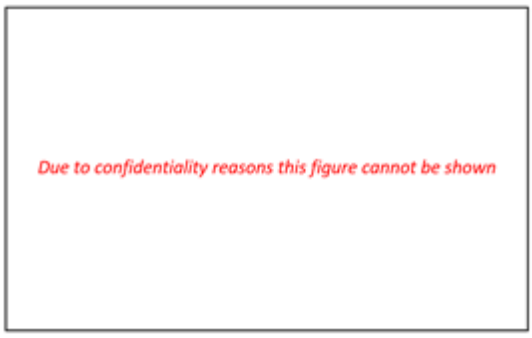

FIGURE 2.6: Distribution of zones used over pools

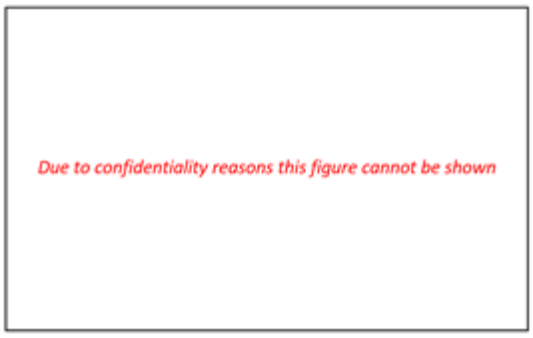

FIGURE 2.7: Average picking time per item in pool

When assigning items to the zones, there are still different strategies how to assign the items to the determined zones. If orders are available in multiple zones, we have to find the right logic that can be applied to achieve the best result in terms of picking productivity. By the historical data shown in 1.7 at Section 1.3.2, a reversely shaped exponential pattern was observed by the relation between the average time per pick and the poolzone sizes. By the historical data, average insights into the performance of zone combinations can be performed. To illustrate, let the variables $n_a, n_b, .., n_z$ be hypothetical poolzone sizes of zones $(a, b, .., z)$ and N be the total orders in the pool ($N = n_a + n_b + .. + n_z$). Furthermore, the obtained data of the average time per pick based on the poolzone is noted as $Averagetimeperpick(n)$. Then, the estimated average time per pick can then be obtained by the formula given in Equation 2.1.

$$\frac{n_a * Averagetimeperpick(n_a) + n_b * Averagetimeperpick(n_b) + ... + n_z * Averagetimeperpick(n_z)}{N} \tag{2.1}$$

In an illustrated example, we vary the distribution over a total of 100 (N = 100) orders over multiple zones. By Equation 2.1, the weighted average picking time over the picked items in the pool is measured. The result of a set of combinations, including a variety of zones that are used, is shown in Table 2.2. As can be seen in Table 2.2, the average time per pick is generally the highest for combinations where the items in poolzone A are maximized. Some results of combinations over 1 up to 4 different zones are shown. As Figure 2.7 showed, the picking time is generally higher the more zones are included in a pool. Next to that, the combinations where the used zones are equal show an interesting insight. Remarkably, the more balanced poolzone sizes are resulting in the highest average picking times. This holds for any number of zones that are used. When varying the N and increasing the number of poolzones even more,

the same pattern is recognized. Thus, maximizing one poolzone size and thereby adding other zones with fewer items outweigh combinations where zones are more balanced in terms of items.

Possible distributions over zones in pool				
zone A (n_a)	**zone B** (n_b)	**zone C** (n_c)	**zone D** (n_d)	**Average time per pick in seconds**
100	-	-	-	1.00[4]
99	1	-	-	1.02
75	25	-	-	1.25
50	50	-	-	1.30
98	1	1	-	1.13
80	10	10	-	1.42
60	30	10	-	1.52
40	30	30	-	1.60
97	1	1	1	1.20
50	30	15	5	1.70
30	30	20	20	1.81
25	25	25	25	1.82

TABLE 2.2: Average time per pick for 100 items over different poolzone distributions

The findings of this section provide the insight that we should focus primarily on using as minimum zones as needed and secondary on the maximization of poolzone size. In other words, all items should ideally be picked in the same zone. However, due to the explained complexity of multi-orders, this is regularly not possible. Therefore, the zones that are used should be exploited to the maximum extent that is possible.

2.3.3 Complexities of multi-orders

After discussing the performances over a broad perspective as a pool collectively, more in-depth insights over the characteristics of multi-orders are considered valuable as well. Insights on which characteristics make it hard to assign orders are explained in this section. Since orders arrive continuously (see Section 2.2.3), bol.com the number of available orders dynamically changes over time. Since bol.com is not operating during the night, average performances could be measured by time periods of one day. For each moment in time, a pick run is requested, the list of orders is updated. Access to these order lists is permitted by bol.com. which enables us to retrieve multiple of these order sets during a day. By filtering the unique orders and the orders that must be picked today, a reliable overview of the order lists that have to be picked each day is obtained. Based on data sets of ten different (randomly chosen) days, analysis is performed on the size, quantity and cut-off time of orders.

The size of orders

Since multi-orders consist of multiple items, they can vary a lot in size. For the size of a multi-order, two dimensions can be interpreted: the number of unique items per order and the quantity per item. In Figure 2.8 and Figure 2.9, the average data of both dimensions are presented in a fraction of the total orders sets. Note that correction errors are applied since not the exact number of orders could be obtained. Nevertheless, both figures represent a reliable reflection on the item distributions of orders over an average operating day.

As can be observed, minor differences are observed between the outbound lines. Note that when an order consists of only one unique item, it has to consist of an item that has a quantity of more than 1. The majority of the orders consist of two unique items, where usually the average quantity per unique item is just one. Exceptional values of unique items or quantities per item more than 5 are very rare.

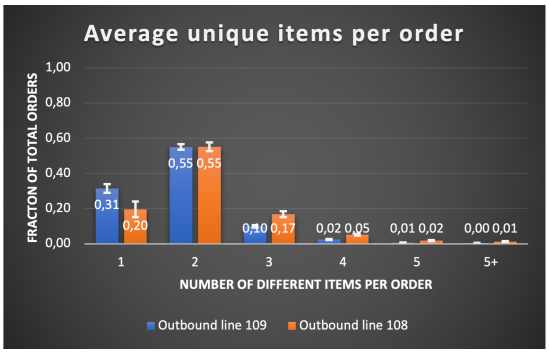

FIGURE 2.8: Average size per order in unique items

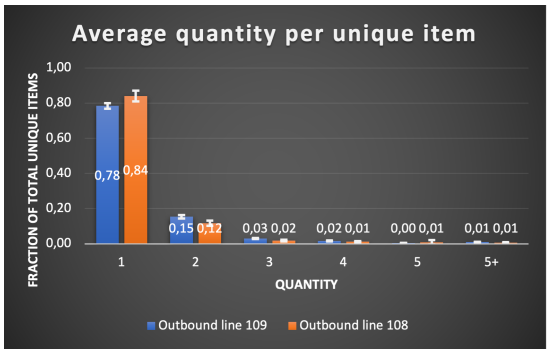

FIGURE 2.9: Average quantity of unique items

Availability of items

Another important factor of items of multi-orders is the stock availability among the potential zones an item could be allocated to. Since the same type of item could be located at multiple zones (Section 2.2.2), the stock per item varies heavily and changes dynamically by the randomized storage policy bol.com applies. Each item has its unique stocking locations, distributed over the 52 zones. In Figure 2.10, the average storage distribution over the number of zones wherein the item is available is shown. Since there exists no discrimination in stock allocations of items belonging to multi-orders per outbound line, all items are considered generally. As the figure shows, most items are available in just a few zones only. Note that by combining zones, the probability of fitting all orders may become complex by the endless combinations that can be made since there are 52 zones in total. For example, when selecting three zones for a pool, already 22.100 combinations are possible.

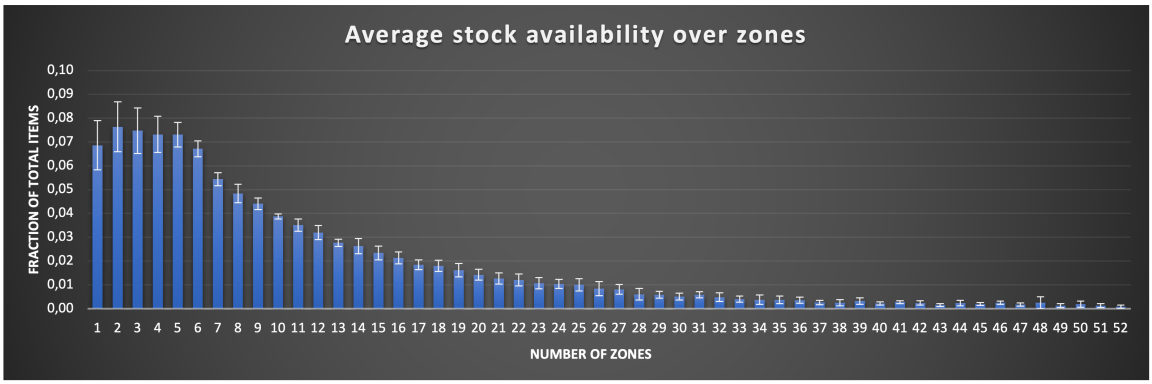

FIGURE 2.10: Stock availability distribution over the zones

Cut-off times of orders

As mentioned, the cut-off time is a hard restriction in the fulfilment of all orders. The cut-off times vary per distribution carrier. In Figure 2.11, a representative distribution over different cut-off times over a day is shown, including its average fraction of occurrence. Again, the results per outbound line show more or less the same pattern. As operating days at bol.com usually end before 1:00 AM, an interesting observation is that in total around 96% of all orders contain a cut-off time afterwards. The fractions of orders having a cut-off time during the day (either at 12:30 PM, 2:15 PM or 10:45 PM) are very small. Since the operation starts early in the morning, that leaves quite some time (and some pick runs) to fulfil all orders on time.

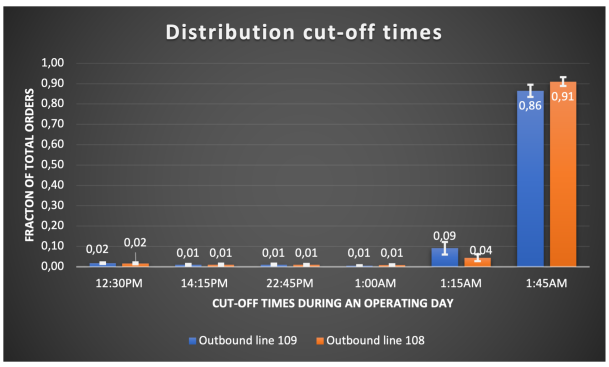

FIGURE 2.11: Cut-off time distribution over the orders

2.4 Conclusion

Analyzing the algorithm step-wise lead to a series of interesting observations. First, it was noted the sorting criteria are in a way related to the cut-off time. Remarkably, no other additional complexities of orders are considered (the difficulty score is currently inaccurate). Nevertheless, investigating the cut-off times of all multi-orders on average, we observe regularly 96% of all warehouse orders contain a cut-off time after the end of a usual operating day. Conspicuously, this high majority of orders have no other prioritization criteria besides its cut-off time. Though, it is shown multi-orders are varying heavily; the number of items per order and the picking possibilities based on the stock availability indicate the needed zones, and thereby the complexity, of assigning a multi-order.
When adding orders to the pools, the seed order strategy appears to be a haphazard way of determining zones of the pools, since this decision is currently based on one single order only. The rest of the orders coincidentally have to fit that zone selection, while we learned that the use of zones is of big impact on the average picking time. When allocating items to the zones efficiently, two learnings are identified: 1. We should minimize the number of zones in use and 2. When allocating the items over multiple zones we should maximize one poolzone to the highest possible extent. The more balanced distributions of items over poolzones are outperformed, as is shown in 2.2.

After determining the zones and selecting the first orders of the pool, the effect of the fill rate optimization rule (step 1b) is vague; since the earlier determined zones remain unchanged probably not many extra orders will fit. Besides, the fill rate estimation is very rough since it is measured in volume and not in exact dimensions, which reflects the historical data; for only 11% a fill rate of more than 90% per tote is achieved. The same dubiety holds for step 2, which combines small pools by the small chance they can be merged; it is more productive to create efficient pools instantly.

Thus, the current seed order selection rule seems to be the bottleneck in the pooling process. The pools are limited by the determined zones for one specific order, while in all likelihood more efficient combinations could be made. In the next section, existing literature is studied to identify related problem environments, and collect a set of appropriate alternative methods. One crucial factor to consider is the allowed computation time, which is only four minutes for all outbound lines in total.

Chapter 3

Literature review

The goal of this literature review is to provide insights into the current state of academic research on how to assign items, belonging to multi-orders, to zones in a pool efficiently in terms of average picking productivity. Existing literature related to constructive heuristics, metaheuristics and other techniques studied in the warehousing field are reviewed on their applicability to the problem environment described in the previous chapter. In section 3.1, related problem definitions are given to identify our research problem by the earlier performed literature studies, including mathematical formulations that might be of use. Based on this problem identification, appropriate methods are explored in sections 3.2 and 3.3. A distinction is made between constructive methods that will create initial solutions by themselves and on the other hand more advanced methods, that strive for optimization of already existing solutions. Furthermore, insights about parameter estimations regarding the picking time are addressed in section 3.4. These insights give valuable inspiration for the determination of our output that we need to define later on in this research.

3.1 Problem related definitions

The pooling allocation problem, exposed in chapter 1, has many specific characteristics which should be considered. First, the order type is of importance, since multi-orders is the main motive why pooling is needed. In the existing literature, the concept of multi-orders is at an early stage: in total, one research is available that includes this order type in their research scope. As a result of this scarcity, pooling strategies are not addressed at all.
A second factor is a continuous arrival that should be included in the decision making of selecting orders during different time periods. In the literature, this element can be identified by the concept of wave picking. This concept is focusing on selecting the best set of orders for a specific moment in time, based on the chosen objective criteria (in literature mainly focused on minimizing latency) (Liang, Wu, Zhu, & Zhang, 2019). Though, many (the relatively older ones) did not take this online order arrival into account. This is explainable by the fact that the speed of customer delivery faced a highly competitive development in the last couple of years.
A third factor is the type of order picking. The order-picking system that is considered in this review is the picker-to-parts system since the order moves to the items to fulfil the orders. Above that, the system picker-to-parts system is a low-level picker-to-parts system, since all the items can be picked without the additional help of machinery (R. de Koster et al., 2007). Within this system, the specific zoning method has a big impact on the picking strategy of bol.com.

For picker-to-parts systems in general, lots of research have been performed and many different problems relating to picking productivity are already thoroughly investigated. The work of van Gils et al.(2017) provides an overview of research performed in all different problems in the warehousing field, where we observe minor research attention is going to zoning related planning problems so far. Above that, studies that are addressing zoning are analyzing the appropriateness of whether to use zoning or not, instead of developing optimization techniques on how to use zones efficiently. These studies support the use of zoning by confirming the hypothesis that for the right batch sizes the mean throughput time of batches can indeed be decreased by using zones (Yu & de Koster, 2008).

In contradiction to zoning, many researchers are investigating problems related to storage assignment, batching in general and routing. Next to that, a lot of research has been performed in integrating and combining these problems. By the study of Scholz et al. (2017), multiple successive problems are combined in one model (order batching, assignment, sequencing and routing). By their research, they conclude integration of multiple problems is pivotal for warehouse organizations to be efficient. Another research by van Gils et al. (2016) is showing the interaction of multiple warehouse problems (even including zoning problems) can significantly contribute to an improved picking productivity. The research of Marchet et al. (2010) is even more interesting, as they consider a pick-and-sort system, where the orders need to be sorted after being sent by the conveyor. They are studying the impact of wavelengths in a number of orders per wave, to minimize idle times and thus, improve efficiency. This side effect of the main objective of this research, the picking productivity, should be considered when evaluating potential solution methods.

A general remark all researchers in the warehousing field agree on is that the computation time remains a difficult issue, and therefore optimal solutions for large-scale instances are very complex. In practically every study case, the order size is usually not higher than 1,000 orders. In integrating problem approaches, many precedence constraints as cut-off times are currently unexploited and need more attention before they can be of practical value to real-world warehouse organizations. Van Gils et al. (2017) supports this argument, by their findings that the simultaneous formation of batches and routing for every pick batch was also very extensive in terms of computation time.

3.1.1 Online order batching problem

Since any similar pooling strategies are not observed in the literature, it is hard to link directly to one of the currently investigated order picking related problems. Therefore, the inspiration of a closely related problem is obtained: the Online Order Batching Problem (OOBP). The Online Order Batching Problem is defined as the problem of how to divide orders in an efficient composition of groups (in literature defined batches). This optimal composition of batches should minimize the total processing time or total lateness while assigning customer orders to batches (Le-Duc, 2005). In contradiction to zoning and multi-order strategies, lots of different studies have been performed in this field.

Comparing the problem to the situation of bol.com, this problem is related to the process of pooling and batching combined (and for mono-orders only). The online aspect included in the problem implies new customer orders can arrive at any moment during an operating day. Since some of these orders have to be picked, packed and shipped the same day due to the delivery services bol.com wants to provide for their customers, the decision set which orders to pick is very dynamic. In general, warehouses applying online batching have a wave-based scheme to keep in control of the responding speed and the operating efficiency. Within this efficiency, the minimization of order picking times is included (Liang et al., 2019).

Multiple mathematical models have been constructed to solve this problem, either to minimize the total distance or to minimize the tardiness of orders. Research has found out that at least 50% of the time spent on picking activities is spent on travelling distances. The remaining fraction includes set up times for each batch, searching time for the items on the shelves and the time to collect the items in totes or on carts (Tompkins et al., 2010).

For an indication of the mathematical notation of a general order batching problem, the basics of this problem is studied. Let set $I = \{1, ..., I_n\}$ be the set of all feasible batches and $J = \{1, ..., J_n\}$ be the set of all orders. The decision variable can be noted as x_i, which expresses the choice whether batch i is chosen $(x_i = 1)$ or not $(x_i = 0)$. The cost function, by (Gademann & van de Velde, 2007) and (Aboelfotoh, Singh, & Suer, 2019) expressed as the the total length of a picking tour of all the order placed in batch i is given by d_i, where $(i \in I)$. This implies an optimal route for a given batch is a picking route that minimizes the length through the warehouse to pick up all the items of one batch. The costs are expressed in terms of distance in this model. Furthermore, let C denote the capacity of

the general picking device and c_j be the required capacity of order j, where $(j \in J)$. The vector a_{ij} denotes every feasible batch of customer orders, for which the entry parameter a_{ij} states whether order j is included in batch i $(a_{ij} = 1)$ or not $(a_{ij} = 0)$. By the given information, the mathematical problem can be constructed in such a way the aim is to minimize the total travelled distance when grouping the orders into batches.

$$\min \sum_{i \in I} d_i x_i \tag{3.1}$$

Subject to:

$$\sum_{j \in J} c_j a_{ij} \leq C, \forall i \in I \tag{3.2}$$

$$\sum_{i \in I} a_{ij} x_i = 1, \forall j \in J \tag{3.3}$$

$$x_i \in \{0, 1\}, \forall i \in I \tag{3.4}$$

In this model, constraint 3.2 restricts the total capacity of the number of orders that could be batched together may not exceed the capacity of one picking device, and thus, the maximum capacity of one batch. Constraint 3.3 ensures that every order is included in exactly one of the total set of batches, while constraint 3.4 supports that by the binary constraint of x_i.

The OBP is classified as an NP-hard problem since it is proven that it is not solvable in polynomial time for each case where batches consist of more than two orders (Gademann & van de Velde, 2007). In addition, many other researchers came to the same conclusion in their studies (Pan & Liu, 1995) (Hsua, Chen, & Chen, 2005) (R. de Koster et al., 2007). According to them, it is complex to achieve optimal solutions for large-scale problems within an acceptable computing time. Due to this complexity, approximation algorithms like local search algorithms are advised to handle this type of problem. These approximation algorithms should be modelled to find acceptable solutions in limited periods of computation time.

3.1.2 Assignment problem

Next to a practical warehouse related view, the problem can be analyzed in a more abstract perspective. Since decision making in pooling regards the assignments of items to specific zones, the problem can be considered as an assignment problem. An assignment problem (also called matching problem) is a fundamental combinatorial optimization problem, which objective is to find optimal matching between a number of agents and a number of tasks (Valencia, Alfaro, Martinez, Magana, & Perez, 2018). The items of multi-orders available in BFC1 at bol.com can be seen as tasks, while the zones are several agents to which the item could be assigned. By an efficient assigning strategy, the objective value in terms of picking times can be reduced. Similar to the order batching problem, the assignment problem is proven to be NP-hard (1997). From an abstract view, the order batching problem is a disguised assignment problem depending on capacity, since the orders are grouped to an agent (batch) that can handle multiple jobs based on capacity restrictions.
Furthermore, an important difference we note is the freedom of choosing orders in the solution or leaving them out (and what the costs are of leaving them out and selecting them later). To the extent of our knowledge, less research has been performed in assignment problems where not all items are obliged to be chosen in the solution, which is the case at bol.com.

Bipartite Graphs and Combinatorial Matching

In most assignment problems, a perfect matching solution is requested. This implies every order in the set can only be assigned to one of the available tasks. This results in a symmetric graph instance where the cardinality of both agents and tasks are equal, of which an illustration is represented in Figure 3.1. When the cardinality is unequal, we speak of asymmetric instances, which can be related to this research case (the tens of thousands of orders per day can be seen as tasks and the 52 zones as agents). The research of Hamzehi et al. (2019) provide insights into the performance of reinforcement learning for both types of problem instances, as they have obtained some good performing results for their reinforcement learning based-algorithms applied to small symmetric bipartite matching problems. However, the same algorithms applied to large-scale asymmetric instances are showing bad results in comparison to linear programming-based and nearest neighbour-based algorithms. The use of reinforcement learning is therefore not recommendable to the large-scale research problem we encounter at bol.com.

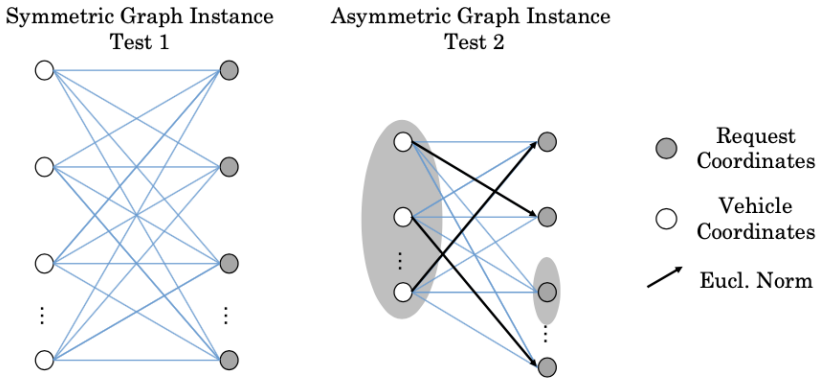

FIGURE 3.1: Illustration of symmetric and asymmetric instances

3.1.3 On-time delivery problems

Generally, in assignment problems, the main goal is to minimize tardiness. However, by measuring tardiness this parameter will only take values for the moment in time it is already too late. Though, the research of Mora et al. (2020) defines an interest objective function to their model, which is used to optimize an efficient schedule of assigning batches of orders to their production furnaces. The basis of the problem defined corresponds to the order batching problem of section 3.1.1, except that there is an additional element included that measures the total tardiness of orders. Further on, both elements (1. the cost value based on the processing time, 2. the tardiness) are prioritized by a factor $W_{production}$ and $W_{tardiness}$. The objective function will then be as follows:

$$\min V = W_{production} * \sum_{m=1}^{M} (F(n_m)) + W_{tardiness} * \sum_{j=1}^{O} (T_j) \tag{3.5}$$

In equation 3.5 above, the total production time is measured by $\sum_{m=1}^{M}(F(n_m))$ for every machine m and $\sum_{j=1}^{O}(T_j)$ is the total sum of the tardiness. Further on in this research, a case study was applied successfully, leading to optimal results including approximation techniques that came quite close to the optimal scenario's. However, the case study consists of too many other parameters and variables that are not comparable to the research problem in this thesis. Nevertheless, the method of prioritizing the processing time and tardiness is quite interesting (Mora et al., 2020).

In another study, another prioritizing rule is performed by assigning weights to customer orders. In this case study, the weight is embedded in the objective function. As a consequence, the orders that are most important (and thus given high weights) are be more likely to ship on time (Azadnia, Taheri, Ghadimi, Saman, & Wong, 2013).

3.2 Constructive heuristics

Constructive heuristics are methods that start with an empty solution and repeatedly extend the current solution until a complete solution is obtained. The process findings of this complete solution rely on predefined, or randomly chosen iterations. In this section, we describe four types of constructive methods: priority rule-based algorithms, seed algorithms, greedy heuristics and savings algorithms.

3.2.1 Priority rule-based algorithms

Priority rule-based heuristics involve two main stages; at first, the available orders are sorted by a given criterion, which determines the priority of the order. Thereafter, the orders are assigned to different batches sequentially according to the priority rule.

The most common example of a priority rule-based heuristic is the First-Come First-Served (FCFS) rule, where the items are assigned in order by the timestamp they arrive into the process. However, for the order batching problem this method is not very effective (Alonso-Ayuso, Albareda-Sambola, Molina, & de Blas, 2009). Often, the FCFS heuristic is used as a baseline to compare the different constructive solution approaches. As an example, the study of Albareda-Sambola et al. (2009) adds the First-Come-First-Served (FCFS) strategy as a reference to benchmark their results of other, more complex, methods. Another common heuristics, mainly used for benchmarking, is the Earliest Due Date (EDD) rule. As the name of the method implies, the order is sorted on their due date. Henn and Schmid (2013) use this method as a benchmark and propose two metaheuristics that significantly improve this standard heuristic; iterated local search and the attribute-based hill climber. These methods are explained later on in this chapter.

The motivation to apply a distinctive priority rule can differ per situation: bol.com manually prioritizes orders because of approaching cut-off times of orders, to ensure orders are shipped on time by the belonging transport carrier. Despite this manual prioritization, creating pools by a standard priority rule as the earliest due date is guaranteed to be inefficient since the availability of items in zones causes additional complexity.

3.2.2 Seed algorithms

Another constructive approach is the use of seed heuristics, which corresponds to the current heuristic of bol.com. The seed algorithms are efficient and easy-to-implement heuristic algorithms to solve the order batching problem (M. de Koster, van der Poort, & Wolters, 1999). The seed algorithm consists of two main steps. First, from the available order set a seed order is selected based on a specific seed selection rule. Secondly, available (and thus not yet selected) orders in the order set are added to the seed order to form a batch (in our research pool) until the capacity constraint is reached. For the selection of orders, additional rules have to be defined. A schematic overview of this logic is shown in Figure 3.2 (Zhang, Zhang, & Zhang, 1999).

For each batch, the procedure starts again till the capacity constraints are met. In most problem instances, the order addition rule is based on a measure of ‘distance’ regarded to the seed order (R. de Koster et al., 2007). The work of Ho et al. (2008) and Ho and Tseng (2006) proposes in total fourteen seed selection rules for their algorithms, which were all aisle-based or distance-based concerning the picking locations. The exact routing was included in their scope of the order batching problem. The study of Pan and Liu (1995) addresses the prioritizing on the weight of orders, while Koster et al. (1999) also include a random seed selection rule in comparison to their other distance-based rules. Since a combination study with either multi-order items or zoning methods are not addressed in the literature, no relevant seed and order addition rules are worth mentioning in detail.

3.2.3 Greedy heuristics

Within greedy algorithms, choices are based on what looks like the best option at one specific moment. A greedy heuristic can be interpreted as an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage. For job scheduling, examples for this optimal choice are

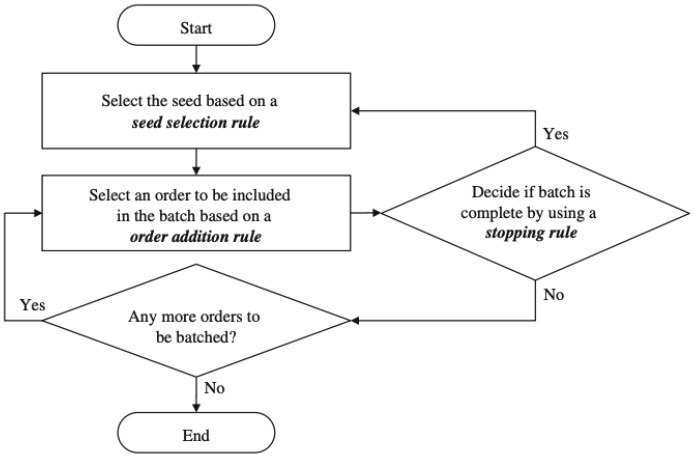

FIGURE 3.2: Flowchart seed algorithm

processing jobs by the closest due date or the earliest completion time. Priority rule-based heuristics are also a type of greedy heuristics. However, in comparison to the general priority rule-based algorithms greedy heuristics can involve more advanced metrics (and even involve multiple decision stages) that are determined in front of the algorithm. The purpose of greedy heuristics is to take no more than a few minutes to obtain relatively results. For real-time requirements and large-scale problems, this is appropriate and very effective (Shao, Shao, & Pi, 2021).

Greedy randomized algorithms

Borras et al. (2020) propose a two-phase method for the online batching problem, where the first part consists of a constructive heuristic that is based on the combination of greediness and randomization in the made decisions. The method is called Greedy Randomized Adaptive Search Procedure (GRASP). The method first initializes a list of candidate solutions and sort them on determining weights by a greedy function prioritized on heaviness (in weight, descending order). Thereafter, they convert this list a restricted candidate list based on a random fraction that splits up the original candidate list; this way the order that is chosen has to come from the randomly generated percentile of best solutions (in this study, 'best' is measured in terms of heavy items). One order in the restricted candidate list is chosen randomly and is added to the solution. This process continues till a max number of iterations of is maximum allowed time is reached.
As becomes clear by this example, greedy algorithms select the optimal target in each round in the attempt to ultimately gain the optimal results. Although greedy methods can not guarantee global optimization, they are easy to use and are effective in producing near-optimal results (Chen & Lin, 2020).

3.2.4 Savings algorithms

Another type of a constructive methodology is the use of savings algorithms, which was introduced by Clarke and Wright (1964). Savings algorithms classify orders based on time savings s_{ij} obtained by combining multiple orders. For instance, combining order i and j in one tour with picking time t_{ij} in comparison to the time to collect both orders individually $(t_i + t_j)$. The savings of combining order i and j can be calculated by $s_{ij} = t_i + t_j$ - t_{ij}. In their example, the savings were expressed in travelling times, where the routing algorithm will work towards an efficient result when iterating over a calculated savings matrix, and thereby picking the largest savings consistently (Clarke & Wright, 1964).

Henn et al. (2012) look critically towards the method of Clarke and Wright in terms of computation time, since (re)calculating the savings matrix for all possible order combinations heavily increases when a large number of orders is considered. By the research of Koster et al (1999), it is shown that in a comparison study one proposed the savings algorithm outperforms other seed order heuristics. However,

the result is very time consuming and therefore not applicable to large warehouses; already 246 seconds were required for a problem size of 30 orders.

3.3 Local search

Metaheuristics are approximation heuristics that are used to find close optimal solutions in an acceptable computation time. For large problem instances, finding the exact optimum can cost an impossible amount of time. For these situations, metaheuristics are an appropriate outcome. In order batching and assignment problems, different metaheuristics have been studied already. Most of the metaheuristics that have been proposed are local search methods, which are methods that try to improve existing solutions by iterating over small, less time-consuming, changes. In this section, we elaborate on the methods iterated local search, variable neighbourhood search and tabu search.

3.3.1 Iterated local search

Iterated local search (ILS) is a method that is used to improve existing feasible solutions based on neighbouring changing operations. By one small, iterated change, a new solution is generated. The iterated local search method generally consists of two different phases: a local search phase and a perturbation phase. In the local search phase, the heuristic starts from an initial solution and determines a sequence set of feasible solutions. Each element in this set is a neighbour of its predecessor and by comparing the objective values of this neighbour solutions, a local optimum can be found (Lourenço, Martin, & Stützle, 2019). The second stage of the algorithm is the perturbation phase, where the local optimum found at that moment is partially modified and a further local search phase is applied to this modified solution. This perturbation phase involves an acceptance ratio, whereby new solutions are only accepted if it passes that certain criterion. When the acceptance criterion is not passed, another perturbation will take place to hopefully find a better solution. The method ends until a termination condition is met, which is usually based on the total number of iterations or time that has passed. It is conducted ILS can still be time-consuming; the computation time linearly increases by the problem size, specified by the number of orders included (Henn & Schmid, 2013).

3.3.2 Variable Neighborhood Search

In the field of local search methods, different variants of neighbourhood solutions have been applied. One interesting and a systematically different method is the Variable Neighborhood Descent (VND) method, which explores the neighbourhood in a deterministic way by using multiple local search procedures. As the local search principle, the method is used for solving combinatorial and global optimization problems based on the idea of systematically changing the considered neighbourhood to help escape from local optima. (Menéndez, Pardo, Alonso-Ayuso, Molina, & Duarte, 2017).
Albareda-Sambola et al. (2009) applied the method for the order batching problem, where the following three moves were used in their study: (i) assigning an order to a different batch, (ii) swapping up to two orders from one batch with other batches, and (iii) swapping up to two orders from either one or two batches to one or two different batches. Initially, each order was assigned to a batch, after which batches were joined to reduce the total number of batches. The size of the problem was limited since the maximum amount of orders that are processed was 250 (Alonso-Ayuso et al., 2009).

Recurring to the work of Borras et al. (2020) mentioned in section 3.2.3, the second step of the method applies Variable Neighborhood Descent. The VND procedure they suggest is based upon searching neighbour solutions by the following three strategies:

1. An **inserting** strategy that considers all possible solutions reached by the interchange of any pair of orders in a different batch in the solution, into all other available batches.
2. A **first swapping** strategy that considers all possible solutions reached by the interchange of any pair of orders in a different batch in the solution, into all available batches.
3. , At last, a **second swapping** strategy is applied, which considers all possible solutions reached by the exchange of every pair of two orders within the same batch.

If a neighbourhood strategy is not able to improve the current solution, then the method jumps to the next available neighbourhood until the maximum number of neighbourhoods is reached. When an explored neighbour solution improves the current solution, the best solution will be updated and the method starts again from the first neighbourhood strategy. An important criterion for each neighbourhood solution is that all solutions that are considered must be feasible. Otherwise, they are skipped immediately (Gil-Borras et al., 2020).

Another recent study by Cano et al. (2021) addresses variable neighbourhood structures, where destroy and repair heuristics are applied to change the solution. Performing this method, a part of the current solution is split up partially and built again by a certain rule of logic. This more rough way of changing the solution prevents the algorithm from being confined to local optima (Kuhn et al., 2021).

Researchers studying the Variable Neighborhood Descent method state the method is able to improve other priority-rule based heuristics significantly (Menéndez, Pardo, et al., 2017) (Alonso-Ayuso et al., 2009) (Menéndez, Bustillo, Pardo, & Duarte, 2017). Due to its generality, they recommended exploiting this method for real-world situations. Though, complex issues like due dates are not yet considered in the solution approaches so far. This corresponds to the overview of Barras et al. (2019), where researchers state the same about the absence of studies combining the order batching problem with on-time delivery constraints.

3.3.3 Tabu search

Another local search method that has been applied to the order batching problem is tabu search. In addition to the other local search procedures, tabu search keeps track of a tabu list that records move applied in previous iterations. Applying moves on the list is forbidden ('tabu') when iteration towards an optimum to avoid cycling and diversify the search. Thus, tabu search considers only those elements of the neighbourhood as new incumbent solutions that can be obtained by a non-tabu move. The tabu list is updated after each iteration (Kulak et al., 2012).
The effect of the tabu search method is studied by Henn and Wascher (2012), who recommend tabu search on large-scale warehouse instances. However, this is only a recommendation since their performed experiments did include at most 100 orders. Nevertheless, the concept of tabu solutions can be integrated into other local search methods when the problem is likely to get stuck in a local optimum quickly. Though, updating the tabu list causes additional computation time, which has to be considered concerning its added value.

3.4 Picking parameter estimations

To measure the picking productivity of pooling, we should define a parameter that could evaluate the performance of pooling strategies. Since the picking productivity is dependent on the pick batches that are created after the pooling strategy, analysis of how this picking time originated is required. According to Caron et al. (2010), the picking time for one pick batch is composed of two elements: a constant value for the set-up time of the pick batch and the time to execute the picking route. The time devoted to executing the picking route by operators is assumed to be constant and can be divided into two different processes: the time to walk between the different locations to pick the items of one pick batch and the time to perform a pick while being at the right location. The walking time is expressed in the time it takes to walk one additional meter by the total distance that is travelled for the pick batch. Next to that, existing literature supports that the expected distance can be estimated by creating an exponential distance savings function, as long as the storage policy and the number of (cross-)aisles remain constant (Duc & Koster, 2005).

3.4.1 Estimation fitting methods

Predicting the picking time per pick batch is useful for evaluating the pooling method since this represents the output of the pooling method in general. To predict the average time of a pick batch, we have to find an approximation function of the expected picking time per pick batch, based on the number of

items we assign to a zone (since this decision determines the pooling performance). One of the most common known tools for fitting an estimation function based on a set of historical data is the (weighted) total least squares method. This method sums for each fitted value the quadratic difference among predetermined sets for all involved parameters. As a result, the best values per parameter are identified that obtain the least-squares. By adding weights to the fitted values, the concept remains the same but more focus is shifting to the heavier weighted values. This might be recommendable by data that is showing inconsistencies across the explainable variable used as input. This accurate parameter estimation method is very easy to use and appropriate for linear and non-linear estimations (Malengo & Pennecchi, 2013).

To evaluate fitting models, the bias can be calculated. The bias is the difference between the actual value and the observed value. The bias is computed by summing all errors (observed value - actual value) per estimate and dividing them by the total number of estimates. When the bias is around zero, the estimates were unbiased, and the function delivers unbiased results. If a function has a consequent bias (either positive or negative), bias-corrections can be applied to reduce the average error of the function (Mejari, Breschi, Naik, & Piga, 2020).

3.5 Conclusion

In this section, we discussed the most important findings the literature could provide us. The concept of pooling is an unknown term in existing literature since multi-orders (or any other equivalent term implying orders that consist of multiple items) are barely addressed. Therefore, the online order batching problem is the most relevant problem related to this research problem, which provided some useful mathematical interpretations which fundamentally serve as input for the mathematical definition of this research problem in the next section. Though, since the online order batching problem is mainly focused on distance as its cost function, so the model should be adjusted to a more abstract (asymmetric) assignment problem we described in subsection 3.1.2. In contradiction to the assignment problems defined in the literature, this model should include non-committal decisions to allocate items to the available zones. Regrettably, warehouse studies including zoning strategies are scarce. For the studies that applied to zone, mainly small warehouse instances were investigated.
Next to that, the output of the pooling method can be estimated by a derivation of the picking time of a pick batch. By comparing the estimation to the historical data of bol.com, where a weighted least squares method should be applied to obtain optimal values for the output function.
Many researchers appoint the complexity of solving online order batching problems in reasonable computation time, as this heavily increases among a wider problem scope. Since bol.com is allowing limited computation time in practice, approximation heuristics must be applied to effective solutions in a time-efficient way. Greedy heuristics and seed order heuristics are considered to be appropriate for time-restricted problems. However, any current logic for the unique problem environment of bol.com is not provided by the literature. So, constructive methods have to be designed by logical reasoning based on the research environment that is discussed.
Furthermore, improvement methods are discussed in this section. Methods like the savings algorithm (3.2.4) and the local search methods (3.3) are showing promising results on small-scale instances. Though, since the limited computation time and the heavily sized problem instance of bol.com, these more advanced methods are not very appropriate. Besides, implementing these methods comes with risk since all of these methods are not yet applied on large-scale instances.

Chapter 4

Mathematical formulation

In this chapter, the mathematical formulation of the problem is introduced. Next to an explanation of the problem context, an approximation function for the pooling performance is derived. Successively, the mathematical model is being defined. Thereafter, the solvability of the modelled problem is discussed.

4.1 Problem context

This problem contains the decision making in assigning items belonging to multi-orders into pools. Within these pools, the items are allocated to zones. These lists of items are the output of the pooling process. Since orders are preassigned to specific outbound lines, the model can be considered for each outbound line separately.

As explained before, one multi-order consists of multiple items. Every unique item of a multi-order must be added to the same pool. The main decision of this research problem is to allocation the items of multi-orders to zones such that the best selection of orders (of all thousands of orders usually available) is chosen in terms of picking productivity. Though, multi-orders vary heavily in the number of unique items and their quantities per item (Section 2.3.3). By formulating the quantity per unique item belonging to a multi-order as an integer parameter, proper constraints can be defined to ensure the solution remains feasible.

The maximum capacity of a pool is determined on order level since that is based on the sorting capacity that plays a role in a later stage of the outbound supply chain (see Appendix A). This implies the size of multi-orders is not considered when selecting orders for the pools. Since this way of working is organized in alignment with other warehouse activities such as sorting and packing, these parameters are assumed to be fixed in this research.

Another important parameter related to this problem is the available stock in BFC1. For each item, we have to know the available quantity in each zone at every moment a pick run is requested. The warehouse management system of bol.com keeps track of this real-time stock inventory, which updates every time a new pick run is sent into the process. Since the exact picking location within the zone is irrelevant in this phase, the total available quantity per zone satisfies the necessary input for this model.
The solution of the mathematical model should indicate the performance of the picking productivity, measured in time that is taken to pick all items of the chosen orders. The size in the number of orders per pick run is varying; the demand of the pick run (request by the control room) can be exceeded since after every created pool it is evaluated if the total picked orders meet the demand of the control room. and the maximum pool size is determined on order level, the number of items in pools is unequal since multi-orders vary in size. For this reasoning, the solution should obtain the average picking time of all items that are included in the pick run. The derivation of the estimation function is explained in detail in the next section of this chapter.

Focusing on the mathematical notation, sets that we need to define are the orders o, items i, zones z and pools p. For simplicity, only the pooling allocation of one pick run is considered. After modelling this problem, we elaborate on how the performance of multiple pick runs should be evaluated (Section 4.4). For a pick run, the requested number of orders should be noted, which is interpreted as the demand of orders that should be picked for one specific outbound line. As this demand is varying over time, the

number of total pools cannot be determined in advance. Also, the pool sizes (expressed in number of orders) are not fixed, as the size varies between a minimum and maximum capacity depending on the outbound line (Again, see Appendix A).

The availability of current stock can be noted by the integer parameter a_{iz}, that represents the quantity of item i in zone z. Secondly, parameter (b_{io}) connects items and orders by denoting the quantity of all items i belonging to order o. By this given information, the sets, input parameters and decision variables can be formulated as follows:

Sets

$I = \{1, ..., \|I\|\}$	Set of unique customer items
$O = \{1, ..., \|O\|\}$	Set of customer orders
$Z = \{1, ..., \|Z\|\}$	Set of the 57 zones
$P = \{1, ..., \|P\|\}$	Set of pools

Parameters

a_{iz} = The amount of stock of item i that is available in zone z
b_{io} = The quantity of item i belonging to order o
D = Demand requested by control room in number of orders, size of the pickrun per outbound line
C_{max} = Maximum capacity of a pool

Decisions

x_{izp} = The quantity of items i assigned to zone z in pool p

$$y_{op} = \begin{cases} 1, & \text{if order o is assigned to pool p} \\ 0, & \text{otherwise} \end{cases}$$

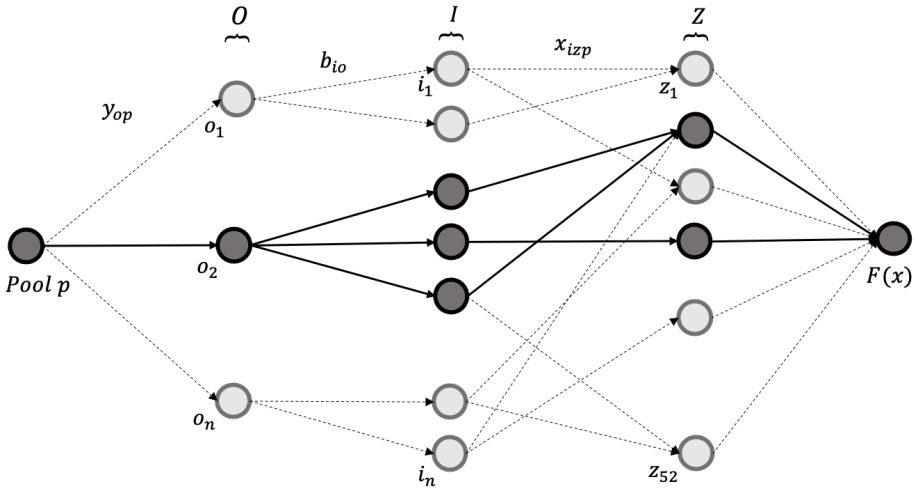

FIGURE 4.1: Schematic view of all involved sets, parameters and decision variables

The decision variable x_{izp} is based on the allocation of all items i, belonging to all orders included in pool p, to zone z. To support the mathematical formulation, a schematic relationship diagram of the set, parameters and variables is presented in Figure 4.1. For example, the decision of allocating the items of order o_2 to zones z_2 and z_4 is highlighted. As can be seen, the decision variable x_{izp} relates to the decision y_{op} and parameter b_{io}. Additional constraints have to ensure these depending variables are defined properly, which implies when orders must be fully allocated by its all belonging items in the same pool, otherwise the order (and thus none of its items) can not be included in the pool.

4.2 Picking time approximation

The feasible solutions of the model are measured by an objective function, which should measure the performance by means of picking productivity. An estimation of the average picking time per item can be obtained by dividing the total expected picking time by the number of all items that are included in all pools. The total expected picking time can be computed by summing the expected time of all individual poolzones, for which we have to construct an estimation function based on the number of items it contains. For one poolzone, the term n_{zp} expresses the total number of all items assigned to zone z in pool p. This number can be derived by Equation 4.1.

$$n_{zp} = \sum_{i \in I} (x_{izp}) \tag{4.1}$$

By knowing the number of items assigned to zones in all the pools, the total number of assigned items in the pick run can then be computed by Equation 4.2.

$$N = \sum_{z \in Z} \sum_{p \in P} n_{zp} \tag{4.2}$$

To approach the expected picking time accurately, we first analyze the expectation of the picking time of one pick batch. By the work of Caron et al. (2010), three main parameters can be defined that influence the duration of a pick batch (see Section 3.4):

1. α - The set-up time of a pick batch
2. β - Travelling speed of an operator
3. γ - Time to perform the pick

By deriving an accurate general function that can be applied to every zone, it is important every zone is every zone is approximately the same in terms of the number of aisles and storage capacity. Therefore, we rule out the five flammable zones (and thereby orders containing items stored in this type of zones) in this model, since these types of zones have a significantly smaller storage surface. Furthermore, all aisles in the well-nigh 52 identical zones BFC1 are wide enough to avoid congestion when multiple order pickers are operating in the same zone; operators can pass each other in an aisle. Therefore, by finding the right values for all parameters we can theoretically calculate the expected picking time precisely. The average expected picking time for one pick batch is based on the formula given in Equation 4.3.

$$\text{Average time pick batch} = \alpha + \beta * Expectedtotaldistance(n_{picksinbatch}) + \gamma * n_{picksinbatch} \tag{4.3}$$

The set-up time (α) and the time needed for one pick (γ) is expressed in seconds. The expected total distance is measured in meters and is currently not known, but an estimation can be derived from the data. Furthermore, note that n_{picks} denotes the total number of picks that are included in the pick batch. When deriving this formula into an approximation of one single pick, the formula should be divided by the total number of picks the pick batch contains. Note that the time needed for one pick (γ) becomes a constant factor in the equation, which is assumed to be the same for every item (Equation 4.4). We can support this assumption by the knowledge every storage location is easily accessible. The size and weight of an item are also disregarded, since these factors can not exceed the capacity restrictions of a tote, meaning picking the largest or heaviest items will not bring any additional difficulty.

$$\text{Average time per pick in pick batch} = \frac{\alpha + \beta * Expectedtotaldistance}{n_{picksinbatch}} + \gamma \tag{4.4}$$

Converting this formula to the average time per pick in a poolzone, the number of batches should be added to multiply this by the required setup time for each pick batch. The expectation of the total travelled distance is modelled the same as for a pick batch, where n_{zp} now denotes all picks that are included in a poolzone, instead of the used term $n_{picksinbatch}$ for all picks in a pick batch. The time needed to perform one pick remains constant, while the expected number of pick batches and the expected total distance are

both depending on the total number of picks included in the poolzone. By this deduction, we derived the estimation represented in Equation 4.5, where $F(n_{zp})$ expresses the average time per pick in a poolzone.

$$F(n_{zp}) = \frac{\alpha * Expectedpickbatches(n_{zp}) + \beta * Expectedtotaldistance(n_{zp})}{n_{zp}} + \gamma \tag{4.5}$$

Estimations of the expected number of pick batches and the expected total travelled distance need to be derived. Per outbound line, an individual function should be derived based on the historical data of that outbound line. The reason to consider each outbound line separately is the variation in orders assigned to the outbound lines (in general more heavy and larger shaped items are assigned to outbound line 108 in comparison to outbound line 109, for instance). For now, the estimation of the picking time for items assigned to outbound line 109 has our priority. In Figure 4.2 and Figure 4.3, the data (corresponding to outbound line 109) of the number of batches and the travelled distance are shown along with the numbers of items assigned to a poolzone. The functions are fundamentally different: the expected number of pick batches follow a linear line since the number of pick batches straightforwardly increases along with more items are involved, while the cumulative travelled distance is exponentially shaped. The exponential decrease over assigning more items to a poolzone can be explained by the fact that the average distance becomes smaller as more items are involved since more absolute distance can be saved by picking items in an efficient route.

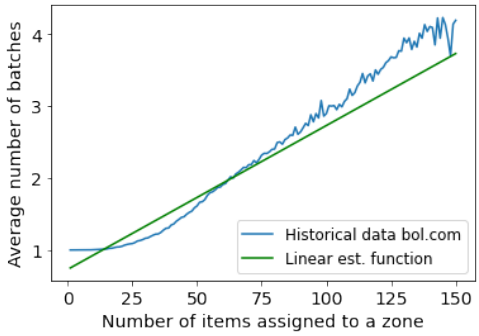

FIGURE 4.2: Distance function outbound line 109

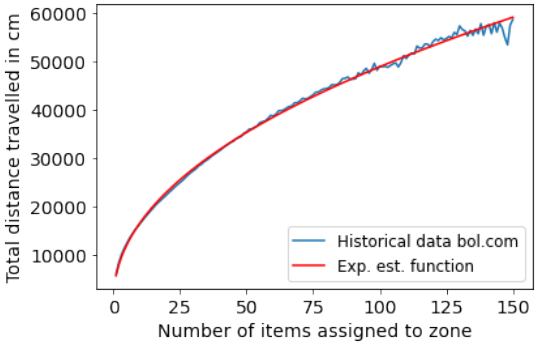

FIGURE 4.3: Pick batches function outbound line 109

In both Figures, the right tail of the graphs show irregularities in the data; this is caused by the relatively low frequencies of these high number of items. To give an indication, XXX% of all the poolzones are containing less than 100 items. Since the lower number of items are more common (and therefore more important to approximate precisely, a weighted least square method is applied to obtain the optimal values for the estimation functions (For more context about the weighted least squares method, see 3.4.1). For the weights, the frequency of each poolzone size expressed as a percentage of the total poolzones is used.
In Equation 4.6 and Equation 4.7, both abstract functions are given. By finding the optimal values of a, b, c and d, accurate functions are created that represent the real-world performance of bol.com.

$$Expectedpickbatches(n_{zp}) = a + n_{zp} * b \tag{4.6}$$

$$Expectedtotaldistance(n_{zp}) = c * n_{zp}^{d} \tag{4.7}$$

By finding the weighted sum of the least-squares error of both functions for multiple parameter sets, the optimal parameters are obtained. In Table 4.1, the optimal values of a, b, c and d for outbound line are presented. In Figure 4.2 and Figure 4.3, the modelled expectation functions are plotted over the historical data of bol.com. The relative high deviation for the higher number of items in a poolzone is explainable by the low frequency (and thus low weights) this number of items occur.

Parameter	Function	Outbound line 109
a	Lower bound for pick batch estimation	XXX
b	Expected pick batch per new item added	XXX
c	Average expected distance per pick in cm	XXX
d	Exponential power fraction the added distance decreases	XXX

TABLE 4.1: Optimal values for estimating functions

As the expected number of batches and the expected total distance can be narrowly estimated by these derived functions, the next step is to find appropriate values for α, β and γ for the main estimation formula (Equation 4.5). Even for this function, we apply a weighted least squares method to derive the optimal values for the approximation of the average picking time per pick in a poolzone. As a result of experimenting with large sets of parameter values and computing the weighted least squared error per combination, the optimal values were obtained. For α, the setup time for one pick batch is determined to be XXX seconds. Secondly, the value of β fits optimally by XXX per meter and the time needed for one pick (γ) should be XXX seconds, on average. Implementing these values, the estimation of the average picking time is shown in Equation 4.8, which is now only based on the number of items that are assigned to a poolzone.

$$F(n_{zp}) = \frac{XXX * Expectedpickbatches(n_{zp}) + XXX * Expectedtotaldistance(n_{zp})}{n_{zp}} + XXX \quad (4.8)$$

The estimation function (without bias-correction) shown in Equation 4.8 comes narrowly close to the current performance of bol.com, which can be seen in Figure 4.4. Again, the deviation at the tail of the function can be explained by the low frequency of these number of items; the weighted least squares method ensures priority for the frequently used number of items, which give the most reliable result. Though, the function currently overestimates all higher numbers of items assigned to zones. The current bias of the function is now 227.8 (for the calculation of the bias, see 4.3.1). To compensate, a linear bias correction is modified, which reduces the total bias to 114.5. By applying a small correction function of $0.01 * n_{zp}$, the increasing bias is tempered. In Figure 4.5, the bias of the original function is plotted together bias including bias-correction is plotted. And as can be seen in Figure 4.4, the function containing the bias correction is more accurate than the original function.

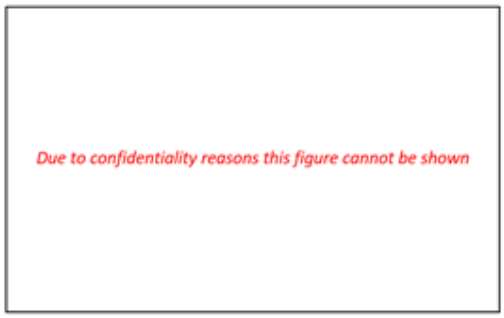

FIGURE 4.4: Estimation function outbound line 109

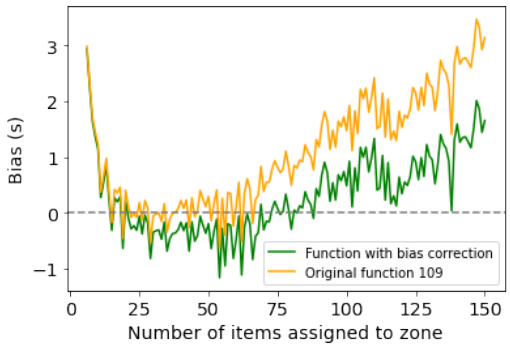

FIGURE 4.5: Bias correction outbound line 109

Thus, by this approach, the average time per pick is successively modelled that only needs the assigned number to a zone as input, which corresponds to the decision making of this model. The parameter derivation of outbound line 108 is shown in Appendix D.

4.3 Mathematical model

By the estimated cost function given in equation 4.8, the estimation of the average picking time can be obtained when evaluating feasible solutions. Since the goal of this research is to optimize the current picking productivity, expressed in terms of average picking times, this estimation is an accurate indication of the real performance. Minimization of this estimation should lead to lower average picking times, and thus a better picking productivity. According to this reasoning, the objective function is given in equation 4.9, which minimizes the estimation of the average total picking time for all assigned items in all pools. The subjected constraints, set to achieve feasible solutions only, are given below.

$$\min \frac{\sum_{z \in Z} \sum_{p \in P} n_{zp} * (F(n_{zp}))}{N} \tag{4.9}$$

Subject to:

$$\sum_{p \in P} x_{izp} \leq a_{iz} \quad \forall i \in I, z \in Z \tag{4.10}$$

$$y_{op} * b_{io} = \sum_{z \in Z} x_{izp} \quad \forall i \in I, o \in O, p \in P \tag{4.11}$$

$$\sum_{p \in P} y_{op} \leq 1 \quad \forall o \in O \tag{4.12}$$

$$\sum_{o \in O} y_{op} \leq C_{max} \quad \forall p \in P \tag{4.13}$$

$$\sum_{o \in O} \sum_{p \in P} y_{op} \geq D \tag{4.14}$$

$$n_{zp} = \sum_{i \in I} (x_{izp}) \quad \forall z \in Z, p \in P \tag{4.15}$$

$$N = \sum_{z \in Z} \sum_{p \in P} n_{zp} \tag{4.16}$$

$$y_{op} \in \{0, 1\} \quad \forall o \in O, p \in P \tag{4.17}$$

$$x_{izp} \in \mathbb{Z}^* \quad \forall i \in I, z \in Z, p \in P \tag{4.18}$$

While Constraint 4.10 makes sure no more items than available can be picked in each zone, Constraint 4.11 connects the orders to the corresponding items of those orders. By this constraint, all the items of an order assigned (y_{op}) to pool p should be assigned to one of the zones in the same pool p, and vice versa. Constraint 4.12 checks that orders can only be assigned to one pool. Capacity Constraint 4.13 ensures that the total orders assigned to one pool may not exceed the maximum allowed pool size, while Constraint 4.14 certifies the sum of all chosen orders of all the pools must at least be as large as the requested demand by the control room. As explained in Section 4.2, the total number of items assigned is can be used to determine the estimated picking time, which is represented in Constraint 4.15 by the sum of decision variable x_{izp} for every poolzone. Furthermore, Constraint 4.16 is used to compute the sum of all items that are included in the pick run.
Constraint 4.17 ensures the assignment of an order to a pool is binary, while Constraint 4.18 ensure the decision how many items should be assigned to zones in a pool can be any non-negative integer value for all items, orders, zones, and pools. This constraint implies n_{zp} are also non-negative integer values, as this parameter relates to x_{izp}.

4.4 Solvability

The formulated model of this research problem raises a few concerns. First, the set of available orders can be extremely large and is varying heavily over time due to the continuous arrival of orders. Every time a pick run is requested, this available order set has to be updated for every outbound line, which consumes time that is included in the allowed time period of solving this problem.
Next to that, the consistently large set of available implies there are a great many decision variables since every order consists of multiple items. For every unique item belonging to a multi-order, there is a non-binary decision variable to add them to one of the 52 zones (which might have the item available) into one of the created pools (of which the number is not fixed in front, as it is depending on the size request each pick run). For instance, when 2 pools are created to meet the pick run demand and in total 10,000 multi-ordered items are available, already a million non-binary decision variables are involved in the problem $(2\ pools\ x\ 52\ zones\ x\ 10,000\ items)$. Note that these numbers represent a very generic scenario occurring in most cases a picking run is requested at bol.com. By not even discussing the largest -above-average- scenarios, it becomes clear that the problem instance is very time-consuming when it would be solved exactly.

Next to the many decision variables, the stated objective function is non-linear, since the average picking time is minimized. This is because the total number of assigned orders is not fixed for each pick run. As a result, solving this problem requires more computation time than linear formulated problems. Above that, we only discussed the problem size of one outbound line yet. In the real-world situation, bol.com demands a solution within four minutes for all outbound lines (mono- and multi outbound line) sequentially.

Furthermore, the formulated mathematical model in this chapter is focused solely on the minimization of the average time per pick, while a consistent fulfilment criterion is stated as a part of the research goal as well. Integrating the cut-off time of each order into the model would make this model even more complex, as additional constraints have to be defined to include timely picking properly. As mentioned in Section 2.1, orders can be prioritized manually (independently of the algorithm) when a cut-off time is approaching. As bol.com is already used to prioritize orders in their current way of working, we decided to not include the cut-off time of the mathematical model. Instead, the performance on fulfilment will be evaluated afterwards.

Because of the above-mentioned arguments, achieving an exact solution within the allowed time period is considered unrealistic. Besides, when evaluating the performance of solution methods the orders that are leftover from each pick run are even as important as the orders that are selected for the pools. Currently, the defined model is not measuring this 'quality' of order selections since the model includes only the scope one pick run. An extension of the model would require a lot more parameters and thereby research how these parameters should be defined and weighted. As no similar problems are addressed by the literature and this model extension is not considered as a prior goal of this research, a more advanced notation of this model is neglected. As a consequence, solutions derived by this model would probably not result in a representative performance in comparison to the real-world scenario at bol.com, since it is focused on only one pick run instead of continuous performance. Nevertheless, the modelled mathematical formulation supports by a better understanding of the problem. Though, it cannot be used to solve the problem. Therefore, the applicability of different approximation techniques is discussed in the next section of this report.

Chapter 5

Solution approach

In this section, we construct solution approaches to solve the introduced problem. As discussed in the literature review, many methods are addressed to optimize problems related to order picking processes. Though, the concept of pooling is not addressed in scholarly literature yet, which inquire new designed approaches than performed so far. By the large-scaled problem environment and additional complexities of the problem, an exact solution cannot be calculated in a reasonable time (Section 4.4). Though, multiple potential approximation heuristics are defined in the literature review, that can be used to obtain near-optimal solutions. Combining the critical factors of the current situation of bol.com by the knowledge the literature provided us, a set of alternative methods is designed to challenge the current performance. This section starts by discussing what factors are essential for the solution to be feasible, that serves as valuable input for the set of solution methods in a second section of this chapter.

5.1 Requirements solution methods

In the dynamic situation of bol.com, customer orders arrive continuously and pooling decisions have to be made under a rolling planning horizon. Therefore, the list of available orders is different for each moment a pick run is requested. The performance of this pick run can be measured by an estimation function of the average picking time based on the number of items assigned to each zone in the created pools of the pick run (Section 4.2). As we stated in Section 4.4, the mathematical model cannot be used to solve the problem exactly. Nevertheless, the formulated notations are used to create feasible heuristics, whereof the performance is measured by the stated objective function.
Next to the objective function, the solution should also be evaluated on the fulfilment criteria, implying every order should be picked on time (before its cut-off time). Although, it could occur orders are not released into the picking process on time by the algorithm itself. As explained in Section 2.1, the control room is manually prioritizing the orders which otherwise would be picked too late. By designing the solution approach of this research, it is assumed bol.com is accepting a certain amount of orders that are not automatically fulfilled by the designed algorithms itself. However, the number of orders this concerns is still analyzed carefully. When too many orders needs prioritization, too many orders have to be released in one time which would disrupt the flow within the warehouse.

The solution methods are designed to provide consistently good performances over multiple pick runs (in contradiction what the model in Chapter 4 suggests). When focusing on only pools in one pick run, the relative easiest orders could be selected over the more complex orders, which would affect the performance of the successive pick runs. The output of the set of orders which are not chosen for a pool is relevant as well, which is the reason we design solution methods to perform consistently on entire operating days. Thus, we want to construct solution methods that are best performing on average, based on all created pool over all requested pick runs of one operating day.

Number of zones per order
Besides the cut-off time of orders, there are other factors we should consider in the decision rules of our solution. Since multi-orders are varying heavily in terms of size and the available stock locations of all items, the complexity of picking each order is based on multiple characteristics. For instance, it is easier assigning an order containing one item that is ordered ten times instead of assigning an order

consisting of ten different items, since likely more zones need to be used. The more different items an order contains, the higher the probability more zones are needed to fulfil the multi-order. This makes the order harder to assign efficiently. By knowing the number of zones orders need, decisions can be made in assigning them effectively.

Allocation possibilities of items

Next to the size of an order, the current stock availability in determining the allocation possibilities, and thus the complexity of the order. Items can be available in multiple zones by the randomized storage strategy bol.com is using (Subsection 2.2.2). For instance, a fraction of items are available in one zone only. These orders are relatively hard to assign since that specific zone need to be used for a pool. When having more items within that multi-orders, the combinations are getting more specific and the order becomes harder to allocate. Therefore, the allocation possibilities is considered in one of the designed algorithms.

Minimizing zones, maximizing items

Furthermore, the solution methods proposed in the next section should consider the obtained strategies in Subsection 2.3.2. First, as minimal zones as possible should be used. One greedy method that focuses entirely on the number of used zones is designed. The second strategy is the maximization of items within a zone relative to other used zones in a pool. Maximization the distribution of items to one zone is performing significantly faster, thus when possible this should be maximized.

5.2 Designed algorithms

In this section, we explain four alternative heuristics that have the potential to improve the current algorithm. The first proposed method is an improved version of the current algorithm, where some decisions are altered. Thereafter, three completely new methods are introduced, whereof its most important decision rules are explained. These decision rules are based on the mentioned complexities, described in the previous section. For each method, we elaborate on the performed decisions and support the methods by pseudo-codes and examples.

For the three newly designed algorithms, a likewise item maximization rule holds when all orders are selected. This rule implies the items of multi-orders are not assigned to zones directly (like the current algorithm), but only after the moment, the last order is chosen. When the pool is complete, all determined zones will be checked on how many items they can fulfil. The zones will be prioritized on this number of items and per zone, the corresponding items will be assigned. When an item is available in multiple zones, it ensures the item will be assigned to the zone that can relatively fulfil most items. This maximization strategy is in contradiction with the current heuristic, since orders are assigned to zones directly. The numbers in parentheses denote the algorithm number, used in the experimental environment later on in this research.

5.2.1 Improvements current heuristic (2)

Analyzing the current heuristic, we see some improvement potentials in the decision making of closing a pool. Therefore, we propose an alternative version of the current heuristic, where decisions regarding the size of a pool are reconsidered. Currently, the algorithm might close a pool when the rest of the available orders do not fit in the already selected zones, even when the maximum pool size is far from reached. Since we observed that adding more items to pools (if there is at least an item added to a zone that was already used) practically always saves picking time, it is recommendable to keep adding zones until the maximum pool size is reached. However, bol.com maintains a maximum number of zones that can be opened in a pool, since that affects the flow at the sorting and packing stations of each outbound line (cause using many zones result in many totes in a pool). As the explained pool size/zone strategy is assumed to have an impact on the picking productivity, two benchmark heuristics are used. In the first version (1) of the current heuristic, we choose to stop adding zones when no other orders fit in the pool. For the second version (2), we choose to keep adding zones until the maximum number of

zones is used in a pool, which is 10. This maximum number is already applied in the original heuristic, as selecting more zones could lead to too many totes in a pool that could disrupt the process at the sorting station.

5.2.2 Maximize minimum zones needed method (3)

The first method we propose contains a priority rule that sorts the orders on the minimum number of zones that are needed to assign all items of the complete order. For each order, it can be determined what the optimal set of zones should be when assigning only this order by considering the current stock availability. Orders containing many different items are likely to need multiple zones, which have turned out to be undesirable for achieving fast average picking times. Nevertheless, these types of orders must be picked at some point during an operating day. This method chooses the strategy to choose these types of orders in an early phase, so the rest of the pool could still be optimized by the large set of orders that is still available; likely no extra zone will be needed to reach the maximum pool size.

The list of available orders are sorted on the minimum zones that are needed for each order, in descending order. When the number of minimum zones is equal for orders, the cut-off time is a second sorting criterion. As the control room manually can prioritize orders (and is planning to keep doing that in the future), we cannot set the cut-off time of an order as a fixed constraint. Instead, we apply the cut-off time as a second sorting criterion to take at least some prioritization into account. As described in section 5.1, the performance of the fulfilment score will be evaluated afterwards.
Let us illustrate this method by the following mini example, where five orders are considered. The pool that is created starts empty and the fictional maximum pool size is four orders. The characteristics of these orders are shown in Table 5.1.

Order	Items	Zone possibilities per item	Minimum zones needed	Cut-off time
o_1	i_1	$[z_1, z_2]$	3	23:00
	i_2	$[z_5]$		
	i_3	$[z_3, z_7]$		
o_2	i_4	$[z_1]$	2	12:30
	i_5	$[z_2, z_4]$		
o_3	i_6	$[z_4]$	2	22:30
	i_7	$[z_3, z_7]$		
	i_8	$[z_4, z_5]$		
	i_9	$[z_7]$		
o_4	i_{10}	$[z_5, z_7]$	2	23:00
	i_{11}	$[z_4, z_6]$		
o_5	i_{12}	$[z_5]$	1	23:00
	i_{13}	$[z_5, z_6]$		

TABLE 5.1: Mini order set - example max min zone method

After the sorting procedure, the first order that will be included in the pool is the order that contains the maximum of these minimum needed zones. This order is chosen as the seed order of a new pool. As can be seen in Table 5.1, the list of orders is sorted on this minimum zone property, where o_1 needs at least three zones of the assignment of all its items. This order is thus chosen as the seed order, for which the most efficient zones (corresponding to the preliminarily determined minimum number of zones) that could be used are selected. Since there is no single combination of zones that could be made, the zones are selected in alphabetical order: zones z_1, z_3, z_5 will be opened for this pool.

The second step of the method iterates over the available sorted order list and checks if other orders can be fulfilled completely by the same zones that are already used for the seed order. When looking at

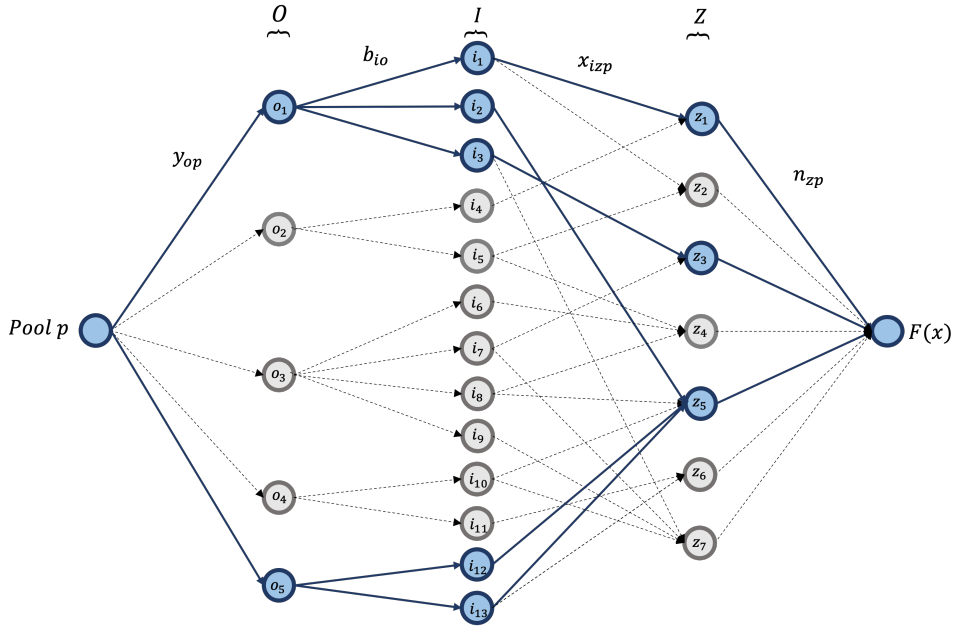

FIGURE 5.1: Illustration of the selected orders - example max min zones method

the zone possibilities in Table 5.1, it is observed that o_2 and o_3 cannot be fully assigned to these zones, since item i_5 of order o_2 and item i_6 and i_9 of order o_3 cannot be assigned to the selected zones of this pool. Looking at order o_5, we see that this order can be assigned, since both of the items of o_5 are available in z_5. When more orders were included, this process would continue till the maximum pool size is reached. The decisions made so far are represented in the illustration shown in Figure 5.2.

Since we have not reached the maximum pool size of four orders yet, an additional step is performed to add more orders to the pool. For every zone that is currently not opened in the pool, we check how many orders could be added when we open just this one zone. For each combination, the potential orders that could be filled by this set of zones are saved. After computing this for every combination, the zone containing the highest number of additional orders will be chosen, and the corresponding orders will be assigned to the pool. To clarify this step, we denote all combinations for the four zones that could still be opened in Table 5.4.

Zone up for check	**Potential zone selection**	**Nr. of fulfilled orders**	**Order selection**
z_2	$[z_1, z_2, z_3, z_5]$	1	$[o_2]$
z_4	$[z_1, z_3, z_4, z_5]$	2	$[o_2, o_4]$
z_6	$[z_1, z_3, z_5, z_6]$	0	-
z_7	$[z_1, z_3, z_5, z_7]$	0	-

TABLE 5.2: Adding orders by checking zone combinations - example max min zone method

Table 5.4 is showing that by opening z_4 two extra orders can be added to the pool. Since this is the maximum number of orders of all checked combinations, these zones will be opened and the corresponding orders will be added to the pool. The maximum pool size is reached, so the order set for this pool is determined ($[o_1, o_2, o_4, o_5]$). Note that when the maximum pool size was not reached, a new zone was added based on the same procedure. This process will continue until the maximum pool size is reached or that in total 10 zones are used. In case the total number of orders becomes higher than the maximum pool size after an iteration, the list of orders is truncated and the highest-ranked orders will be assigned to the pool.

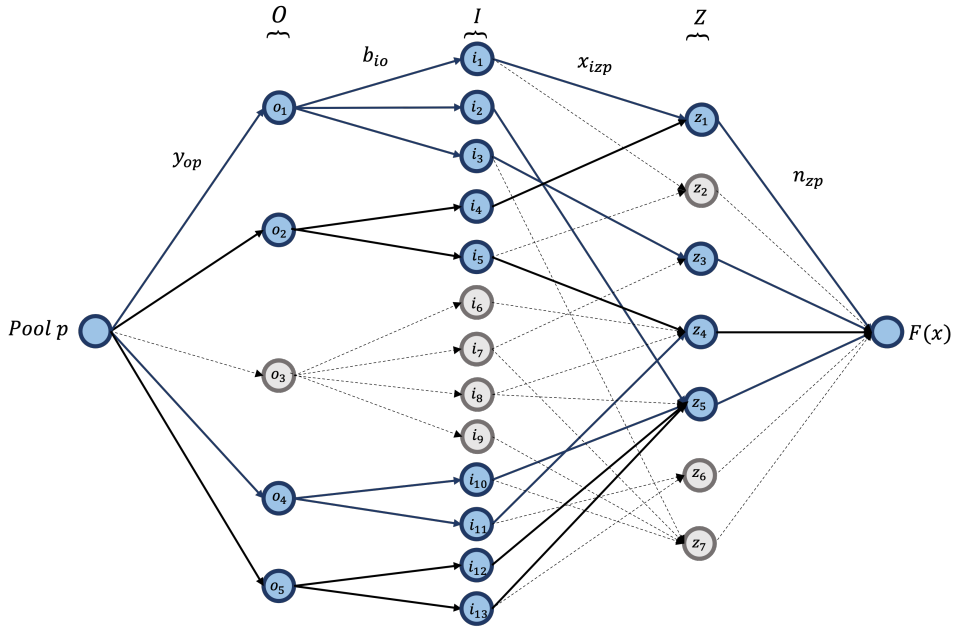

FIGURE 5.2: Final pool containing four orders - mini example max min zones method

At this moment, a pool is created. In Figure 5.2, the decisions made in this method are highlighted, even as the chosen orders. The items of the four assigned orders are distributed over zones $[z_1, z_3, z_4, z_5]$ and have poolzone sizes of $[1, 1, 2, 4]$, respectively. This is the output, for which the average time per pick can be calculated by the formula derived in Section 4.2.
To fulfil a pick run, multiple pools are created by the same procedure until the total orders that will be picked exceed the requested demand of the pick run. Additionally to this explanation, a pseudo-code is provided in Appendix F.

5.2.3 Minimal possibilities method (4)

A second method that is considered to have the potential of outperforming the current pooling method is another constructive method focusing on the limited availability of items. Comparing this method to the first method and the approach bol.com is currently applying one main difference should be noted; the strategy of this method is adding zones based on the order list instead of adding orders one by one.

The first step of this method consists of checking for each item in all orders in how many zones it is available; the minimum availability of an item in the number of zones is noted for each order. The orders are sorted on this minimum availability first, where the cut-off time will become a second sorting criterion. Also for this method, the approach will be explained by an illustrated mini example. A sorted data set of five orders is shown in Table 5.3.

All orders containing one or more items being available in one zone only are considered as a subset of the entire order list. This subset has priority in this method. Over this subset, zones are added by a greedy rule used to avoid the need of checking all possible zone combinations. The first zone is determined by picking the zone that is storing most of the items being only available in that zone. In the example, orders o_1, o_2 and o_3 form the subset that consists of items that are available in one zone only (items i_1, i_2, i_4, i_6 have this property). For the items $[i_1, i_2, i_4, i_6]$ the available zones where the items are placed are in zones $[z_1, z_2, z_3, z_3]$ in respective order. Since zone z_3 contains most of this type of items, this zone is chosen as the starting zone of the pool.

Thereafter, we iterate over each zone combination by this starting zone and each of all the other zones. By alternately adding each zone to the zone selection, we can check which orders of the subset can be picked by this hypothetical zone selection. The zone that can serve the highest number of orders

Order	Items	Zone possibilities per item	Number of possibilities per item	Cut-off time
o_1	i_1	$[z_1]$	**1**	13:00
	i_2	$[z_3]$	**1**	
o_2	i_3	$[z_1, z_3]$	2	22:30
	i_4	$[z_2]$	**1**	
	i_5	$[z_2, z_4]$	2	
o_3	i_6	$[z_3]$	**1**	22:30
	i_7	$[z_2, z_7]$	2	
o_4	i_8	$[z_5, z_7]$	2	21:00
	i_9	$[z_6, z_7]$	2	
o_5	i_{10}	$[z_4, z_5]$	2	23:00
	i_{11}	$[z_5, z_7]$	2	

TABLE 5.3: Mini order set - example zones on min possibilities method

is chosen and will be added to the real zone selection. Recurring to the example, all combinations of the zones by starting zone z_3 are given in Table 5.4. As can be seen, the zone selection $[z_2, z_3]$ consist of the most orders that can be assigned completely, so these zones will be the zone selection. This implies the corresponding orders o_2 and o_3 are added to the pool simultaneously. The decisions made so far are highlighted in Figure 5.3

Zone up for check	Potential zone selection	Nr. of fulfilled orders	Order selection
z_1	$[z_1, z_3]$	1	$[o_1]$
z_2	$[z_2, z_3]$	2	$[o_2, o_3]$
z_4	$[z_3, z_4]$	0	-
z_5	$[z_3, z_5]$	0	-
z_6	$[z_3, z_6]$	0	-
z_7	$[z_3, z_7]$	0	-

TABLE 5.4: Order fulfilment on zone combinations - example zones min possibilities method

This process is repeated until a fixed number of zones are included in the pool or the maximum pool size is reached. Since zones are added by the algorithm and not every order one by one, there might fit more orders in the zone selection than the maximum pool size. If this is the case, the orders are added to the pool will be selected in order of the sorting criteria.

When the maximum pool size is not reached after selecting the determined fixed number of zones, the rest of the available orders (excluded from the subset) will be checked if they can be assigned completely to the already used zones. At the moment the maximum pool size will be reached, the pool is closed.
In case the maximum pool size is not reached by checking all orders (even the orders outside the subset), extra zones will be added to the pool until 10 zones in total will be used (same zone criteria as the previous method). The search which zone should be included in the pool follows the same procedure as the first method, where from now on the entire list of available is considered instead and not the subset of orders containing items with an availability of one zone only.

The example of this method has resulted in a pool of three orders ($[o_1, o_2, o_3]$, distributed over the zones $[z_1, z_2, z_3]$. When the maximum pool size was higher than 3, the method would seek more zones to add. Though, the understanding should be clear. The poolzone sizes of the zones in this example are respectively $[1, 3, 3]$. For more context about this method, an additional pseudo code is provided in

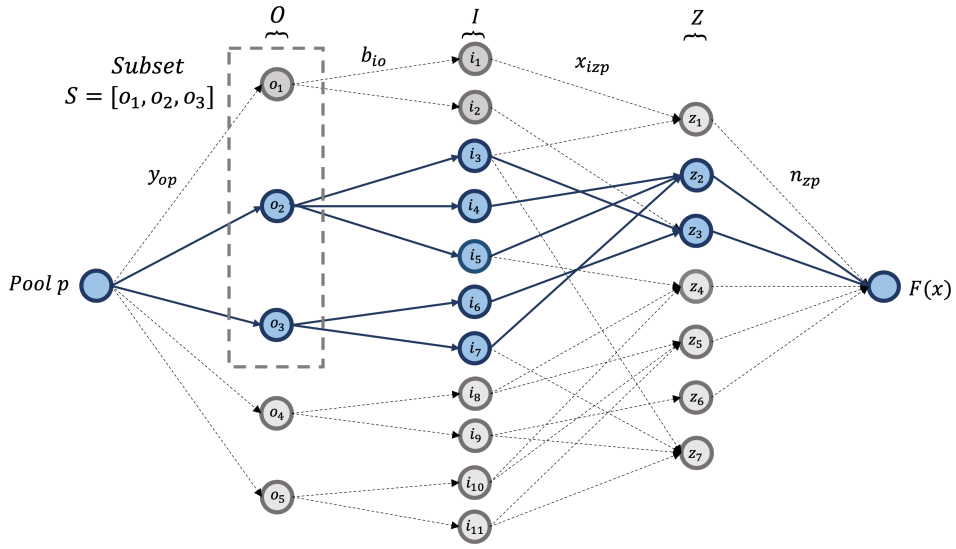

FIGURE 5.3: Final pool - example minimum possibilities method

Appendix F.

5.2.4 Greedy zones method (5)

A third considered method is a greedy method that determines the zones instead of orders. By the same approach as the second method, this method is adding zones by a greedy rule and checks at each iteration which orders can be fulfilled.
The orders will be sorted on their cut-off time first. When the cut-off time is equal the minimal possibilities criteria of 5.2.2 and the minimum zones metric of 5.2.1 are the second and third sorting criteria, respectively. In Table 5.5, an overview of the corresponding example to this method is given, including four orders in total.

Order	Items	Zone possibilities per item	Cut-off time
o_1	i_1	$[z_1, z_2, z_3]$	22:00
	i_2	$[z_1]$	
	i_3	$[z_4, z_6]$	
o_2	i_4	$[z_1, z_4]$	23:00
	i_5	$[z_4, z_5]$	
o_3	i_6	$[z_1, z_4]$	23:30
	i_7	$[z_3, z_7]$	
	i_8	$[z_4, z_5]$	
	i_9	$[z_7]$	
o_4	i_{10}	$[z_4]$	23:30
	i_{11}	$[z_6, z_7]$	

TABLE 5.5: Mini order set - example greedy zones method

The algorithm iteratively checks for each zone how many complete orders can be fulfilled. The zone containing the highest number of orders will be selected and moved on the successive iteration, where potential zone combinations will be checked by the chosen zone of the previous iteration. In Table 5.6 is the first iteration represented, where can be observed that zone z_4 can fulfil order o_2. Since this zone consists of the highest potential order set, this zone will be selected as the first zone in the zone selection.

Zone up for check	Number of potential orders	Selection of orders
z_1	0	-
z_2	0	-
z_3	0	-
z_4	1	$[o_2]$
z_5	0	-
z_6	0	-
z_7	0	-

TABLE 5.6: Order fulfilment on zones - example greedy zones method

The second iteration is shown in 5.7. The potential orders by all two-zone combinations where zone z_4 is included are checked. While combination $[z_1, z_4]$ can add o_1 to their current order selection, combination $[z_4, z_7]$ can add o_3 and o_4. As this is the best combination so far, zone z_7 will be included in the zone selection.

Zone up for check	Potential zone selection	Nr. of fulfilled orders	Order selection
$[z_1, z_4]$	2	$[o_1, o_2]$	
$[z_2, z_4]$	1	$[o_2]$	
$[z_3, z_4]$	1	$[o_2]$	
$[z_4, z_5]$	1	$[o_2]$	
$[z_4, z_6]$	1	$[o_2]$	
$[z_4, z_7]$	3	$[o_2, o_3, o_4]$	

TABLE 5.7: Order fulfilment on zone combinations - example greedy zones method

A schematic representation of the decisions made in the example is shown in Figure 5.4. The zone selection now consists of zone set $[z_4, z_7]$, that can fulfil order set $[o_2, o_3, o_4]$. This process of adding zone for zone by the same greedy rule will repeat till one of two stopping criteria are reached, whereof the first one is the exceedance of the maximum pool size: at this point, the order set will be truncated along with its sorting criteria (the highest-ranked orders will be included). A second stopping criterion is a situation when the 10th zone is selected for the pool since more than 10 zones can disturb the flow at the sorting station of the outbound line.

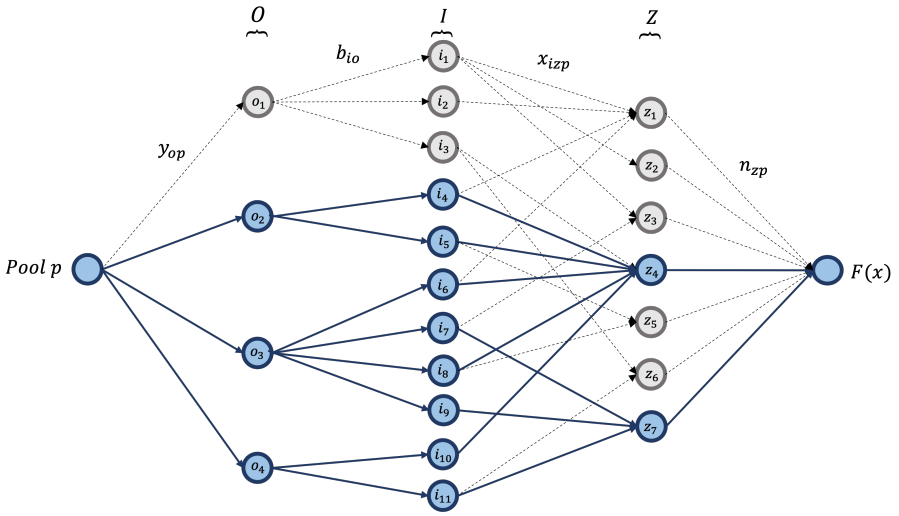

FIGURE 5.4: Final pool - example greedy zones method

Chapter 6

Experimental set-up

This chapter describes the experimental set-up that is used to test the designed algorithms.

6.1 Testing environment

This section discusses the assumptions needed to create an effective testing environment, wherein the designed algorithms are tested.

6.1.1 Time horizon

By evaluating the performance of all the different algorithms, entire operating days are considered. As mentioned in Section 5.1, creating one efficient pick run of pools is not solving the problem; this pick run has impact on the rest of the available order set and thus the rest of the pick runs. Since operating days end around midnight and all orders that **'must'** be picked is known, the performance of solution methods can be tested in a simulation model that represents these operating days. Within this model, the request of pick runs can be simulated over time, as the cut-off time of orders is also known. In addition, bol.com is merely focusing on this type of must go orders (orders are usually not picked a day in advance, which are labelled as 'could' go orders). Thus, filtering out this type of order results in a representative order set that can be simulated to challenge the current performance of bol.com to a series of newly designed methods. To state once more for clarity, these methods are tested on each outbound line separately, since the orders are preassigned to outbound lines.

As mentioned, new orders may arrive continuously into the system (Section 2.2.3). Consequently, the list of available orders is far from complete at the beginning of an operating day and changes dynamically over time. Every moment in time a pick run is requested, the set of available orders is updated. Therefore, the entry time of each order should be included in the simulation model; it is infeasible to select an order before it has entered the system.
Next to that, the orders must consistently be checked on their availability regarding the current stock levels, which is updated when items are allocated. At the end of an operating day at bol.com, the control room manually could decide to convert multi-orders into mono-orders, as one item of the multi-order is not available while the rest of the items is. Since the mono-orders are out of scope for this research environment, this conversion of multi-orders into mono-orders is neglected in the testing environment of this research. When one item of a multi-order is not available, the entire order will be excluded from the list of available orders.

When a low amount of orders is leftover for an outbound line at the end of an operating day, bol.com applies a sweep run (Section 2.2.4) to collect the last set of orders. Since we want to measure the difference of the regular method bol.com is applying, the current logic for the sweep run is constantly applied at every last pick run for each method that is executed in the testing environment. The size of this sweep run is equal to the rest of the pick runs.

6.1.2 Data retrieval

The Warehouse Management System of bol.com keeps track of the available orders and contain files of each moment a pick run is requested. Since orders may arrive continuously, the number of available

orders change dynamically over time (Section 2.2.3). To obtain a complete list of online arrived orders, multiple data files of the available orders have to be imported to obtain a representative list of orders. Per operating day, ten separate data files of timestamps -spread over the same operating day- are merged into one complete list of orders that must be picked on that operating day.

Next to the online arrival of orders, the available stock in BFC1 will also be supplied by new items continuously. As a joint decision with bol.com, the solution approaches are tested under the assumption that no new orders arrive in the system. This implies a snapshot is taken at a specific moment in time that shows the stock levels at that specific moment in time. This specific moment is chosen to be the beginning of the day, whereof the first order file is also obtained.
The reason why to test the solution approaches as described above is based on the accessibility of data. No specific data is available on what items were put in stock at which specific moment in time; the same holds for all picked items. Retrieving reliable data to represent the stock levels dynamically is too complex and will take too much time for this research. Besides, by setting fixed equal stock levels the performance of the algorithms is measured more effective.

When simulating over an operating day, the entire list of orders obtained by the data files will be processed in line with the available stock. Time parameters need to be included since not all orders are available at the beginning of a day. The pick runs are simulated constantly over time, where an operating day starts at 8:00 am and ends around midnight (00:00 am). By simulating the time for each pick run, it enables us to evaluate the cut-off time of orders. Furthermore, in this experimental approach, a constant request of the number of orders is assumed each quarter. To clarify, for instance at moment t_1, the control room requests a constant number of orders for every outbound line. By the data available, the stock levels per zone of all items and the list of available orders are known. At moment t_2, the control room makes a new request for the same number of orders for that outbound line, where t_2 is now 15 minutes later than moment t_1. In this time interval, new orders may have arrived and the processed orders in the pick run of t_1 are excluded from the available order list.
Except in rare cases, this way of experimenting represents the actual situation accurately. The results of processing an entire day enable us to interpret the performance behaviour between all pick runs during the day. In the beginning, all demand and stock will be available to choose from and all algorithms will likely create efficient pools. As the operating day ends, the set of allocation possibilities will become smaller and the algorithm are expected to perform differently in terms of picking productivity, based on its decision rules.

6.1.3 Test settings

Per outbound line, the utilization in terms of orders varies (see Section 2.2.2). Also, the parameters for the estimation function of the average time per pick differ per outbound line (Section 4.2). However, the behaviour of the poolzone size related to the average time per pick is more or less the same. To analyze the designed algorithms on different parameters, the algorithms are tested on multiple outbound lines. As explained, outbound lines 108 and 109 will be included in the experimental scope of this research. Because these two outbound lines have processed **XX%** of all multi-orders last year, they are most valuable for this research.
Roughly, the set of orders assigned to outbound line 108 vary between **XX-XX** orders per day and the orders assigned to outbound line 109 vary between **XX-XX** orders per day. Since differences in the total number of orders processed per day could influence the performance, the order sets are equalized for each outbound line; respectively 2,000 orders and 10,000 orders for outbound lines 108 and 109 are processed per experiment, which are randomly selected from the total list of orders processed that day. We assume the results represent the average performance accurately by these number of orders since these are average production numbers for bol.com. Experiments by larger or smaller order sets per outbound line are not expected to differ significantly in performance, since the characteristics of order sets (regardless its size) are corresponding to each other in terms of cut-off times, items per order and quantities per order etc., discussed in Section 2.3.3.

For each of the two outbound lines, multiple data sets are tested on two scenarios. As we observed the stock availability in determining the combinations of item allocations to the zones (see Section 2.3.3), this is considered of impact to the performance of the proposed algorithms. To measure this effect, the scenario where the total stock level at bol.com was relatively low will be compared to periods where the total stock level was relatively high. Summarizing the influencing factors, four situations will be examined:

1. High stock levels for outbound line 108
2. High stock levels for outbound line 109
3. Low stock levels for outbound line 108
4. Low stock levels for outbound line 109

For each scenario, experiments for five different days are executed. The low stock level scenario includes around **0.75 X** items in total at the beginning of the day, while the high stock level scenario includes around **1.00 X** items in total at the beginning of the day. The data sets of the operating days are randomly chosen over a selection of days that match the mentioned criteria in this paragraph, including conditions that at most one day of the week can be chosen to include spread over different periods, and all days should be in the year 2021 for an actual representation of the performance. Above that, weekend days are avoided, since generally a lower number of orders are processed.

6.2 Performance measures

This section explains the different performance measures in more detail. These performance measures are used to compare the performance of the proposed heuristics against the current pooling logic, considered as the benchmark heuristic.

6.2.1 Average picking time

It is already addressed multiple times the performance of the pooling algorithm is expressed by the average time it takes to pick one item. This average time per pick can be obtained by an estimation based on the number of items that are assigned to a zone. Recall that this estimation is calculated by the formula in Equation 6.1, derived in Section 4.2.

$$F(n_{zp}) = \frac{\alpha * Expectedpickbatches(n_{zp}) + \beta * Expectedtotaldistance(n_{zp})}{n_{zp}} + \gamma \qquad (6.1)$$

Equation 6.1 gives the average time per pick for the number of items assigned to one poolzone. The cumulative average time per pick over the total picked items of an operating day is obtained by taking the weighted average of all poolzone sizes within each pool for every requested pick run. Next to that, the average picking time per individual pick run is also calculated to identify patterns of solution methods during an operating day.

6.2.2 Cut-off time

Next to reducing the main objective function, the average time per pick, it should also be examined whether the fulfilment score remains comparable to the current fulfilment score. As is explained in Section 2.1, the control room is prioritizing orders manually to ensure timely delivery. The current algorithm is not able to fulfil 100 % orders on time automatically. However, as is also mentioned in Section 5.1, it is still important to know how many orders concern this prioritization per for every tested algorithm. If this number of orders become too large, the workload for operators might become too high and orders might still be picked late. Experts in the field at bol.com state the workload will not be affected as long as the number of prioritized orders for all outbound lines remains below 500 per moment in time.

To measure the fulfilment, we note the fraction of orders that are not directly fulfilled by the algorithm itself. Furthermore, as the impact of this prioritization on the average picking time is important, we will include this prioritization in the simulation: an additional condition will be built so that all orders having

a cut-off time within the upcoming hour will be added in the next pick run, regardless of the decision rules of each tested method.

6.2.3 Calculation time per pick run

Moreover, the allowed calculation time of the algorithm may exceed at most four minutes per pick run for all outbound lines (Section 2.2.5). Thus, the proposed algorithms should be able to find a solution relatively fast. Because of a fourfold argumentation, based on the calculation times in the testing environment it is hard to state whether the algorithms would fit in the allowed time period: first, the programming software bol.com is more advanced than the software that is used for the simulation model. Secondly, the calculation time is built up differently. Extracting and updating the data of orders and stock consumes time that is included in the calculation time (as it is also in the real-world situation). A third reason is the requested pick run size is now constant but may vary over time in the real-world situation. The total calculation time is depending on the pick run size, as this determines how many orders should be assigned per outbound line. A fourth and last argument is that the tested benchmark algorithm is recreated along with a series of simplifications, whereof the savings in computation time are unknown.

Thus, the absolute calculation times will be incomparable. Therefore, the ratio of the calculation time of each algorithm and the calculation time of the current algorithm (measured in the testing environment) is computed. This ratio is based on the average running time for all pick runs of both outbound lines. Besides, outliers of the average calculation time are carefully analyzed and discussed when needed.
For this performance measurement, bol.com states it is worrying if an algorithm runs more than ten times longer than the current algorithm.

6.3 Validation

To draw well-founded conclusions based on the results, the experimental outcomes should be validated first. The steps that are performed to verify whether the differences between methods are significant for each scenario and each outbound line are explained in this section.

6.3.1 Feasibility

An evident property of the solution should be that the solution is feasible. This means it should be checked if every order is processed and that the number of items assigned to the total number of pools is the same as the number of items that had to be planned. The method should ensure that each order is only assigned once and that every order is fulfilled. Furthermore, all pools should correspond to the maximum number of orders that are set for the outbound line and the size of the pick run must exceed the constant request of the control room.

6.3.2 Paired sample t-test

A different method used to test whether or not the results are significantly different is the paired sample t-test. The paired sample t-test can be used to test for a significant difference between two dependent variables. This research can be used to test the mean seconds per pick of the new proposed heuristics against the mean seconds per pick of the benchmark heuristic.

The tested hypothesis is:

Null hypothesis (H_0): $u_d = 0$, indicating a mean difference between the seconds per pick of the benchmark heuristic and the seconds per pick of the new proposed heuristics equal to 0

Alternative hypothesis (H_A): $u_d \neq 0$, indicating a mean difference between the seconds per pick of the benchmark heuristic and the seconds per pick of the new proposed heuristics not equal to 0

In case the p-value is less than 0.05, the null hypothesis can be rejected. Rejecting the null hypothesis means that there is a statistically significant decrease or increase in the mean value.

Chapter 7

Experimental results

In this chapter, the results of the designed methods are provided. By a simulation approach, experiments have been performed that measure the performance during an entire operating day. For each of the two scenarios regarding the total stock levels, results on both outbound lines are provided. The current heuristic bol.com is tested in the same experimental environment and is set as a benchmark heuristic. Based on the benchmark heuristic, a variant is designed. Next to that, three new designed methods are tested, resulting in the following five methods:

1. Benchmark heuristic
2. Benchmark heuristic - maximized pools
3. Max minimum needed zones method [1]
4. Minimum possibilities method [2]
5. Greedy zones method

For the average time per pick, paired sample t-tests are performed to state whether the difference in results is significant. Next to the average time per pick, the fulfilment score and the calculation time of all methods are evaluated as a second and third criterion. Furthermore, this section provides additional experimentation of combinations of methods. Also, insights into the number of zones used per pool are shown in a separate section. At last, the most important findings of this chapter are discussed.

7.1 Seconds per pick

This section shows the results of the performance measure 'average picking time', calculated according to the estimation function derived in Section 4.2. The goal of this analysis is to create an understanding if there is a significant positive difference in the average time per pick of the different methods, compared to the currently applied method. By the use of paired sample t-tests, this significance is measured. A calculated T value based on the difference is used to obtain a p-value (see Section 6.3.2). In case the p-value is less than 0.05, the null hypothesis can be rejected. Rejecting the null hypothesis means that there is a statistically significant decrease or increase in the mean value. The green marked p-values can be rejected and thus show a significant difference in the mean value, while the red marked p-values cannot be rejected and therefore these algorithms are not able to show a significant decrease or increase in the mean value. Furthermore, the average performance per pick run of each algorithm is given for outbound line 109 per scenario, which provide insights into the behaviour of the algorithms during an operating day. Since the behaviour of these pick runs is quite comparable, performance per pick run of outbound line 108 is provided in Appendix F.

7.1.1 Scenario 1 - high stock levels

For the scenario of high stock levels, BFC1 had stored around **1.00 X** items in total at the beginning of all simulated operating days. For outbound lines 108 and 109, around 2,000 and 10,000 orders are processed per experiment, respectively. For both outbound lines, the results in terms of the average time per pick are presented in Table 7.1 and Table 7.2.

[1] In legendas called 'max min zone needed'

[2] In legendas called 'min possibilities'

[3] Again, the mean is expressed as ratios of the best performance of both scenarios over both outbound lines

Algorithm	Mean	T-value	p-value
1	**1.44**[3]		
2	1.38	10.255	9.636 e-5
3	1.37	8.985	1.816 e-4
4	1.37	10.091	1.041 e-4
5	1.34	8.311	2.629 e-4

TABLE 7.1: t-test high stock levels - outbound line 108

Algorithm	Mean	T-value	p-value
1	**1.09**		
2	1.06	5.377	0.002
3	1.08	0.380	0.343
4	1.05	3.478	0.012
5	1.00	9.637	1.299 e-4

TABLE 7.2: t-test high stock levels - outbound line 109

A first observation is that the results of the benchmark heuristic, show similar performances that are obtained by the historical data of bol.com. The results correspond to the average picking times per outbound line, shown in Figure 2.5 in Section 2.3.1. This suggests the testing environment represents the current performance accurately.
The difference in the performance of both outbound lines is explainable for two reasons. First, the maximum pool size is lower for outbound line 108 than for outbound line 109 (see Appendix A for exact numbers). Secondly, the processed items on outbound line 108 are generally shaped by larger dimensions and contain more weight.
Looking at the p-values, all alternative methods on outbound line 108 show a significant decrease in the average time taken per pick in comparison to the benchmark algorithm. For outbound line 109, it is noted algorithm 3 is the only method not showing a significant decrease. The algorithm resulting in the lowest time per pick (for both outbound lines) is algorithm 5, which show the average time per pick could decrease around **0.10 X** seconds and **0.09 X** seconds for outbound lines 108 and 109, respectively.

In Figure 7.1, the average performance per pick run on outbound line 109 is shown. Starting an operating day at 8:00 AM, a constant request of 150 orders is requested every 15 minutes. The pick runs including cut-off order prioritization are represented in pick run 14 and pick run 21, which were around 12:30 PM and 2:15 PM respectively. As expected, the performance of these pick runs is relatively bad in terms of the average time per pick, since all algorithms need to prioritize a fraction of orders at these points in time.

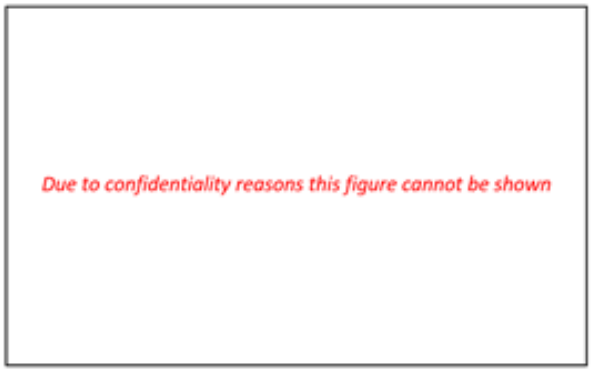

FIGURE 7.1: Performance pick runs - high stock levels - outbound line 109

Furthermore, it is observed the average time per pick of all methods increase heavily at the end of the operating day. Also, the performance of the greedy zones method remains consistently good for the first half of the operating day. By achieving this good performance, the algorithm succeeds in using only one zone for as long as possible (since the method will only use more zones when the maximum pool size is not reached). The max min zone needed method (algorithm 3) is scoring the highest time per pick for its first series of pick runs since relatively many zones are in use. Nevertheless, at the end of the day, this algorithm is scoring the best results (from pick run 52-60). The effect of picking more complex orders

first seems to have its effect, only on average, it is not resulting in the best performance.
Note that algorithm 4 needed a few pick runs less on average than the other methods. This does not imply fewer orders are processed; the pools vary in size and since the pick run request may be exceeded, the number of pick runs is not fixed.

7.1.2 Scenario 2 - low stock levels

In this scenario, operating days with low stock levels are considered. On average, BFC1 had around **0.75 X** items in stock at the beginning of the simulated operating days. The same amount of orders is processed for each outbound line as in scenario 1. The mean time per pick and the additional p-values are given in Table 7.3 and Table 7.4.

Algorithm	Mean	T-value	p-value
1	**1.46**		
2	1.41	4.426	0.004
3	1.41	2.993	0.020
4	1.40	2.108	0.058
5	1.37	10.652	8.024 e-4

TABLE 7.3: t-test high stock levels - outbound line 108

Algorithm	Mean	T-value	p-value
1	**1.09**		
2	1.05	11.407	5.760 e-5
3	1.08	1.365	0.144
4	1.04	10.601	8.212 e-5
5	1.01	13.818	2.262 e-4

TABLE 7.4: t-test high stock levels - outbound line 109

As can be observed in Table 7.3 and Table 7.4, also in this scenario all results of the designed algorithms are performing better than the benchmark algorithm on average. Again, the difference in performance of algorithm 3 on outbound line 109 is not significant in comparison to the benchmark heuristic. Remarkably, the results of the benchmark and the fourth algorithm differ also not significantly on outbound line 108, while its mean is performing almost **0.06 X** seconds faster per pick. This implies the results per experiment are not consistent for this algorithm.

By the performance per pick run on outbound line 109 shown in Figure 7.2, it can be observed that the greedy zones method (algorithm 5) is, again, performing better in the first series of pick runs. During the day the average time per pick increases, but the same holds for all other methods. The behaviour of all algorithms per pick run is similar to scenario 1. The relation between the number of used zones and the average time per pick is analyzed in a later section of this report.

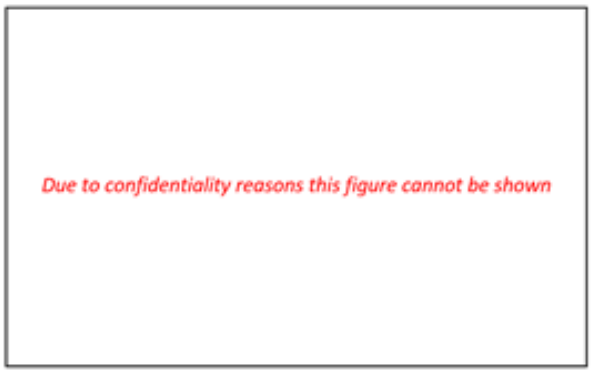

FIGURE 7.2: Performance pick runs - low stock levels - outbound line 109

Comparing the results of this scenario to scenario 1, the mean time per pick of outbound line 108 is slightly higher by a range of **0.02-0.04 X** seconds for all algorithms. A possible explanation for this is the smaller solution space when the stock level is lower, which results in fewer assignment possibilities per multi-ordered item. However, for outbound line 109, no clear differences are identified, as the results are approximately the same for each algorithm (the largest difference per algorithm is **0.01 X** seconds).

Also in this scenario, the best performing algorithm for both outbound lines is the greedy zones method (algorithm 5). Compared to the benchmark algorithm, on average **0.09 X** and **0.08 X** seconds can be saved per pick for outbound lines 108 and 109, respectively. Even when comparing this algorithm to the other methods, there is a large gap between this algorithm and the second-best algorithm (algorithm 4).

7.2 Fulfilment

As mentioned in Subsection 6.2.2, the fulfilment score is converted to the fraction of all processed orders that are not automatically picked by the tested algorithms. Although bol.com prioritizes orders manually (and the impact on the average time per pick is included in the performance), we still analyze how many orders concern this prioritization. Too many prioritized orders could affect the flow within the warehouse. In terms of orders, experts in the field at bol.com state an unfulfillment of more than 500 orders summed over all outbound lines at one time period would be alarming; if less than this amount of orders need to be processed urgently the orders can be assigned in the upcoming pick runs while the workload remains constant. Next to the fraction of unfulfilled orders, the number of unfulfilled orders are presented in Table 7.5. Note that these numbers of orders are obtained by multiplying the processed demand of each outbound line times the fraction of unfulfilled orders. In Appendix G, the fulfilment performance is compared to the average time per pick.

	Scenario 1				Scenario 2			
	Line 108[4]		Line 109		Line 108		Line 109	
Alg.	**Fraction**	**Orders**	**Fraction**	**Orders**	**Fraction**	**Orders**	**Fraction**	**Orders**
1	1.01 %	20	0.40 %	40	0.16 %	3	0.33 %	33
2	-0.09 %	-2	-0.06 %	-6	+0.05 %	+1	+0.03 %	+3
3	+0.45 %	+9	+0.22 %	+22	+0.53 %	+11	+0.14 %	+14
4	+0.95 %	+19	+1.35 %	+135	+0.77 %	+15	+0.62 %	+62
5	+1.06 %	+21	+1.48 %	+148	+0.78 %	+16	+0.76 %	+76

TABLE 7.5: Fulfilment performance per algorithm per scenario

The fulfilment performance of algorithm 4 and algorithm 5 are marked orange for outbound line 109 in scenario 1 since they have the highest number of unfulfilled orders. Note the arbitrary difference in performance between scenarios is based on the different sampled data sets; there is no relation between the stock levels and the fulfilment.

A pattern is observed regarding the algorithms based on their focus on the cut-off time. The benchmark algorithm and its variant are scoring almost the same in terms of fulfilment, as the cut-off is the prior sorting criteria. In comparison, algorithm 3 is showing a slight increase in unfulfilled orders since the cut-off time is used as a second sorting criterion. The focus on cut-off time in algorithm 4 is even less, while algorithm 5 is barely considering the cut-off time in its decision rules. The last mentioned is scoring the lowest in terms of automatic fulfilment. Nevertheless, the increased fraction of unfulfilled orders are still only varying in a range around 1-2 %. Since outbound line 109 processes, the most orders this fulfilment makes the most impact, where it is noted the number of orders extra this relates to yields a range of 92-169 (sum of both outbound lines) orders in comparison to the benchmark algorithm.

As mentioned, the impact of this fulfilment is yet included in the average time taken per pick. By prioritizing orders because of its cut-off time relatively inefficient pools are created (the irregular peaks shown in Figure 7.1 and Figure 7.2. In comparison to the benchmark heuristic and its variant, more others have to be prioritized manually for the other three methods; the performance of these pick runs is worse in terms of average picking times. Though, on average the other algorithms are still outperforming the current algorithm in almost all situations. Since the number of automatically unfulfilled orders of both outbound lines for each scenario is far below 500 orders, we conclude the small decrease in fulfilment does not create a significant impact for each of the tested algorithms.

[4]To simplify the notation, the term 'outbound line' is shortened by 'Line' in this Table

7.3 Calculation time

A third performance measurement is the calculation time per algorithm. As stated in Section 6.2.3, we cannot give certainty whether this algorithm fits within the allowed calculation time of four minutes. The absolute expected running time of the algorithm itself is therefore hard to identify. Nevertheless, indications in comparison to the benchmark heuristic are computed. Bol.com stated a ratio of algorithms performing ten times longer than the benchmark heuristic is worrying. For both scenarios, the average computation time per pick run in seconds for both outbound lines are provided in Table 7.6, including the sum of both outbound lines. Also, the ratio of the running time in comparison to the benchmark heuristic is provided.

	Scenario 1				Scenario 2			
Algorithm	**Line 108**	**Line 109**	**Total**	**Ratio**	**Line 108**	**Line 109**	**Total**	**Ratio**
1	77 s	94 s	171 s		56 s	72 s	128 s	
2	91 s	88 s	179 s	x 1.05	68 s	69 s	137 s	x 1.07
3	138 s	184 s	321 s	x 1.88	117 s	149 s	266 s	x 2.08
4	219 s	233 s	453 s	x 2.65	168 s	196 s	363 s	x 2.84
5	232 s	291 s	522 s	x 3.05	169 s	202 s	371 s	x 2.90

TABLE 7.6: Average calculation time per pick run per scenario in seconds

Table 7.6 only provides the average calculation times measured over every pick run. Based on analysis of the calculation time per pick run individually, all methods similarly tend to increase at the end of a day. Since the ratio between all algorithm remains approximately constant during the day, the averages themselves represent an accurate indication of the calculation time of every algorithm.
Looking at Table 7.6, the calculation time of the three new algorithms are reasonably higher in comparison to the benchmark algorithm and its variant. Furthermore, the difference in the results per scenario can also be observed, since the calculation times are way lower for scenario 2. Higher stock levels imply there are more allocation possibilities, whereby the algorithms have to perform more iterations to check to which zones all items have to be allocated. Though, the ratio's per algorithm are comparable between the scenarios.
Algorithm 5 consume the most calculation time on average, which is caused by the large set of possible combinations it seeks for choosing its zone selection. For this algorithm, the ratio of the calculation time is about three times as high in comparison to the benchmark algorithm. Algorithm 4 is behaving similarly but is a bit more time-efficient (that comes at a cost of higher average picking times).
In the testing environment, the average time per pick run is higher than the allowed four minutes (240 seconds), not even considering the outliers. However, as explained in Section 6.2.3, this absolute number is not a proper measurement of whether this algorithm would fit in the current system of bol.com.

7.4 Combination of methods

As is observed in section in Section 7.1, the greedy zones method (algorithm 5) is performing best in terms of the average time per pick. In this section, an additional experiment is performed to verify if the performance of algorithm 5 could be improved by testing a combination approach of the tested methods. Also in Section 7.1, it is shown the max min zone needed method (algorithm 3) is, relatively, performing most stable by selecting at least one 'hard' order in every pool. By analyzing Figure 7.1 and Figure 7.2, a combination of algorithm 3 and algorithm 5 is having the most potential to algorithm 5.

To check if a combination of methods would result in a significant difference, algorithm 5 is now set as the benchmark algorithm and multiple combination approaches are tested. These combinations differ in their moment in time where the strategy change from algorithm 5 to algorithm 3. Also in this additional experiment, paired sample t-tests are performed to state whether the differences are significant. Since

the previous results between both scenarios did not show any significant differences, this experiment is initially performed for scenario 1 only. Above that, the simulations are performed on the most utilized outbound line only, which is outbound line 109. The combinations which are tested are shown in Table 7.7, including its results.

Combination algorithm	Mean	T-value	p-value
Greedy zones method - full	1.00		
Combination 1 - change method after 16:00 (pick run 32)	1.008	0.448	0.332
Combination 2 - change method after 18:00 (pick run 40)	0.999	1.147	0.184
Combination 3 - change method after 20:00 (pick run 48)	0.996	1.604	0.108
Combination 4 - change method after 22:00 (pick run 56)	0.996	-1.289	0.157

TABLE 7.7: T-test combinations greedy zones and max min zone needed - outbound line 109

By the mean performance presented in Table 7.7, we see the average time per pick is narrowly close for all tested combination methods by a very small range of [**-0.004 X**, **0.008 X**] compared to the greedy zones method. The p-values of the results indicate the difference are not significant, as all p-values are above 0.05. By this same approach, other combinations of the newly designed methods are tested. However, none of the combinations of the created methods showed promising results.
Therefore, we conclude a combination of the tested methods is not able to improve the greedy zones method significantly. In other words, of all the tested methods the greedy zones method is best performing at every time period during an operating day.

7.5 Zone performance

Next to the stated performance measurements, the number of zones used per pool give valuable insights regarding the supply chain in the warehouse. More used zones per pool generally relate to more pick batches per pool, which could affect the flow between the picking process and the packing activities at the outbound lines. A stated boundary condition by bol.com is the total number of zones used for a pool may at most be 10 zones (Section 5.2). This maximum may only be exceeded by prioritized orders and the sweep runs at the end of an operating day. In Figure 7.3, the distribution of the number of zones used per pool is presented per method. The average frequencies per number of zones are taken over all simulated days in both scenarios.

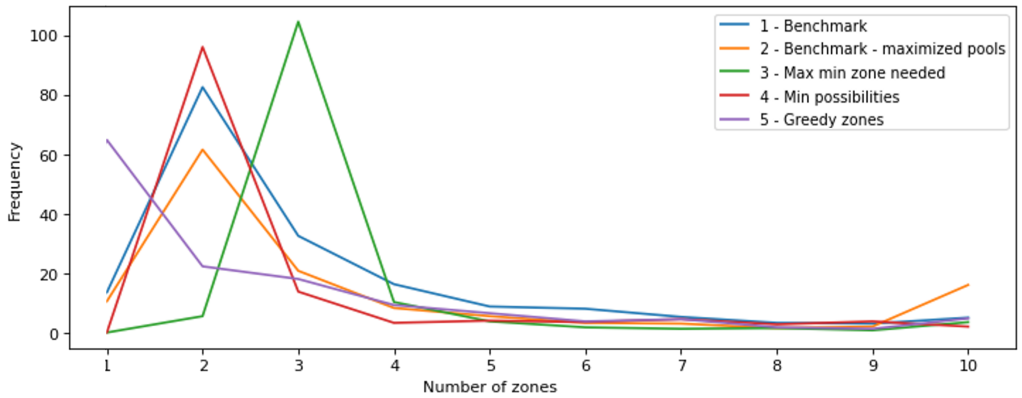

FIGURE 7.3: Number of zones per pool

Looking at the number of zones used per pool in Figure 7.3, large differences are observed by the number of pools that contain only one zone; the greedy zones method (algorithm 5) succeed to create pools by using one zone only, while the rest of the algorithms do this sporadically. The other algorithms are using most of the time two or three zones. Also, we note that the variant on the benchmark method

is consistently using fewer zones than the original version of the benchmark method, as the pools are maximized and thus fewer pools are needed. Furthermore, we see the maximum number of zones is not frequently needed for all algorithms. As the newly designed algorithms do not need more zones than the benchmark algorithm, this indicates the flow within the warehouse would not be affected negatively.

In Figure 7.4, the average number of zones used per pool are presented per pick run over the simulated days. Since the results over outbound line 108 and outbound line 109 are corresponding, we focus only on the results for outbound line 109. Again, the averages are computed over scenario 1 and scenario 2 together, since the differences between the results are negligible.

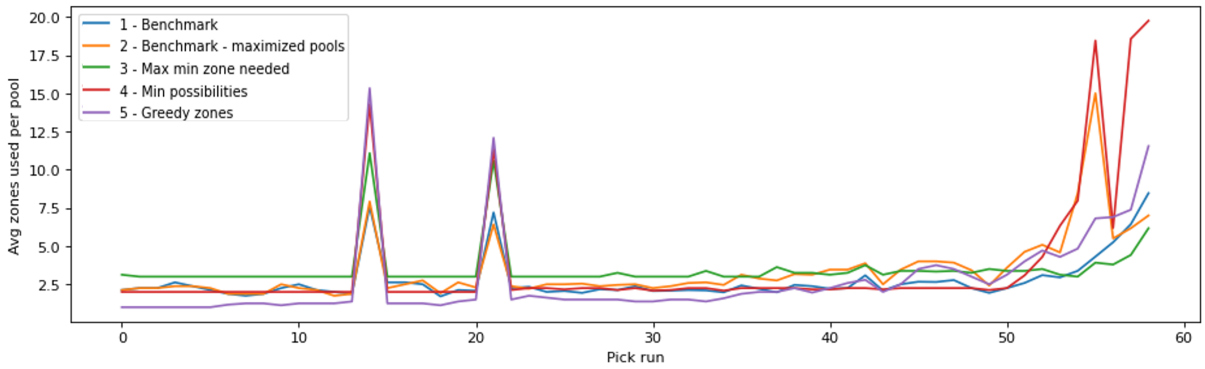

FIGURE 7.4: Average number of used zones per pool per pick run

Aligning Figure 7.4 to the Figures 7.1 and 7.2 of Section 7.1, clear relationships between the average used zones and the average time per pick run can be observed. Algorithms 3, 4 and 5 have a constant number of zones at the first half of a day, which is only disturbed by cut-off prioritization, where more zones are required. The benchmark algorithm chooses an order on its cut-off time, thus the number of zones in use vary (which can be related to the fluctuating performance in average time per pick). Algorithm 3 choose to assign the hardest order (in terms of minimum needed zones), which results in many zones for the first pick runs. As a result, constantly higher average times per pick are obtained. Algorithm 4 selects at least two zones per pool, which also result in a constant performance (besides the peaks due to cut-off prioritization orders). Algorithm 5 choose preferably only one zone when it is possible to fill a full pool by orders that can all be allocated to that zone. The performance of this algorithm is way better at the beginning of an operating day, while it is also not performing worse than order algorithms at the end of an operating day.
Furthermore, it is noted the number of zones per pool increases along the operating day ends. For all algorithm holds: more zones are needed to add more orders to the pools. As the benchmark algorithm suggests, bol.com is experiencing this in its current way of working. Experts in the field of bol.com state as long as the number of used zones do not increase drastically in comparison to the current situation, this is acceptable.

7.6 Discussion results

In this section, the main findings of this section are summarized. When comparing the average time per pick for both scenarios regarding the stock levels, more or less the same performance is observed. Despite a small difference in the performance of outbound line 108, this is not sufficient proof to state the scenarios are notably different. And since the performance of all algorithms is similar, we can conclude the simulated scenarios do not have impact on which method to use.
As expected, the results per outbound line differ in terms of the average time per pick. The explanation for this is twofold: the maximum pool size is smaller for outbound line 108 and the processed items are generally heavier and larger in size dimensions.

The benchmark algorithm performed similar results to what is known from the available historical data, which indicates the testing environment is representative. In comparison to the benchmark algorithm, we note that a variant (algorithm 2) of this algorithm can save around **0.033 X** seconds by adjusting the decision to obtain larger pools, which is significant by the performed paired sample t-tests. The strategy of algorithm 3 is choosing an order that requires the highest number of zones, which results in a constant performance where relative many zones are used. This high number of zones is performing slightly better on outbound line 108 than the benchmark algorithm, while the results on outbound line 109 are not performing significantly better. Despite this method being the most constant of all methods, in terms of average time per pick, this is not the best performing method.
Except for outbound line 108 in scenario 2, algorithm 4 is showing significant differences in the results per pick in all other situations. Regarding the average time per pick, it is the second-best performing algorithm. By prioritizing items that are scarcely available (in only one zone), some complex orders are dealt with effectively, which can reduce the current performance.

According to the average time per pick, the best algorithm is the greedy zones method (algorithm 5). For every scenario and every outbound line, the decrease in average time per pick is at least **0.075 X** seconds in comparison to the benchmark algorithm, which is in every case significant according to the paired samples t-tests. The method suggests the best strategy to use is to keep minimizing the used zones while creating full pools. During the day, more zones are needed to fill full pools and consequently, the average time per pick increase heavily by this greedy decision making. Though, this increase is even as worse as the benchmark algorithms and the other designed algorithms, which makes this strategy on average the best one to use.
Comparing the algorithms on fulfilment, we observe an increase in the number of orders that are automatically unfulfilled by the algorithm. Nevertheless, the number of orders this concerns is manageable for bol.com, as this amount of orders usually could be processed in one pick run.

A drawback of the greedy zones method is the calculation time. The summed calculation time for both outbound lines is expected to be three times higher than the benchmark heuristic, which must be considered when advising the best algorithm to bol.com.

Since the greedy zones method is best performing, additional experimenting is performed to investigate whether the average time per pick could be exploited further. By experimenting with combinations of the created methods and varying the changing moment over time, no significant differences are obtained. To improve the greedy zones method, even more, modifications have to be made or a completely new method should be introduced.

At last, insight into the distributions over the used zones are provided. Clear correlations between the number of zones used and the average time per pick are identified by comparing the performances per pick run. Furthermore, insights by the created pools indicate no more zones are used by the greedy zones method, in comparison to the current situation.

Chapter 8

Conclusion, limitations and recommendations

This chapter points out the most important findings of this report into a conclusion, which provides an answer to the main research question of this research. Further on, the limitations of this research are discussed, followed by a set of recommendations.

8.1 Conclusion

The stated goal of this research was to optimize the pooling allocation problem for bol.com. In accomplishing this, the logic of the current pooling algorithm had to be altered, such that the average time per pick decreases, among the allocation of items of multi-orders to the zones. While reducing the average time taken per pick, the fulfilment is carefully analyzed.

How can the time per pick for multi-orders in the BFC be reduced by improving the current pooling algorithm, while maintaining the fulfilment criteria?

By means of theoretical research, an estimation function was defined that uses historical data to predict the average time per pick based on the number of items that are assigned to a poolzone, which is the output of the pooling algorithm. By this estimation function, the pooling performance can be evaluated. In addition, a simulation model was created to test a set of solution methods, designed in this research. Experiments are performed to evaluate the average time per pick over entire operating days, including online arrival and varying cut-off times of orders.

Based on the results of the experiments in this research project, it can be concluded that the simulation model represents the current performance. Next to the current method, four other methods are tested. These methods are all constructive heuristics since the mathematical formulated problem cannot be solved exactly within the limited calculation time that is required. The results of the tested constructive heuristics obtained the following insights:

1. One main finding of this research is that by maximizing the pools to their maximum extent (in terms of orders) the picking performance increases. Comparing a variant on the current method to the current method that applies this maximization of pools, the average time taken per pick could be reduced by an average of **0.033 X** seconds (weighted average of both outbound lines).

2. In comparison to the current algorithm, the best strategy to reduce the average time per pick is to minimize the number of zones that is used per pool. By analysis, it is derived the majority of multi-orders can be fulfilled by assigning all its items to one zone. As a consequence, by minimizing the number of zones an algorithm succeeds in needing only one zone for more than half of an operating day. When the order set becomes smaller at the end of an operating day, the average time per pick increases as more zones are needed to fill up pools to their maximum extent. However, this increase seems inevitable, as all other algorithms (including the current algorithm) are showing the same behaviour.

(a) The method that applies this strategy showed the average time per pick can be reduced to **- 0.075 X** seconds per pick. Based on the demand over 2020 and the hourly wage of operators, this reduction saves an annual amount of at least **€ XXX,XXX** expressed in terms of labour costs.

(b) This decrease in average time per pick of this method is accompanied by a decrease in fulfilment, as more orders are not automatically fulfilled by the algorithm. However, the number of unfulfilled orders that need manual prioritization is not that high, as this concerns at most 1.5 % more of the orders, which is in total still 2 % of the total processed multi-orders. This decrease in fulfilment is acceptable for bol.com, as this fraction of orders will be prioritized in the real-world scenario. The impact on the average time per pick is included in the simulation model, which implies this method is still performing better than the current algorithm.

(c) The calculation time of the best performing method is expected to be three times as long as the current algorithm. However, the exact calculation time should be investigated in a further experiment, since the calculation time of the testing environment is not comparable to the real-world situation. At this point, we cannot guarantee this method provide solutions for all outbound lines within the allowed time period of four minutes.

8.2 Limitations

During this research, assumptions and simplifications are made that resulted in a set of limitations:

- The created estimation function which measures the output of all tested pooling algorithms is derived by a very generic approach. This function estimates the average time per pick under the assumption every item is equal in size and weight, which is in practice varying by a large extent of different items bol.com is storing in BFC1. Based on the estimation function, a very rough estimate is created which calculates how many pick batches per pool are expected. The number of expected pick batches would help to indicate the maximum pool size, since the number of pick batches (and thus, totes) determines the flow of the stingray (capacity buffer), and thereby the process at the outbound lines. To achieve a refined estimation of the expected pick batches, size dimensions and weight of each item should be included. However, this comes at a cost of more computation time, which should be considered if a more accurate estimation is desired.

- A second limitation is the simulated request of pick runs over the day is remain constant. In practice, bol.com has operators controlling the flow within the warehouse and steering the process continuously. Pick runs could vary heavily from size and even the time between successive pick runs fluctuates in the real situation. This could affect the process in terms of fulfilment since the number of available orders is more varied over time. Also, a higher pick run request implies more orders have to be added while the allowed calculation time is still only four minutes. Varying pick run requests are not considered in this research yet.

- Another limitation is the mathematical model is not representing this research problem entirely, since the cut-off time of orders and measurement of the quality of leftover orders are not included. As only one pick run is considered, the objective function could only be used to evaluate the performance of all methods afterwards.

- Despite the testing environment representing the current performance in terms of average time per pick and fulfilment, the calculation time is measured inaccurate. The performed experiments can only provide rough indications related to the current algorithm. Since the simulation model is built by other software that bol.com is using, calculation times are hard to compare. Besides, the extraction of the data and the updating procedures (stock levels and available orders) are expected

to consume more time in the simulation environment than in reality. Furthermore, the recreated current algorithm contains some simplifications in the testing environment, which is also saving calculation time. However, the extent of all these differences is unclear.

8.3 Recommendations

1. **Verify calculation time in further experimentation**
   The first recommendation is to experiment further with the greedy method that minimizes the number of used zones per pool. As this is the best performing algorithm based on the different performance measures, implementing this algorithm would lead to significant cost savings. Experiments performed in the WMS of bol.com should point out whether the calculation time of this strategy would fit in practice. Also, the impact on the parameters should be carefully analyzed, since simplifications are made in this research.
   In case this experiment demonstrates an actual unaccepted calculation time, multiple decisions can be made. As a first option, bol.com can consider extending this allowed running time. In that case, the control room should move its decision making slightly forward in comparison to their current way of working. The impact of this running time should be further investigated among all involved stakeholders within bol.com.
   As a second option, the method could be applied for specific cases only. For instance, since outbound line 109 processes the most orders the combination of running the greedy zones method on outbound line 109 and for the other outbound lines the improved version of the original algorithm. Another possibility is to run the greedy zones method at peak moments when a large set of orders is available. The results per pick run have shown in all scenarios, the main improvement of the solution method is caused by the better performance in the first series of pick runs (when many orders are available).
   In case neither an extension of the calculation time nor a combination of methods is acceptable for bol.com, the remaining advice is to apply the tested variant of the current algorithm. By the performed experiments, this algorithm is able to reduce the average time per pick by 0.8 seconds for items of multi-orders. By the change in decision making, larger pools are created in terms of orders, which would not harm the calculation time.

2. **Monitor processed multi-orders**
   A second advice to the company is to monitor the characteristics of multi-orders on a monthly basis. Insight into the type of multi-orders that are regularly processed support the decision-making of which strategy will perform best. For instance, when in a year significantly more multi-orders are containing a cut-off time during the operating day, other methods might perform better than the results of this research point out. A dashboard of the aspects like size, quantities, the availability of items per zone, minimum needed zone of orders and its cut-off time will provide continuous insight into the distribution of multi-orders.

3. **Align storage strategies**
   As is observed by this research, the available stock is influencing the allocation possibilities of items and is thereby indicating how hard items are to assign. Bol.com is currently applying random storage policies to distribute items over multiple zones. However, these strategies are not aligned to the picking process of multi-orders yet. When developing more aligned logic into the put-away strategies this would support the picking performance. For multi-orders, the time per pick would decrease if items are available in as many zones as possible. Additional research should be performed to define the best stock allocation strategy for multi-orders, including analysis of which indicators need to be defined to measure the relation between the stocking strategy and the picking productivity.

4. **Analyse impact pool parameters**
   Another piece of advice is to analyse the impact of the parameters that are affecting the pool sizes. As this research concludes, larger pools contribute to lower average times per pick. Currently, the maximum size of a pool in terms of orders is depending on the capacity of the sorting station per outbound line. The impact of extending the sorting station of outbound lines (and thereby the maximum pool sizes) should be investigated. Furthermore, the maximum number of zones to be chosen for a pool was set at 10 zones in this research. Since the impact of this parameter is currently still vague, experiments that measure the effect of changing this parameter could give valuable insights regarding the flow within the warehouse. Increasing the number of zones could lead to disruptions at the sorting and packing process, whereof the accompanying costs are not clearly defined yet. As a first step, insights into the costs of these inefficiencies concerning the flow should be defined, which would help determine the optimal parameters regarding the pool sizes.

5. **Reinforcement learning**
   In the performed research, the designed logic was focused on the set of orders that had to be chosen at that point in time. No attention was paid to the remaining order set, which is of impact for the performance of the successive pick runs that follow. Since the dynamic order arrival system of bol.com change over time, learning methods could be appropriate to evaluate the available order set that is left over.
   An interesting field within these learning methods is reinforcement learning, in which methods are designed to learn from themselves and perform actions based on the interaction it receives based upon their previous decisions. Recent advancements in the field of reinforcement confirm that learning techniques are able to solve large-scale combinatorial optimization problems.
   Also in warehousing, the first studies are performed recently. In a recent study of Cals et al. (2021), the applicability of deep reinforcement learning into order batching was introduced. Inspired by this research, reinforcement learning could be used to optimize the decision making of selecting orders, based on multiple sets of indicators that can measure the complexity factors of multi-orders in terms of its size, stock availability among the zones and its cut-off time. By the use of reinforcement learning the model can evaluate the real-time performance and learn the optimal weights for these indicators, to come up with the best strategy at every point in time. Applicable methods in this field might be relevant for bol.com in the near future, which the company should watch for.

Appendix A

Additional warehouse context bol.com

A.1 Overview warehouses bol.com

Name	Location	Characteristic
BFC1	Waalwijk	Small and medium products
Veerweg	Waalwijk	Medium and big products
Centraal boekhuis	Culembourg	Books only
BFCXL	Nieuwegein	XL products
BRC	Waalwijk	Returning centre

TABLE A.1: All operating warehouses of bol.com in 2021

A.2 Overview multi-order outbound lines BFC1

Number outbound line	Name outbound line	Minimum pool size	Maximum pool size
106	Multi high risk	20	60
107	Multi manual VAS	40	60
108	Multi manual	20	60
109	Multi automatic sorter	50	80

TABLE A.2: All operating warehouses of bol.com in 2021

Appendix B

Flowchart current pooling algorithm

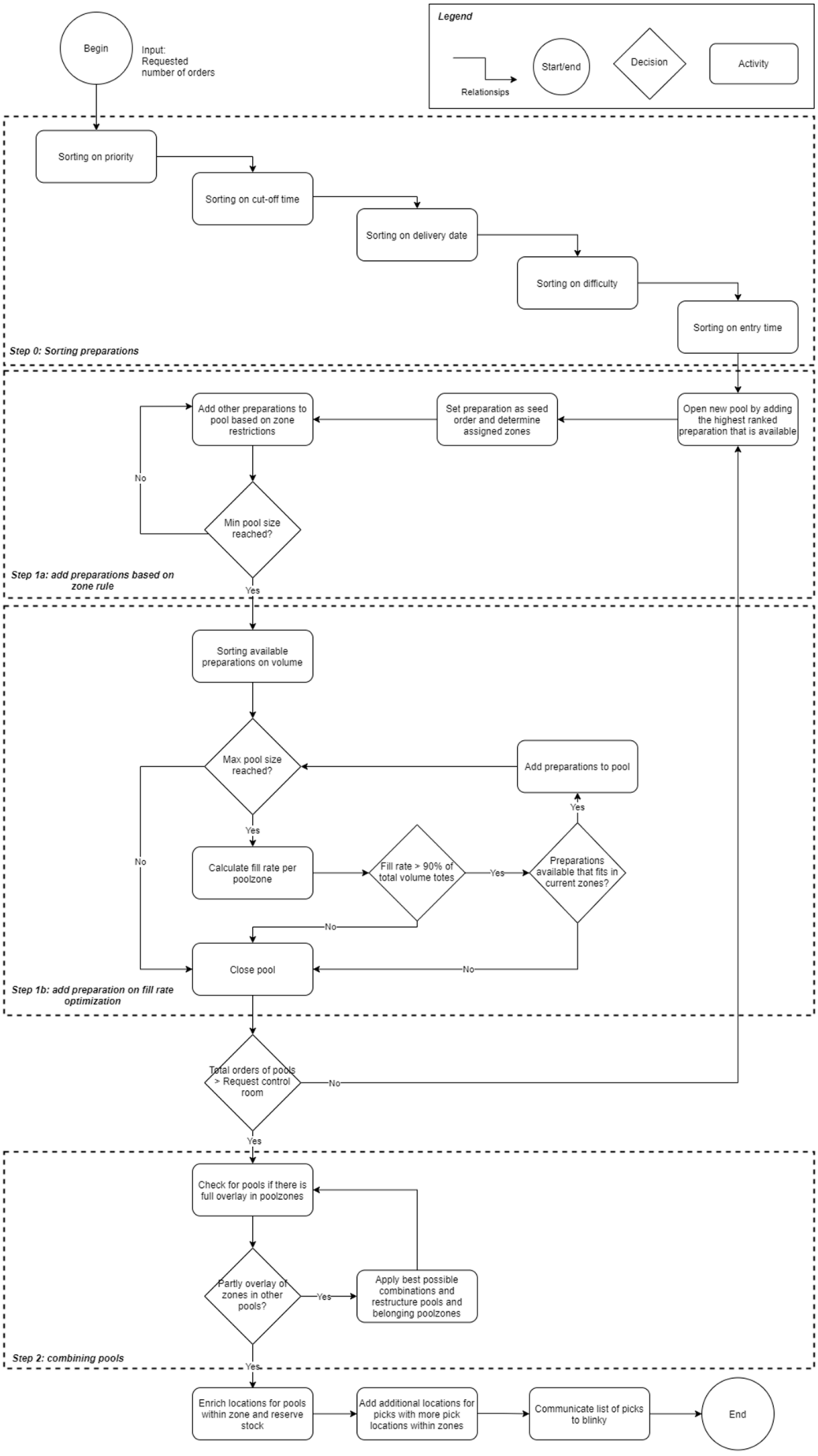

FIGURE B.1: Flowchart pooling algorithm

Appendix C

Pseudo codes current pooling algorithm

C.1 Pseudocode step 1 pooling algorithm

Algorithm 1: Assigning orders to pools

```
Input: Request number of control room, sorted list of available orders
Result: List of items assigned per poolzone that is between the allowed range of a poolsize
for outbound line in total set of outbound lines do
    while requested quantity of orders per outbound line not yet reached do
        while current pool size in orders is lower than the minimum pool size do
            Pick highest ranked order on available list;
            Set picked order as seed order of new pool;
            Determine zones ;
            for all available orders in the sorted list of available orders do
                if order contains same zones as seed order then
                    Add order to pool;
                end
            end
            for all available orders in the sorted list of available orders do
                if order contains same zones as seed order +1 extra zone then
                    Add order to pool;
                    Rule out relaxation zone;
                    (Continue by the updated set of zones);
                end
            end
        end
        while maximum pool size is not reached do
            sort the list of available orders on volume for each pool do
                for each item set assigned to a zone in a pool do
                    Calculate fillrate (volume of total items/volume of totes);
                    if fillrate per tote is < 90% then
                        for order in list of available items do
                            if item set match to zones and updated fill rate <100% then
                                Add order to pool;
                            end
                        end
                    end
                end
            end
        end
        Close pool;
    end
end
```

C.2 Pseudocode step 2 pooling algorithm

Algorithm 2: Assigning orders to pools

Input: : List of items assigned per poolzone that is between the minimum and maximum of a poolsize
Result: Combined list of items assigned per poolzone (if possible)
for *each pool in total created pool* **do**
    for *each zone in pool* **do**
        Check overlay in zones at other pools;
        Save overlay in assigned zones;
    end
end
while *overlay exists* **do**
    if *maximum number of orders in pool is not exceeded* ***and*** *Maximum number of zones per pool is not exceeded* **then**
        sort all possible combinations based on criteria [a];
        Apply best combination;
        Restructure pools and the corresponding assigned zones;
    end
end

[a] Criteria are based on:

1. The number of pools that are used for combinations (the less pools involved, the more efficient).
2. The number of zones per pool that are used (ascending order, the less the better)
3. The number of orders in the new pool (descending order, preferably as close as possible to the maximum pool size)

Appendix D

Estimation function outbound line 108

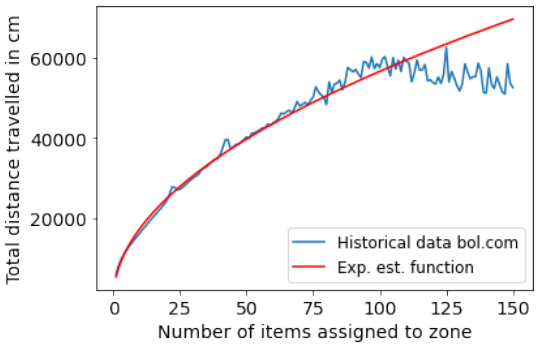

FIGURE D.1: Distance function outbound line 108

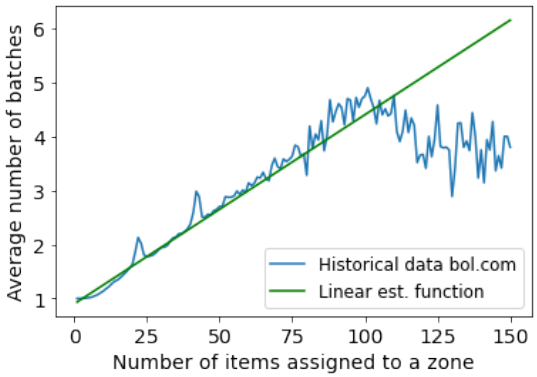

FIGURE D.2: Pick batches function outbound line 108

$$Expectedpickbatches(n_{zp}) = a + n_{zp} * b \tag{D.1}$$

$$Expectedtotaldistance(n_{zp}) = c * n_{zp}^{d} \tag{D.2}$$

In Table D.1, the parameters for the estimation functions in Equation D.1 and Equation D.2 are given. Furthermore, the values for α, β and γ are also provided and implemented in the estimation function of the average picking time of items on outbound line 108, represented in Equation D.3. For all parameter values, the same weighted least squares method was applied as for outbound line 109.

Parameter	Function	Outbound line 108
a	Lower bound for pick batch estimation	XXX
b	Expected pick batch per new item added	XXX
c	Average expected distance per pick in cm	XXX
d	Exponential power fraction the added distance decreases	XXX
α	The set-up time of a pick batch in seconds	XXX
β	Travelling speed of an operator in meter per second	XXX
γ	Time to perform the pick in seconds	XXX

TABLE D.1: Optimal values for estimating functions

$$F(n_{zp}) = \frac{XXX * Expectedpickbatches(n_{zp}) + XXX * Expectedtotaldistance(n_{zp})}{n_{zp}} + XXX \tag{D.3}$$

The current bias of the function is now 647.7 (for the calculation of the bias, see 4.3.1). To compensate, a linear bias correction is modified, which reduces the total bias to 308.0. By applying a small correction function of $0.03 * n_{zp}$, the increasing bias is tempered. In Figure D.4, the bias of the original function is plotted together bias including bias-correction is plotted. And as can be seen in Figure D.3, the function containing the bias correction is more accurate than the original function.

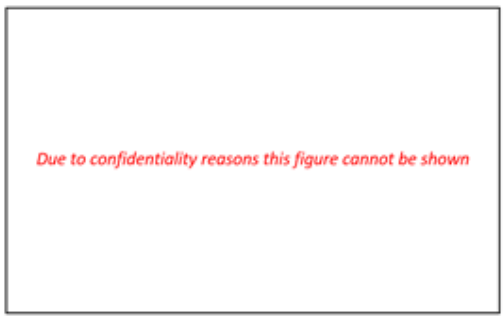

FIGURE D.3: Estimation function outbound line 108

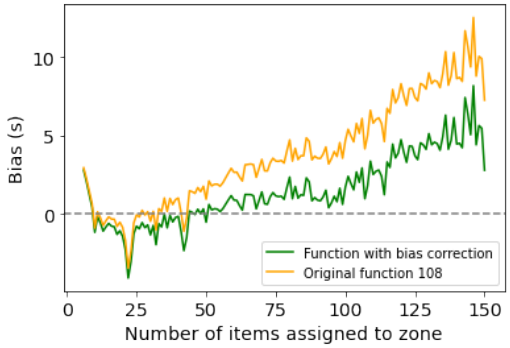

FIGURE D.4: Bias correction outbound line 108

For comparison, the function of outbound lines 108 and 109 are shown in Figure D.5.

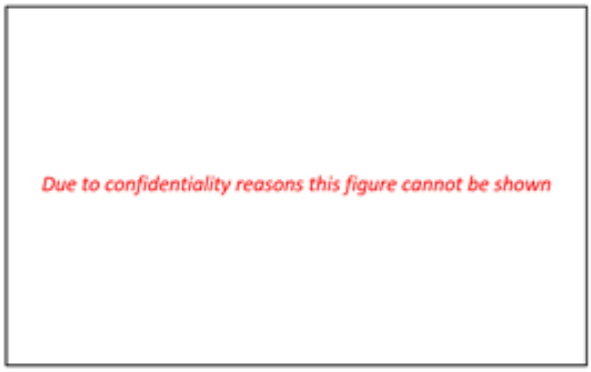

FIGURE D.5: Estimation functions outbound line 108 and outbound line 109

Appendix E

Pseudo codes constructive methods

E.1 Maximum minimum zones needed method

Algorithm 3: The maximum minimum zones needed method

```
Input: Request number of orders control room, sorted list of available orders on minimum
  zones needed (descending)
Result: Lists of orders containing items assigned to zones and belonging to pools
while requested number of orders is not reached yet do
    Create new pool;
    Pick highest ranked order on available list;
    Set order as seed order of new pool;
    Determine zones used for the pool;
    for all available orders in the sorted list of available orders do
        if order can be assigned to selected zones in pool then
            Add order to pool;
            if maximum poolsize is reached then
                Close pool;
            end
        end
    end
    if minimum poolsize is not reached then
        for all zones not used in pool do
            Append zone to set of used zones;
            for all available orders in the sorted list of available orders do
                if order can be assigned to selected zones in pool then
                    Add order to potential pool list;
                end
                Determine zone with the highest available orders;
                Set potential order list of zone as final pool list;
                if pool list > maximum pool size then
                    Truncate pool list on sorted ranking;
                end
            end
        end
    end
    while not all items are assigned do
        for all unassigned items in pool do
            for all zones in zone selection do
                Check assignment of item in zone;
            end
        end
        Assign items to zone with most left items;
    end
    Close pool;
end
```

E.2 Minimum possibilities method

Algorithm 4: Minimum possibilities method

```
Input: Request number of orders control room, list of orders sorted ascending on availability of
  most scarce item of the order
Result: Lists of orders containing items assigned to zones and belonging to pools
while requested number of orders is not reached yet do
    Create new pool;
    for all orders containing an item which is only available in one zone do
        Count zone of item that is available in one zone
    end
    Set zone with highest count as starting zone in zone selection;
    while less than 4 zones are used or pool size => maximum pool size do
        for zones not in zone selection do
            for all orders containing an item which is only available in one zone do
                if order can be completely assigned to zones in zone selection then
                    Save order in potential pool list;
                end
            end
        end
        Add zone that can fulfil most orders to zone selection;
    end
    Set potential order list of best combination as pool list;
    if pool list > maximum pool size then
        Truncate order list on sorted ranking;
    end
    if minimum poolsize is not reached then
        for all zones not used in pool do
            Append zone to set of used zones;
            for all available orders in the sorted list of available orders do
                if order can be assigned to selected zones in pool then
                    Add order to potential pool list;
                end
                Determine zone with the highest available orders;
                Set potential order list of zone as final pool list;
                if pool list > maximum pool size then
                    Truncate pool list on sorted ranking;
                end
            end
        end
    end
    while not all items are assigned do
        for all unassigned items in pool do
            for all zones in zone selection do
                Check assignment of item in zone;
            end
        end
        Assign items to zone with most left items;
    end
    Close pool;
end
```

E.3 Greedy zones method

Algorithm 5: The greedy zones method

Input: Request number of orders control room, list of orders sorted cut-off time primarily, minimum possibilities secondary and the minimum number of zones needed as a tertiary criterion

Result: Lists of orders containing items assigned to zones and belonging to pools

```
while requested number of orders is not reached yet do
    Create new pool;
    Initialize the zone selection as an empty set;
    while maximum pool size is not reached and used zones of pool is less than 10 do
        for all zones do
            Add zone to potential zone selection;
            for all available orders in sorted list do
                if order can be completely assigned to zones in potential zone selection then
                    Save order in potential pool list;
                end
            end
        end
        Add zone with the longest potential pool list to the definite zone selection;
    end
    Add potential order list of definite zone selection to the pool order list;
    if pool list > maximum pool size then
        Truncate order list on sorted ranking;
    end
    while not all items are assigned do
        for all unassigned items in pool do
            for all zones in zone selection do
                Check assignment of item in zone;
            end
        end
        Assign items to zone with most left items;
    end
    Close pool;
end
```

Appendix F

Results per pick run - outbound line 108

F.1 Scenario 1 - high stock levels

In Figure F.1, the average result per pick run for outbound line 108 is shown. Note that the x-axis denotes the pick run number, which relates to timestamps over a simulated day. Starting an operating day at 8:00 AM, a constant demand of 100 orders per hour is requested for outbound line 108. At pick run 4 and pick run 6, an occasional increase in the average picking time is observed. This pick runs consists of items that are prioritized by its cut-off time, which were around 12:30 PM and 14:15 PM respectively.

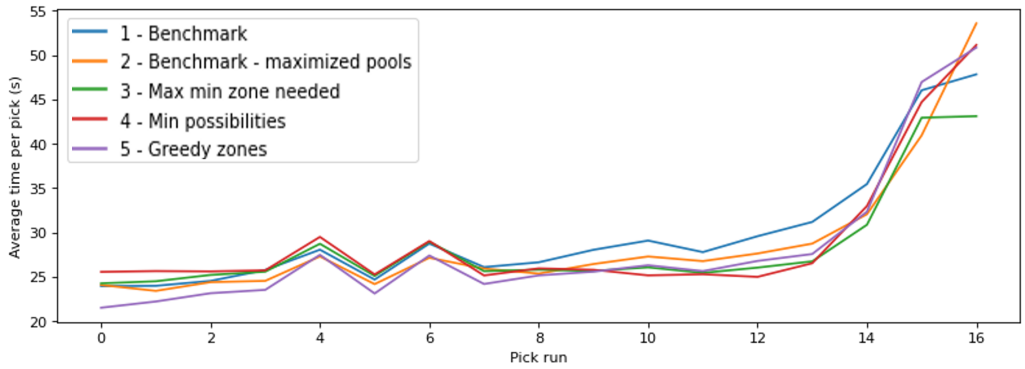

FIGURE F.1: Performance pick runs - high stock levels - outbound line 108

F.2 Scenario 2 - low stock levels

In Figure F.2, the results per pick run of all algorithms are shown for outbound line 108. Even for this scenario, the drastic increase in average time per pick during the day is inevitable for all tested algorithms; all algorithms perform badly at the end of an operating day. From pick run 6, the benchmark algorithm is consistently scoring the highest average time per pick, which declares the highest average performance.

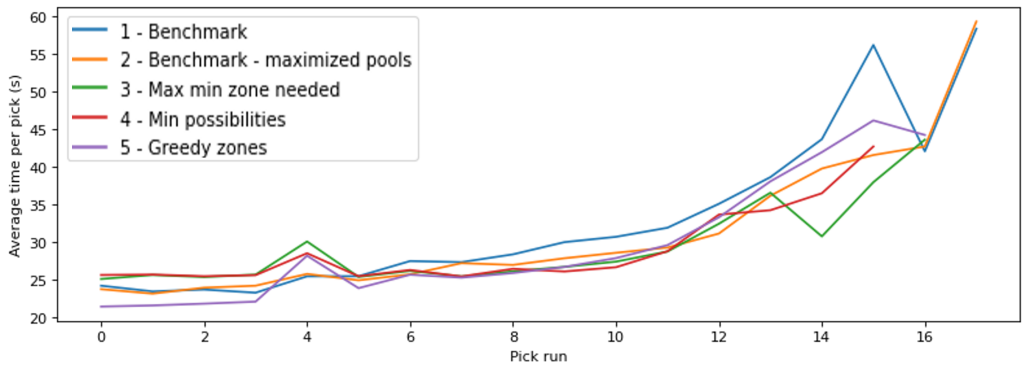

FIGURE F.2: Performance pick runs - low stock levels - outbound line 108

Appendix G

Comparison performance measurements

In Table G.1, the average time per pick and the number of unfulfilled orders per algorithm are given. The results on both performance measurements are compared to the current algorithm that is set as a benchmark algorithm. The performances in average time per pick which show a significant decrease (according to the paired sample t-tests) are marked green, while the other average picking times are marked red (thus, the results of these algorithms do not show a significant difference). For the number of unfulfilled orders, order numbers above 100 are marked orange. As the number of unfilled orders was above 500 orders, it was marked red since that would be alarming for bol.com.

	Scenario 1				**Scenario 2**			
	Line 108		**Line 109**		**Line 108**		**Line 109**	
Algorithm	**Mean**	**U.f.f.o.**[1]	**Mean**	**U.f.f.o.**	**Mean**	**U.f.f.o.**	**Mean**	**U.f.f.o.**
1	1.44 X	20	1.09 X	40	1.46 X	3	1.09 X	33
2	-0.061 X	-2	-0.033 X	-6	-0.054 X	+1	-0.034 X	+3
3	0.066 X	+9	-0.013 X	+22	-0.053 X	+11	-0.004 X	+14
4	-0.064 X	+19	-0.045 X	+135	-0.063 X	+15	-0.042 X	+62
5	-0.101 X	+21	-0.091 X	+148	-0.090 X	+16	-0.079 X	+76

TABLE G.1: Fulfilment performance per algorithm per scenario

[1] Number of automatically unfulfilled orders per algorithm

References

Aboelfotoh, A., Singh, M., & Suer, G. (2019). Order batching optimization for warehouses with cluster-picking. *Procedia Manufacturing*, *39*, 1464-1473.

Alonso-Ayuso, A., Albareda-Sambola, M., Molina, E., & de Blas, C. S. (2009, October). Variable neighborhood search for order batching in a warehouse. *Asia Pacific Journal of Operations research*.

Azadnia, A. H., Taheri, S., Ghadimi, P., Saman, M. Z. M., & Wong, K. Y. (2013). Variable neighborhood search for the order batching and sequencing problem with multiple pickers. *The Scientific World Journal*(3).

Bol.com. (2021). *Geschiedenis van bol.* Retrieved from https://www.bol.com/nl/nl/m/geschiedenis-van-bolcom/

Cals, B., Zhang, Y., Dijkman, R., & van Dorst, C. (2021). Solving the online batching problem using deep reinforcement learning. *Computers Industrial Engineering*, *156*.

Chen, R.-C., & Lin, C.-Y. (2020). Optimizing a dynamic order-picking process. *The Journal of Supercomputing*, *76*, 6258-6279.

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

de Koster, M., van der Poort, E., & Wolters, M. (1999). Efficient orderbatching methods in warehouses. *International Journal of Production Research*, *39*(7), 1479-1504.

de Koster, R., Le-Duc, T., & Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, *182*, 481-501.

Duc, T. L., & Koster, R. B. D. (2005). Travel distance estimation and storage zone optimization in a 2-block class-based storage strategy warehouse. *International Journal of Production Research*, *43*, 3561-3581.

Franco Caron, G. M., & Parego, A. (2010). Routing policies and coi-based storage policies in picker-to-part systems. *International Journal of Production Research*, *36*, 713-732.

Gademann, N., & van de Velde, S. (2007). Order batching to minimize total travel time in a parallel-aisle warehouse. *IIE Transactions*, *37*, 63-75.

Gil-Borras, S., Paardo, E. G., Alono-Ayuso, A., & Duarte, A. (2019). Basic vns for a variant of the online order batching problem. *Pergamon*.

Gil-Borras, S., Paardo, E. G., Alono-Ayuso, A., & Duarte, A. (2020). Grasp with variable neighborhood descent for the online order batching problem. *Journal of Global Optimization*, *78*, 295-325.

Hamzehi, S., Bogenberger, K., Franeck, P., & Kaltenhauser, B. (2019). Combinatorial reinforcement learning of linear assignment problems. *Intelligent Transportation Systems Conference*.

Heerkens, H. (2012). *Geen probleem.* van Winden Communicatie.

Henn, S., Koch, S., & Wäscher, G. (2012). Order batching in order picking warehouses: A survey of solution approaches. , 105-137.

Henn, S., & Schmid, V. (2013). Metaheuristics for order batching and sequencing in manual order picking systems. *Computers Industrial Engineering*, *66*.

Henn, S., & Wäscher, G. (2012). Tabu search heuristics for the order batching problem in manual order picking systems. *European Journal of Research*.

Ho, Y.-C., Su, T.-S., & Shi, Z.-B. (2008). Order-batching methods for an order-picking warehouse with two cross aisles. *Computers Industrial Engineering*, *55*, 321-347.

Ho, Y.-C., & Tseng, Y. (2006). A study on order-batching methods of orderpicking in a distribution centre with two crossaisles. *International Journal of Production Research*, *44*, 3391-3417.

Hsua, C.-M., Chen, K.-Y., & Chen, M.-C. (2005). Batching orders in warehouses by minimizing travel distance with genetic algorithms. *Computers in Industry*, *56*, 169-178.

Kovalyov, M. Y. (1997, June). Batch scheduling and common due date assignment problem: an np-hard case. *Discrete applied mathematics*.

Kuhn, H., Schubert, D., & Holzapfel, A. (2021). Integrated order batching and vehicle routing operations in grocery retail – a general adaptive large neighborhood search algorithm. *European Journal of Operational Research*, *294*.

Kulak, O., Sahin, Y., & Taner, M. E. (2012). Joint order batching and picker routing in single and multiple-cross-aisle warehouses using cluster-based tabu search algorithms. *Flexible Services and Manufacturing Journal*, *24*, 51-80.

Le-Duc, T. (2005). *Design and control of efficient order picking processes* .

Liang, J., Wu, Z., Zhu, C., & Zhang, Z.-H. (2019, May). An estimation distribution algorithm for wave-picking warehouse management. *Journal of Intelligent Manufacturing*, *156*.

Lourenço, H. R., Martin, O. C., & Stützle, T. (2019). Iterated local search: Framework and applications. *Handbook of Metaheuristics*, 129-168.

Malengo, A., & Pennecchi, F. (2013). A weighted total least-squares algorithm for any fitting model with correlated variables. *Metrologia*.

Marchet, G., Melacini, M., & Perotti, S. (2010). A model for design and performance estimation of pick-and-sort order picking systems. *Journal of Manufacturing Technology Management*, *22*, 261-282.

Mejari, M., Breschi, V., Naik, V. V., & Piga, D. (2020). A bias-correction approach for the identification of piecewise affine output-error models output-error models. *IFIAC PapersOnLine*, *53*, 1096-1101.

Menéndez, B., Bustillo, M., Pardo, E. G., & Duarte, A. (2017). General variable neighborhood search for the order batching and sequencing problem. *European Journal of Operations Research*, *263*, 82-93.

Menéndez, B., Pardo, E. G., Alonso-Ayuso, A., Molina, E., & Duarte, A. (2017). Variable neighborhood search strategies for the order batching problem. *Computers Operations Research*, *78*, 500-512.

Mora, J. F., Nait-Abdallah, R., Lozano, A. J., Montoya, C., & Otero-Caicedo, R. (2020). Batch assignment of parallel machines in an automotive safety glass manufacturing facility. , *24*.

Nieuwenhuyse, I. V., de Koster, R. B., & Colpaert, J. (2007). Order batching in multi-server pick-and-sort warehouses.

Pan, C.-H., & Liu, S.-Y. (1995). A comparative study of order batching algorithms. *Pergamon*, *23*(6), 691-700.

Scholz, A., Schubert, D., & Wäscher, G. (2017). Order picking with multiple pickers and due dates – simultaneous solution of order batching, batch assignment and sequencing, and picker routing problems. *European Journal of Research*.

Shao, W., Shao, Z., & Pi, D. (2021). Effective constructive heuristics for distributed no-wait flexible flow shop scheduling problem. *Computers and Operations Research*, *136*.

Tompkins, J. A., White, J. A., Bozer, Y. A., Frazille, E. H., Tanchoco, J., & Trevino, J. (2010). *Facilities planning* (2nd ed.). John Wiley Sons Inc.

University, C. (2021). *Dictionary cambridge.* Retrieved from https://dictionary.cambridge.org/dictionary/english/e-fulfilment

Valencia, C. E., Alfaro, C. A., Martinez, F. J. Z., Magana, M. C. V., & Perez, S. L. P. (2018). Outperforming several heuristics for the multidimensional assignment problem. *International Conference on Electrical Engineering*.

van Gils, T., Ramaekers, K., Caris, A., & de Koster, R. B. M. (2017, September). Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. *European Journal of Operational Research*.

van Gils Kris Braekers Katrien Ramaekers Benoît Depaire, T., & Caris, A. (2016, September). Improving order picking efficiency by analyzing combinations of storage, batching, zoning and routing policies. *Computational logistics*.

Yu, M., & de Koster, R. B. (2008). The impact of order batching and picking area zoning on order picking system performance. *European Journal of Operational Research*, *198*, 480-490.

Zhang, J., Zhang, Y., & Zhang, X. (1999). The study of joint order batching and picker routing problem with food and nonfood category constraint in online-to-offline grocery store. *International Transactions in Operational Research*, *28*, 2440-2463.