Meta-heuristic improvement of order picking efficiency

Exploiting cross-order similarity in a goods-to-person warehouse

T. H. R. Biemans

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Meta-heuristic improvement of order picking efficiency

Exploiting cross-order similarity in a goods-to-person warehouse

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Student:	T. H. R. Biemans
Supervisors:	prof.dr.ir. B. H. K. De Schutter
	ir. F. B. Gorte
Readers:	dr.ir. T. J. J. van den Boom
	dr.ir. J. Alonso-Mora
	ir. J. F. van der Maesen

March 31st, 2020

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by Picnic Technologies BV. Their cooperation is hereby gratefully acknowledged.

Abstract

This research aims at determining a suitable order release strategy for a warehouse featuring a shuttle-based storage and retrieval system with multiple goods-to-person picking workstations. These picking workstations facilitate efficient picking; however, current release strategies do not take similarity between orders into account, while this could potentially lead to better capacity utilisation. A meta-heuristic approach has been developed in literature to exploit this similarity, thereby using less stock retrievals to fulfil customer orders. However, its capabilities are limited. In this thesis, the meta-heuristic is extended to deal with multiple pick stations and a stock-multiplicity constraint. The algorithm consists of multiple steps. First, a random-start greedy algorithm generates an initial solution for a large set of orders. However, this initial solution will be of poor quality in general. As the computation cost scales exponentially with the order set size, heuristically optimising the entire order set is infeasible in practice. Secondly, the initial order sequence is split into subsets of equal length, each of which will be picked by a separate picking workstation. The picking workstations are then sequentially optimised using the meta-heuristic approach assuming an infinite stock. Afterwards, a check on the stock-multiplicity constraint is conducted, i.e. it is checked if the warehouse is capable of supplying all picking stations simultaneously. If an order line violates this constraint, the order is selected and put in a random position of the order completion sequence of that picking station. It was found that this method is capable of avoiding violations of the stock-multiplicity constraints, although it was also found that this constraint will hardly be violated in a realistic goods-to-person warehouse setting, i.e. with a large number of orders, picking stations, and order lines per order. Furthermore, our simulations show that the algorithm is capable of significantly reducing the number of stock retrievals per picking workstation.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

More and more shopping happens online instead of in physical shops. As a result, fulfilment centres now need to focus on picking individual orders instead of picking stock per store. Fulfilment centres can be roughly split into two categories: Picker-To-Goods (PTG) warehouses and Goods-To-Picker (GTP) warehouses. In PTG warehouses, a worker (picker) walks or drives through the storage aisles in order to fulfil orders. According to Kulak et al. (2012); De Koster et al. (2007), 50-70% of the total operating costs of a PTG warehouse are attributed to the labour cost of order picking. Travel time accounts for at least 50% of the picking tour (Tompkins et al., 2003). An alternative approach to PTG is the GTP approach.

In a GTP warehouse, the worker is stationary, i.e. the worker is located at a picking station and the required products are brought to the station by a system of conveyors or autonomous robots. According to Azadeh et al. (2019) 63 fulfilment centres featuring some form of robotisation have been built in the Netherlands between 2012 – 2016. It therefore makes sense to research automated fulfilment. An overview of this kind of warehouses will be given in Section 1-1. Warehouse control logic strongly influences the efficiency of automated logistics solutions. The challenges arising in the automated fulfilment operation are explained in Section 1-2. A problem statement will be drawn up, which will be elaborated upon in Section 1-3.

1-1 Automated fulfilment centres

The fulfilment operation can be split into inbound and outbound operations. In order fulfilment, a physical customer order is created from a list of requested products and the corresponding stock. Inbound operations break down bulk goods into stock totes, which are then stored in the storage system. Unit totes are crates of the same size. These are used by the Shuttle-Based Storage/Retrieval System (SBS/RS) to store the products as it is easier for the SBS/RS to handle unit totes than to handle products of different size. Outbound operations create orders by picking goods from stock totes to customer totes. Completed orders are stored in the storage area until they are received for shipping. Since this work specifically

focuses on the outbound operations in GTP warehouses, this section will elaborate on the picking station and the storage system.

Picking stations

The picking station can be located *end-of-aisle* or *remote*. *End-of-aisle* means that the picking station is located close to the input/output of the storage and the product totes are delivered to the picking station directly by the storage/retrieval machine. A disadvantage of this set-up is that, in general, only the products of the corresponding aisle can be picked by a picking station. On the other hand, a *remote* picking station means that conveyors or robots are needed to transport the product totes to the picking station. In most warehouses the picking station receives Stock Keeping Units (SKUs) in unit load stock totes from the storage system. A schematic overview of a remote order picking system is provided by Tappia et al. (2019) and shown in Figure 1-1. The storage consists of consist of multiple tiers, multiple columns, and multiple aisles of racks. The storage is considered to be single-deep, i.e. each storage location can fit one tote. At the picking stations, SKUs are transferred from their stock tote to a customer tote. If the stock tote has not been depleted, it will be returned to the storage system. A depleted stock tote will be replenished at a different location by inbound operations. Multiple customer totes can be filled simultaneously. This style of order picking is depicted in a schematic way in Figure 1-2a (Füßler and Boysen, 2019). Please note that at Picnic the customer tote is divided in three parts and thus not necessarily coincides with a single order. Other setups are possible as well. Füßler and Boysen (2017) research the need for scheduling in the sequence of arriving SKUs in an inverse order picking system. In such a system the picking station has a different layout. The customer totes are located along a conveyor, on which a stock tote is transported. A stock keeping unit is transferred from the storage to the customer tote and the stock tote moves to the next location. If a customer tote is fulfilled, it is replaced. This style is depicted in Figure 1-2b.

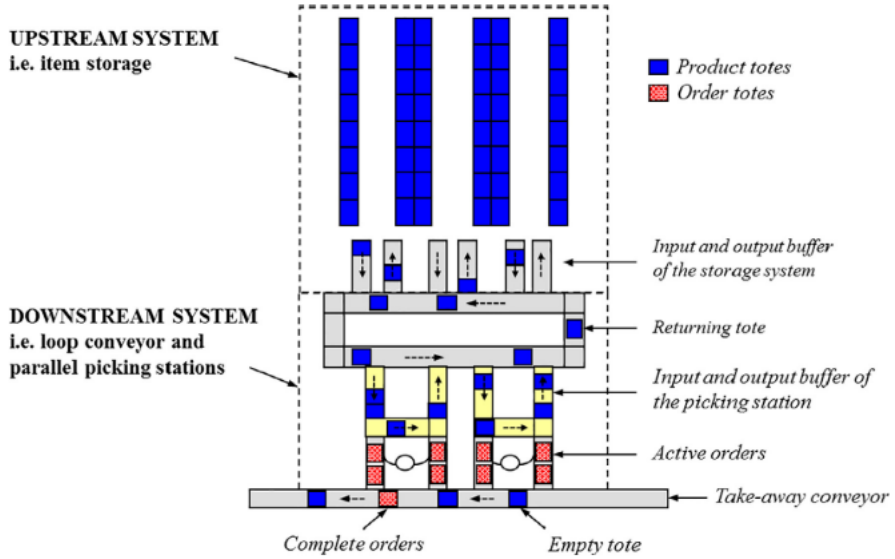
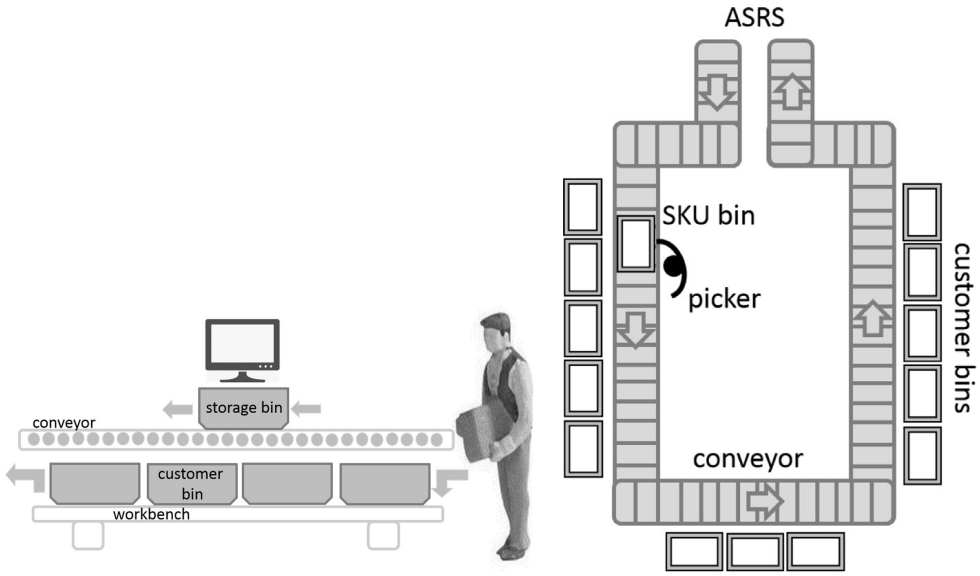


Figure 1-1: An overview of a remote order picking system (Tappia et al., 2019)



(a) Conventional order picking station (Füßler and Boysen, 2019)

(b) Inverse order picking station (Füßler and Boysen, 2017)

Figure 1-2: Two kinds of order picking

It is clear that the aforementioned picking styles differ slightly. The most important difference is that in inverse order picking a subsequent customer tote cannot receive the current stock keeping unit since it has moved on. In conventional order picking, a subsequent customer tote can receive the current SKU. It therefore provides more flexibility.

Storage/retrieval systems

In a remote order picking system, both an Automated Storage/Retrieval System (AS/RS) or a SBS/RS can be used to transport the product totes to the conveyor. Both systems will be elaborated upon in this sections.
In an AS/RS the product totes are stored using a crane that can move horizontally and vertically simultaneously. The crane can visit any column and any tier within its aisle. In an SBS/RS, the horizontal and vertical movements are split. Shuttles take care of the horizontal movements in a specific tier and lifts take care of the vertical movements per aisle. Once a shuttle has picked up a product tote, it will transfer it to a buffer point near the lift. As soon as the lift is available, it will pick up the product tote and transfer it to the conveyor system. An SBS/RS can achieve a higher throughput capacity than an AS/RS. A downside is that an SBS/RS has a higher investment cost per aisle (Tappia et al., 2015).

Boysen et al. (2017) research another possible setup that uses rack-moving mobile robots, also called a Kiva system. The robots are remotely controlled vehicles that move racks containing several types of SKUs. The fact that one robot carries several SKUs makes the planning of robots different from an AS/RS. For a thorough survey on the different possible set-ups the interested reader referred to (Roodbergen and Vis, 2009).

The current Picnic fulfilment centres are all PTG warehouses. However, to increase throughput and decrease labour costs, work has started on an automated fulfilment centre. Since the

company will most likely implement an SBS/RS with remote picking stations, this system will be considered from now on.

1-2 Challenges in automated fulfilment centres

Automated fulfilment is extremely complex. Therefore, several challenges arise, some of which are listed below.

1. Suppliers of warehouse material handling systems deliver to traditional e-commerce companies. In traditional e-commerce orders are typically of small size, consisting of few different order lines. As a result, cross-order similarity is ignored in the fulfilment operation. At Picnic, however, the average order consists of some 30 different order lines, making it interesting to research if cross-order similarity can be exploited.
2. Within the SBS/RS several problems can be optimised, e.g. shuttle dwell-point location, storage policy, etc. The problem is that all these factors are all interdependent, so tuning one process may result in a severe loss of performance in another process.
3. Even in an automated fulfilment centre, human pickers are used for the operation. Human picker performance varies over time as it is repetitive work, usually with poor ergonomics. As this is the most expensive process within the picking operation, it makes sense to let the human pickers be the bottleneck of the picking operation. This can be achieved in a few different ways: create large stock tote buffers at the picking station or make sure that the SBS/RS will not be the systems' bottleneck. The latter can be achieved by investing in extra aisles of SBS/RS technology, or by making sure that the SBS/RS works efficiently. Buffers at the picking stations are easy to implement but create different problems, e.g. a stock tote that is stuck in a buffer cannot be used at another picking station. It is also trivial that extra investments might not be an option.
4. One has to take into account that the actual operation will deviate from the planning. One can devise a (near-)optimal order release strategy, but if a crucial shuttle is broken, it needs maintenance; this can potentially block a proper execution of the planning. Therefore, some form of real-time control and physical redundancy is needed to steer the system back to the (near-)optimal strategy.

It is clear that several challenges arise in automated fulfilment centres and that it is hard to take every aspect of the operation into account. Lastly, many different set-ups or picking procedures are available, each featuring their own advantages and disadvantages. Luckily, these choices had already been made by Picnic. However, even at the Picnic automated fulfilment centre, fast-moving products will be picked in a different way than the slow-moving products. The choice was made to focus on the most commonly used GTP picking stations. Considering this, the next section will draw up a problem statement that will serve as a base for this research.

It can be seen that, for now, it is close to impossible to optimise an entire automated fulfilment centre. Trade-offs have to be made while designing and operating such a centre. As Picnic is interested in the performance within the picking stations, the scope was set on the picking

part of the operation. More specifically, how can picking be made more efficient, thereby decreasing the chance that the SBS/RS becomes the bottleneck.

1-3 Problem statement

In this section a problem definition will be drawn up. In the end, a fulfilment centre needs to make profit for it to stay active. As mentioned before, 50-70 % of the total warehouse costs are part of the picking costs. In conventional PTG warehouses a large amount of this is attributed to labour costs, more specifically the cost of travel time. This is not cost efficient, so it makes sense to replace human workers with some form of automation. However, a single AS/RS aisle costs around $\$630,000$ (Roodbergen and Vis, 2009) and an SBS/RS aisle will cost even more (Tappia et al., 2015). In order to get return on investment as soon as possible, the SBS/RS and human workers need to be used as efficiently as possible. To do so, it is important to make sure that the workers do not experience down time. Using the storage/retrieval machines as efficiently as possible means that all the customer totes are filled with as little storage/retrieval movements as possible, so the storage/retrieval system is less likely to be the bottleneck of the warehouse. This goal can be achieved indirectly by making use of cross-order similarity. If multiple customer totes demand the same SKU at the same time, only one stock tote is needed to fulfil the customer totes. This way, the number of stock totes retrieved is reduced while the number of customer totes that can be fulfilled is increased. The sequence of the customer totes to be fulfilled heavily influences the number of stock totes needed. Furthermore, current order release strategies do not take this into account, as it is uncommon in traditional e-commerce to have orders existing of more than a few different order lines. Considering these facts, the following problem statement can be drawn up:

Can cross-order commonality be exploited to minimise the number of storage/retrieval transactions, considering multiple picking stations, stock tote multiplicity, and a realistic order set in a Goods-To-Picker (GTP) warehouse?

From this problem statement, the contribution to the current literature of this MSc thesis can be derived. In this report a strategy from current literature will be extended to deal with larger problem sizes, multiple picking stations and stock multiplicity. To the best of the author's knowledge, this combination has not been researched before. The structure of the rest of this MSc thesis report is as follows. In Chapter 2 the state-of-the-art of order picking efficiency improving strategies is elaborated. Chapter 3 will explain the proposed improvements that have been implemented in this research. Chapter 4 discusses the numerical experiments and its results. It gives an insight on the benchmark algorithms used for this cause. Lastly, in Chapter 5 we will try to draw conclusions and give recommendations for follow-up research in the area of this work.

Chapter 2

Strategies to improve order picking efficiency: State-of-the-art

In this chapter, the state-of-the-art on improving order picking efficiency is discussed. It is important to note that, to the best of the author's knowledge, little research has been conducted on the picking process within a picking station. Therefore, most of the algorithms and approaches that are discussed in Section 2-1 have been applied to PTG warehouses. However, the approaches used in PTG warehouses can be adapted to GTP warehouses. Section 2-1 therefore discusses literature in both the PTG and GTP case. Section 2-2 explains the exact way of solving the problem described in Section 1-3. A meta-heuristic way to solve the problem is elaborated in Section 2-3. This meta-heuristic has its limitations; these are addressed in Section 2-4. Lastly, conclusions are drawn in Section 2-5.

2-1 Algorithms for more efficient order picking

In this section, strategies and algorithms that have been used to make order picking more efficient are discussed. First it is necessary to understand the picking process, and the strategies that can be used to make picking more efficient. As mentioned, quite some research has been done on PTG warehouses. Therefore, the strategies to improve order picking efficiency for PTG warehouses are first explained. Afterwards the GTP case and some heuristic algorithms will be elaborated in this section

2-1-1 Efficient order picking in Picker-To-Goods (PTG) warehouses

In PTG warehouses pickers walk or drive through the aisles of the warehouse. Customer orders consist of SKUs, of which multiple are picked per picking tour. This can be done per order *(pick-by-order)* or for a combination of orders *(pick-by-batch)*. Pickers use a picking device, usually a cart, to carry the picked items. The capacity of the picking device cannot be exceeded. Usually, all items of a specific order have to be collected in a single tour, i.e.

customer orders cannot be split as this means an extra sorting step. This is different when a zoning strategy is applied, as will be explained later in this section. A nice overview of the supply chain within warehouses is provided by Zhang et al. (2016).

The total processing time of a picking tour consists of (Henn, 2015; Henn and Schmid, 2013):

1. travel time, i.e. the time spent on travelling between pick locations, from the depot to the first pick or from the last pick back to the depot;
2. search time, i.e. the time spent on locating the SKUs to pick;
3. pick time, i.e. the time spent on picking the SKUs, i.e. moving the SKUs from the racks to the picking cart;
4. setup time, i.e. the time spent on administrative tasks per tour.

Bartholdi III and Hackman (2017) stated: "Travel time is waste. It costs labour hours but does not add value." However, the travel time accounts for at least 50% of the total processing time of a picking tour (Tompkins et al., 2003). In order to minimise warehouse operation costs, it is makes sense to minimise travel time. In PTG there are four strategies to minimise travel time: picker routing, zoning, storage assignment, and order batching. These strategies will be elaborated in the following paragraphs.

Routing

The first strategy is picker routing, where the travel distance is minimised directly. Assuming a constant travelling speed, this is equivalent to minimising travelling time. Most of the developed routing strategies are sub optimal, yet commonly applied. Ratliff and Rosenthal (1983) developed a solvable optimal routing strategy by modifying the travelling salesman problem. A problem with 50 aisles takes around a minute to solve. However, it is often not implemented since optimal routes are not intuitive and lead to confusion among the pickers (De Koster et al., 1999; Koch and Wäscher, 2016). Furthermore, the most common routing strategies reduce the in-aisle congestion as opposed to an optimal routing strategy (De Koster et al., 2007). As picker routing is irrelevant in a GTP warehouse, the routing strategies will not be further investigated.

Zoning

Under the zoning strategy, the warehouse is divided into several disjoint zones. A picker is assigned to one zone and will only pick SKUs in his own zone. There exist two types of zoning. The first type is called progressive zoning, where the picker stays within his zone, but the pick cart is transferred to the next zone. This differs from synchronised zoning, where the pickers from different zones pick in parallel and the orders are consolidated afterwards (De Koster et al., 2007). Zoning has received plenty of attention in literature, but the consolidation step is hardly researched (Boysen et al., 2018). Since zoning is out of scope for this research, the interested reader is referred to De Koster et al. (2007); Gu et al. (2007, 2010) for extensive reviews.

Storage assignment

The third strategy is storage assignment. In this strategy, SKUs are stored according to some property. Several storage assignment methods exist: *dedicated, random, closest open location, full-turnover-based,* and *class-based* (Roodbergen and Vis, 2009). All these strategies utilise the storage capacity differently, e.g. SKUs are stored according to their demand frequency, weight and size, or randomly. Storage assignment is an interesting approach for either PTG warehouses or when the performance of the SBS/RS is researched directly. For a research on the behaviour within a picking station, it is not of use. It will therefore not be further discussed.

Order batching

Order batching means that orders are grouped into batches and picked in a single tour to save on travel time. These methods are the ones most researched and reviewed (De Koster et al., 2007; Gu et al., 2007, 2010). Order batching is considered to be the most important strategy to save on travel times (Tsai et al., 2008). Batching creates the need for an order consolidation step after picking, as the pick cart has to be sorted into individual orders. Boysen et al. (2018) try to minimise the total consolidation time by sequencing partial orders using multiple heuristics. However, their research features an automated storage/retrieval system as an interface between the picking and the consolidation. They show that the release sequence of storage bins influences the efficiency of the consolidation process as soon as packing is the bottleneck of the warehouse. It is easy to see that the *pick-by-batch* strategy will generally need less travel time and is therefore more efficient than a *pick-by-order* strategy. Minimising travel time will minimise operational costs, as labour is expensive.

Batching can be done on-line and off-line. In on-line batching, orders are not known in advance and can become available over time. Batching is mostly done per time window, i.e. when the order is received. In off-line batching, all orders are known beforehand. This is also the case at the Picnic warehouse, since the 'shop' closes at 22:00 the evening before. Therefore, this research will mainly be focused on off-line batching and sequencing. Henn and Schmid (2013) batch and sequence orders such that the total travel distance as well as the total order delay time, or tardiness, is minimised. They use iterated local search and an attribute-based hill climber algorithm. Order delay time is the time interval between the fulfilment deadline the actual fulfilment of the order. Order delay time is minimised to ensure a timely delivery to the customers. Similarly, Henn (2015) uses meta-heuristics to solve the order batching and sequencing problem. Both variable neighbourhood descent and variable neighbourhood search are used to solve this problem. These algorithms are proposed as they can vary the neighbourhood structure during the optimisation as opposed to e.g. iterated local search. Henn and Schmid (2013) manage to improve their initial solutions by around 44% on average. However, their initial solution is of poor quality as the orders are only sequenced by due date. On the other hand, Henn (2015) manages to improve their initial solution by around 39% on average. As their initial solution is created by a constructive algorithm, it will be of higher quality than the initial solutions from Henn and Schmid (2013). Therefore, it can be concluded that the research by Henn (2015) yields superior results. Scholz et al. (2017) also take the batch assignment to pickers and picker routing into account in their algorithm. That is, the algorithm by Scholz et al. (2017) actively optimises the picker routes instead of

optimising the order batching and sequencing problem and calculating a route afterwards. They therefore achieve smaller order processing times than Henn (2015).

As order batching is so important in PTG warehouses, it has been researched extensively. However, most papers research more or less the same problem and just use different algorithms to solve the problem. Bozer and Kile (2008) use a Mixed-Integer Linear Programming (MILP) approach in order to solve a relatively simple order batching problem to optimality for small instances up to 25 orders. They use the MILP approach to find upper and lower bounds on the solutions to the order batching problem by heuristics. The authors state that the heuristics are reasonably good, but take quite some computation time for larger problems. It is good to keep in mind that this paper was written in 2008, and thus the performance of heuristics will have improved. Žulj et al. (2018) also use meta-heuristics and can solve the order batching problem up to 100 orders. The authors propose a hybrid algorithm and compare it to some of the methods used in literature before. Their algorithm is capable of outperforming all of the other algorithms. This paper gives a good overview of the capabilities of many algorithms that have been proposed for order batching. Henn and Wäscher (2012); Henn et al. (2010); Henn (2012); Koch and Wäscher (2016) all cover the order batching problem and integrate some form of routing into the problem.

There are papers that combine order batching, batch sequencing and picker routing into one problem. Tsai et al. (2008); Azadnia et al. (2013); Chen et al. (2015) all use a Genetic Algorithm (GA) to solve this problem. Tsai et al. (2008) used a nested GA to solve the order batching problem and routing problem. Azadnia et al. (2013) propose a hybrid between a data-mining algorithm and a GA. Their algorithm outperforms a simple GA by 68% on average. However, no analysis is shown on computation times, which is a weak point to this paper. Chen et al. (2015) proposed a hybrid between a GA and an ant colony optimisation. Their algorithm managed to improve the solution by 9% for some simple problems. A downside is that the problem instances used for the experiments were extremely small; the largest problem instance featured 8 orders.
Henn and Schmid (2013); Henn (2015); Scholz et al. (2017) all used different meta-heuristics. These papers are hard to compare, since they all use different constructive heuristics as their benchmark. Henn (2015) and Scholz et al. (2017) both propose a Variable Neighbourhood Descend, but Scholz et al. (2017) take multiple pickers into account. Therefore these papers are also hard to compare. Judging by the reviewed literature and the fact that the order batching and batch sequencing are $\mathcal{NP}-$hard problems, we can conclude that it is impossible to solve these problems to optimality for now, and that (meta-)heuristic approaches are the only way to produce near-optimal solutions (Matusiak et al., 2014; Henn, 2012).

2-1-2 Efficient order picking in Goods-To-Picker (GTP) warehouses

In a GTP configuration, the picker is stationary at a picking station, and the stock totes, i.e. crates containing stock SKUs, are brought to the picker. This can be done using e.g. a conveyor belt system or rack moving robots. Similarly as in a PTG setting where the pick cart has a certain capacity, the picking station has a certain capacity as well. In a GTP warehouse batching cannot be used to minimise travel time, but to synchronise different flows in the picking station. This is called sequencing and is used to minimise storage movements. This way, energy is saved and the machines use less of its capacity. An example of this is

given by Füßler and Boysen (2019). In this paper, orders and stock totes are sequenced in order to retrieve as little stock totes as possible. The authors also prove that the problem is $\mathcal{NP}-$hard and find out that it is thus only solvable to optimality for small instances of the problem. Since small instances are not realistic, they propose a three-stage heuristic to solve bigger instances. This meta-heuristic will be elaborated in Section 2-3.

A similar problem is covered by Füßler and Boysen (2017). Here inverse order picking is used, i.e. the stock totes pass along the customer totes on a conveyor. A critical note is provided by the authors themselves: inverse order picking is especially attractive if each SKU is demanded by multiple orders. That implies that inverse order picking will be less efficient for slow moving items or very large assortments. This problem is also proven to be $\mathcal{NP}-$hard and therefore a two-stage heuristic is proposed. The authors obtain multiple initial solutions, which are then improved using a sequencing algorithm. At Picnic Supermarkets the assortment is quite large, around 10.000 different products, so inverse order picking will probably not be useful.

Boysen et al. (2017) try to synchronise inbound and outbound flows in a picking station of a rack-moving mobile robot system, also known as a Kiva system. Once again, the synchronisation is carried out in order to prevent excess movements by the robots and ultimately to limit the fleet as this decreases investment costs. This time, a heuristic is used to solve two sub problems in an alternating fashion. Furthermore, Boysen et al. (2017) recognise the scheduling problems arising with multiple picking stations but leave these open for future research, as is the case with Füßler and Boysen (2017, 2019). These scheduling problems have thus not been researched yet for either the Kiva system or an SBS/RS for multiple picking stations. To the best of the author's knowledge, these papers are the only ones to consider the optimisation within the picking station.

2-1-3 Heuristic algorithms to improve order picking efficiency

Now, we can review the algorithms that have been used in literature. We only consider the most widely used algorithms: simulated annealing and a GA, both of which are considered to be suitable to solve scheduling problems. Please note that the simplest form is discussed here. Also, note that these algorithms are often embedded in a hybrid algorithm, as will be shown in the next sections. One can think of a greedy algorithm to create an initial solution that can be improved by simulated annealing.

Simulated annealing

In simulated annealing, a local search based algorithm, the cooling of metal is mimicked. During quick cooling of metal a meta stable equilibrium can be reached, which is similar to a local minimum. When cooling slowly, a nice crystalline structure is achieved, which is similar to a global minimum. Simulated annealing represents the slow cooling of metal, while the quick cooling is similar to a local search. Simulated annealing is capable of escaping local minimums by sometimes accepting a solution that increases the objective function value (Eglese, 1990). The acceptance of worse solutions is done using a probability factor: $\mathbb{P} = \exp(-\delta/T)$, where $\delta = f(x^{\text{neighbour}}) - f(x^{\text{incumbent}})$, $f(x^{\text{neighbour}})$ is the objective function value of the neighbour solution, and $f(x^{\text{incumbent}})$ is the objective function value of the incumbent solution. The temperature control parameter is called T. It can be seen that large values of

T create a higher probability of accepting a worse solution. The lower T becomes, the less likely that worse solutions are accepted. This results in the algorithm moving through the search space more randomly at high temperatures (Delahaye et al., 2019).

When using a simulated annealing algorithm, a few choices need to be made. First of all, it is important that the initial temperature and the temperature function are determined. Both have an effect on the acceptance of worse solutions. An often used temperature function is as follows: $T(t+1) = \alpha T(t)$, where α is a constant that is typically between 0.8 and 0.99. Secondly, it is important to define after how many iterations the temperature will be altered. Lastly, the termination criterion should be defined. Matusiak et al. (2014) combine the principle of simulated annealing with other algorithms to solve the order batching problem while taking precedence constraints into account. Simulated annealing is responsible for batching orders in this case. An extensive comparison with heuristics from literature is made. This comparison shows that the simulated annealing-based algorithm outperforms all other algorithms in terms of solution quality when the problem features more than 40 orders. The other algorithms in this comparison were the C&W(ii) savings algorithm, attribute-based hill climber, and variable neighbourhood search. Boysen et al. (2017) use simulated annealing to define near optimal order and rack sequences in a GTP warehouse using rack-moving robots. They use the simulated annealing approach on either the order sequence, or the rack sequence. They show that optimising the order sequence provides better solutions than optimising the rack sequence. The authors define the initial temperature by generating 100 initial solutions and neighbourhoods with a strictly worse objective function value. Afterwards, the authors set the initial temperature such that around 80% of the neighbourhoods get accepted.

Threshold acceptance is a variant of simulated annealing. Essentially, the only difference between threshold acceptance and simulated annealing is the way the acceptance criterion is defined (Gilli et al., 2011). Where the acceptance criterion is probabilistic in simulated annealing, it is deterministic in threshold accepting. The acceptance criterion is defined as a sequence of different values. Dueck and Scheuer (1990) introduced the method and conducted numerical experiments and found that threshold accepting produces very-near-optimal results in a fraction of the amount of CPU seconds compared to simulated annealing. Füßler and Boysen (2019) used threshold accepting in order to improve a given order sequence. After a predetermined number of iterations, the threshold is multiplied with a sink rate. This makes the search behave more and more like a local search with increasing iterations. Their threshold accepting-based algorithm outperforms a conventional MILP approach and can solve problems up to 100 orders. A weak point is that their proposed algorithm is not benchmarked to other meta-heuristics from literature, but only to simplifications of itself.

Genetic Algorithm

A GA is a population based meta-heuristic that was first introduced by Holland (1975). It is especially useful for scheduling problems, but can cover a wide variety of problems (Whitley, 2019). The GA mimics the evolution of a species and uses the fact that only the fittest individuals get to reproduce. In a GA, the solutions get encoded into (possibly binary) strings that store all the information needed. The following operators are of importance: selection, crossover, and mutation. The selection operator determines which parents are allowed to reproduce. This is mostly done using a roulette wheel selection where each individual gets

assigned a selection probability proportional to its fitness, i.e. individuals with a higher fitness have a higher chance of breeding (García-Martínez et al., 2018).
The other two operators are genetic operators and determine how the reproduction will happen. The crossover operator determines at what points the encoded string is cut and switched with a sub string of the other parent. A mutation can happen after the crossover procedure with a certain probability. It is common for the probability of mutation of one bit, p_m to be proportional to the length of the string: $p_m = 1/n$ (Whitley, 2019).

Tsai et al. (2008) used a multiple GA method for solving the batch picking problem. They constricted the search space to make sure that the algorithm would not become too computationally expensive. Tsai et al. (2008) used two GAs in a nested way; the first one creates the batches by minimising the travel cost, earliness and tardiness of all orders. The second GA takes care of the routing strategy by minimising the travel distance. The authors benchmarked their proposed nested algorithm against two variants; one variant that only minimises travel cost, and against one variant that just minimises earliness and tardiness. Their proposed algorithm outperformed both variants. This is, however, hardly a useful conclusion as both variants were simplifications of the proposed algorithm, so the fact that these simplifications produce solutions of worse quality seems trivial. The research by Tsai et al. (2008) did not benchmark against another algorithm than their simplifications, which is a weak point as it does not provide any reference to the state-of-the-art.
Koch and Wäscher (2016) developed a group-oriented GA and benchmarked it to a standard item-oriented GA developed by Hsu et al. (2005). Hsu et al. (2005) were already capable of cutting travel length with more than 50% for a 3D warehouse. The computation time for this problem, however, was 5.4 hours on average. The group-oriented GA by Koch and Wäscher (2016) created better solutions than the item-oriented GA. Koch and Wäscher (2016) state that the item-oriented already improved a solution from a C&W(ii) savings algorithm by 3.15% on average, while the group-oriented GA could improve the C&W(ii) solution by 3.75%. However, for larger problem instances, the improvements in comparison with the C&W(ii) solution are minimal.

Chen et al. (2015) created a hybrid coded GA in combination with an ant colony optimisation. Hybrid coded in this case means that the encoded string contains information on two different objectives; the first part of the string contains the batch sizing information, the second segment contains information on the batch sequencing. The fitness evaluation is done by ant colony optimisation, that calculates the total tardiness values of each individual solution that is given by the GA. These fitness values are then fed back to the GA, which determines if the termination conditions are satisfied. Chen et al. (2015) apply an elitist selection strategy, i.e. the best performing solution is automatically transferred to the next generation. They manage to obtain near-optimal solutions, but only on rather small problems with at most 8 orders.

2-2 Base MILP approach for GTP picking

To the best of the author's knowledge the only papers that have researched the GTP picking process within picking stations are Füßler and Boysen (2019), Boysen et al. (2017) and Füßler and Boysen (2017). The paper by Füßler and Boysen (2019) will be followed because the authors describe a way of picking that is similar to the way of picking that will be used by

Table 2-1: Notations

$\mathcal{T}$	Number of time slots ($t = 1, \ldots, \mathcal{T}$)
n	Number of picking orders ($i = 1, \ldots, n$)
S	Set of SKUs (and storage bins; $s \in S$)
o_i	Set of SKUs demanded by order i
C	Capacity of the workbench
$x_{s,t}$	Binary variables: 1, if SKU s is delivered in slot t; 0, otherwise
$z_{s,i,t}$	Binary variables: 1, if SKU s is delivered for order i in slot t; 0, otherwise
$y_{i,t}$	Continuous variables: 1, if order i is processed in slot t; 0, otherwise
δ_t	Continuous variables: 1, if the SKUs delivered in $t-1$ and t differ; 0, otherwise

Picnic in their automated fulfilment centre. Füßler and Boysen (2019) try to exploit cross-order similarity, thereby limiting the number of stock retrievals. Using the notation from Table 2-1 the authors defined a MILP problem is defined as

$$\text{minimize} \quad F = \sum_{t=2}^{\mathcal{T}} \delta_t \tag{2-1a}$$

subject to

$$\sum_{s \in S} x_{s,t} \leq 1 \qquad \text{for } t = 1, \ldots \mathcal{T}, \tag{2-1b}$$

$$\sum_{i=1}^{n} y_{i,t} \leq C \qquad \text{for } t = 1, \ldots \mathcal{T}, \tag{2-1c}$$

$$y_{i,t} + y_{i,t'} \leq 1 + y_{i,t''} \qquad \text{for } i = 1, \ldots, n; \quad 1 \leq t < t'' < t' \leq \mathcal{T}, \tag{2-1d}$$

$$\sum_{t=1}^{\mathcal{T}} z_{s,i,t} \geq 1 \qquad \text{for } i = 1, \ldots, n; \quad s \in o_i, \tag{2-1e}$$

$$2z_{s,i,t} \leq y_{i,t} + x_{s,t} \qquad \text{for } i = 1, \ldots, n; \quad s \in o_i; \quad t = 1, \ldots, \mathcal{T}, \tag{2-1f}$$

$$\delta_t \geq x_{s,t} - x_{s,t-1} \qquad \text{for } t = 2, \ldots, \mathcal{T}; \quad s \in S, \tag{2-1g}$$

$$x_{s,t} \in \{0, 1\} \qquad \text{for } t = 1, \ldots, \mathcal{T}; \quad s \in S, \tag{2-1h}$$

$$z_{s,i,t} \in \{0, 1\} \qquad \text{for } i = 1, \ldots, n; \quad s \in o_i; \quad t = 1, \ldots, \mathcal{T}, \tag{2-1i}$$

$$0 \leq y_{i,t} \leq 1 \qquad \text{for } i = 1, \ldots, n; \quad t = 1, \ldots, \mathcal{T}, \tag{2-1j}$$

$$\delta_t \geq 0 \qquad \text{for } t = 2, \ldots, \mathcal{T}. \tag{2-1k}$$

Here, Equation (2-1a) is the objective function to be minimised. It minimises the amount of SKU changes, so $F + 1$ equals the number of stock retrievals. Equation (2-1b) ensures that only one SKU is processed per time slot, while Equation (2-1c) makes sure that at most C orders are processed concurrently. Equation (2-1d) states that an order will be processed in consecutive time slots, i.e. it cannot leave the picking station. Equation (2-1e) says that every SKU required by order i should be delivered. For this to be possible, both order i and SKU s should be present in the picking station, which is ensured by Equation (2-1f). Equation (2-1g) tracks the number of SKU changes and Equations (2-1h)-(2-1k) define the domains of the variables. An upper bound on the number of slots is set as $\mathcal{T} = \sum_{i=1}^{n} |o_i|$. This upper

bound corresponds with the situation when every SKU is pick separately for every order. This is also considered to be the worst case scenario in this research.

Please note that the above MILP only minimises the number of SKU changes. It does not minimise the number of time slots that orders or SKUs are present in the picking stations. For this to be achieved, the objective function has to be extended to

$$F = \sum_{t=2}^{\mathcal{T}} \delta_t t + \sum_{t=1}^{\mathcal{T}} \sum_{i=1}^{n} y_{i,t} + \sum_{t=1}^{\mathcal{T}} \sum_{s \in S} x_{s,t}. \tag{2-2}$$

In Equation (2-2) the number of time slots is minimised by adding the two double sums. These double sums make sure that any order or SKU occupies the picking station for the least possible number of time slots. Since the problem uses these time slots, it is a discrete optimisation problem. Furthermore, the variable δ_t is forced to take either the value 0 or value 1. Therefore, multiplying δ_t with t has an integrating effect. As each SKU change δ_t is now multiplied with the time slot, the solver now forces the minimisation problem to use the earliest time slots possible.

2-3 The base meta-heuristic approach for efficient order picking

As solving a MILP problem will take too much computation time in practice, Füßler and Boysen (2019) developed a meta-heuristic method. This meta-heuristic approach consists of three stages: a greedy algorithm to create an initial solution, a heuristic improvement stare, and a small MILP problem in order to eliminate as many SKUs as possible. The different stages will be elaborated in this section.

2-3-1 A greedy algorithm to create an initial solution

An initial feasible solution is needed before heuristic improvement can take place. This initial solution is created using a greedy algorithm. This algorithm will create 3 different sequences: an order sequence, a completion sequence, and a SKU sequence. First, the order sequence is derived; it defines in what succession the free spaces in the picking station are filled with orders. A random order is chosen to be the first order of the order sequence. Let this order be called order i; its set of SKUs is therefore called o_i. A degree of similarity with every other order j is calculated according to

$$S(o_i, o_j) = \frac{2|o_i \cap o_j|}{|o_i| + |o_j|}. \tag{2-3}$$

The next order in the order sequence is then chosen according to a probability that is proportional to this similarity degree. The newly chosen order that is now last in the order sequence is then set to be the new order i and the process is repeated until all orders are placed in the order sequence.

Secondly, the completion sequence can be defined. This sequence determines in what succession the orders are completed and thus removed from the picking station. It is clear that

the completion sequence is strongly dependent on the order sequence: an order that is not available at the picking station simply cannot be completed. In order to define the completion sequence, the first C orders from the order sequence are considered, as these are the first to be available at the picking station. The first order to be completed is randomly chosen from these C orders. Afterwards, the next orders are randomly drawn from the first C, not yet completed, orders of the order sequence. To keep orders from blocking a spot in the picking station, an order is forced into the completion sequence if it has not been chosen for $2C$ times.

Lastly, the SKU sequence is created. The SKU sequence is created by considering the current order in the completion sequence. The SKUs required by that order need to be retrieved first. Then we consider the order that takes the place of the current order in the completion sequence after it has been fulfilled is determined, i.e. the subsequent order. The overlapping SKUs are determined, one of which will be the last SKU to retrieve for the current order. This ensures that the subsequent order can still benefit from that SKU and it will save a stock retrieval. All SKU positions to fulfil that have not been predetermined to be the last SKU for the current order, or the last SKU from a preceding order, are fulfilled randomly.

It can be seen that although a greedy similarity measure is applied, there is quite a lot of randomness present in this algorithm. It is therefore highly unlikely that the initial solution will be of good quality. Therefore it will be improved in the following stages.

2-3-2 Heuristic improvement

To heuristically improve the initial solution, threshold accepting is applied. Threshold accepting is similar to the simulated annealing algorithm. Threshold accepting uses swap moves to define solutions in the neighbourhood of the incumbent solution. First, a random order is selected from the completion sequence. This order can then be swapped with one of the orders that is simultaneously active at the picking station. The definition of the set of orders that can be swapped with the order on the i-th position of the completion sequence is defined as

$$O_i^s = \{\pi_j^o | j = 1, \ldots, \min(n; i + C - 1)\} \setminus \{\pi_j^c | j = 1, \ldots, i\}. \tag{2-4}$$

In Equation (2-4), π_i^o is the order at position i of the order sequence and π_i^c is the order at position i of the completion sequence.

The threshold accepting procedure is then relatively straightforward. First, Equation (2-4) is used to apply a swap move to the incumbent solution. It's objective function value is then compared to the objective function value of the incumbent solution. If the difference is smaller than the current threshold, the neighbour solution is accepted as the new incumbent solution. When a new incumbent solution is found, the threshold is multiplied with a sink rate. This means that the threshold will become lower over time, thereby reducing the chance that a deteriorating move will be accepted. In the paper by Füßler and Boysen (2019), a sink rate sr = 0.999 was used. Preliminary tests showed that this sink rate made the algorithm exceptionally slow. Therefore it was decided to adjust the sink rate to sr = 0.99. The other parameters were chosen in accordance with Füßler and Boysen (2019): threshold $\tau = 10$ and stopping threshold $\tau_s = 0.001$. The pseudo code for threshold accepting is given in Algorithm 1.

Algorithm 1 Threshold Acceptance

Input: set n_{steps}, threshold τ, sink rate sr, compute current solution x^c (randomly)
  while $\tau > \tau_{\text{stop}}$ **do**
    for $i = 1 : n_{\text{steps}}$ **do**
      generate $x^n \in \mathcal{N}(x^c)$;
      $\Delta = f(x^n) - f(x^c)$;
      if $\Delta < \tau$ **then**
        $x^c := x^n$
      else
        $i = i + 1$
      end if
      $\tau := \tau \cdot \text{sr}$
    end for
  end while
  $x^{\text{solution}} = x^c$
Output: x^{solution}

2-3-3 Elimination of SKUs by MILP

Stage three of the meta-heuristic algorithm solves an interval scheduling problem. Let π^s be the SKU sequence that is returned by stage 2 and π^s_i the SKU at the i-th position of the SKU sequence. Furthermore, T is the total number of stock totes in π^s, i.e. the objective function value. Using π^s, all possible fulfilment intervals are determined for each order j. An interval starts at a position i, where an SKU required by the order is active on the picking station. The interval ends when when order is fulfilled entirely, i.e. at the smallest possible $i' = i, \ldots, T$ for which $o_j \subseteq \bigcup_{k=i}^{i'} \pi^s_k$ holds. Then, let Γ_j be the set of all possible intervals for order j. Now two binary variable can be preprocessed. First we process a_{jmt}, a variable that states if an interval m is active for order j at SKU sequence position t. We set $a_{jmt} = 1$ if interval m is indeed active, or $a_{jmt} = 0$ otherwise. Secondly, we process b_{jmt}, that states if an interval m for order j ends at SKU sequence position t. If the interval m indeed ends at position t, we set $b_{jmt} = 1$. Otherwise we set $b_{jmt} = 0$.

Now, the optimisation variables are declared as

$$x_{jm} = \begin{cases} 1, & \text{if interval } m \text{ is selected for order } j, \\ 0, & \text{otherwise,} \end{cases} \tag{2-5}$$

$$y_t = \begin{cases} 1, & \text{if the SKU as position } t \text{ is required,} \\ 0, & \text{otherwise.} \end{cases} \tag{2-6}$$

Then, the MILP problem is defined as

$$\text{minimize} \quad P = \sum_{t=1}^{T} y_t \tag{2-7a}$$

subject to

$$\sum_{m \in \Gamma_j} x_{jm} = 1 \qquad \forall j \in O, \tag{2-7b}$$

$$\sum_{j \in O} \sum_{m \in \Gamma_j} a_{jmt} \cdot x_{jm} - b_{jmt} \cdot x_{jm} \leq y_t \cdot C \quad \text{for } t = 1, \ldots, T, \tag{2-7c}$$

$$\sum_{j \in O} \sum_{m \in \Gamma_j} b_{jmt} x_{jm} \leq y_t \cdot M \quad \text{for } t = 1, \ldots, T, \tag{2-7d}$$

$$x_{jm} \in \{0, 1\} \quad \forall j \in O; \quad m \in \Gamma_j, \tag{2-7e}$$

$$y_t \in \{0, 1\} \quad \text{for } t = 1, \ldots, T. \tag{2-7f}$$

In Equation (2-7) O is the set of all orders and M is a large number. It uses the objective function in Equation (2-7a) to minimise the number of required SKUs. Equation (2-7b) ensures that exactly one processing interval is chosen per order. Equation (2-7c) states that at most C orders can be simultaneously active at the picking station for a requires SKU at sequence position t. It is important to ignore the orders that end at sequence position t as they free up space on the picking station. Equation (2-7d) makes sure that an interval can only end at a SKU position where the SKU is required. Lastly, Equation (2-7e) and (2-7f) set the binary domains of the optimisation variables.

2-4 Remarks on the meta-heuristic approach

In initial testing, the meta-heuristic approach showed promising results. It is capable of handling larger order sets in less computation time in comparison with the MILP approach. However, some remarks have to be made.

First, some unwanted behaviour was found. Equation (2-7c) is used to determine the maximal number of orders that can be simultaneously active on the picking station. An important part of this algorithm is that order intervals that end on SKU sequence position t are not counted in this capacity constraint, since these will be replaced by a subsequent order that can still benefit from the current SKU at the picking station. During testing it was found that this results in the algorithm planning single-item orders on intervals that are infeasible in essence. This behaviour is shown in Figure 2-1. It shows the chosen fulfilment intervals per order. From this graph we can read that, e.g. order 0 is active at the picking station between SKU sequence positions $0 - 9$. In this instance, the meta-heuristic was solving a small order set, consisting of 30 orders of between 1-10 items per order. The capacity of the picking station was set to 3 orders. It can be seen that order 27 is placed at a spot where no capacity is left; orders 0, 2, and 16 already are already active on the picking station leaving no capacity for order 27. Since it only consists of one item, the order interval ends in the same sequence position as it starts. Its presence at the picking station is therefore neglected by Equation (2-7c). Picnic uses a minimal order value of €25, so single-item orders will not occur. Taking this

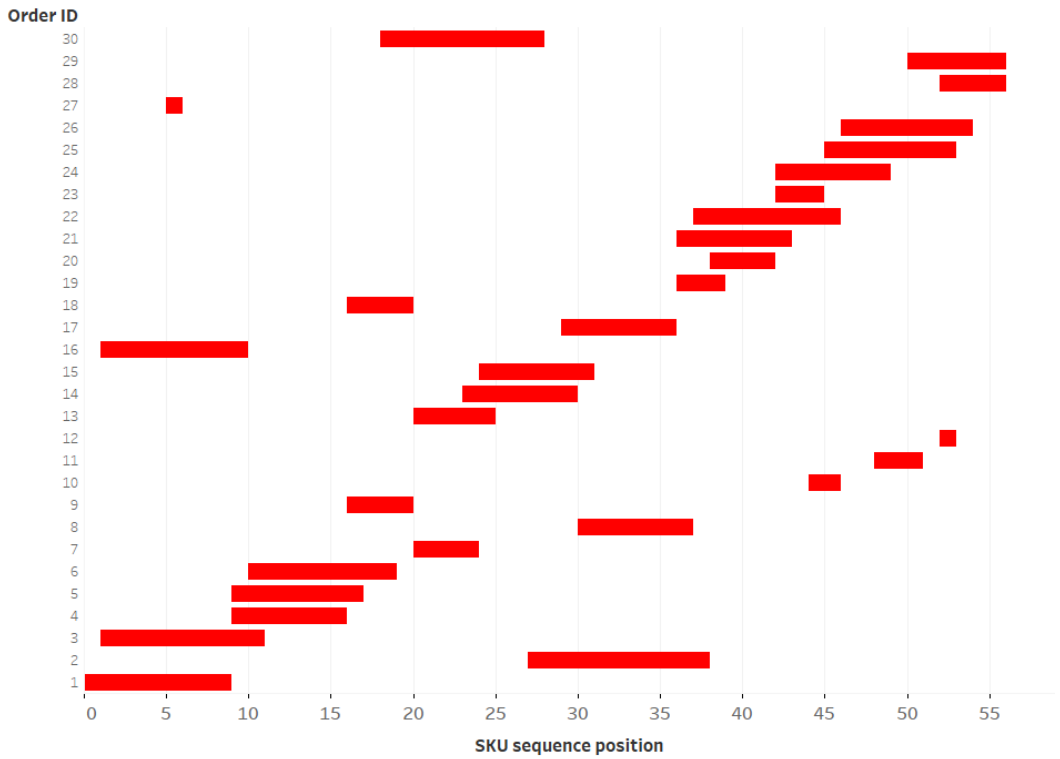


Figure 2-1: Unwanted behaviour on a small and simple order set

into consideration, no effort was made to fix this error. In traditional e-commerce however, this should be fixed as single-item orders are much more common there.

Secondly, the benchmark conducted by Füßler and Boysen (2019) only uses the separate stages and the MILP approach to benchmark their proposed algorithm. The author considers this to be insufficient to be called a benchmark as it does not compare with algorithms for order sequencing from literature. However, it does suffice as a proof of concept. A different approach was chosen for this research. It will be elaborated in Chapter 4

Thirdly, a remark has to be made on the order sets used by Füßler and Boysen (2019). The variables shown in Table 2-2 were combined in a full factorial manner, i.e. all possible combinations of parameters are considered. It can be seen that at most 50 different SKUs were used to create the orders and that each order contained at most 10 SKUs. Furthermore, a set of 100 orders is small in reality. Once again, this paper was used as a proof of concept. An order of at most 10 SKUs seems reasonable in a conventional e-commerce fulfilment centre. However, at Picnic orders generally consist of around 30-40 different SKUs. Also, using only 50 different SKUs is not a realistic amount for any e-commerce fulfilment centre.

Lastly, the meta-heuristic cannot cope with multiple picking station. This is a severe limitation as it does not represent the real GTP order picking operation. As multiple picking stations have not yet received attention in literature, it is considered a research gap.

Table 2-2: Parameters of the numerical experiments of Füßler and Boysen (2019)

Description	Values
Number of picking orders	$10, 30, 50, 70, 100$
Number of SKUs	$10, 30, 50$
Interval of SKUs per order	$[1; 3], [7; 10], [1; 10]$

2-5 Conclusions

In this section, conclusions on the state-of-the art will be drawn. It is clear that the process within the picking stations of an GTP warehouse has hardly had any attention in literature so far. The problem described in Chapter 1 is proven to be $\mathcal{NP}-$hard, as are the order batching problem and its' variants. A consequence is that one cannot use a commercial solver to solve the MILP formulation to generate a feasible solution in a reasonable computation time. The use of meta-heuristic methods for optimising the order picking operation is currently the most suitable way to address this limitation.

The most used strategy to make order picking more efficient in the PTG case is called order batching. In order batching, several orders are picked in one picking round. This will save on travel time as the picker does not have to walk to and from the consolidation area for each order. However, if several orders with a high similarity are to be combined the saving can become even higher; It is easy to see that less distance needs to be travelled in order to pick all products, since a high fraction of the products overlap. In GTP warehouses, the picker is stationary, so no travel time can be save. However, the amount of stock retrievals, i.e. the number of machine movements, can be minimised. This is called sequencing and it can have a severe impact on the utilisation rate of the SBS/RS. In this chapter, these approaches have been reviewed and the most used heuristic algorithms have been elaborated. Simulated annealing, threshold acceptance and GAs all seem to be suitable for (meta-)heuristic optimisation methods for order sequencing.

Please note that PTG warehouses have been researched extensively, but these papers mostly consider the operation within the SBS/RS. To the best of the author's knowledge, only three papers have been published on the picking process in picking stations in a GTP warehouse. One of these papers considers the picking strategy that will be implemented by Picnic. The paper, by Füßler and Boysen (2019) tries to find a near optimal stock tote sequence using a meta-heuristic approach. The meta-heuristic uses a Greedy algorithm to generate an initial solution, which is then heuristically improved by threshold accepting. Lastly, a small MILP is used to eliminate as many SKUs as possible. The meta-heuristic approach shows promising results. However, as already mentioned in Section 2-4, it does have its' own limitations; one constraint was not entirely thought out, the benchmark process only suffices as a proof of concept, and the problem size and order size are not realistic. In the end, the paper suffices as a basis for this research. However, it does need to be extended. This will be explained in Chapter 3.

Chapter 3

Improvements to the base meta-heuristic

In Section 2-4 we learned that the base heuristic by Füßler and Boysen (2019) poses limitations that prevent it from being implemented in the Picnic GTP picking operation. In this chapter, several improvements are proposed to extend the heuristic to more realistic problems. The current limitation of the unrealistic order sets is addressed in Section 3-1. Afterwards, the extension to multiple picking stations is elaborated in Section 3-2. In Section 3-3, the stock multiplicity constraints considered. Lastly, conclusions are drawn in Section 3-4.

3-1 Optimising realistic order sets

As mentioned previously, Füßler and Boysen (2019) used the variables of Table 2-2 in their numerical experiments in a full factorial manner. As a result the most complex problem the authors researched featured 100 orders, each of which consisted of between 1 and 10 items that were chosen from 50 different SKUs. This problem size does not represent the problem sizes encountered in typical e-commerce operations. At Picnic each warehouse produces between 2000 and 3000 orders per day and this is expected to grow by a factor of 10 with the introduction of GTP picking. Furthermore, Picnic now has an assortment of over 10.000 different SKUs and orders consist of 30-40 different SKUs on average.

In this research the algorithm by Füßler and Boysen (2019) was adapted to deal with realistic warehousing problems. The first step is to use more realistic order sets as the inputs. In order to achieve this, a realistic order data-set was supplied by Picnic. Picnic uses three temperature zones in their warehouses and some products will not be picked using the GTP picking stations. The temperature zones are called *frozen, chilled,* and *ambient*. As will be shown in this section, the realistic data set was filtered on the products that are picked using the GTP picking stations in the ambient temperature zone. This resulted in a data set of 4490 different SKUs.

In this research the large Picnic data set is filtered on the *ambient* temperature zone and on the products that are actually picked using the GTP picking stations. For some very fast-moving products, it makes no sense to decant them into stock totes and store them in the warehouse. These products are picked directly from the pallets in different and separate picking stations. As these picking stations make use of a different picking process, they are ignored in this research. Devising a strategy that combines both types of picking stations could be an interesting opportunity for follow-up research.

As fewer SKUs are available for orders to consist of, the computational complexity of the problem will decrease. This is a realistic assumption since *chilled* and *frozen* SKUs are picked into different, insulated, totes. These SKUs are therefore picked in different picking stations. No adjustments need to be made to the algorithm for it to optimise *chilled* or *frozen* picking. Supplying the data set of the appropriate temperature zone suffices. It is interesting to point out that, since less SKUs are available for optimisation, the similarity of the orders will be higher than when the warehouse is optimised as a whole. This is expected to be beneficial for the performance of the algorithm.
Lastly, the data set is grouped on the *Tote ID*, as an order can be larger than one tote and the capacity of the picking station is measured in totes. In the remainder the word *order* is used instead of *order tote*.

We can now define what a realistic problem will be for the meta-heuristic to optimise. A realistic problem will be defined as a problem that features around a days worth of orders, each of which can consist of around 30-40 SKUs, chosen from these 4490 SKUs. The orders are then filtered on GTP picking and on the *ambient* temperature zone. For now, a days worth of orders coincides with the number of orders the current PTG warehouses produce, which is around 2000-3000 orders each day.

Determine the order set size

This section will demonstrate the obtaining of a favourable order set size that is to be optimised. To determine the order set size the meta-heuristic was run 20 times for each order set size and the results were analysed. Specifically, the percentage of improvement from the worst case scenario and the computation time were plotted in order to find an order set size with a favourable trade-off between the two. Recall from Section 2-3-3 that the worst-case scenario is defined as the scenario where every order line is picked separately for every order. This is defined as

$$\text{worst case} = \sum_{i=1}^{n} |o_i|. \tag{3-1}$$

In Equation (3-1) n is the number of orders, and o_i is the set of SKUs required by order i. The percentage of improvement is defined as the number over stock retrievals saved in comparison with the worst case scenario, which is defined as

$$\text{improvement} = \frac{\text{worst case} - \text{objective}}{\text{worst case}} \cdot 100. \tag{3-2}$$

It was expected that the algorithm would find a larger improvement for larger order sets due to the fact additional orders will largely consist of already used SKUs. Therefore, more

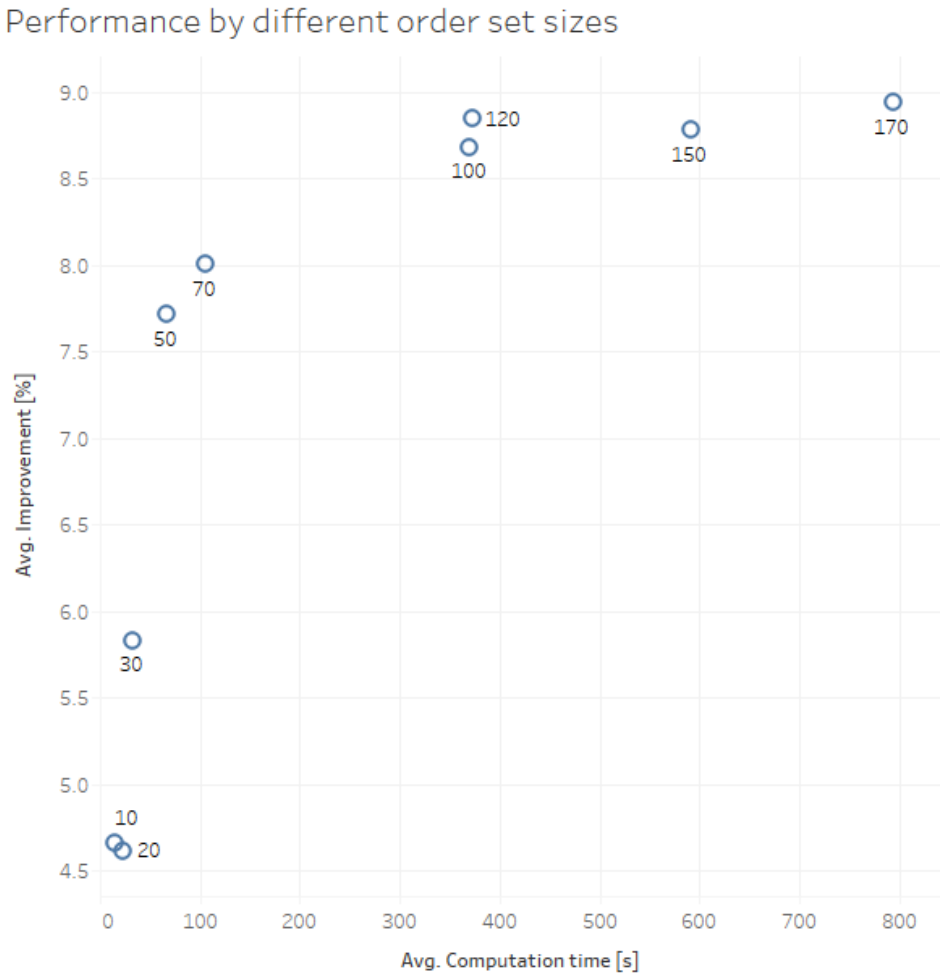


Figure 3-1: The average percentage of improvement from worst-case vs. the average computation time on one pick station

options with a high similarity degree are available. Also, it was expected that the increase of improvement would become smaller as the order sets grows, as the number of extra orders becomes relatively smaller. It is evident that a larger order set will take more computation time. In this section, the relationship between improvement and computation time is investigated. As shown in Figure 3-1 the improvement, as defined in Equation (3-2), increases with an increasing order set size. For larger order set sizes the extra improvement obtained is marginal. The computation time however, increases exponentially with the order set size. Using these results an order set size of 70 orders was chosen as it provides a significant improvement from the worst-case scenario, while needing only around 100 seconds of computation time on average. It can also be seen that the algorithm takes more than three times as long for an order set of 100 orders than a set of 70 orders, while the average extra improvement is slightly less than 10 % extra. The average improvement for 120 orders is sightly more than 10% extra in comparison with 70 orders, however this still needs more than three times as much computation time. Note that if a similar experiment is conducted on a different machine or on parallel machines, Figure 3-1 can look rather different. It is therefore recommended to re

do this simple analysis when implementing the meta-heuristic on another machine.

Assumptions

In modelling automated warehouses with GTP picking stations the following assumptions are made. The GTP warehouses are complex, and some assumptions are needed to be able to model the behaviour. The assumptions made in this research are now elaborated.

1. Each stock tote contains only one kind of SKU. A stock tote that contains multiple different SKUs would limit the capacity of the warehouse, and increase the complexity of the problem. It is easy to imagine the scheduling problems that could arise in such a situation. Let SKU A and SKU B be stored in the same stock tote. If picking station *1* requires SKU A, SKU B is cannot be required by a different picking station simultaneously. This assumption is valid, as stock totes will only carry one SKU in the Picnic fulfilment operation.

2. It is assumed that each tote carries enough stock to fulfil all orders that are currently active on the picking station. In reality, this might not be true since a stock tote will only be replenished when it is depleted. When a stock tote is stored in one location only, this assumption could potentially be a problem. However, if a stock tote is stored at multiple locations, the system should know the number of SKUs stored in each stock tote. It can then choose to send a tote that can indeed fulfil the orders at the picking station. Another possibility is that, next to returning an order release strategy, the algorithm determines a stock tote depletion strategy. It can then send a stock tote that will be completely depleted, such that it can be replenished immediately after fulfilling the orders. However, as we assume sufficient stock per stock tote, this problem is out of scope for this research. It is therefore an interesting problem for follow-up research.

3. Before extending the algorithm to multiple pick stations, an assumption is made that there is always a crate available to fulfil the required SKU. This assumption is made since in general, an SKU is stored in multiple locations, and thus always one stock tote will be available for picking. When dealing with only one picking station, this assumption is valid, since at most one picking station can request an SKU simultaneously. However, when multiple picking stations are considered, multiple picking stations can request the same SKU simultaneously. If that SKU is stored on just one location in the warehouse, this will yield an infeasible solution. How to deal with this constraint is explained in Section 3-3.

4. If an order requires multiple order lines of one product, these are ignored. First of all, in Assumption 2 we just assumed that each stock tote carries enough stock to fulfil every order tote that is active on the picking station. Secondly, we take the number of stock retrievals as the performance metric and that number does not change when an order line is required multiple times by the same order. Therefore, by ignoring multiple order lines of the same SKU, we do not change the problem or the model.

5. When a stock tote has fulfilled orders at a picking station, it is sent back to the storage. Because the trip back to the storage takes time, the tote cannot be requested again

immediately. In this research, the optimisation problem is solved using SKU sequence positions. An SKU sequence position is defined as the interval in time where a specific SKU is active at the picking station. These time slots are of variable time length. It is assumed that a stock tote that is required in time slot t, cannot be required by another picking station in the same time slot. However, it can be required by another picking station in time slot $t+1$. A downside of this approach is that it is hard to say anything about the total processing time of an order. In the case of only one picking station, this assumption means that no time limits are used during optimisation and a specific stock tote can, in theory, be requested in two consecutive time slots.

6. Lastly, standardised customer totes and stock totes are assumed. This makes it possible to express the capacity of the picking station in a number of totes that can be filled simultaneously. Standardised totes have been used throughout the Picnic operations and will be used in their GTP warehouse as well, as it is easier for the SBS/RS to handle standardised totes.

It is clear that some assumptions had to be made in order to be able to solve the order sequencing problem. Now, the validity of these assumptions is discussed. Assumptions 1 and 6 are valid, as they mimic the picking operation in reality. Assumption 2 is not always realistic. In reality, stock totes are filled and only replenished once they have been completely depleted. In can be interesting to let the warehouse control system choose a stock tote that can be replenished directly after fulfilling the orders at a picking station. In other words, a depletion strategy can be developed. As this is a different field of research this assumption was made. Assumptions 3 is valid for the situation when only one picking station is considered. As the algorithm will be extended to multiple picking stations, this assumption simplifies the algorithm as it eliminates the stock multiplicity constraints from Section 3-3. As said, Assumption 4 is made since we assume a sufficient number of SKUs per stock tote in Assumption 2; it is therefore a valid assumption. Lastly, Assumption 5 is not so realistic. In a real situation, the travel time of a stock tote will vary, deadlocks can occur, lifts and shuttles can break down, and SKUs can come from different locations in the warehouse. Stock totes will thus not be available for request for varying amounts of time. These stochastic situations are not captured when using SKU sequence positions as the time variable. The SKU sequence positions greatly simplify the model of the picking station. The SKU sequence positions also provide an opportunity for future research, where the stochastic behaviour of the system can be incorporated in the problem.

3-2 Extending to multiple picking stations

As we have defined the assumptions in the previous section, we can now extend the algorithm to service multiple picking stations. In this section, a strategy is proposed to create this extension.

As mentioned in Section 2-3, the current heuristic developed by Füßler and Boysen (2019) consists of three stages; a greedy initial solution generator stage, a heuristic improvement stage, and an SKU elimination MILP problem. In Section 3-1, we determined a suitable number of realistic orders that can be optimised by the original meta-heuristic. It was found

that a set of around 70 realistic orders yields the most favourable trade-off between the percentage of improvement and the computation time. As the Picnic warehouse will feature around 30 picking stations, an order set of 2100 orders was created by randomly sampling the entire order data set. It is clear that this large order set should be split into 30 pieces of an equal number of orders, each one representing a picking station. At least two possible strategies for this division exist. The large order set can simply be split or it can be optimised first. As the large order set is created using random sampling, simply splitting it means that random orders end up at the same picking station; this can possibly diminish the performance of the meta-heuristic. Therefore, the large order set is first sorted on cross-order similarity by the Greedy algorithm that has been explained in Section 2-3-1.

After sorting the large order set on cross-order similarity, it can be split into manageable chunks per picking station. Due to the sorting process, orders that have a high similarity degree are likely to end up in the same picking station. Remember from Section 2-3-1 that the Greedy algorithm arranges orders by sampling based on a probability that is proportional to the similarity degree. That means that randomness is present in this greedy algorithm, as it can happen that an order that does not have the highest similarity degree is chosen as the next order in the order sequence.

Now that the orders have been sorted and split into chunks per picking station, an initial solution is determined per picking station by creating a completion sequence and an SKU sequence. Due to the randomness that is present within the Greedy algorithm, these chunks will in general not result in a solution of good quality. Therefore, stage 2 from Section 2-3-2 and stage 3 from Section 2-3-3 are used to improve the initial solution. This is done for each picking station sequentially while assuming infinite stock, i.e. no stock multiplicity constraints are taken into account. After each picking station has been optimised all generated SKU sequences are returned and the total objective function value is calculated by summing all SKU retrievals.

It is important to check if the assumption of infinite stock is valid. A simple analysis was conducted on this by running a global GA and letting it create solutions. These solutions were checked for feasibility by comparing the number of requested SKUs to the stock multiplicity for each SKU at each SKU sequence position. If an infeasible solution was found, it was stored, along with the multiplicity degree of the SKU that violated the constraint. It was found that a completely decoupled solution, as is described in this section, would create solutions that are infeasible in practice. Therefore, the check on the stock multiplicity constraints was added to the meta-heuristic algorithm, as will be elaborated in Section 3-3. It was also found that a large portion of the violations were caused by SKUs with just one storage location. Almost 99.5 % of the violations were caused by SKUs that are only stored in one place in the warehouse. This is shown in Figure 3-2. It is therefore recommended to make sure that every SKU is stored in at least two different storage locations.

In short, in this section a strategy to take multiple picking stations into account has been elaborated. First a large order set is sorted based on cross-order similarity, after which it is split into parts of an equal number of orders. Each of these chunks is regarded as a picking station. Then for each picking station, the heuristic improvement stage and the SKU elimination stage are used to improve the solution quality; these stages are iterated five times. In this section, it was also found that the assumption of infinite stock would cause infeasible solutions in reality. Therefore, a strategy on how to take stock multiplicity into account is

explained in the following section.

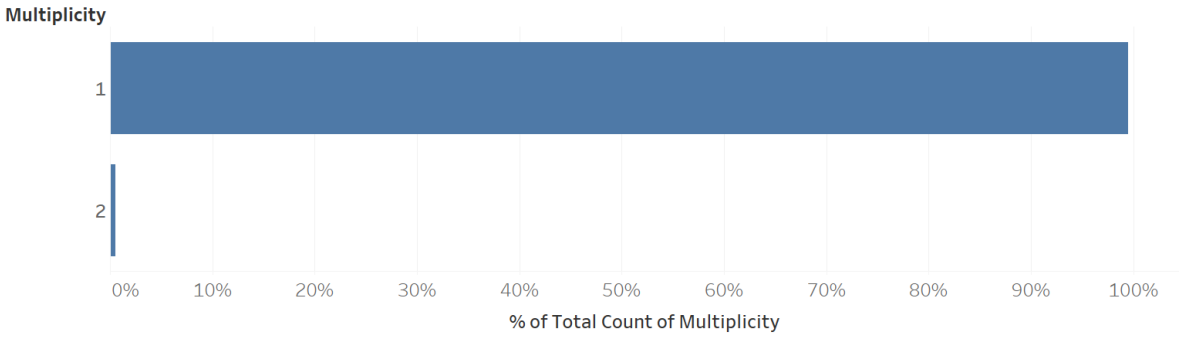


Figure 3-2: The count percentages per multiplicity degree for violations

3-3 Considering stock multiplicity

In the previous section, an extension to multiple picking stations was proposed. However, as infinite stock is assumed, this proposal is neither realistic, nor of any use in reality. It will plan critical SKUs on multiple picking stations simultaneously and thus create infeasible solutions. For the algorithm to be useful to Picnic, or any other e-commerce company, it is necessary to take stock multiplicity into account. In other words, if a certain SKU is stored on only one location in the warehouse, it cannot be requested by multiple picking stations simultaneously. To take this multiplicity into account, a database on storage locations was provided by Picnic. It featured a possible configuration of the storage locations in the future automated fulfilment centre. The proposed strategy from Section 3-2 is slightly adapted. After each optimisation of a picking station, a check is conducted over all currently known SKU sequences. If all these SKU sequences combined do not violate the multiplicity constraints, the overall solution is accepted and the next picking station will be optimised, i.e. the next SKU sequence is created. An example of this is given in Figure 3-3. The corresponding stock multiplicities are given in Table 3-1. In this figure, each row represents a picking represents a picking station, of which m are present. Each column represents a position in the SKU sequence. There are n positions in these sequences and it is assumed that every picking station features an SKU sequence of equal length, although this might not be true in reality. Each time after a new SKU sequence created, i.e. a row is added to the SKU sequence matrix, a multiplicity check is conducted. This check will count all SKUs per SKU sequence position, i.e. per column. As soon as a violation is found, in this case at picking station m and SKU sequence position 3, the order containing that specific SKU is placed at a random spot in the completion sequence of that picking station. Recall from Section 2-3-1 that a different completion sequence will yield a different SKU sequence. The pseudo code of this process is given in Algorithm 2. The following behaviour is expected from this strategy. It is easy to see that the (near-)optimal solution within the picking station is adjusted. This will probably result in a solution of lower quality in that picking station. However, a solution of lower quality is more useful than an infeasible solution. Another expectation is that this strategy will hardly cost any computation time. A different strategy could have been to simply redo the local heuristic optimisation when a violation is found. However, this would result in the heuristic improvement stage as well as

the SKU elimination stage restarting every time a violation is found. As this will result in a computationally extensive algorithm, this strategy is not used.

Algorithm 2 Pseudo code for the stock multiplicity check

Input: order sequence per picking station, stock multiplicity data, number of picking stations m, overall_solution = $\emptyset$
  for i = 1:m **do**
    create order completion sequence;
    create SKU sequence;
    append overall_solution with SKU sequence;
    n = max(length(overall_solution));
    for j = 1:n **do**
      count SKUs used in SKU sequence position j;
      if stock multiplicity is violated **then**
        find violating order and place at the end of order completion sequence
        create new SKU sequence
        break
      end if
    end for
  end for
Output: overall_solution

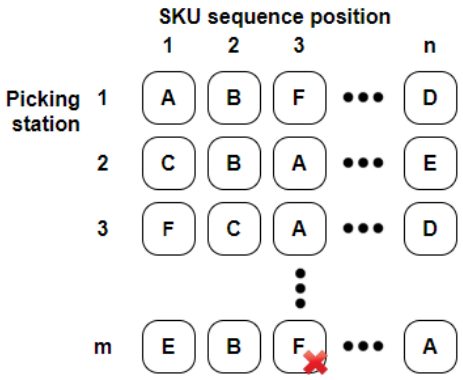


Figure 3-3: An example of a multiplicity constraint check

Table 3-1: The stock multiplicity for Figure 3-3

SKU	Multiplicity
A	2
B	3
C	3
D	3
E	1
F	1

3-4 Conclusions

In this chapter, several different improvements to the base line algorithm by Füßler and Boysen (2019) have been proposed. Firstly, the problem is made more realistic by increasing the number of SKUs and the number of orders, as opposed to the original algorithm. An analysis showed that an order set of 70 orders yields a good trade-off between the improvement

from the worst case scenario and the computation time. Secondly, the meta-heuristic was extended to deal with multiple picking stations. At first, an infinite stock was assumed, i.e. the picking stations were completely decoupled from each other and interdependencies were ignored. However, from analysis this was found to be unrealistic as infeasible solutions were created. For instance, an SKU with only one storage location could be requested by two picking stations simultaneously. Therefore, it was proposed to take the so called stock multiplicity constraints into account in order to ensure feasible and realistic solutions.

It is evident that the extension to multiple picking stations will have a large effect on the computation time. As stage 2 from Section 2-3-2 and stage 3 from Section 2-3-3 have to be repeated for every picking station, the computation time will increase severely, as both stages are computationally extensive. On the other hand, the improvement will probably not be affected, since all stages are still used to optimise each picking station. The check on the multiplicity constraints is expected to slightly lower the performance of the meta-heuristic. As explained, it will alter a local (near-)optimal solution to make it suitable for the global solution. In other words, the solution of one picking station is changed so the overall solution for all picking stations is feasible. The proposed strategy for this, takes the order that violates the stock multiplicity constraint and places it at a random spot in the completion sequence for that picking station. It will then not be improved heuristically anymore, so it takes only a small amount of computation time.

These proposed improvements of course have their effect on the performance of the algorithm. It is expected that the solution quality of the algorithm will be slightly worse due to the increased number of SKUs. It is easy to see that a larger set of SKUs to choose from has a lower cross-order similarity as an effect. Therefore the algorithm will probably find fewer opportunities to exploit the cross-order similarity. On the other hand, this is expected to be cancelled out by the fact that the meta-heuristic sorts a large order set based on cross-order similarity. This will result in more common orders ending up at the same picking station. This creates an opportunity to eliminate more SKUs, thereby causing a larger improvement in the overall solution. Furthermore, as a more realistic order set contains more orders, a longer computation time is expected.

Lastly, a takeaway from this chapter is that the number of multiplicity constraint violations can be reduced significantly by ensuring multiple storage locations for all SKUs. In the following chapter, the numerical experiments will be elaborated that are conducted using the proposed improvements from this chapter.

Chapter 4

Numerical experiments for the proposed meta-heuristic

In this chapter, the numerical experiments for the proposed meta-heuristic will be elaborated. First the set-up will be explained in Section 4-1. Afterwards, the benchmark algorithms are elaborated in Section 4-2. We will try to explain the results in Section 4-3, after which some conclusions will be drawn in Section 4-4.

4-1 Set-up

Picnic provided a large order data set consisting of realistic orders. From this large order set, 20 different smaller order sets were created using random sampling. Multiple data sets were used to eliminate the chance that one of the data sets featured some advantage for one of the algorithms. Each of these smaller order sets consists of 2100 orders, i.e. 30 picking stations that have to pick 70 orders. This number was chosen since the Picnic automated fulfilment centre will feature around 30 picking stations. The picking stations all pick 70 orders since it was found that an order set of 70 orders featured the most favourable trade-off between improvement and computation time. This has been described in Section 3-1.
As previously explained, the algorithm improves an initial solution in a heuristic way. The fact that a heuristic is used, may result in a different solution every run. Similarly, the other benchmark algorithms feature some form of randomness. For instance, the Greedy algorithm uses random sampling, as will be explained in Section 4-2-2. Therefore, every algorithm was run 5 times per data set. In order to say something about the spread of the solutions, the standard deviation is also considered in the experiments.

It is expected that the MILP approach cannot handle the problems consisting of 2100 orders. It was found by Füßler and Boysen (2019) that their MILP approach could not handle problems consisting of more than 30 orders. It has to be noted that their problems were of lower complexity than the ones in this research. This could mean that a MILP approach will

perform worse in this set-up. Therefore, different experiments were set up to test the performance of the proposed algorithm on a smaller scale and to test if the proposed algorithm can actually produce (near-)optimal solutions. Several data sets were created, again using random sampling. The data sets consisted of 10, 20, 50, and 70 orders. For each order set size 20 data sets were created. As mentioned, the hardware used throughout this research is a Dell laptop using a Intel(R) Core(TM) i5-8350U CPU with a base speed of 1.90 GHz and 16 GB RAM.

4-2 Benchmark algorithms

In order to be able to make a meaningful conclusion on the proposed algorithm, it is important that its performance is checked against the performance of different algorithms. The benchmark algorithms chosen in this research are a First-Come-First-Serve (FCFS) algorithm, a Greedy algorithm, and a GA. These will all be explained in this section.

4-2-1 First-Come-First-Serve (FCFS)

In order to draw meaningful conclusions on the ability of every algorithm to improve the solution quality, an FCFS algorithm is used as the simplest benchmark. As the name implies, the algorithm uses the first order it reads in the data set as the first order of the order sequence. In other words, the algorithm does not shuffle the order set base on some property of the orders. After the order sequence is created, it creates a completion sequence and a SKU sequence. While creating the SKU sequence, it will take the cross order similarity of the simultaneously active orders into account, i.e. it will use one stock tote to fulfil multiple orders at once. The resulting solution will therefore still be an improvement from the worst case scenario. Remember that the worst case scenario is defined as every SKU is picked separately for all orders. This simple FCFS algorithm will combine overlapping SKUs while fulfilling the orders and therefore improve the worst case scenario. It is still necessary to check for multiplicity violations. If the multiplicity constraint is violated, an entirely new SKU sequence is constructed for the picking station that causes the violation. Remember from Section 2-3 that SKUs that are not predetermined to be the first or last SKU of a particular order, are fulfilled in a random order. Creating a new SKU sequence can thus potentially solve the violation. The pseudo code for the FCFS algorithm is not shown, as it is similar to the pseudo code for a Greedy algorithm that is shown in Algorithm 3. The only difference is that the FCFS algorithm does not feature a `CreateOrderSequence` function, as it derives the order sequence directly from the data set.

4-2-2 Greedy algorithm

The second algorithm that was chosen to use as a benchmark is the Greedy algorithm. This algorithm is based on stage 1, as described in Section 2-3-1. It has been slightly adapted, which will be elaborated in this section. Similarly as previously explained, the large order set of 2100 orders is sorted based on the cross-order similarity degree that was given in Equation (2-3). After this sorting step, the algorithm has created an order sequence of 2100 orders. This sequence is then split into 30 pieces of equal length; each piece is regarded

as a picking station. The algorithm now calculates the completion sequence and the SKU sequence per picking station. These SKU sequences are stored in a matrix, and after every picking station, a check on the multiplicity constraints is conducted as has been described in Section 3-3. When a violation of the stock multiplicity constraint is found, the algorithm is restarted for the picking station that caused the violation. The pseudo code for the Greedy algorithm can be found in Algorithm 3. As mentioned, the only difference between the Greedy algorithm and the FCFS algorithm is the `CreateOrderSequence` function that sorts the order set according to the similarity degree. The result is that more similar orders are active on the picking station simultaneously. This is expected to yield a solution of better quality, since more stock totes can be used by multiple orders. The function `CheckMultiplicityConstraints` is described in Algorithm 2.

Algorithm 3 Greedy algorithm

Input: order set S, number of picking station n, stock multiplicity file
```
CreateOrderSequence
restart = True
sku_sequence_matrix = ∅
for i = 1:n do
  while restart do
    restart = False
    CreateCompletionSeqeunce
    CreateSKUSequence
    CheckMultiplicityConstraints
    if multiplicity constraint violated then
      restart = True
    end if
  end while
end for
```
Output: sku_sequence_matrix, objective

4-2-3 Global Genetic Algorithm (GA)

According to Whitley (2019), a GA is especially useful for scheduling problems. A global GA was created as a benchmark for the proposed meta-heuristic as it is considered to be a competitive algorithm. The global GA will be elaborated in this section.
The GA is set up to optimise the 2100 orders globally, thereby possibly converging to a global optimum. In order to achieve this, an individual consists of all 2100 order IDs. The sequence of the order IDs is regarded as the order sequence. From this sequence, all other information can be calculated. The toolbox used for creating the GA is called `DEAP` (Fortin et al., 2012). `DEAP` features several evolution operators. The ones that were used in this research will be discussed below. The pseudo code of the GA that was used as a benchmark is given in Algorithm 4.

Ordered cross-over

An ordered cross-over is useful when an individual consists of alleles that all need to be present in each individual. An example of this could be the solution to a Travelling Salesman Problem, or our problem where we need each order to be fulfilled. The cross-over operator needs two individuals as input, an example is

$$\begin{aligned} P_1 &= (8\ \ 4\ \ 7\ \ 3\ \ 6\ \ 2\ \ 5\ \ 1\ \ 9\ \ 0), \\ P_2 &= (0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9). \end{aligned} \tag{4-1}$$

At random, two cross-over points are determined for the first parent P_1. All the alleles that are in between the two cross-over points of parent P_1 are placed in the child O_1 in the exact same positions as they were found in P_1. Then the remaining positions are filled with alleles from P_2 in the order in which they appeared in P_2. Determining the order is fairly simple. If a P_2 allele is positioned before the second cross-over point, it will be placed before the P_1 alleles in the child. Any allele that was positioned after the second cross-over point is placed behind the P_1 alleles in the child. This is visualised as follows:

$$\begin{aligned} P_1 &= (8\ \ 4\ \ 7\,|\,3\ \ 6\ \ 2\ \ 5\ \ 1\,|\,9\ \ 0), \\ P_2 &= (0\ \ \cancel{1}\ \ \cancel{2}\ \ \cancel{3}\ \ 4\ \ \cancel{5}\ \ \cancel{6}\ \ 7\ \ 8\ \ 9). \end{aligned} \tag{4-2}$$

The resulting child is defined as

$$O_1 = (0\ \ 4\ \ 7\ \ 3\ \ 6\ \ 2\ \ 5\ \ 1\ \ 8\ \ 9). \tag{4-3}$$

If another child is desired, P_1 and P_2 are swapped and the operation is repeated.

Population size

Another parameter that influences the behaviour of a GA is the population size. As mentioned before, a population consists of individuals, each of which represents a solution. Note that in this case, not every solution is feasible per definition. The feasibility of each solution is checked using the pseudo code from Algorithm 5, which adjusts the fitness value if the solution is infeasible. It is easy to see that a larger population will explore more of the search space than a smaller population; this can potentially lead to a solution of better quality. However, a larger population means that more calculations are needed in each iteration. A larger population will therefore take a longer computation time.
In order to tune the GA a data set of 70 orders was optimised for different population sizes. The mutation probability used was 0.01 and the tournament size was 5 individuals. These parameters were tuned separately afterwards, as will be explained in the following section. The results from the experiment are summarised in Table 4-1. In this table, the results from 5 runs per population size are averaged. The column *Avg. best solution* therefore is the average of the best solution found in the last generation of each run. It can be seen that the expectations are fulfilled, i.e. the computation time increases with a larger population and the objective function decreases. This means that for a larger population, the GA finds a solution

Table 4-1: Results from the population size tuning

Population size	Avg. Computation time	Avg. best solution	Std. dev. of best solution
20	32.1	690.0	3.5
30	47.1	686.8	2.7
40	65.2	684.8	3.1
50	96.2	686.6	3.0
60	117.2	684.4	2.1
70	138.2	682.4	4.2
80	155.6	682.4	2.2
90	173.2	681.8	2.8
100	192.7	682.4	2.1
110	235.1	680.8	2.0
120	258.0	680.8	0.8
130	282.3	681.4	1.5
140	302.1	679.8	3.1
150	322.7	677.6	2.5
200	430.6	678.0	1.6

that needs fewer retrievals, and has therefore a better quality. In the end a population size of 90 individuals was chosen. This because this size features a significantly improved average solution quality in comparison with a population size of 20 individuals, while still using a reasonable computation time. Lastly, the standard deviation is reasonable as well, at only 0.41% of the average best solution.

Selection tournament and mutation

In a GA, a mutation operator is important for the performance of the algorithm. It can implement some random behaviour in solutions of already good quality. Thereby, it makes sure that the search space is thoroughly explored. In this research, a mutation happens via the `mutShuffleIndexes` operator. The operator uses an individual and a mutation probability P_m as an input. The mutation probability is the probability that an allele is shuffled with another allele. The way that the individuals are defined, a shuffle of an allele will yield a shuffled order sequence.

$P_m = 1/n$ is widely accepted as a suitable mutation probability, where n is the number of alleles of the individual (Whitley, 2019). In this research $n = 2100$, since that is the number of orders that are optimised. This means that each orders has a chance of $1/2100 = 0.00047$ to be swapped with another order. The mutation operator prevents the algorithm from getting stuck in a local minimum. With a mutation probability as low as 0.00047, mutation will hardly take place. This can result in a GA that does not explore the search space sufficiently. However, a too high mutation probability will prevent the algorithm from remembering useful features of a solution, thereby diminishing the performance.

A 2D experiment was run on the size of the selection tournament and the mutation probability. An order set of 70 orders was optimised by the global GA with tournament sizes ranging from 2 to 50 participants and mutation probabilities ranging from 0.0005 to 0.5. Each combination

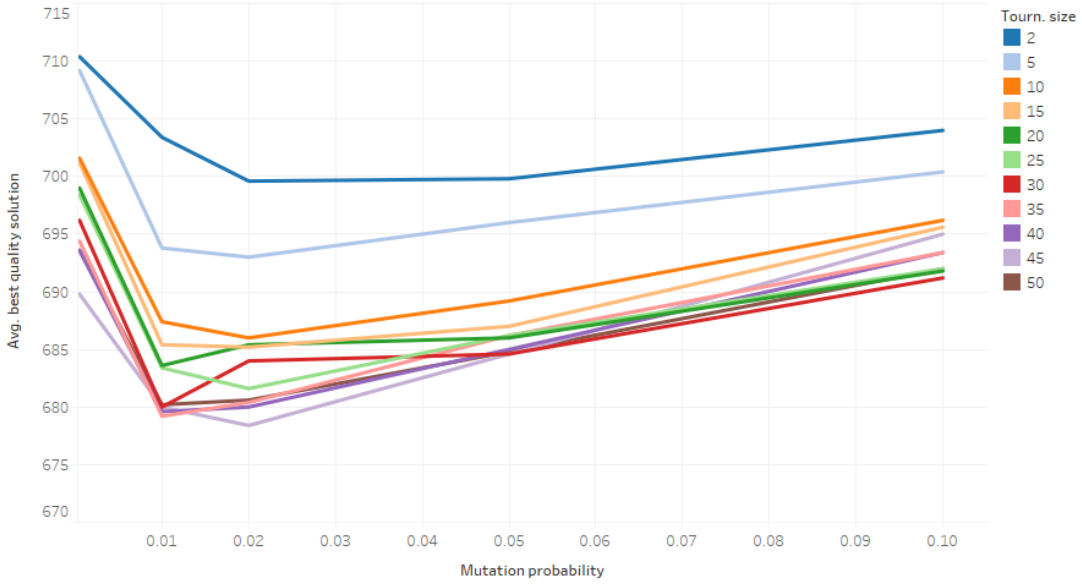


Figure 4-1: Results of the 2D analysis split per tournament size

of tournament size and mutation probability was ran 5 times to average out some of the statistical noise. The results of the analysis on mutation probability are shown in Figure 4-1. Please note that the qualities corresponding to a mutation probability of 0.5 are left out, as these runs performed badly and distorted the figure. It can be seen that the best results were obtained when a mutation probability of either 0.01 or 0.02 was used. The results on the tournament size can be found in Figure 4-2. It can be seen that especially the lower tournament sizes produce bad quality solutions, while for larger tournament sizes the incremental improvement is small. The computation time however, increases almost linearly with a larger tournament size. This can be seen in Figure 4-3.

The selection is carried out using a tournament. For a tournament x individuals are randomly picked from the current population. From these x individuals, the individual with the best fitness is transferred to the next generation. This means that a tournament of size one equals a random sampling selection operator. This is expected to diminish the performance of the GA as no useful features of a solution will be remembered. On the other hand, a tournament with the same size as the population results in the same individual getting selected every tournament. This will thus result in an entire population consisting of the same individuals. It is easy to see that this will not be beneficial for the solution quality.

A heat map of the combined results of the 2D experiment is shown in Figure 4-4, where it can indeed be seen that a mutation probability of either 0.0005 of 0.5 performs badly, as well as a tournament size of 2 as this is close to using random sampling for selection. In this figure, it can be seen that the lowest average minimums are found when using a mutation probability of 0.01 or 0.02 and a tournament size between 30 and 50. Within this area of possible combinations, the differences between average solution quality are minimal and could as well be attributed to statistical noise. Therefore, we take the computation time from Figure 4-3 into account. This led to choosing a mutation probability of 0.01 and a tournament size of 35 individuals.

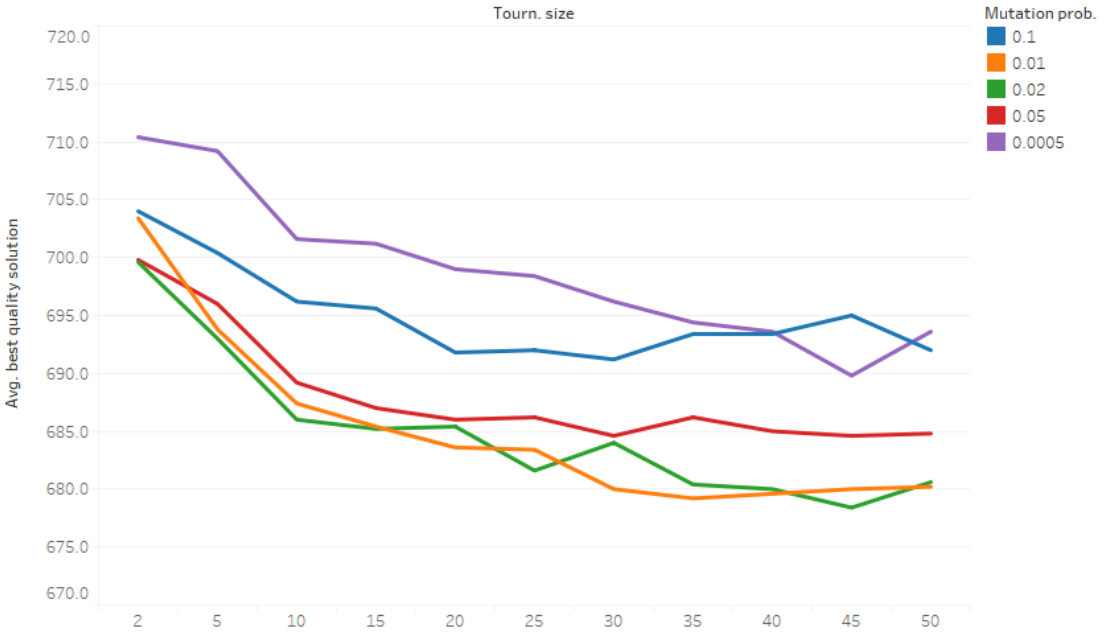


Figure 4-2: Results of the 2D analysis split per mutation probability

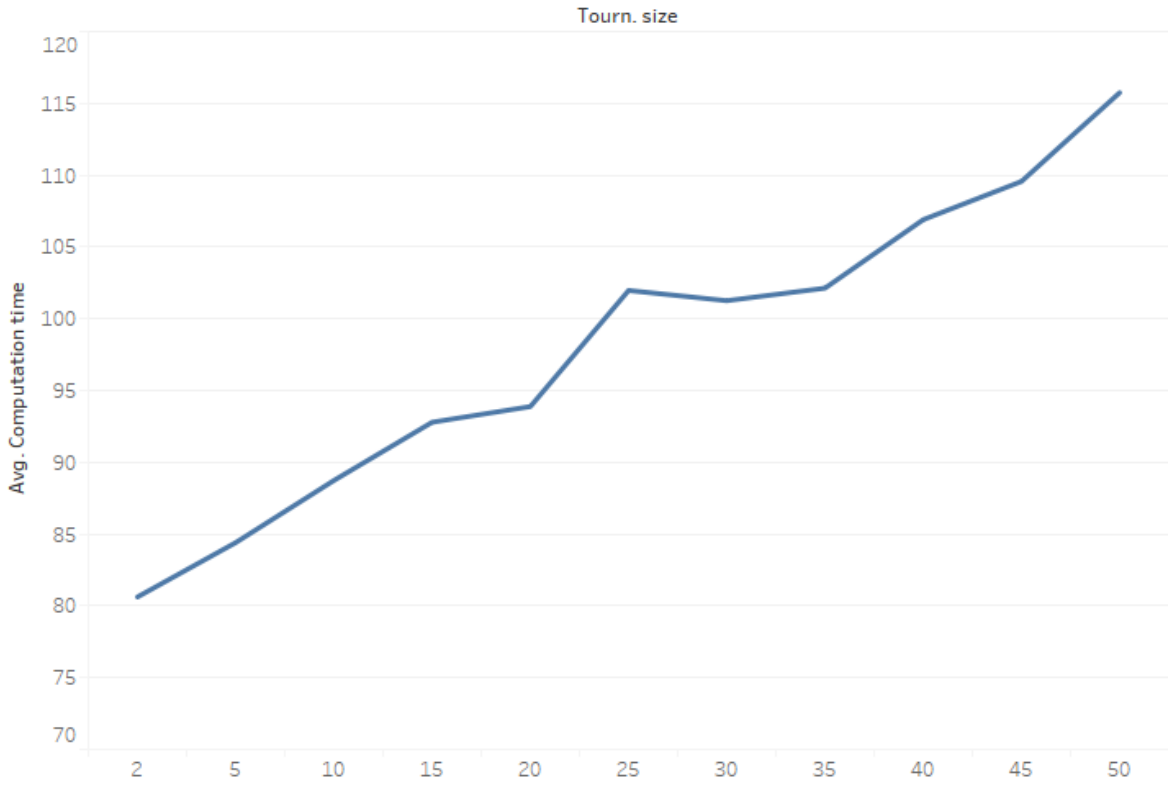


Figure 4-3: Tournament size vs computation time

Figure 4-4: A heat map of the results of the 2D analysis on GA parameters

Algorithm 4 General pseudo code for a GA

Input: Initial population, N_p: population size, P_c: crossover probability, P_m: mutation probability, n: number of generations

```
//Initial population:
for i = 1 to N_p do
    p_i := Generate solution;
    p_i.fitness := Evaluate(p_i);
end for
//Evolution
P := initial population
for generation < n do
    //Offspring generation
    for j = 1 to 5 do
        P_0 := P
        parents := Sample(P);
        offspring := Crossover(parents); //according to P_c
        offspring := Mutation(offspring); // according to P_m
        offspring.fitness := Evaluate(offspring);
        P_0 := P_0 ∪ offspring;
    end for;
    P := Selection(P_0); //according to N_p
end for
```

Output: Best generated solution;

Algorithm 5 General pseudo code for the fitness evaluation

Input: current population, n: number of picking stations, M: a large number
  for individual in current population **do**
    order_sequence := SplitIndividual(individual)
    sku_sequence_matrix := $\emptyset$
    fitness = 0
    for $i = 1 : n$ **do**
      compl_seq := GenerateCompletionSequence(order_sequence[n])
      sku_sequence := GenerateSKUSequence(compl_sequence)
      sku_sequence_matrix.append(sku_sequence)
      CheckMultiplicityConstraint
      if MultiplicityConstraint violated **then**
        individual.fitness := length(sku_sequence) + M
      else
        individual.fitness := length(sku_sequence)
      end if
    end for
  end for
Output: population.fitness

4-3 Results

In this section, the results of the different numerical benchmark experiments are discussed. In order to properly check the performance of the meta-heuristic and benchmark algorithms some smaller instances were checked, as these are more likely to be solvable by the MILP approach. This way, it was checked if the meta-heuristic is capable of producing optimal solutions for smaller, thus less complex, instances. These smaller instances featured 10-70 orders per data set. We chose at most 70 orders for the small scale experiments, as that is the amount of orders that will be optimised per picking station. For each number of orders, 20 different order sets were created by random sampling orders from the large data set supplied by Picnic. In each table in the appendix of this section, the results for different order set sizes are shown. In each row, the worst case number of retrievals for is given per order set. Recall that the worst case is defined as picking all order lines separately for all orders, in other words no cross-order similarity is exploited. The column $\bar{I}[\%]$ gives the average improvement achieved by each algorithm over a specific data set according to Equation (3-2). The average objective function value that has been obtained by the algorithm over a specific data set in a number of retrievals, is denoted by $\bar{O}[\#]$. Lastly, $\bar{T}[s]$ shows the average computation time used by the algorithm for a specific data set in seconds and the column *Std.* denotes the standard deviation of the objective. Each algorithm was run 5 times per data set in order to average out some of the statistical noise and randomness that is embedded in most of the algorithms.

All algorithms have been coded using the Pycharm (x64) 2018.3.7 edition IDE with the Python 3.7 interpreter. The linear programming was conducted using the `PuLP 1.6.10` package and using and `Gurobi Optimizer 8.1.1` as the MILP solver (Gurobi Optimization, 2020). Lastly, the GA was coded using the `DEAP 1.3.1` package (Fortin et al., 2012). As mentioned, the

hardware used throughout this research is a Dell laptop using an Intel(R) Core(TM) i5-8350U CPU with a base speed of 1.90 GHz and 16 GB RAM.

10 orders

The results from the experiment with an order set of 10 orders are shown in Table A-1 and summarised in Figure 4-5. It is clear that the MILP approach is more suitable for the simplified problem by Füßler and Boysen (2019). In their results, the authors showed that the MILP algorithm was capable of solving instances up to 30 orders. From Table A-1 it is clear that the MILP approach performs poorly and struggles to solve problems of just 10 orders. A solving time limit was set on 1200 seconds using the `Gurobi Optimizer` package. This was done in order to prevent the computer from running out of memory. An empty row for the MILP algorithm means that the solver could either not find a feasible solution within the solving time limit, or the computer ran out of memory, i.e. the data set seemed too complex for the solver to handle. In Figure 4-5, just the results for the data sets that were solved by all algorithms are shown, making it easier to make a meaningful comparison.

The MILP is outperformed by all other algorithms, both on the average improvement of the solutions and the average computation time, as can be seen in Table A-1. Figure 4-5 also shows that the solution quality of the MILP algorithm is generally worse than the quality of the other algorithms. This seems strange, as a MILP algorithm should output optimal results. However, it is stated in the Gurobi documentation that the solver will output the best found solution when the solving time limit is reached. This means that the solver can output sub optimal solutions.

Table A-1 shows the full results from the experiments with 10 orders. Please note that the total average objective of the MILP algorithm is a bit misleading in this table. This is due to the fact that the solver was not capable of generating feasible solutions for the somewhat larger problems. It could only generate feasible solutions for the smaller problems with a worst case scenario of less than 110 retrievals. As a result, the average objective function is significantly lower than for the other algorithms. When the average results are filtered by the instances that the MILP solver was capable of actually solving, the results are different; these results are depicted in Table 4-2. It can be seen that still the MILP approach performs the worst. This is due to the output of sub optimal solutions when using the time limit with the Gurobi solver. The FCFS and Greedy algorithm performed similarly. Both provided solutions of better quality than the MILP approach in a short computation time. The solutions from the Greedy algorithm feature a slightly larger standard deviation than the solutions of the FCFS algorithm. This was to be expected since the experiments featured a small order set size and the FCFS does not create the order sequence based on a property of the order. The Greedy algorithm does use the similarity between orders to create the order sequence, but some randomness is present in this process. This can therefore result in different order sequences for each run, which explains the slightly larger standard deviation. Even so, both the GA and the proposed meta-heuristic approach perform similarly and produce solutions of better quality than the MILP approach. The proposed meta-heuristic, however, uses only around a third of the computation time of the GA.

A key take away from this small scale experiment is that both the GA and the proposed meta-heuristic are capable of producing (near-)optimal solutions. However, since the time

Table 4-2: the results for 10 orders, filtered on the instances that were solvable by every algorithms

	$\bar{I}[\%]$	$\bar{O}[\#]$	$\bar{T}[s]$
MILP	2.16	100.57	1,319.97
FCFS	2.46	100.26	0.10
Greedy	2.35	100.37	0.10
GA	2.78	99.93	12.54
Heuristic	2.78	99.93	4.06

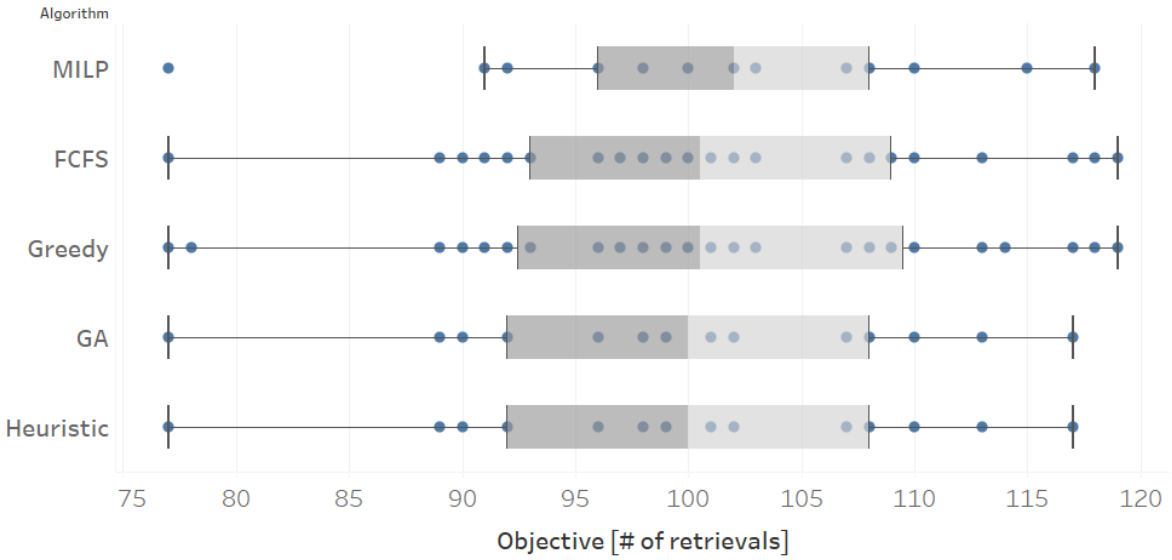


Figure 4-5: The distribution of the quality of the solutions for 10 orders, filtered on the data sets that were solvable by all algorithms

limit is used, it is hard to prove optimality for the solutions generated by the MILP. Another interesting phenomenon is that the GA and the proposed meta-heuristic take much longer to compute a feasible solution than both the FCFS and the Greedy algorithm. This was expected, as the proposed meta-heuristic and the GA improve an initial solution iteratively.

20 orders

It is now interesting to check how the behaviour of the algorithm scales with the order set size. It is important to note that the MILP approach is not featured for the experiment that are larger than 10 orders. This choice was made since the MILP approach was struggling to provide feasible solutions for 10 orders, let alone optimal solutions. It was found that the computer ran out of memory for order sets of 20 orders or larger, even with the solving time limit set on 20 minutes.

The results of the experiments featuring 20 orders are as expected; they are shown in Table A-2 and summarised in Figure 4-6. As the order set is now larger, there are more possibilities to differentiate for the algorithms. It can be seen that the worst performance in terms of average improvement is generated by the FCFS algorithm. Note that this simple algorithm is still

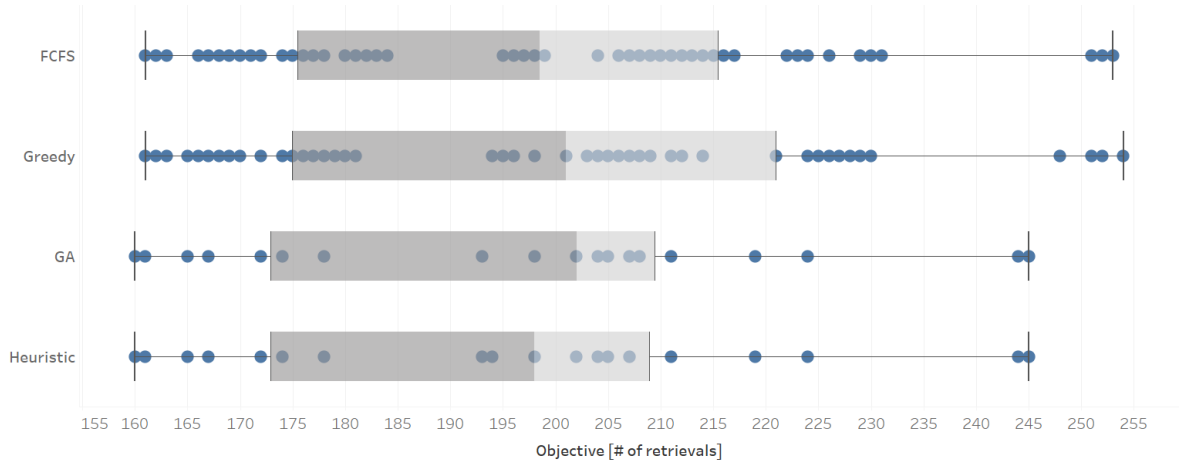


Figure 4-6: The distribution of the quality of the solutions for 20 orders

capable of improving the worst case scenario by 2.25%, i.e. the algorithm that is currently used in most automated fulfilment centres.

Once again, the proposed meta-heuristic and the GA produce similar results. Both these algorithms improve the worst case scenario by 4.75% on average with a minimal average standard deviation. The only big difference is the average computation time. The proposed meta-heuristic needs less than half the computation time of the GA. However, since the order set size is still rather small, and the GA and the proposed meta-heuristic perform similarly, it is hard to draw conclusions on what algorithm would be the better choice for solving large scale order sequencing problems. The performance of the algorithms is visualised in a box plot in Figure 4-6, where we can see that in general, the meta-heuristic returns the best quality solutions.

50 orders

The experiments consisting of 50 orders again yielded results that were expected, as can be seen in Table A-3. This time, as the order set size was larger than in the previous, there was even more room for the algorithms to differentiate. Again, the FCFS algorithm performed relatively poorly, improving the worst case scenario by only 3.28% on average. The Greedy approach to solving the order sequencing problem performs slightly better, improving the worst case scenario by almost 5.5%. The standard deviation is slightly higher for the Greedy algorithm than for the FCFS algorithm. This is to be expected as has been discussed previously. Still, the standard deviation accounts for less than a percent of the average objective value. In these experiments, we can now see that the proposed meta-heuristic is well suited for the order scheduling problem. It achieves the best quality solutions, improving the worst case scenario with around 9.6% on average. Also, the solutions are quite close together with an average standard deviation of just 2.09 retrievals. We can also see that the proposed meta-heuristic uses less computation time than the GA, around 60% of it. The GA also performs slightly worse than the meta-heuristic, improving the worst case scenario just with 8.95% on average. The results have been summarised in Figure 4-7.

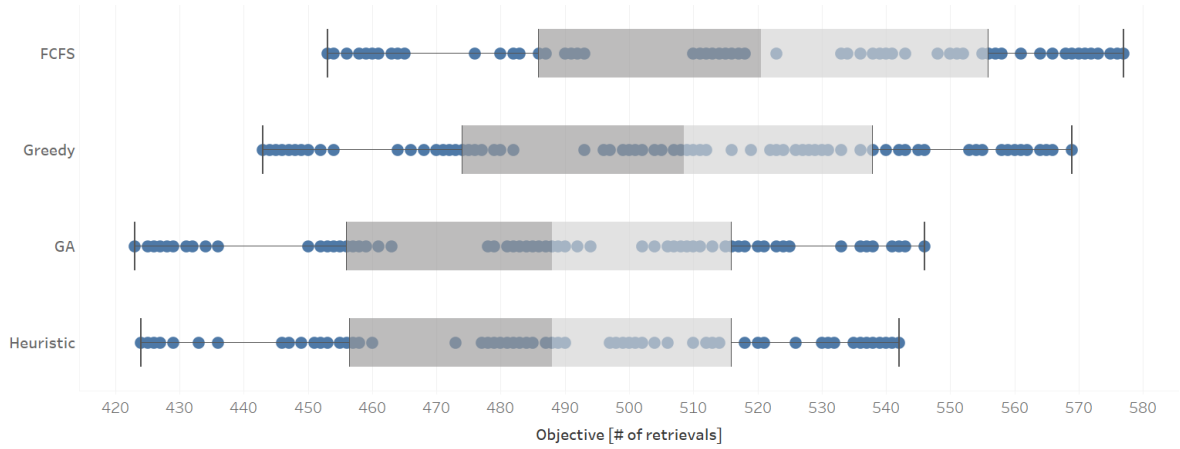


Figure 4-7: The distribution of the quality of the solutions for 50 orders

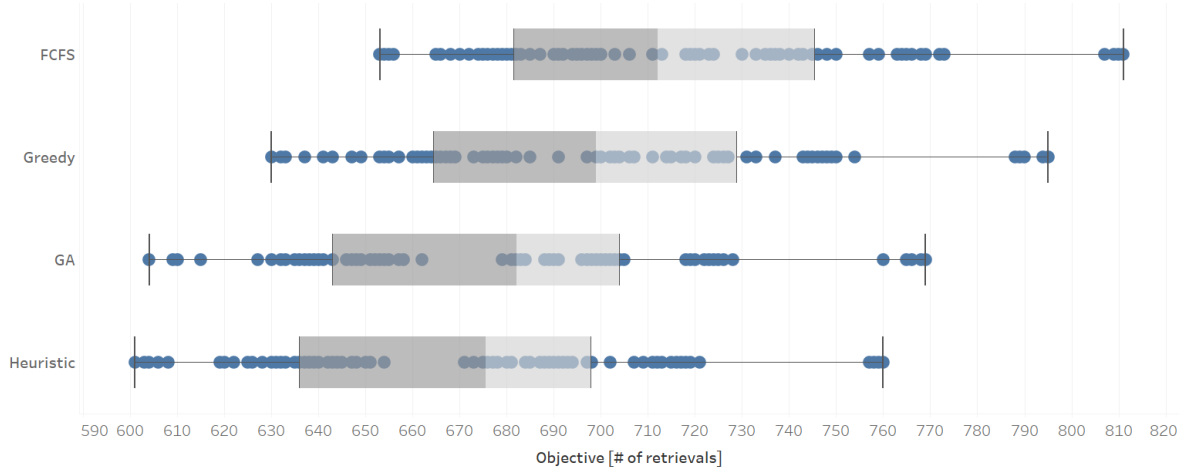


Figure 4-8: The distribution of the quality of the solutions for 70 orders

70 orders

Now the results on the experiments of 70 orders are discussed; they are summarised in Figure 4-8. This is the largest order set size that was tested in a single picking station setup, due to what has been discussed in Section 3-1. It can be seen in Table A-4 that the FCFS performs quite poorly again by improving the worst case by around 3.5%. The Greedy algorithm again performs significantly better by improving the worst case scenario by 5.9%; it does yield a slightly higher average standard deviation. Furthermore we can see that the proposed meta-heuristic improves the worst case scenario the most, but it needs quite some computation time. It does, however, perform better than the GA on all facets, as it produces better solutions in less computation time. The GA features a lower average standard deviation and a lower average objective, which means that it will produce solutions using more retrievals on a more consistent basis in comparison with the meta-heuristic.

2100 orders

In the previous paragraphs, the results of the experiments of relatively small scale have been discussed. In this paragraph we will discuss the results of the large scale experiments of 2100 orders. All algorithms have been adapted such that they can deal with multiple picking stations. For the FCFS and Greedy algorithm, this is done in a similar way as was proposed for the meta-heuristic. This has been described in Section 3-2; each algorithm will optimise the picking stations sequentially and store the SKU sequence per picking station. After a picking station has been optimised, the multiplicity constraint is checked. Before the different SKU sequences can be determined, it is important to know which orders are assigned to which picking station. The FCFS algorithm is the most straightforward in this; it reads the data set and the first 70 orders are assigned to the first picking station, and so on. The Greedy algorithm uses the similarity degree to assign orders to picking stations, as has been explained in Section 3-2.

Again, as expected, the FCFS algorithm performs poorly. This can be explained by the fact that the FCFS algorithm does not sort the orders on similarity. It will therefore most likely assign orders with a low similarity degree to the same picking station. It is then incapable of creating a SKU sequence of good quality and is does not use a improvement stage. This all sums up the relatively low improvement of the FCFS algorithms. Interesting to see is also the large average standard deviation in comparison with the other algorithms. The FCFS has an average standard deviation of around 1.1% of the average objective value; this is an order of magnitude larger than the other algorithms.

In this large scale experiment, it becomes clear what the benefits of sorting orders based on similarity are. The Greedy algorithm, like the FCFS, does not use any heuristic improvement stages in order to solve the order sequencing problem. Still, it is capable of improving the worst case scenario by almost 8.67% on average, 4,78 percent point more than the FCFS algorithm. It achieves this with a significantly smaller standard deviation.

Figure 4-9 shows that the proposed meta-heuristic approach is far superior to the other algorithms. Not only is the distribution of the objective values positioned at a much better quality than the other algorithms, the distribution is also quite narrow in comparison. This means that the proposed meta-heuristic produces high quality solutions relatively consistently. A downside is, of course, the computation time it needs to arrive at these solutions. On average, the meta-heuristic takes 3072 seconds of computation time, or around 51 minutes. Considering the order set size of 2100 orders this could potentially become a problem. Currently, between 2000 and 3000 orders are produced in Picnic's PTG fulfilment operations. This number is expected to grown with a factor 10 when GTP picking is used, thereby increasing the computation time for devising a suitable order release strategy. On the other hand, it should be noted that this research is intended as a proof of concept and not as an implementation study. It is expected that the computation time used by meta-heuristic can be shortened by using more efficient code.

Looking at Figure 4-9 and Table A-5, some interesting results are visible for the GA. While the GA performed well in the small scale experiments, the performance diminishes with larger order set sizes. The GA was only capable of improving the worst-case scenario with around 4.4% on average, while it was capable of improving the worst case scenario with around 9.2% on average with the experiments on 70 orders. A few different hypotheses were formulated.

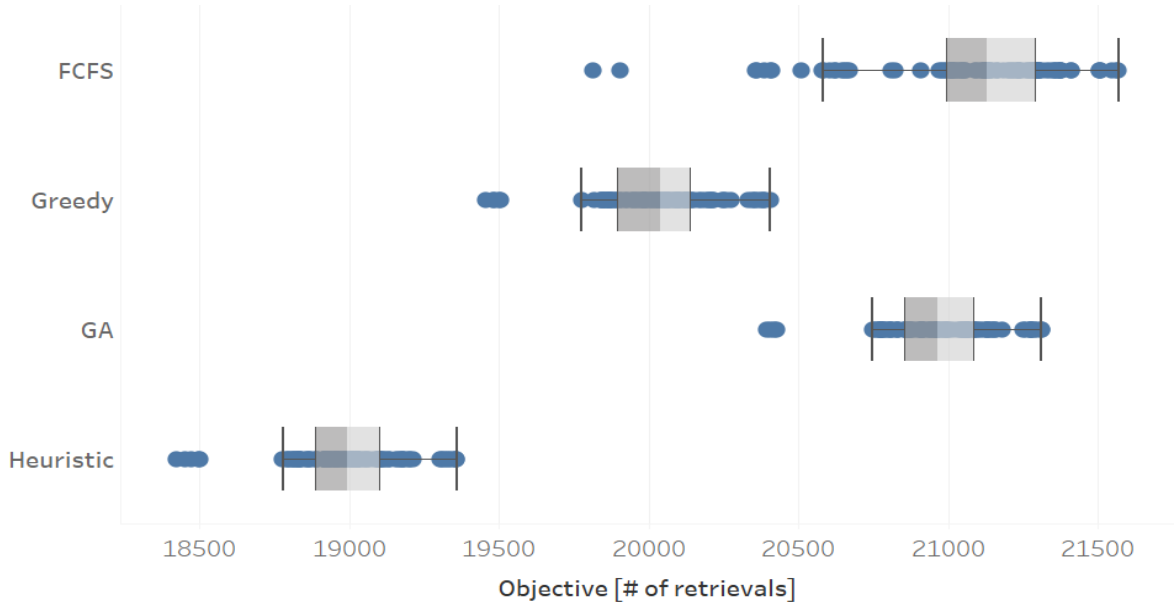


Figure 4-9: The distribution of the quality of the solutions for 2100 orders

The first hypothesis was that the GA was not properly tuned for such large order sets. Remember from Section 4-2-3 that different settings were tested to tune the GA properly. The data sets on which the GA was tuned however, consisted of only 70 orders. It is possible that the scale of the data set changes the most suitable settings for a GA. For instance, remember that the mutation probability was set to 0.01, while according to literature it should have been set to 0.0005. A second hypothesis was that, since the experiments features many more orders, it needs more generations to return a good quality solution. Lastly, a hypothesis was that a violation of the a stock multiplicity constraint should have been featured explicitly in the genome of the individuals. This is explained in the next paragraph.

The biggest difference between the small scale experiments and the large scale experiment is the fact that small scale experiments are small enough to use just one picking station. That is, each picking station will process 70 orders, since this was decided in Section 3-2. As a result, at most one of each SKU can be requested at any time step, thereby eliminating the chance of violating the stock multiplicity constraints. So the only experiments in which a stock multiplicity constraint can be violated are the experiments with 2100 orders. Remember from Section 3-3, Section 4-2-1, and Section 4-2-2 that a multiplicity violation can be dealt with directly. This is done by either re-positioning the order that caused the violation in the proposed meta-heuristic or by restarting the algorithm when a violation is detected for both the FCFS and Greedy algorithms. Also note that the GA can create infeasible individuals, where each allele corresponds to an order ID. The sequence of the order IDs in the individual thus determines the order sequence. The GA then creates a completion sequence and SKU sequence as has been described in Section 2-3. These procedures use certain random processes, such as sampling. Afterwards, the GA will evaluate the fitness of every individual using the algorithm described in Algorithm 2. In short, the sequence of the alleles does influence the chance of a multiplicity violation, but it does not directly determine if a multiplicity violation will occur. The GA severely punishes stock multiplicity constraint violations by worsening

the fitness of an individual with such a violation. However, since the SKU sequence is not directly determined by the sequence of the alleles, it can happen that an order sequence that features a good solution is punished due to a randomly formed stock multiplicity violation. As a result, areas of the search space featuring potentially good quality solutions can be ignored, thereby diminishing the performance of the GA.

In order to test the hypotheses, the GA was rerun using different settings. To test if the (lack of) tuning of the GA was responsible for the bad performance, the mutation probability was set to 0.00047. To test if too few generations were used, the GA was set to 300 generations instead of 100. Lastly, to test if the stock multiplicity check was responsible for the bad performance, the GA was run without the stock multiplicity check, i.e. an infinite stock was assumed. If the GA were to provide good quality solutions on one of these set-ups, the corresponding hypothesis can be accepted as true. The data set that featured the largest average improvement by the meta-heuristic was chosen for these tests. The results of these extra experiments are compared to the runs of the GA including the stock multiplicity check, and to the meta-heuristic runs. Each version of the GA was run 5 times on the same data set.

In Table 4-3 it is shown that the tuning of the mutation probability is probably not the cause of the GA's bad performance, as the average improvement is slightly worse for a mutation probability of 0.0005. Ignoring the stock multiplicity constraint does yield a larger improvement, though not a significantly larger one. The largest improvement by a GA was generated by increasing the number of generations from 100 to 300. This also resulted in the smallest standard deviation so a GA that uses more generations is more consistent in producing better quality results. A disadvantage is the computation time, that almost tripled in comparison with the computation time of the original GA. Another way to explore more of the search space is use a larger population. This will also increase the computation time. It also needs to be noted that still the meta-heuristic produces far superior solutions than the best performing GA.

Lastly, it can be seen in Table A-5, that the original GA already takes the longest average computation time of all algorithms. This in combination with the bad performance in terms of solution quality, makes this specific GA not suitable for this particular problem. In order for it to be a more suitable alternative to the proposed meta-heuristic, it needs to be adapted. We can conclude that the GA does not seem a suitable algorithm for the case of 2100 orders. However, the potential of a GA has been shown during the small scale experiments. The problem is that the experiments of 2100 orders feature 30 times more orders, and therefore needs more exploring moves. As the computation time almost tripled when we increased the number of generations from 100 to 300, one can expect enormous computation times when more generations are added in order to increase the GA's performance. It will therefore not be useful in daily picking operations.

4-4 Conclusions

In this section, conclusions will be drawn on the results of the numerical experiments. The conclusions will be split between the small scale and the large scale experiments. In the small scale experiments, the most interesting results are summarised in Figure 4-5 and Table 4-2,

Table 4-3: The results of GA experiments with different settings

Algorithm	$\bar{I}[\%]$	$\bar{O}[\#]$	Std.	$\bar{T}[s]$
GA lower mutation	4.18	20,862	20.69	3,188.67
GA original	4.42	20,811	19.38	3,495.49
GA inf stock	4.45	20,804	31.85	2,925.81
GA 300 generations	5.00	20,684	17.96	9,846.62
Meta-heuristic	13.52	18,830	26.47	3,013.42

where it is visible that the MILP approach performed quite poorly. A time limit of 1200 seconds was needed; without it, the computer would run out of memory. A result of this, is that the Gurobi solver will return a sub optimal solution when the time limit is reached. It can also be seen in Table A-1 that for 6 of the 20 data sets, either Gurobi could not find a feasible solution or the computer ran out of memory. Another interesting thing from the 10 order experiments is that the GA and the proposed meta-heuristic perform similar. They return the same results in every run. This insinuates that a global optimum is found for these experiments, however this cannot be proven as the MILP approach returned a worse solution for 50% of the problems it was capable of solving. The computer ran out of memory while trying to solve an experiment of 20 orders with the MILP approach. It can therefore be concluded that solving a realistic order scheduling problem is too complex to be done using a MILP formulation

The rest of the small scale experiments returned results that were largely expected. In the experiments with 20 orders, the GA and the proposed meta-heuristic yielded superior results compared to the FCFS and the Greedy algorithm. The experiments with 50 order show a slight advantage for the proposed meta-heuristic, which returns solutions of better quality in less computation time and with a lower standard deviation compared to the GA. The solution quality that is returned by either the FCFS or the Greedy algorithm is significantly worse than the solution qualities from the GA and the proposed meta-heuristic. This pattern is continued in the experiments with 70 orders, however the advantage of the meta-heuristic is slightly larger. From the small scale experiments it can be concluded that the proposed meta-heuristic is a promising algorithm to solve the order scheduling algorithm. It is capable of returning good quality solutions in a reasonable computation time.

Even more interesting are the results of the large scale experiments. Currently mostly the FCFS algorithm, or a sorting algorithm based on due dates is used for determining a suitable order release strategy. Furthermore, cross-order similarity is not exploited due to the fact that in traditional e-commerce, orders consist of few order lines. These algorithms will result in a fairly random order sequence in terms of cross-order similarity. It can be seen in Table A-5 that this results in a relatively low average improvement of around 3.9% in comparison with the worst case scenario. Sorting the orders beforehand based on cross-order similarity, is beneficial for the solution quality. This sorting step is a relatively simple procedure that significantly improves the solution quality. It can be seen that the Greedy algorithm improves the worst case scenario by almost 8.7% on average, thereby reducing the average number of retrievals needed by 1047. Please note that in the Greedy algorithm, no heuristic improvement has been applied. The results show that a heuristic improvement stage is also beneficial for the solution quality. The proposed meta-heuristic achieved an average improvement of around 13.4% in

comparison to the worst case scenario. The performance of the GA, however, diminished when solving problems of larger scale. This behaviour is probably due to the fact that not enough of the search space can be explored with the settings used. In order to overcome this, more generations are needed, or a larger population should be used. It can be concluded that this GA is not well suited to solve such complex problems in a reasonable computation time. Even if more generations are used, the question is if a GA will be able to produce better quality solutions than the proposed meta-heuristic, as the GA was outperformed by the meta-heuristic in the small scale experiments.

In the end, the conclusions on the meta-heuristic approach should be based on the most realistic problems, i.e. on the experiments with 2100 orders. Based on the average improvement and the average number of retrievals, it can be concluded that the meta-heuristic is the most suitable algorithm due to the large average improvement it generates. A down side on the meta-heuristic is the long computation time, of around 50 minutes on average. It needs to be noted that a day's orders are known at 22:00 the evening before; the entire night is available for calculating a suitable order release strategy. Furthermore, the code used for this algorithm will not be the most efficient, meaning that the computation time can potentially be decreased with developing effort. This means that the computation times in this chapter should be looked at as an indication of the computation time of a fully developed algorithm.

Chapter 5

Conclusions and recommendations

In this chapter, conclusions will be drawn on this research. The conclusions of previous chapters are summarised in Section 5-1. These conclusions will be discussed in Section 5-2. Afterwards, in Section 5-3, some recommendations for follow-up research are given.

5-1 Summary and conclusions

The aim of this research is to develop an algorithm that can return a (near-)optimal order release strategy. Hardly any research has been done on improving the picking efficiency by exploiting cross-order similarity. To the best of the author's knowledge, only a few papers have tried this method. Füßler and Boysen (2019, 2017); Boysen et al. (2019) all try to exploit cross-order similarity in different operational set-ups. In these papers, it was concluded that such an order sequencing problem is $\mathcal{NP}-$hard and that (meta-)heuristic methods are the most suitable way to solve such problems. In these papers a lot of assumptions are made; the problems solved in these papers are therefore simplified versions of realistic problems. The orders in the paper by Füßler and Boysen (2019) contain at most 10 Stock Keeping Units (SKUs), chosen from at most 50 different kind of SKUs. In e-commerce, typically more different SKUs are sold and at Picnic the orders consist of around 30-40 SKUs. Furthermore, the aforementioned papers consider only one picking station, thereby ignoring interdependencies between multiple picking stations. This also significantly simplifies the algorithm.

In order to consider multiple picking stations, it is first important to determine how many orders a picking station will fulfil. In this research an analysis was done on the trade-off between the percentage of improvement from the worst case scenario and the computation time. It was found that 70 orders per picking station yielded a favourable trade-off. Note that the relation between the percentage of improvement and the computation time will differ per machine. Therefore, it is recommended to redo this small analysis when implementing the meta-heuristic on another machine. The algorithm was extended to deal with multiple picking stations, assuming infinite stock. That is, the picking stations are decoupled completely and no interdependencies are taken into account. It was found that such a scenario is not

realistic, as SKUs with a low stock multiplicity can be required by multiple picking stations simultaneously. A strategy was devised to deal with these stock multiplicity violations. If an order was found to violate the stock multiplicity constraint, it was placed at a random position in the completion sequence. It was found that this method is capable of dealing with such stock multiplicity constraint violations. It was expected that this meta-heuristic would yield a lower performance in terms of solution quality in comparison of the meta-heuristic that optimises just one picking station. However, it did not. A possible explanation for this behaviour is the fact that all orders are sorted on similarity before the heuristic improvement and SKU elimination take place. Since the orders have been sorted beforehand, these two stages are capable of generating higher quality solutions.

Numerical experiments were carried out on different order sets. In the smaller order sets, the results were largely as expected. First of all, the MILP approach seemed unsuitable for realistic order sets. It was not capable of returning optimal solutions for the smallest order sets, while for larger order sets the computer ran out of memory. The proposed meta-heuristic was benchmarked against a simple FCFS algorithm, a Greedy algorithm consisting of stage one of the meta-heuristic, and a global Genetic Algorithm (GA). It was found that for order sets up to 70 orders, using one picking station, the meta-heuristic returned solutions of superior quality in comparison with the benchmark algorithms. Even a properly tuned GA, which is widely considered to be a useful algorithm for sequencing, provided lower quality solutions while needing more computation time.

In the experiments with larger order sets, multiple picking stations were used. Here it was found that the proposed meta-heuristic returned results of far superior quality. It managed to improve the worst case scenario by 13.37% on average. However, this came at a large computational cost: the algorithm needed around 51 minutes of computation time on average to optimise 2100 order divided over 30 picking stations. In comparison, the FCFS algorithm performed poorly, with an average improvement of the worst case scenario of only 3.89%. It was much quicker than the meta-heuristic, with an average computation time of just 1.4 seconds. The Greedy algorithm performed reasonably good. It improved the worst case scenario by 8.67% on average while only needing around 90 seconds of computation time. The GA disappointed in the large scale tests with an average improvement of only 4.43%. It was found that by increasing the number of generations, the performance of the GA improved. However, this came at a cost of extra computation time. The GA already used the longest average computation time. It is therefore questionable if adding more generations would yield a suitable algorithm.

In the end it is concluded that the proposed meta-heuristic is a suitable algorithm for solving large scale order sequencing problems. It is capable of handling multiple picking stations and of resolving potential stock multiplicity constraint violations. A critical note is the current average computation time, which is significant. If such a computation time is not an issue, the meta-heuristic is recommended for implementation. It could, for instance, be used to create a (near)-optimal order release strategy before the actual operation starts. If, however, issues occur during the operation, it will be too slow to solve these issues in real time. Fur such a cause, the Greedy algorithm shows a more favourable trade-off between solution quality and computation time.

During this research, a gap has been found in the current literature. To the best of the author's knowledge, only three papers have contributed to literature on order sequencing to improve

order picking efficiency within Goods-To-Picker (GTP) picking stations. This MSc thesis has contributed to that knowledge by extending a meta-heuristic approach from literature to be able to deal with larger, more realistic problem sizes. Furthermore, the meta-heuristic is now capable of devising a suitable order release strategy for a GTP warehouse featuring multiple picking stations while taking stock multiplicity into account.

5-2 Discussion

In this research several assumptions have been made, as has been described in Section 3-1. These assumptions will be discussed in this section.

First of all, it has been explained that the order set has been filtered on temperature zone and on the GTP picking stations. The Picnic operation will be divided in three temperature zones: *ambient* for non-perishable goods, *chilled* for products that need to be cooled, and *frozen* for frozen products. Leaving certain SKUs out of the problem may result in higher cross-order similarity values, and therefore a higher performance of the algorithm. However, it can be justified when looking at the actual picking operation that will take place in Picnic's automated fulfilment centre. First of all, the *frozen* and *chilled* picks will happen in completely separate picking stations. It therefore makes sense to not include them in this problem. Secondly, in a different picking strategy, called *zone picking*, products are picked straight from pallets. This is specifically useful for fast moving products where it does not make sense to put the products away in stock totes. As *zone picking* works differently than GTP picking, it needs to be modelled differently and it was not incorporated into this research. It does make sense to devise a strategy how to combine *zone picking* and GTP picking in a (near-)optimal way. As this was out of scope for this research, it provides a nice opportunity for a follow-up research.

In order to simplify the modelling, time slots are used in the MILP approach. A time slot is defined as the interval of time in which a certain subset of the order set is active on the picking station, as well as a specific SKU. In the meta-heuristic algorithm similar SKU sequence positions are used. A big disadvantage of this method is that it is hard to quantify actual order fulfilment times. This is due to the fact that a time slot or SKU sequence position is not of a fixed time length. This is especially fruitless when certain orders might need to be prioritised, for instance because of an early due date. If the system cannot calculate the order fulfilment time properly, this may cause delays. As a result of this assumption, stochastic behaviour has not been taken into account in this research.

As mentioned, the proposed meta-heuristic needs a lot of computation time and does not take stochastic behaviour into account. It is therefore useless as a real-time optimisation method. Nonetheless, it can be used to compute a near-optimal order release strategy that can serve as an input for the warehouse management system. Also, at Picnic the orders are known at 22:00 the evening before the actual operation. As a result, the entire night is available for calculating such a strategy. It needs to be noted however, that this may cause a problem in the future, when the automated fulfilment centre produces more orders than are currently optimised. It also needs to be noted that the current algorithm may not be written in the most time efficient way. It is expected that, if Picnic chooses to implement the algorithm, it will need less computation time after the code is improved by professionals. Furthermore,

the algorithm is currently run at a laptop, while it will be run in parallel on multiple larger machines when it is implemented into the Picnic GTP picking operation. As this research serves as a proof-of-concept, the computation time should not yet be treated as the most important performance indicator. However, the current computation times could serve as an estimate for the computation times of fully developed future versions.

5-3 Recommendations

The discussion points given in Section 5-2 give rise to some recommendations for future research. For Picnic, it does make sense to use precedence constraints. Such a constraint will guarantee the quality of service and makes sure that no product gets squashed underneath other, heavier products. In order to do so, the precedence constraints state the order in which the products need to be picked. Such constraints, however, have not been taken into account in this research. It is expected that precedence constraints will decrease the quality of the solutions that are currently returned by the meta-heuristic. This is due to the fact that such a constraint may force a non-overlapping SKU to be the last SKU to be picked for an order. An interesting question is how much a pick sequence constraint will affect the performance of the algorithm.

In this research, an algorithm has been created that is capable of dealing with the stock multiplicity constraints. However, it is unclear if the strategy used is the most optimal. Future research should try to find different strategies to deal with the stock multiplicity constraint, or eliminate the need for it altogether. As mentioned, around 99.5% of the stock multiplicity constraint violations were caused by SKUs that are stored in just one place. Future research could devise a strategy to adjust the stock multiplicity to the actual demand.

This research has developed an algorithm to devise a suitable order release strategy. It has been mentioned that no stochastic behaviour has been included in this research. Therefore, there is a large chance that the actual operation will differ from the planned strategy. It would be interesting to optimise the fulfilment operation in real-time. This way, stochastic issues, such as an Shuttle-Based Storage/Retrieval System (SBS/RS) aisle suffering down time, can be dealt with swiftly. This should of course be done with real time data, and requires a different approach than has been used in this research. This can also help in determining when to start or shut down certain parts of the operation.

As mentioned in Section 5-2, the order sets have been filtered on temperature zone and picking strategy. This was done purposefully, as different picking strategies need to be modelled differently. An interesting topic however is how to combine different picking strategies into one order fulfilment strategy. For instance, would the picking efficiency improve if an order fulfilment operation always starts in the *zone pick* area, or how to deal with order lines in several picking zones while taking precedence constraints into account? It is, for instance, easy to see that it will not be beneficial to always start with picking the fast-moving products if a large portion of the fast-moving products is fragile. On the other hand, letting the order tote travel between a GTP picking station and a *zone picking* station could potentially violate the capacity of the conveyor system. It can be seen that such a strategy needs to take a large portion of the warehouse operations into account, which will be extremely complex.

Lastly, as technology advances, robotised picking stations are becoming more and more suitable for the order picking task. Currently several companies create robotised picking stations that can easily pick box-like order lines. However, for more fragile products, human pickers do a much better job. This provides an opportunity to partially robotise the picking stations and let the robotic picking operation start during the night. This can potentially unburden the human pickers during the day, thus leading to more capacity. For this cause however, a suitable strategy should be developed that takes precedence constraints and multiple picking strategies into account.

Appendix A

Experimental results

Table A-1: The results of experiment with 10 orders

	MILP				**FCFS**				**Greedy**				**GA**				**Meta-heuristic**			
Worst Case [#]	$\bar{I}[\%]$	$\bar{O}[\#]$	Std.	$\bar{T}[s]$	$\bar{I}[\%]$	$\bar{O}[\#]$	Std.	$\bar{T}[s]$	$\bar{I}[\%]$	$\bar{O}[\#]$	Std.	$\bar{T}[s]$	$\bar{I}[\%]$	$\bar{O}[\#]$	Std.	$\bar{T}[s]$	$\bar{I}[\%]$	$\bar{O}[\#]$	Std.	$\bar{T}[s]$
138	–	–	–	–	1.01	136.60	0.89	0.10	1.45	136.00	0.00	0.09	1.45	136.00	0.00	17.10	1.45	136.00	0.00	5.45
98	2.04	96.00	0.00	1,298.54	1.63	96.40	0.55	0.09	1.63	96.40	0.55	0.10	2.04	96.00	0.00	11.77	2.04	96.00	0.00	3.78
103	0.97	102.00	0.00	1,314.08	1.75	101.20	0.45	0.11	1.75	101.20	0.45	0.11	1.94	101.00	0.00	12.72	1.94	101.00	0.00	4.10
121	–	–	–	–	2.64	117.80	0.84	0.09	3.14	117.20	0.45	0.10	3.31	117.00	0.00	14.69	3.31	117.00	0.00	4.88
104	3.85	100.00	0.00	1,317.88	4.42	99.40	0.55	0.10	4.04	99.80	0.84	0.10	4.81	99.00	0.00	12.56	4.81	99.00	0.00	4.52
91	0.00	91.00	0.00	1,279.88	2.20	89.00	0.00	0.09	1.10	90.00	1.00	0.10	2.20	89.00	0.00	11.07	2.20	89.00	0.00	3.21
109	1.83	107.00	0.00	1,340.44	1.65	107.20	0.45	0.10	1.65	107.20	0.45	0.10	1.83	107.00	0.00	13.36	1.83	107.00	0.00	3.93
100	2.00	98.00	0.00	1,304.79	1.60	98.40	0.55	0.10	2.00	98.00	0.00	0.10	2.00	98.00	0.00	11.89	2.00	98.00	0.00	3.34
105	1.90	103.00	0.00	1,322.44	2.10	102.80	0.45	0.09	2.29	102.60	0.55	0.10	2.86	102.00	0.00	12.82	2.86	102.00	0.00	4.09
94	2.13	92.00	0.00	1,287.83	1.70	92.40	0.55	0.09	1.91	92.20	0.45	0.10	2.13	92.00	0.00	11.43	2.13	92.00	0.00	3.55
94	3.19	91.00	0.00	1,287.72	3.19	91.00	1.00	0.09	3.83	90.40	0.55	0.10	4.26	90.00	0.00	11.80	4.26	90.00	0.00	3.98
110	1.82	108.00	0.00	1,341.06	1.64	108.20	0.45	0.09	1.27	108.60	0.89	0.10	1.82	108.00	0.00	13.74	1.82	108.00	0.00	4.23
155	–	–	–	–	4.00	148.80	0.84	0.09	4.13	148.60	0.55	0.10	4.52	148.00	0.00	19.32	4.52	148.00	0.00	7.11
83	–	–	–	–	0.00	83.00	0.00	0.09	0.00	83.00	0.00	0.11	0.00	83.00	0.00	9.99	0.00	83.00	0.00	2.66
114	–	–	–	–	4.21	109.20	0.45	0.09	4.04	109.40	0.55	0.10	4.39	109.00	0.00	13.78	4.39	109.00	0.00	4.64
78	1.28	77.00	0.00	1,250.90	1.28	77.00	0.00	0.10	1.03	77.20	0.45	0.10	1.28	77.00	0.00	9.64	1.28	77.00	0.00	3.07
147	–	–	–	–	3.54	141.80	0.84	0.09	2.99	142.60	0.89	0.11	4.08	141.00	0.00	18.40	4.08	141.00	0.00	6.86
112	1.79	110.00	0.00	1,349.55	1.79	110.00	0.00	0.10	1.79	110.00	0.00	0.10	1.79	110.00	0.00	13.51	1.79	110.00	0.00	4.69
120	4.17	115.00	0.00	1,386.01	5.83	113.00	0.00	0.09	5.33	113.60	0.55	0.11	5.83	113.00	0.00	14.10	5.83	113.00	0.00	5.14
122	3.28	118.00	0.00	1,398.50	3.61	117.60	0.89	0.09	3.28	118.00	0.71	0.10	4.10	117.00	0.00	15.11	4.10	117.00	0.00	5.19
Average	**2.16**	**100.57**	**0.00**	**1,319.97**	**2.49**	**107.04**	**0.49**	**0.09**	**2.43**	**107.10**	**0.49**	**0.10**	**2.83**	**106.65**	**0.00**	**13.44**	**2.83**	**106.65**	**0.00**	**4.42**

Table A-2: The results of experiments with 20 orders

	FCFS				**Greedy**				**GA**				**Meta-heuristic**			
Worst Case [#]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]
219	1.74	215.2	1.48	0.10	2.47	213.6	0.89	0.10	3.65	211.0	0.00	34.49	3.65	211.0	0.00	13.13
213	2.07	208.6	2.07	0.10	3.66	205.2	1.92	0.10	5.16	202.0	0.00	32.75	5.16	202.0	0.00	13.63
261	3.45	252.0	0.71	0.10	3.75	251.2	2.17	0.11	6.36	244.4	0.55	40.95	6.44	244.2	0.45	19.22
173	1.50	170.4	1.34	0.10	2.66	168.4	0.55	0.10	3.47	167.0	0.00	25.45	3.47	167.0	0.00	11.30
211	2.09	206.6	1.95	0.10	4.27	202.0	1.41	0.10	6.16	198.0	0.00	31.20	6.16	198.0	0.00	13.99
221	3.71	212.8	1.92	0.10	4.16	211.8	0.45	0.10	6.24	207.2	0.45	33.30	6.33	207.0	0.00	15.10
172	1.05	170.2	1.30	0.10	1.74	169.0	0.71	0.10	2.91	167.0	0.00	25.77	2.91	167.0	0.00	8.85
215	1.49	211.8	1.10	0.10	3.72	207.0	1.41	0.10	5.12	204.0	0.00	33.10	5.12	204.0	0.00	13.23
184	0.87	182.4	1.14	0.10	2.50	179.4	1.14	0.10	3.26	178.0	0.00	27.43	3.26	178.0	0.00	10.06
200	1.50	197.0	1.58	0.10	2.20	195.6	1.52	0.09	3.50	193.0	0.00	31.08	3.40	193.2	0.45	10.67
232	3.71	223.4	1.67	0.10	3.36	224.2	1.92	0.11	5.60	219.0	0.00	36.62	5.60	219.0	0.00	15.01
164	1.46	161.6	0.89	0.09	1.71	161.2	0.45	0.10	2.44	160.0	0.00	23.84	2.44	160.0	0.00	7.94
236	2.46	230.2	0.84	0.10	3.47	227.8	1.92	0.11	5.08	224.0	0.00	35.78	5.08	224.0	0.00	14.73
214	2.90	207.8	0.45	0.10	3.08	207.4	0.89	0.10	4.67	204.0	0.00	32.26	4.67	204.0	0.00	12.46
218	2.75	212.0	1.41	0.10	3.76	209.8	1.64	0.09	5.96	205.0	0.00	32.20	5.96	205.0	0.00	14.48
183	2.84	177.8	1.48	0.09	2.40	178.6	1.14	0.10	4.92	174.0	0.00	28.12	4.92	174.0	0.00	10.42
182	2.31	177.8	1.48	0.10	3.41	175.8	1.30	0.10	5.49	172.0	0.00	27.72	5.49	172.0	0.00	10.68
172	2.33	168.0	1.87	0.10	3.26	166.4	0.89	0.10	4.07	165.0	0.00	25.33	4.07	165.0	0.00	9.30
173	2.54	168.6	1.14	0.09	4.74	164.8	1.64	0.10	6.94	161.0	0.00	25.19	6.94	161.0	0.00	10.76
179	2.23	175.0	0.71	0.09	2.79	174.0	1.22	0.11	3.91	172.0	0.00	26.13	3.91	172.0	0.00	9.36
Average	**2.25**	**196.5**	**1.33**	**0.10**	**3.16**	**194.7**	**1.26**	**0.10**	**4.75**	**191.4**	**0.05**	**30.44**	**4.75**	**191.4**	**0.04**	**12.22**

Table A-3: The results of experiments with 50 orders

	FCFS				**Greedy**				**GA**				**Meta-heuristic**			
Worst Case [#]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]
534	3.33	516.2	1.30	0.12	5.39	505.2	6.22	0.12	8.95	486.2	1.92	89.21	9.55	483.0	1.22	53.02
477	3.14	462.0	2.83	0.11	5.62	450.2	2.86	0.12	9.18	433.2	2.17	78.19	9.81	430.2	4.21	45.28
475	2.82	461.6	1.34	0.12	6.02	446.4	3.51	0.12	9.73	428.8	1.92	76.85	10.40	425.6	1.14	44.08
532	2.56	518.4	2.70	0.11	4.32	509.0	2.55	0.12	8.16	488.6	3.91	90.78	8.46	487.0	1.87	56.41
555	3.60	535.0	2.00	0.12	5.77	523.0	4.42	0.12	8.50	507.8	2.86	92.78	10.02	499.4	1.82	55.29
505	2.61	491.8	0.84	0.12	6.14	474.0	3.39	0.12	9.43	457.4	0.55	83.04	10.46	452.2	2.17	49.47
502	3.27	485.6	3.21	0.11	4.70	478.4	3.05	0.13	8.29	460.4	2.79	84.40	8.88	457.4	1.95	49.03
508	4.29	486.2	2.49	0.11	6.73	473.8	3.42	0.12	10.28	455.8	2.86	87.38	10.67	453.8	3.03	49.63
592	3.28	572.6	3.36	0.12	5.37	560.2	2.17	0.12	8.45	542.0	0.00	100.32	9.26	537.2	1.92	62.55
528	2.61	514.2	1.48	0.11	5.68	498.0	3.16	0.12	8.98	480.6	1.52	85.91	9.24	479.2	1.92	51.47
595	4.87	566.0	2.00	0.11	6.62	555.6	2.30	0.12	9.85	536.4	2.07	94.91	10.79	530.8	3.27	62.20
533	3.98	511.8	1.30	0.11	5.48	503.8	4.71	0.12	9.53	482.2	1.92	86.16	10.21	478.6	3.51	50.93
535	3.51	516.2	1.48	0.12	5.08	507.8	4.09	0.11	8.52	489.4	2.19	87.66	8.71	488.4	1.52	52.49
573	2.86	556.6	3.65	0.12	5.10	543.8	1.64	0.12	8.80	522.6	2.07	95.08	9.25	520.0	1.22	61.77
600	4.37	573.8	2.17	0.12	5.90	564.6	3.21	0.12	9.50	543.0	1.87	96.89	9.93	540.4	1.14	60.03
491	2.32	479.6	2.19	0.12	4.93	466.8	2.28	0.12	7.86	452.4	1.82	81.01	8.68	448.4	1.95	44.04
566	2.44	552.2	3.35	0.11	4.98	537.8	3.49	0.12	8.76	516.4	2.07	93.31	9.72	511.0	3.16	58.55
471	3.40	455.0	2.00	0.11	5.18	446.6	1.52	0.11	9.47	426.4	2.97	75.69	9.51	426.2	1.10	43.48
556	2.95	539.6	2.70	0.12	4.82	529.2	1.64	0.12	7.63	513.6	2.79	93.56	7.91	512.0	1.22	55.83
558	3.37	539.2	3.35	0.12	5.73	526.0	2.74	0.12	9.18	506.8	3.27	90.41	10.25	500.8	2.39	56.58
Average	**3.28**	**516.7**	**2.29**	**0.11**	**5.48**	**505.0**	**3.12**	**0.12**	**8.95**	**486.5**	**2.18**	**88.18**	**9.59**	**483.1**	**2.09**	**53.11**

Table A-4: The results of experiments with 70 orders

	FCFS				**Greedy**				**GA**				**Meta-heuristic**			
Worst Case [#]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]
745	3.09	722.0	2.55	0.12	5.61	703.2	2.86	0.14	8.40	682.4	1.52	124.09	9.32	675.6	1.82	95.58
711	3.32	687.4	3.65	0.13	5.82	669.6	3.71	0.14	9.03	646.8	2.49	119.02	10.18	638.6	3.36	87.73
706	3.77	679.4	4.51	0.14	6.15	662.6	2.70	0.15	9.89	636.2	2.77	119.05	9.97	635.6	4.56	87.41
705	3.91	677.4	2.19	0.12	5.90	663.4	2.70	0.14	9.59	637.4	3.05	121.16	10.41	631.6	3.29	85.58
796	4.32	761.6	4.67	0.12	6.08	747.6	4.16	0.13	9.20	722.8	2.77	137.26	10.78	710.2	3.03	105.96
676	3.20	654.4	1.14	0.12	6.12	634.6	4.39	0.13	9.85	609.4	3.91	117.10	10.59	604.4	2.70	77.79
707	3.96	679.0	2.12	0.13	6.14	663.6	2.88	0.13	10.01	636.2	2.59	122.05	10.75	631.0	3.39	83.41
765	4.03	734.2	2.77	0.12	6.59	714.6	2.51	0.13	9.91	689.2	1.30	137.55	11.01	680.8	3.77	97.77
794	3.17	768.8	3.83	0.12	6.15	745.2	5.07	0.14	8.99	722.6	3.58	143.88	9.75	716.6	1.82	102.83
698	2.98	677.2	2.59	0.12	5.90	656.8	3.03	0.13	9.48	631.8	2.95	122.12	9.97	628.4	5.03	83.10
787	3.05	763.0	2.55	0.12	5.34	745.0	2.55	0.14	8.23	722.2	4.02	142.75	8.97	716.4	3.85	105.22
738	2.90	716.6	4.28	0.12	4.93	701.6	3.36	0.14	7.72	681.0	1.87	131.62	8.48	675.4	3.51	93.06
768	2.71	747.2	2.59	0.13	5.86	723.0	4.42	0.14	8.75	700.8	1.64	137.69	9.40	695.8	5.67	99.04
768	3.70	739.6	2.61	0.12	5.44	726.2	2.77	0.14	8.96	699.2	1.79	136.25	10.13	690.2	1.48	101.00
726	4.30	694.8	3.35	0.13	6.50	678.8	3.56	0.14	10.00	653.4	1.82	127.02	11.49	642.6	3.85	91.25
722	2.74	702.2	4.02	0.13	6.20	677.2	5.85	0.13	9.83	651.0	2.74	128.65	10.53	646.0	3.61	91.01
724	3.98	695.2	3.03	0.12	5.83	681.8	5.40	0.13	9.17	657.6	3.21	127.32	10.22	650.0	3.24	90.02
688	2.88	668.2	2.86	0.12	5.64	649.2	4.49	0.13	8.49	629.6	2.51	119.85	9.56	622.2	2.49	82.57
773	4.11	741.2	4.44	0.12	6.00	726.6	3.65	0.14	9.29	701.2	3.70	140.33	10.35	693.0	2.92	103.33
840	3.67	809.2	1.48	0.13	5.81	791.2	3.11	0.14	8.86	765.6	3.51	154.72	9.69	758.6	1.14	117.92
Average	**3.49**	**715.9**	**3.06**	**0.13**	**5.90**	**698.1**	**3.66**	**0.14**	**9.18**	**673.8**	**2.69**	**130.47**	**10.08**	**667.2**	**3.23**	**94.08**

Table A-5: the results of the experiments of 2100 orders, using multiple picking stations

	FCFS				**Greedy**				**GA**				**Meta-heuristic**			
Worst Case [#]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]	$\bar{I}$[%]	$\bar{O}$[#]	Std.	$\bar{T}$[s]
21752	4.01	20,880	293.72	1.48	8.66	19,868	16.38	77.68	4.39	20,798	25.35	3,430.88	13.32	18,854	13.54	2,942.61
22285	3.97	21,400	323.74	1.45	8.65	20,356	18.74	92.59	4.52	21,279	4.44	3,536.91	13.24	19,334	24.32	3,117.37
21908	3.28	21,189	15.66	1.40	8.62	20,019	32.84	108.12	4.29	20,967	28.43	3,472.94	13.35	18,984	28.74	3,164.02
22019	4.61	21,005	386.10	1.41	8.79	20,083	18.40	123.41	4.51	21,025	25.03	3,567.07	13.47	19,053	35.50	3,155.47
21826	3.79	20,999	327.69	1.39	8.55	19,960	23.42	109.69	4.40	20,866	10.03	3,531.69	13.34	18,913	20.65	3,112.62
22046	3.38	21,301	23.64	1.39	8.73	20,122	34.93	93.01	4.39	21,077	22.93	3,598.90	13.41	19,090	26.10	3,139.09
21728	3.36	20,997	20.55	1.38	8.69	19,839	36.41	77.80	4.43	20,766	14.47	3,522.63	13.47	18,800	16.30	3,027.40
21842	3.97	20,975	318.53	1.37	8.68	19,947	39.04	93.13	4.34	20,895	18.78	3,611.12	13.36	18,923	8.06	3,019.91
22022	3.35	21,284	18.65	1.39	8.61	20,125	19.19	93.08	4.35	21,064	14.77	3,602.03	13.27	19,100	24.50	3,071.37
21773	4.06	20,889	295.96	1.37	8.84	19,848	19.60	78.35	4.42	20,811	19.38	3,495.49	13.52	18,830	26.47	3,013.42
21918	3.15	21,228	9.79	1.39	8.61	20,031	27.50	110.15	4.39	20,956	14.00	3,483.37	13.30	19,004	9.45	3,057.68
21787	3.42	21,043	11.34	1.38	8.72	19,886	16.84	77.96	4.58	20,789	12.90	3,432.10	13.48	18,851	28.54	3,079.34
22038	5.22	20,887	628.19	1.40	8.66	20,131	33.70	78.17	4.43	21,061	5.81	3,480.22	13.31	19,105	20.28	3,130.24
22293	4.69	21,248	355.96	1.40	8.58	20,380	19.34	78.21	4.52	21,285	30.02	3,550.20	13.30	19,327	20.80	3,124.74
21905	4.24	20,977	329.06	1.41	8.63	20,014	17.49	108.84	4.43	20,935	24.20	3,439.05	13.49	18,951	19.93	3,040.85
22105	3.30	21,375	20.82	1.39	8.56	20,212	26.60	78.34	4.41	21,131	19.84	3,464.41	13.22	19,183	23.48	3,079.00
22139	4.07	21,238	318.86	1.39	8.58	20,239	26.75	78.18	4.47	21,150	20.62	3,465.25	13.36	19,182	12.52	3,119.29
21955	4.58	20,948	634.93	1.39	8.64	20,059	22.10	77.72	4.39	20,990	19.51	3,451.81	13.34	19,027	8.64	3,093.43
21880	4.07	20,988	268.70	1.41	8.77	19,961	27.25	77.60	4.49	20,898	22.60	3,485.96	13.40	18,947	26.88	3,091.59
21353	3.33	20,642	18.25	1.36	8.74	19,487	19.20	93.13	4.42	20,409	14.45	3,348.30	13.50	18,470	33.15	2,982.96
Average	**3.89**	**21,075**	**231.01**	**1.40**	**8.67**	**20,028**	**24.79**	**90.26**	**4.43**	**20,958**	**18.38**	**3,498.52**	**13.37**	**18,996**	**21.39**	**3,078.12**

Bibliography

Azadeh, K., De Koster, R., and Roy, D. (2019). Robotized and automated warehouse systems: Review and recent developments. *Transportation Science*, 53(4):917–945.

Azadnia, A. H., Taheri, S., Ghadimi, P., Mat Saman, M. Z., and Wong, K. Y. (2013). Order batching in warehouses by minimizing total tardiness: A hybrid approach of weighted association rule mining and genetic algorithms. *The Scientific World Journal*, 2013:1–13.

Bartholdi III, J. J. and Hackman, S. T. (2017). *Warehouse & Distribution Science*. Retrieved from: https://www.warehouse-science.com/book/index.html.

Boysen, N., Briskorn, D., and Emde, S. (2017). Parts-to-picker based order processing in a rack-moving mobile robots environment. *European Journal of Operational Research*, 262(2):550–562.

Boysen, N., De Koster, R., and Weidinger, F. (2019). Warehousing in the e-commerce era: A survey. *European Journal of Operational Research*, 277(2):396–411.

Boysen, N., Fedtke, S., and Weidinger, F. (2018). Optimizing automated sorting in warehouses: The minimum order spread sequencing problem. *European Journal of Operational Research*, 270(1):386–400.

Bozer, Y. A. and Kile, J. W. (2008). Order batching in walk-and-pick order picking systems. *International Journal of Production Research*, 46(7):1887–1909.

Chen, T.-l., Cheng, C.-y., Chen, Y.-y., and Chan, L.-k. (2015). An efficient hybrid algorithm for integrated order batching , sequencing and routing problem. *International Journal of Production Economics*, 159:158–167.

De Koster, R., Le-Duc, T., and Roodbergen, K. J. (2007). Design and control of warehouse order picking: A literature review. *European Journal of Operational Research*, 182(2):481–501.

De Koster, R., Roodbergen, K. J., and Van Voorden, R. (1999). Reduction of walking time in the distribution center of de bijenkorf. In *New trends in distribution logistics*, pages 215–234. Springer.

Delahaye, D., Chaimatanan, S., and Mongeau, M. (2019). Simulated annealing: From basics to applications. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 1–35. Springer International Publishing, Cham.

Dueck, G. and Scheuer, T. (1990). Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90(1):161–175.

Eglese, R. W. (1990). Simulated annealing: A tool for operational research. *European Journal of Operational Research*, 46(3):271–281.

Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., and Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175.

Füßler, D. and Boysen, N. (2017). Efficient order processing in an inverse order picking system. *Computers & Operations Research*, 88:150–160.

Füßler, D. and Boysen, N. (2019). High-performance order processing in picking workstations. *EURO Journal on Transportation and Logistics*, 8(1):65–90.

García-Martínez, C., Rodriguez, F. J., and Lozano, M. (2018). Genetic algorithms. In Martí, R., Pardalos, P. M., and Resende, M. G. C., editors, *Handbook of Heuristics*, pages 431–464. Springer International Publishing, Cham.

Gilli, M., Maringer, D., and Schumann, E. (2011). Chapter twelve - heuristic methods in a nutshell. In Gilli, M., Maringer, D., and Schumann, E., editors, *Numerical Methods and Optimization in Finance*, pages 337 – 379. Academic Press, San Diego.

Gu, J., Goetschalckx, M., and Mcginnis, L. F. (2007). Research on warehouse operation: A comprehensive review. *European Journal of Operational Research*, 177:1–21.

Gu, J., Goetschalckx, M., and Mcginnis, L. F. (2010). Research on warehouse design and performance evaluation : A comprehensive review. *European Journal of Operational Research*, 203(3):539–549.

Gurobi Optimization, L. (2020). Gurobi optimizer reference manual.

Henn, S. (2012). Algorithms for on-line order batching in an order picking warehouse. *Computers and Operations Research*, 39(11):2549–2563.

Henn, S. (2015). Order batching and sequencing for the minimization of the total tardiness in picker-to-part warehouses. *Flexible Services and Manufacturing Journal*, 27(1):86–114.

Henn, S., Koch, S., Doerner, K. F., and Strauss, C. (2010). Metaheuristics for the order batching problem in manual order picking systems. *Business Research*, 3(1):82–105.

Henn, S. and Schmid, V. (2013). Metaheuristics for order batching and sequencing in manual order picking systems. *Computers and Industrial Engineering*, 66(2):338–351.

Henn, S. and Wäscher, G. (2012). Tabu search heuristics for the order batching problem in manual order picking systems. *European Journal of Operational Research*, 222(3):484–494.

Holland, J. H. (1975). Adaptation in natural and artificial systems ann arbor. *The University of Michigan Press*.

Hsu, C.-M., Chen, K.-Y., and Chen, M.-C. (2005). Batching orders in warehouses by minimizing travel distance with genetic algorithms. *Computers in Industry*, 56(2):169–178.

Koch, S. and Wäscher, G. (2016). A grouping genetic algorithm for the order batching problem in distribution warehouses. *Journal of Business Economics*, 86(1-2):131–153.

Kulak, O., Sahin, Y., and 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(1):52–80.

Matusiak, M., De Koster, R., Kroon, L., and Saarinen, J. (2014). A fast simulated annealing method for batching precedence-constrained customer orders in a warehouse. *European Journal of Operational Research*, 236(3):968–977.

Ratliff, H. D. and Rosenthal, A. S. (1983). Order-picking in a rectangular warehouse: A solvable case of the traveling salesman problem. *Operations Research*, 31(3):507–521.

Roodbergen, K. J. and Vis, I. F. A. (2009). A survey of literature on automated storage and retrieval systems. *European Journal of Operational Research*, 194(2):343–362.

Scholz, A., Schubert, D., and 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 Operational Research*, 263:461–478.

Tappia, E., Marchet, G., Melacini, M., and Perotti, S. (2015). Incorporating the environmental dimension in the assessment of automated warehouses. *Production Planning and Control*, 26(10):824–838.

Tappia, E., Roy, D., Melacini, M., and De Koster, R. (2019). Integrated storage-order picking systems: Technology, performance models, and design insights. *European Journal of Operational Research*, 274(3):947–965.

Tompkins, J. A., White, J., Bozer, Y., Tanchoco, J., and Trevino, J. (2003). Facilities planning.

Tsai, C. Y., Liou, J. J. H., and Huang, T. M. (2008). Using a multiple-ga method to solve the batch picking problem: Considering travel distance and order due time. *International Journal of Production Research*, 46(22):6533–6555.

Whitley, D. (2019). Next generation genetic algorithms: A user's guide and tutorial. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, pages 245–274. Springer International Publishing, Cham.

Zhang, J., Wang, X., and Huang, K. (2016). Integrated on-line scheduling of order batching and delivery under b2c e-commerce. *Computers and Industrial Engineering*, 94:280–289.

Žulj, I., Kramer, S., and Schneider, M. (2018). A hybrid of adaptive large neighborhood search and tabu search for the order-batching problem. *European Journal of Operational Research*, 264:653–664.

Glossary

List of Acronyms

AS/RS	Automated Storage/Retrieval System
FCFS	First-Come-First-Serve
GA	Genetic Algorithm
GTP	Goods-To-Picker
MILP	Mixed-Integer Linear Programming
PTG	Picker-To-Goods
SBS/RS	Shuttle-Based Storage/Retrieval System
SKU	Stock Keeping Unit