

Lars Eirik Bergland
Isak Rathe Støre
Stian Westerhus Vale

Solving the Storage Location Assignment Problem for a Pick-and- Pass Warehouse

A simulation-based genetic algorithm approach

Master's thesis in Industrial Economics and Technology
Management
Supervisor: Magnus Stålhane
June 2019



Lars Eirik Bergland
Isak Rathe Støre
Stian Westerhus Vale

Solving the Storage Location Assignment Problem for a Pick-and-Pass Warehouse

A simulation-based genetic algorithm approach

Master's thesis in Industrial Economics and Technology Management
Supervisor: Magnus Stålhane
June 2019

Norwegian University of Science and Technology
Faculty of Economics and Management
Department of Industrial Economics and Technology Management

Preface

This master's thesis completes our Master of Science at the Norwegian University of Science and Technology(NTNU). The work done on this thesis was conducted during the spring of 2019 as a continuation of the Specialization Project TIØ4500 Managerial Economics and Operations Research during the fall of 2018. The thesis is part of the specialization Applied Economics and Operation Research at the Department of Industrial Economics and Technology Management. The problem studied was done in collaboration with Kolonial.no, and we are thankful to Sveinung Hartberg, Anders Hennum, and the rest of the team at Kolonial.no for their support. We would also like to thank Professor Keith L. Downing for contributing with his knowledge on genetic algorithms. Lastly, the authors would like to thank supervisor and Associate Professor Magnus Stålhane for his valuable guidance.

Trondheim, June 11th 2019


Lars Eirik Bergland


Isak Rathe Støre


Stian Westerhus Vale

Abstract

In this thesis, the Storage Location Assignment Problem for the Norwegian e-commerce grocery company Kolonial.no is considered. The problem is concerned with assigning goods into a storage space in order to improve order-picking efficiency. Kolonial.no uses a pick-and-pass layout; instead of having pickers move around in a warehouse to collect goods, they are stationed along an assembly line. Boxes move on a main conveyor belt, and pickers fill them with goods according to customer orders. The goods are stored along the assembly line and split into zones called stations. If a box requests goods from a station, it leaves the main conveyor belt and enters the station's buffer zone. This is a separate conveyor belt for each station that allows several boxes to wait at a station without affecting the flow on the main conveyor belt. The goal of this thesis is to find the assignment of the goods that minimizes the makespan of processing a given set of boxes. The makespan is the time it takes from the first box enters the assembly line to the last box exits the assembly line.

In the literature, extensive work has been done on the Storage Location Assignment Problem. It can be solved with regards to different performance measures and for different warehouse layouts. Additionally, several solution methods have been studied. Because of the NP-hard complexity of the problem, most solution methods are heuristics. In this thesis, we look into minimizing the makespan for a pick-and-pass layout by using a heuristic and a metaheuristic approach.

The work done in this thesis is a continuation of previous work by Bergland et al. (2018). Here, we presented a precise MIP-formulation of the problem. However, we were unable to solve the model for the real-sized instance of Kolonial.no. Consequently, two new solution methods are proposed in this thesis, which manages to solve the full instance. A simple greedy algorithm is able to solve the problem in 0.04 seconds. By simulation, the solution is compared to Kolonial.no's current goods assignment. Results show an improvement of 6.0% on a known set of boxes. The main contribution of this thesis, however, is a simulation-based genetic algorithm. This is a genetic algorithm that uses simulation to evaluate the chromosomes. To the best of the author's knowledge, this approach has never been used to solve the Storage Location Assignment Problem. The algorithm is computationally expensive, but the results are promising. An improvement of 14.9% from the current solution is achieved on a known set of boxes. These results indicate that the proposed genetic algorithm significantly outperforms today's solution.

Sammendrag

I denne oppgaven undersøkes Storage Location Assignment Problem for den norske nettbaserte matbutikken Kolonial.no. Problemet handler om å plassere varer på lagerplasseringer slik at plukking av varer effektiviseres. Kolonial.no har et 'pick-and-pass'-oppsett på lageret sitt: I stedet for at plukkere går rundt i lageret for å plukke varer, er de i stedet stasjonert langs et samleband. Bokser beveger seg langs et hovedband hvor plukkere fyller dem med varer i henhold til kundeordre. Varer plasseres langs samlebandet, og er fordelt inn i soner kalt stasjoner. Hvis en boks skal ha varer som finnes på en stasjon, forlater den hovedbandet og går over på stasjonens buffersone. Dette er et separat band for hver stasjon, hvor flere bokser kan vente uten at det påvirker flyten langs hovedbandet. Målet med denne oppgaven er å finne en vareplassering som minimerer totaltiden det tar å prosessere en gitt mengde bokser.

I tidligere litteratur har omfattende arbeid blitt gjort på Storage Location Assignment Problem. Problemet kan løses med hensyn til forskjellige objektiver og ulike lageroppsett. I tillegg har det blitt brukt flere ulike løsningsmetoder. På grunn av NP-hard-kompleksiteten til problemet, er de fleste løsningsmetodene heuristikker. I denne oppgaven minimerer vi totaltiden for et 'pick-and-pass'-oppsett ved å bruke en heuristikk og en metaheuristikk.

Arbeidet med denne oppgaven er en fortsettelse av Bergland et al. (2018). Her presenterte vi en nøyaktig MIP-formulering av problemet. Vi fikk imidlertid ikke løst modellen for størrelsesordenen som Kolonial.no opererer med. Derfor blir to nye løsningsmetoder presentert i denne oppgaven, hvor begge klarer å løse problemet med den riktige størrelsesordenen. En enkel grådig algoritme er i stand til å løse problemet på 0.04 sekunder. Ved å simulere løsningen, kan den sammenliknes med Kolonial.nos nåværende vareplassering. Resultatene viser en forbedring på 6.0% for en kjent mengde bokser. Hovedbidraget i denne oppgaven er imidlertid en simulator-basert genetisk algoritme. Denne bruker simulering til å evaluere kromosomer. En slik metode har aldri, til vår kjennskap, blitt brukt for å løse Storage Location Assignment Problem. Algoritmen er beregningstung, men finner lovende løsninger. På en kjent mengde bokser finner den en vareplassering som er 14.9% bedre enn Kolonial.nos nåværende vareplassering. Disse resultatene indikerer at den genetiske algoritmen kan forbedre dagens vareplassering betydelig.

Table of Contents

1	Introduction	1
2	Industry and Company Insight	3
2.1	Market Overview	3
2.2	Company overview	4
2.3	Supply Chain	5
2.4	In summary	7
3	Problem Description	10
4	Literature Review	12
4.1	Storage Location Assignment Problem	12
4.2	Our contribution	19
5	Assembly Line Routing Model	22
5.1	Assumptions	22
5.2	Notation	24
5.3	Objective function	25
5.4	Constraints	25
6	Genetic algorithm with an adaptive penalty for the SLAP	30
6.1	Overview	30
6.2	Chromosome Representation	31
6.3	Evaluation of chromosomes	32
6.4	Termination criteria	33
6.5	Parent Selection	33
6.6	Crossover	34
6.7	Mutation	35
6.8	Algorithm Complexity	36
7	Simulator	37
7.1	Simulator properties	37
7.2	The Simulator Algorithm	38
7.3	Algorithm Complexity	43
7.4	Walkthrough Example	44

8 Greedy approach to the SLAP	46
8.1 Algorithm Complexity	47
9 Computational Study	48
9.1 Description of the provided data	49
9.2 Data split into Opt Data and Unseen Data	50
9.3 Estimation of simulator parameters	51
9.4 Calibration of genetic algorithm parameters	52
9.5 Significance of the instance size	63
9.6 Performance of the algorithms	64
9.7 Comparing Solutions	66
9.8 Uniqueness of the best solution	72
9.9 The value of shorter processing times	74
9.10 Further improvements	74
10 Conclusion	77
Appendix A	84
A.1 Assembly Line Routing Model	84
Appendix B	88
B.1 Additional plots	88
Appendix C	92
C.1 Run summary	92

List of Tables

4.1	Comparative overview of related literature	20
7.1	Event attributes	39
9.1	Hardware and software specifications	48
9.2	Files provided by Kolonial.no	49
9.3	Provided picking times	51
9.4	Max picking times	51
9.5	Parameter notation	53
9.6	Preliminary test results	56
9.7	Calibration of population size	57
9.8	Calibration of mutation rate and number of swaps	59
9.9	Calibration of crossover rate	59
9.10	Calibration of crossover operator	60
9.11	Calibration of generation scaling	61
9.12	Makespan time tuned on different input sizes	63
9.13	Performance of the algorithms	64
9.14	Location of the most frequent goods	71
9.15	Comparison with respect to similarity	73
9.16	Symmetry of a solution	73
9.17	Goods at multiple stations	75
9.18	Fewer pickers at work	76
C.1	List of all calibration testing	92

List of Figures

2.1	Kolonial.no's annual revenue and profits	4
2.2	Facility layout for Kolonial.no	6
2.3	Workstations along the assembly line	8
2.4	Assembly line entry/exit and box overtaking	8
2.5	Details at a workstation	9
3.1	Assembly line for dry picking at Kolonial.no	10
4.1	Storage Layouts	14
4.2	Pick-and-pass warehouse layout	15
4.3	Solution methods	16
4.4	Metaheuristic methods	17
5.1	Description of node sets	22
5.2	Description of node numbering	23
6.1	Genetic algorithm flowchart	31
6.2	Representation of a chromosome	32
6.3	An illustrative example of how Tournament Selection works.	34
6.4	An illustrative example of how One-point Crossover works.	35
6.5	An illustrative example of how Uniform Crossover works.	35
6.6	Tournament Selection	36
7.1	Simulator GUI	37
7.2	Flowchart of simulator	38
7.3	Flowchart of Box-Arrived event	40
7.4	Flowchart of Box-Ready event	41
7.5	Flowchart of Node-Empty event	42
7.6	Walkthrough part 1	44
7.7	Walkthrough part 2	45
9.1	Number of goods and boxes per date	49
9.2	Good frequency	50
9.3	Picking time distribution	52
9.4	Makespan times for Kolonial's goods assignment	54
9.5	Number of goods in boxes	55

9.6	Performance of different population sizes	58
9.7	Generation scaling penalization	61
9.8	Considering termination threshold	62
9.9	Elite relative computational time	65
9.10	Elite relative to generations	65
9.11	Picking time per station, training	66
9.12	Picking time per station, training	67
9.13	Picking time per station, test	68
9.14	Queue and waiting time	70
9.15	Cumulative good frequency	71
B.1	Mean makespan through generations	88
B.2	Hamming distance	89
B.3	Max - Min gap	90
B.4	Number of goods at each station	90
B.5	Walkthrough part 3	91

Introduction

"Groceries online will never pay off. As a matter of fact, this is absolutely impossible." These are the words of e-commerce entrepreneur Einar Brandsdal, translated into English (Weldeghebriel, 2018). He is referring to the fact that in an e-commerce grocery store, pickers are paid to do the picking of goods that is usually done by the customers. Besides, products in this segment typically have high weights and volumes relative to profit margins compared to other product segments. There are also expiration dates to consider. Getting the cost per pick down to sustainable levels is then impossible, Brandsdal claims.

Kolonial.no is a Norwegian e-commerce grocery company that aims to prove this statement invalid. Succeeding in taking groceries into e-commerce is challenging, but possible. This requires cost-effective warehouse processes. The costs of operations in their warehouse facility are about 2/3 of their total operational costs, excluding the cost of goods. To increase effectiveness and cut operational costs, they continuously work on improving every element of their supply chain. In 2018, they developed a semi-automated assembly line to streamline the processing of customer orders. Instead of having warehouse operators move through a storage facility and collect goods for an order, the warehouse operators are stationed along the assembly line. Boxes move on a conveyor belt and are to be filled with goods according to customer orders. The goods are stored along the assembly line and split into zones called stations. There is one warehouse operator at each station, and they are responsible for picking the goods located at their station. A box only stops at a station if the station stores goods that are in the customer order of the box. Because almost no box contains the same combination of goods, the movement of the boxes is unpredictable. If one station is overloaded with work, this creates a bottleneck that affects the flow along the assembly line.

Kolonial.no's objective is to process customer orders as quickly as possible. They believe that the storage location of each good has a significant impact on how fast the customer orders may be processed. This belief is based on the fact that the good locations determine the workload of each station and the flow along the assembly line. Therefore, the purpose of this thesis is to find a goods assignment that is considerably better than the one Kolonial.no uses today. This is measured in terms of the change

in makespan of processing a set of boxes.

In the literature, the problem of Kolonial.no is called the Storage Location Assignment Problem. This problem has been studied for several warehouse layouts and performance measures. The layout Kolonial.no use is called a pick-and-pass-system. With regards to performance measure, Kolonial.no want to minimize the makespan of processing a set of boxes. The makespan is the time it takes from the first box enters the assembly line to the last box exits the assembly line. With regards to solution methods, multiple have been used for the Storage Location Assignment Problem in previous literature. This includes exact methods, heuristic methods, and metaheuristic methods, all of which are used in this thesis.

This thesis is a continuation of the work done by Bergland et al. (2018). Here, a Mixed-Integer-Programming model was formulated and implemented. It was solved with a branch-and-bound approach, but only for less than 0.05% of the real-sized instance. This solution method was therefore considered unfit for the real-sized instance of Kolonial.no. Therefore, we look more into heuristic and metaheuristic approaches in this thesis. Although these methods do not guarantee optimality, they aim to find solutions that are considerably better than Kolonial.no's current solution. A simple greedy algorithm is created to have a basis for comparison. The primary solution method of this thesis, however, is a simulation-based genetic algorithm. Instead of using a mathematical expression as the fitness function, the fitness is computed by running the chromosome through a simulation. This is a solution method that, to the best of the authors' knowledge, never has been used to solve this problem before. This thesis may, therefore, be an essential contribution to existing literature.

The outline of this thesis is as follows. Chapter 2 provides an insight into Kolonial.no and their industry. Chapter 3 formally describes the problem studied in this thesis and its respective decision variables. Related literature is reviewed in Chapter 4. This chapter also includes our contribution to the existing literature. Chapter 5 presents a mathematical integer-programming model of the problem. In Chapter 6, the genetic algorithm solution method is presented. Next, Chapter 7 presents the simulator that is used to evaluate goods assignments. Chapter 8 describes the greedy algorithm. The performance of the solution methods is assessed in Chapter 9. Finally, Chapter 10 is a conclusion of the thesis with suggestions for further research.

Industry and Company Insight

This chapter provides industry and company insights about our collaboration partner, Kolonial.no AS (hereafter referred to as Kolonial.no), specifically their operating market and their value chain. This chapter aims to set the context of the problem. It is necessary to give an idea of why Kolonial.no can significantly benefit from warehouse optimization to reduce costs and increase service levels. If no source is cited, the information about Kolonial.no is obtained directly from internal sources within the company.

2.1 Market Overview

Since the e-commerce grocery market is a submarket of the grocery market, it is important to look at the e-commerce market in light of the market for traditional groceries. Adjusted for inflation, the Norwegian grocery market has stagnated in recent years (Myrset et al., 2016), and the market is considered to be in a mature state and is sensitive to product pricing. The grocery market consists of several players, but is dominated by COOP Norge SA, NorgesGruppen ASA, and Reitangruppen AS, which have a total market share of 96,2% (Wifstad et al., 2018). The rest of the market is owned by smaller players such as Kolonial.no, Bunnpris AS and Adams Matkasse AS. Kolonial.no's market share in the traditional grocery market is only between 0.5-0.8%, and the total grocery market was in 2017 171.3 billion NOK. Big international companies like Lidl have tried to establish operations in Norway. However, due to the strong competition in the Norwegian grocery market, the company failed to make profits and shut down their operations in the country back in 2008 (Hagen, 2008). Considerable entry barriers are making it hard for new companies to establish a profitable operation. Economies of scale allow the bigger players in the market to negotiate better purchasing agreements from wholesalers. (Skaar, 2017).

Over the last years, several e-commerce companies have entered the market and tried to make a profit on selling and distributing groceries from an online platform. Marked.no AS and Brødboksen Twd AS are companies that went bankrupt due to considerable operational costs. The meal kit delivery services Adams Matkasse and Godtlevant.no merged in 2017 because they were too small against the more prominent players to operate separately in the market (Valvik, 2017). Just like small players in the traditional

grocery market, the e-commerce companies also struggle to get affordable purchasing agreements from wholesalers and lack the benefits of economies of scale (Hogseth and Lorch-Falch, 2018), (Sundberg and Wig, 2018). In order for e-commerce on groceries to succeed, they need both scale and customer location density to keep distribution costs low. Both of these conditions are challenging to achieve for smaller companies. A report conducted by the consulting company Qvartz indicates that an e-commerce grocery company operating in Denmark would go breakeven if it could reach revenues of around 2 billion DKK. This research could be comparable to the corresponding Norwegian market (Hopland, 2019). In 2017, Kolonial.no reported 801 million NOK in revenues (Proff.no, 2017).

As a large actor in the e-commerce grocery market, Kolonial.no has a major market ownership. The e-commerce market grew 20 % in 2018, and groceries are the largest growing product segment in this market (Høgseth, 2018). Because of a growing trend in e-commerce, the service of selling and delivering groceries from an online platform faces high customer demand (Lorch-Falch, 2018). Companies like Kolonial.no struggle to increase operational capacity to meet necessary service levels.

2.2 Company overview

Kolonial.no is the biggest e-commerce company for groceries in Norway. As of the first quarter of 2019, they had around 500 employees, 20000 weekly customers and yearly revenues of just under one billion NOK. Figure 2.1 shows how the revenues have increased in the last years. Since the beginning of 2013, Kolonial.no has never had any profits. In addition, the negative profits have increased. Kolonial.no believes that the key to positive profits in the future lies in increasing volumes and cost-effective production processes. To increase production capacity and lower the cost per customer order, Kolonial.no developed a semi-automatic assembly line in the summer of 2018. The company is continuously working on cutting operational costs and hopes to make many of the warehouse components fully automatic in the future.

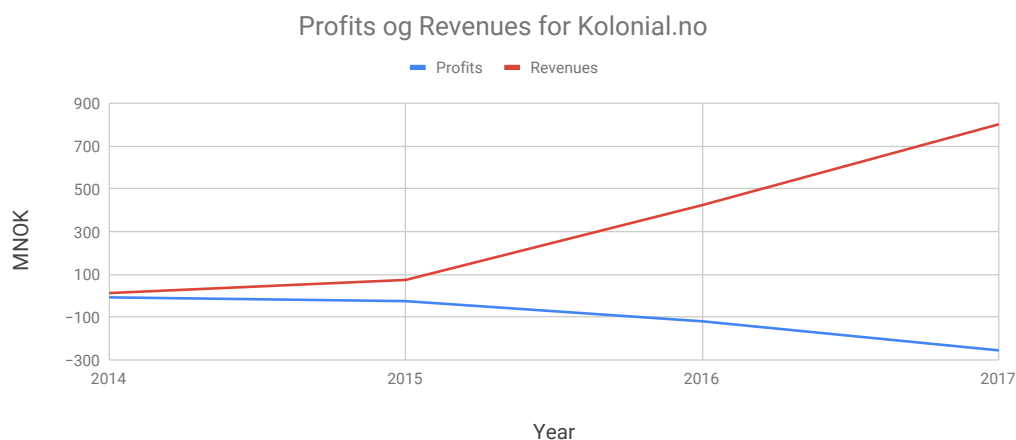


Figure 2.1: The plots shows Kolonial.no's annual revenues and profits since 2014. Data was only available up to 2017.

The company operates solely in Eastern Norway as of 2019. Oslo and Akershus county are the areas of operation that have the highest customer densities. The company seeks to expand their operations to other big cities in Norway over the next years if they manage to achieve profitability in the current domain.

2.3 Supply Chain

This section presents the supply chain for Kolonial.no. The process from the ordering of goods on the company's website to end delivery is explained in detail.

2.3.1 Customer website

Customers order goods from Kolonial.no's own website. Every night at 01:00, the order window closes for delivery the same day. When customers order goods, the cost of delivery comes in addition. This is based on the distance from the main warehouse, delivery time and date. The customer also selects the time window in which they want to receive the goods. Because all browsing and shopping are done through the website, Kolonial.no can gather important data. Big data on customer behavior are of high value to the company. It can be utilized to create customer-tailored product representation and marketing, detect market trends, and precise real-time product demand.

2.3.2 Wholesalers

The company receives its inventory from different wholesalers, and this is delivered to the main warehouse at Lørenskog, Norway. The wholesalers have lead time and shifting delivery dates/times at the main warehouse. Receiving stocks needs to be handled and placed on respective shelves. This creates an irregular workload in the production facility. Kolonial.no has its own bakery in the warehouse, producing baked goods to be delivered every day. Net fresh is a fruit and vegetable wholesaler who, similarly to the bakery, has its wholesaler warehouse and operations within Kolonial.no's main facility.

2.3.3 Restock and Production

The warehouse is divided into three production zones: dry, cold and freeze. Goods are located according to their respective temperature requirements. In these three zones, the customer orders are being assembled, and each has its own restock section, see Figure 2.2. Each zone has its own assembly line. Along each of the three assembly lines, the goods are stored on respective shelves for each station. Customer demand varies; thus, minor changes to the goods assignments are done continuously for improved efficiency. Seasonal changes in demand cause perhaps the biggest changes to the assignment of the goods on the assembly line. During the assembly of customer orders, the goods are being picked from the shelves and placed in NLP-boxes that move along the assembly line. NLP-boxes are industry standard in grocery logistics and have dimensions suited for euro pallets. The boxes are preassigned the goods it needs before it enters the assembly line. One box contains goods from only one customer, but if a

customer orders more goods than one box can carry, several boxes are assigned to that customer. The assignment of goods to boxes, and the order in which the boxes are fed to the assembly line, is done by a software developed by the company. To prevent stockout in the shelves, workers from the restock section fill the respective shelves continuously. Some of the inventory is temporarily stored in the restock section to be restocked at a later time. The dry and cold departments are significantly larger than freeze, and these zones have semi-automatic production systems. Products ordered by customers are assembled in NLP-boxes that move through the assembly line. NLP-boxes are industry-standard storage boxes that carry the plastic bags with orders from production and all the way to the end customer.

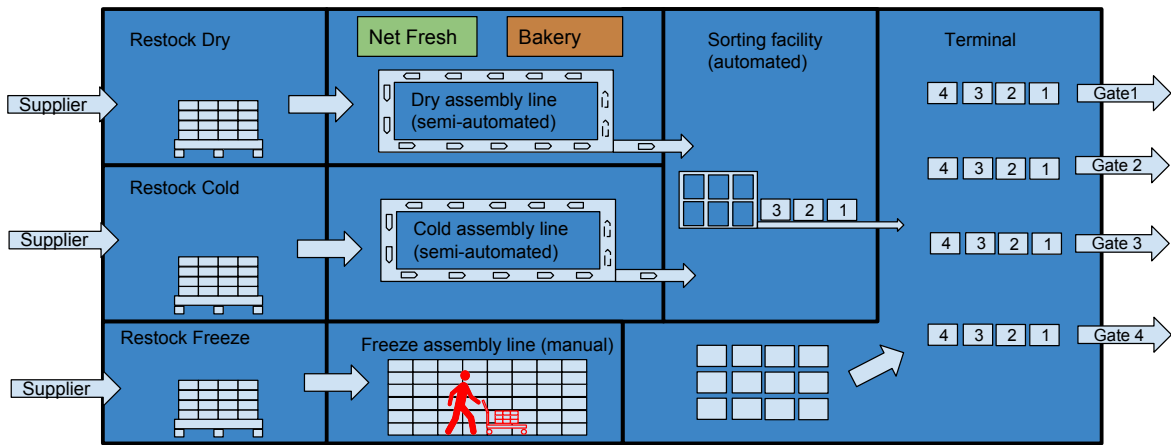


Figure 2.2: This figure depicts the layout of Kolonial.no's warehouse facility. All departments and the sequence in which they function are illustrated.

The semi-automatic assembly line is based on a pick-and-pass warehousing system, see figures 2.3, 2.4 and 2.5. A Pick-and-pass warehousing system is a material handling concept where boxes/containers travel to pick locations and receives the items needed by a person. The system is well suited to assemble small to medium-sized stock keeping units. The system's popularity in warehousing has grown much in the last decades due to the growth of e-commerce and the operational need for cost-effective production lines (Chen and Wu, 2005) (Petersen and Gerald, 2002). Chapter 3 provides more details on the assembly line process.

One weakness of such a system is that every station has to be operated for the whole system to work. If a shortage in labor or demand, one can not scale down operations by dropping stations. The warehouse workers still have to pick goods to boxes on all stations. If a station has many high demand goods and is not processed quickly enough, it serves as a bottleneck for the entire assembly line. This poses a system reliability threat. If the hardware on one station stops working, the whole assembly line is affected.

As of the first quarter of 2019, the goods along the assembly line for all temperature zones are assigned based on a greedy algorithm. Since the start of the semi-automatic

assembly lines, minor changes to the greedy solution is being made on a daily basis. The reason for those changes varies. Some changes are because of empirical testing and reasonable restrictions. Seasonal variations in demand for certain goods also require adaption on a timely basis.

2.3.4 Sorting facility and outgoing terminal

After the dry and cold assembly lines, orders move on to the sorting facility. Here, the orders are sorted in the correct order according to delivery by an automated sorting system. Then, they move on to the outgoing terminal. The orders from the freeze assembly line are being sorted manually and integrated with the order from the sorting facility in the terminal. After being sorted, the set of orders for each vehicle is placed in queue in front of the terminal docks. The orders are placed in a LIFO order - Last In, First Out.

2.3.5 Distribution

The last stage of the supply chain is distribution. Kolonial.no Distribusjon AS is a company owned by Kolonial.no. It is a distribution company with Kolonial.no as its only customer. Their vehicle fleet consists of approximately 50 vans, and distribute the orders directly to the customer or to their nearest collection point where they may collect it themselves. The company uses a third party distribution service to handle high volume fluctuations. The third party uses semi-trailers to distribute larger quantities of orders to intermediate storage facilities.

2.4 In summary

Kolonial.no is operating in a very price sensitive and competitive market. Their need to scale operations and make them more cost effective is imperative for the company's success. As a step towards more cost-effective production, Kolonial.no recently developed their own semi-automatic assembly lines. Their production facility contains three temperature zones, dry, cold and freeze. Dry and cold uses the newly developed semi-automatic system while the freeze assembly line is fully manual. Assembled customer orders go through a sorting facility before ending up at the outgoing terminal. At the terminal, goods are being distributed to the end customers. This thesis takes on a complex problem on the assembly line in the dry zone. The problem is presented in the next chapter.



Figure 2.3: *This figure shows the stations along the assembly line in the dry stock section of the warehouse. There are 13 stations down this aisle. The belt furthest on the right is the main line. The belt second most to the right is the box positions for the stations.*

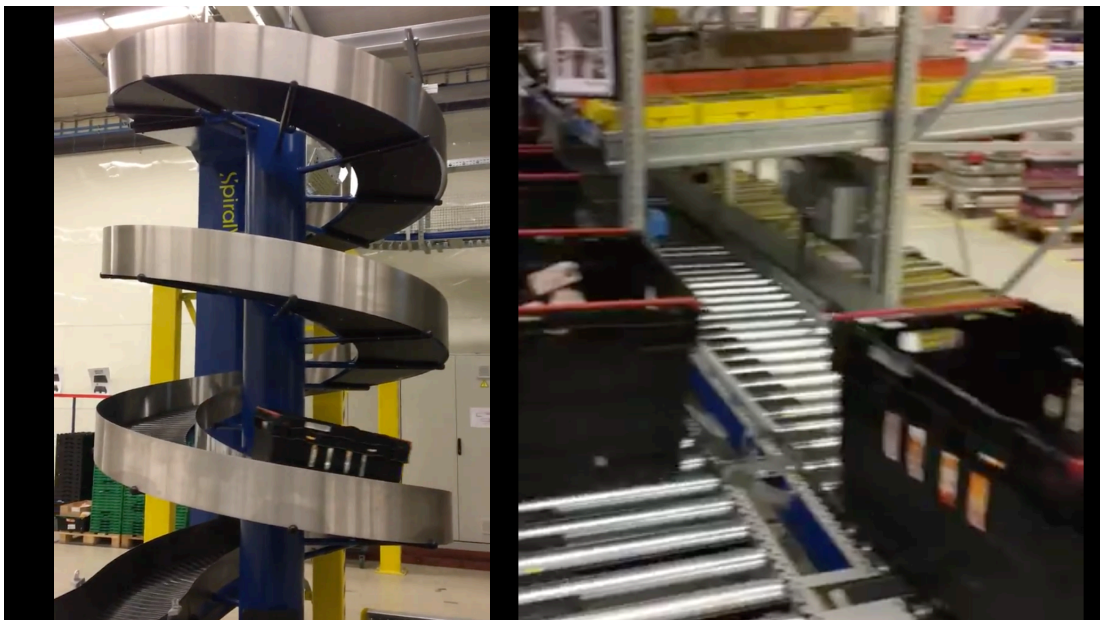


Figure 2.4: *On the left, it is displayed how the boxes are fed onto the assembly line. They are similarly fed off the assembly line. On the right, a box is overtaking some boxes at the adjacent station.*



Figure 2.5: *The picture on the left shows a station from the worker's perspective. The iPad displays what to be picked and the quantity. In the picture on the right, one of the authors epitomizes work in progress.*

Problem Description

This chapter presents the problem description for Kolonial.no's storage assignment problem. Figure 3.1 illustrates the assembly line. The red squares illustrate storing locations where goods are stored. The storing locations are separated into a set of zones called stations. There is a given set of goods, and each good is placed at a station in its distribution unit. The distribution units have a given width. There is a maximum width capacity at each station. It is assumed that stock-outs does not happen. At each station, there is one warehouse operator, hereafter referred to as a picker. These are illustrated with yellow in the figure. It is assumed that the pickers work at a constant speed with no interruptions. The time a picker uses to pick a good, is dependent on the good and the quantity requested.

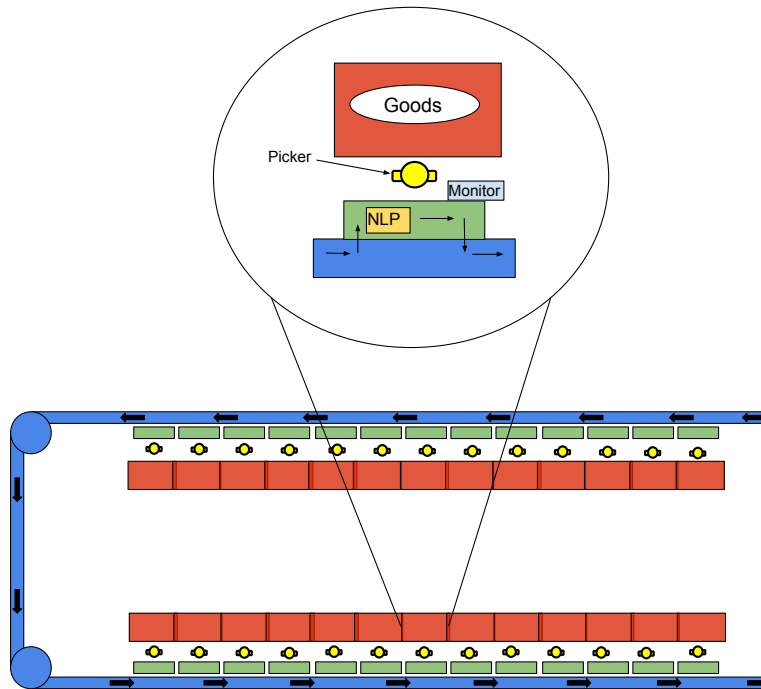


Figure 3.1: The assembly line for good picking. The main line is coloured in blue. The buffer zones are green. The stations' storing locations are marked in red. The pickers are yellow.

There is one main conveyor belt that passes all the stations. This is marked with blue in the figure and is referred to as the main line. The main line carries a given set of boxes through the assembly line. The sequence in which the boxes enter the main line is given. A new box enters the start of the main line as soon as there is available space. Each box is linked with a customer order, which specifies the goods and the quantity to be placed in the box. A box only visits a station if the station stores goods that are in the customer order of the box. If a box is visiting a station, it is pushed off the main line and onto a buffer zone which is a separate conveyor belt for the current station. Here, the box waits to be processed by the picker. A station's buffer zone is decoupled from the main line. Therefore, several boxes can wait at a station without causing a queue on the main line. However, each buffer zone has a maximum capacity of boxes. If a box is going to be pushed to a certain station and the station's buffer zone is full, the box has to wait on the main line. This may cause a queue on the main line. The buffer zones are illustrated with green in the figure.

At a station, a box is ready to be processed by a picker when it is the first box in the buffer zone queue. Then, a monitor shows which goods the box needs from that station. The picker then processes the box by placing the requested goods in the box. After a box is processed at a station, it is pushed back onto the main line. The boxes on the main line have a duty to yield for boxes coming from a station. When a box arrives at the end of the main line, it contains all the requested goods.

Kolonial.no's goal is to minimize the time it takes from the first box enters the assembly line to the last box exits the assembly line. This measure is hereafter referred to as the makespan of the boxes. Many operational improvements can be made to achieve lower makespan. In this thesis, however, we focus solely on the decision of storage location for each good. All goods are to be placed, and they should be placed at exactly one station.

Literature Review

This chapter presents a thorough review of the existing literature for the studied problem in this thesis. The problem is an instance of the Storage Location Assignment Problem, hereafter referred to as SLAP. SLAP is reviewed in Section 4.1. First, the formal definition of SLAP is presented. Then, the literature is reviewed categorically concerning performance measures, storage layouts, industrial sectors, and solution methods. Lastly, our contribution is presented in Section 4.2.

4.1 Storage Location Assignment Problem

In Gu et al. (2007) SLAP is formally defined as follows:

Given:

1. Information on the storage area, including its physical configuration and storage layout.
2. Information on the storage locations, including their availability, physical dimensions, and location.
3. Information on the set of items to be stored, including their physical dimensions, demand, quantity, arrival and departure times.

Determine:

The physical location where arriving items will be stored.

Subject to performance criteria and constraints such as:

1. Storage capacity and efficiency.
2. Picker capacity and efficiency based on the picker cycle time.
3. Response time.
4. Compatibility between products and storage locations and the compatibility between products.

5. Item retrieval policy such as FIFO (first-in, first-out), LIFO (last-in, first-out), BFIFO (batch first-in, first-out). When using the BFIFO policy, items that arrived in the same replenishment batch are considered to be equivalent.

In summary, SLAP is concerned with assigning goods into storage spaces in order to optimize a performance measure, e.g., minimize picking time or maximize operational efficiency.

To the best of the authors' knowledge, the first usage of the term 'SLAP' was in Berg and Zijm (1999). There is, however, much relevant literature preceding this, without referring to the problem as SLAP. Additionally, it is worth mentioning that the literature uses different terms to refer to SLAP. Reyes et al. (2018) did a review of SLAP-literature between 2005 and 2018 and found that the terms *storage assignment* and *storage location assignment* made up for 81% of the reviewed papers. However, some papers used *product allocation/location*, *storage location/space*, *reserve/space allocation* or *slotting* to refer to SLAP.

4.1.1 Performance Measures

There are variations of SLAP concerned with different performance measures. Reyes et al. (2018) categorize them into Operational efficiency, Human factors, Space and distance, Handling costs, Time and Infrastructure. Franzke et al. (2016) are concerned with throughput times and compared the throughput for seven routing policies and four storage assignment rules. Their particular focus was the effect of picker blocking on the mean order throughput times. Most published papers, however, tend to focus on either optimizing the order-picker's travel distance or travel time. According to Reyes, 38% of the relevant papers between 2005 and 2017 were concerned with travel distance and 32% with travel time. Travel time is the focus of the remainder of the paragraph. Chen et al. (2010) propose an integer linear program that minimizes the travel time for performing storage/retrieval operations. By minimizing the travel time, system throughput can be increased. In addition to assigning goods locations in the storage, the model also decides the interleaving sequence of storage/retrieval operations. This adds another dimension to the problem. Ene and Öztürk (2012) propose a SLAP-model for a specific warehouse in the automotive industry. This study aims to minimize total travel time for storing and picking in the warehouse.

4.1.2 Storage Layouts

SLAP has been studied for multiple storage layouts.

Planar SLAP

Park and Seo (2009) and Park and Seo (2010) studies a storage layout that is divided into $m \times n$ cells in the plane. Each cell may only store one item, i.e., items cannot be stored on top of each other. Items may only be moved horizontally in either X or Y direction, i.e., width or depth direction in Figure 4.1. Items may only arrive or depart from one of the sides. If items are on the moving path of an inbound or outbound item, the obstructive items must first be retrieved from the storage in order to provide

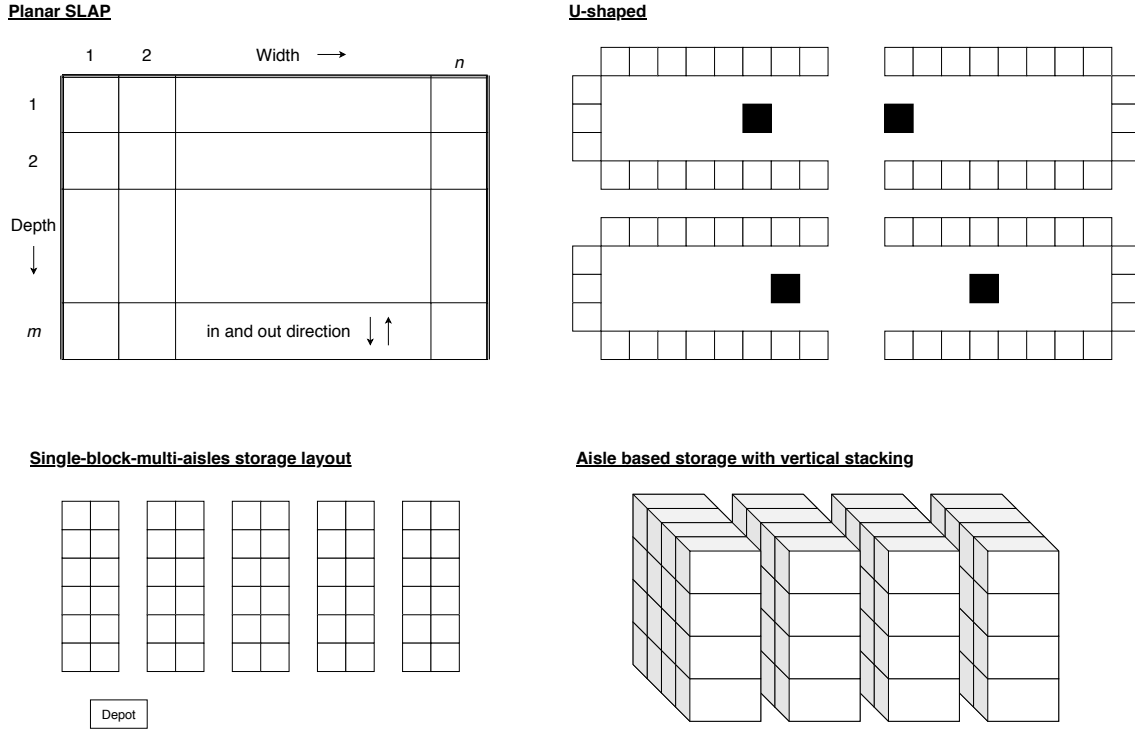


Figure 4.1: The Planar SLAP figure is inspired by Park and Seo (2009). The warehouse with u-shaped picking zones figure is inspired by Glock and Grosse (2012). The single-block-multi-aisles storage layout figure is inspired by Xiao and Zheng (2010). The aisle based storage with vertical stacking layout figure is inspired by Chen and Lu (2010).

the inbound or outbound item with a free moving path. After finishing storing or retrieving items, the obstructive items are placed back to the original locations. The movement of obstructive items is an unproductive activity that should be eliminated. The store and retrieve work may be done by a machine and is then an automatic store and retrieve system (AS/RS).

U-shaped Warehouse Layout

Glock and Grosse (2012) consider a warehouse with zones where shelves are arranged as a U, see figure 4.1. There is a picker in each zone. The picker picks all the items of an order in his/her zone before the rest of the order is collected by workers in the subsequent zones. The black square in each zone is the picker's base and where the picker hands over all the goods. The white squares indicate item locations.

Single-block-multi-aisles warehouse layout

Xiao and Zheng (2010) investigate a single-block-multi-aisles storage layout. This layout can be seen in figure 4.1. Similarly to the layout used in Planar SLAP, only one object may occupy one cell. However, in this layout, each object is accessible without the movement of obstructive objects. Objects are retrieved in batches. The workers collect a list of items before returning to the depot. Objects locations are optimized

to minimize the summed travel distance. This type of retrieval system is sometimes referred to as a picker-to-parts system.

Aisle based storage with vertical stacking

Chen et al. (2010), Chen and Lu (2010) and Ene and Öztürk (2012) consider an aisle based storage layout with vertical stacking, see figure 4.1 for an example. This layout is similar to a single-block-multi-aisles storage layout, except it allows multiple objects to be placed on top of each other in each cell. The store and retrieve work may be done by a machine and is then an automatic store and retrieve system (AS/RS).

Pick-and-pass Layout

The warehouse layout studied in this thesis is an instance of the pick-and-pass layout, also called the pick-to-belt system. An instance of this system can be seen in Figure 4.2. In a pick-and-pass system, pickers are stationed at separate zones along a conveyor belt. When an order arrives at a zone, the picker picks the parts located in that zone. The collection of an order continues at the subsequent zones. The order is passed from one zone to the next via a conveyor belt. Solution methods for the pick-and-pass layout are elaborated on in Section 4.1.4.

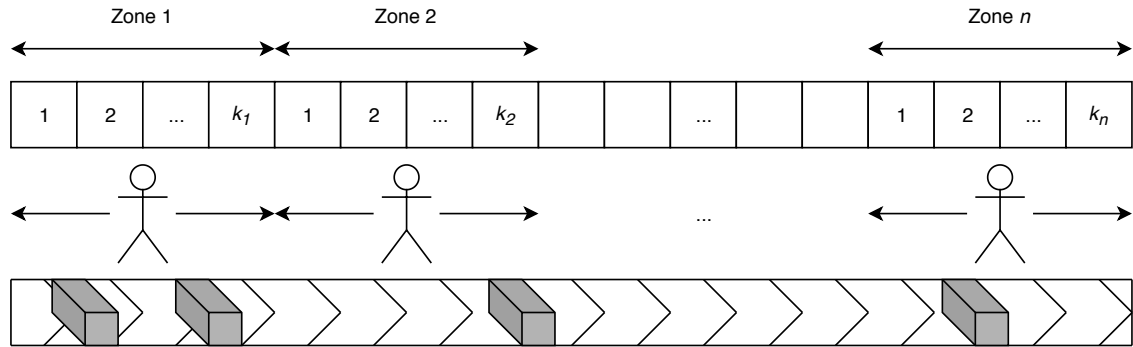


Figure 4.2: *The figure is inspired by Pan and Wu (2009). This pick-and-pass system uses a single conveyor belt.*

4.1.3 Industrial Sectors

Several of the papers on SLAP does a case study with application to an industrial sector. Of particular interest are the food manufacturing and distribution industry. Bindi et al. (2009) does a case study of an Italian food service company and develop a correlated assignment policy. This suggests that products that often are ordered together should be stored close to each other. In the foodservice industry, where multi-item orders are common, this can be particularly efficient. Chiang et al. (2011) develop an assignment approach based on mining a big dataset. To test the model, a case study for a grocery distribution center in Taiwan is done. Among the findings is the advantage of adjusting the allocations based on order patterns. This does, for instance, take seasonality into consideration. Lastly, Hui et al. (2016) propose a SLAP that is adapted to food products requiring special care. This means including parameters as temperature, humidity, insolation control, and expiry dates.

4.1.4 Solution Methods

SLAP has been proven NP-hard, see Abdel-Hamid and Borndörfer (1994) for a proof. Hence, numerous solution methods have been studied for SLAP, see figure 4.3. Many papers have an exact mathematical model formulation. However, it should be noted that the models were often not implemented and solved. Those that were implemented either did not scale polynomially or only scaled polynomially for special cases. Therefore, other papers have developed policies, rules, heuristics, and metaheuristics. The most relevant will be presented. Lastly, a brief history of different solution methods used for the pick-and-pass layout is presented.

Number of references associated with types of solution methods

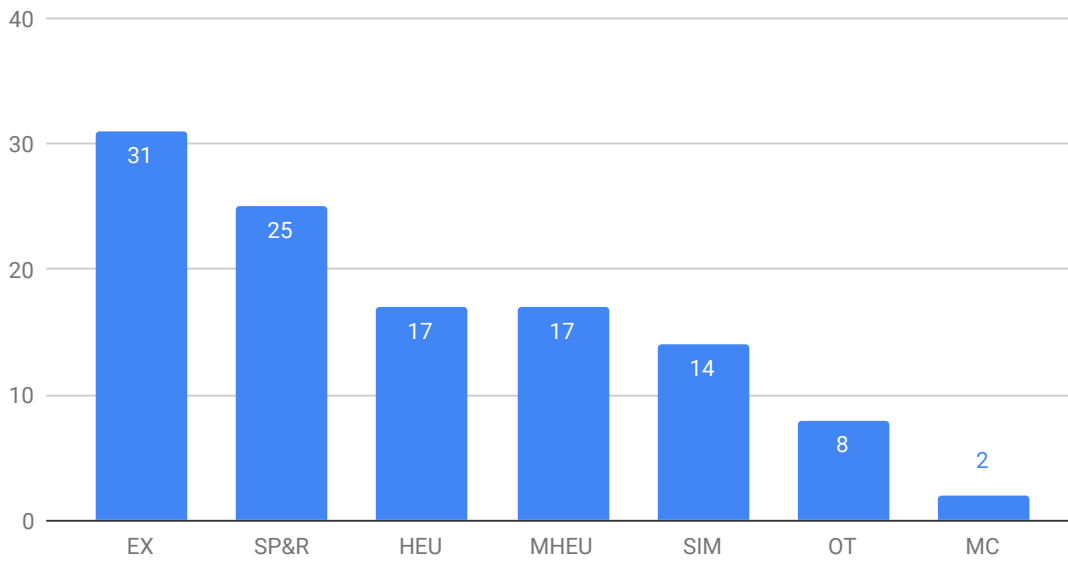


Figure 4.3: The chart shows the solution methods tried in the SLAP literature from 2005 to 2018, see Reyes et al. (2018). The different abbreviations are Exact (EX), Storage Policies and rules (SP&R), Heuristics (HEU), Metaheuristics (MHEU), Simulation (SIM), Other trends and support tools (OT) and Multi-criteria methods (MC).

Storage policies and rules

Apart from exact solutions, storage policies and rules are most used in recent years, see Figure 4.3. Storage policies and rules are quite simple heuristic methods that can quickly assign products to storage locations. Their recurring trend in the literature may be explained by the need for quick decisions and adaptability to incoming orders. Hausman et al. (1976) provides one of the earlier mentions of storage policies. Three rules are presented: *random* assignment, *full turnover-based* assignment and *class-based turnover* assignment. Random assignment assigns a good to whichever location that has space. An extension of this is *nearest free location* where a good is placed at the open space closest to the entrance or exit. This was a finding in Kay et al. (1968). Full turnover-based assignment places the most popular goods at the easiest accessible locations. Class-based turnover assignment classifies goods and assigns them

to a location based on their class. In addition to these, the policy of *dedicated storage* is much used. This policy assigns a good to a fixed location over time. The previously mentioned study of Glock and Grosse (2012) looks into dedicated storage for the U-Shaped Layout. The five aforementioned policies make up the most common storage policies. de Koster et al. (2007) gives an overview of them all.

Genetic Algorithms

The genetic algorithm, hereafter referred to as GA, was first introduced by Fraser in 1957 (S. Fraser, 1957) and later popularized by Holland (Holland, 1975). The algorithm is a metaheuristic that builds on the principles of natural selection. How the algorithm works is presented in Chapter 6. Considering the complexity of the SLAP and the fact that it is a rather short-term problem, a GA is often used to get acceptable solutions within reasonable time. Figure 4.4 shows that GA is the most used metaheuristic method for SLAP, alongside taboo search.

Metaheuristic methods

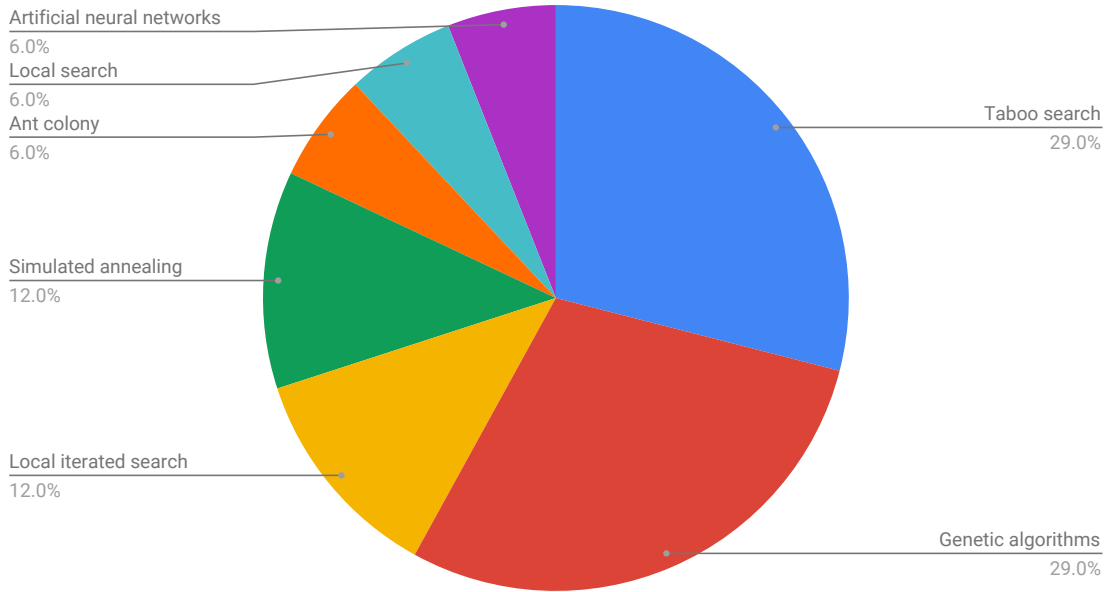


Figure 4.4: The chart shows the distribution of metaheuristic methods tried in the SLAP literature from 2005 to 2018, see Reyes et al. (2018).

Pan et al. (2015) uses a genetic algorithm to balance the workload of each picking zone in a pick-and-pass system with multiple pickers. To mitigate infeasible solutions, they propose a correction algorithm to repair solutions that break the capacity constraints. As fitness function, they use a mathematical equation that describes the level of workload balance for the given storage assignment. Kuo et al. (2016) implements both a GA and a Particle Swarm Optimization algorithm, hereafter referred to as PSO. For their particular warehouse configuration, which is a synchronized zoning system, the PSO performs slightly better than the GA, especially on larger instances. Ene and Öztürk

(2012) created an integer programming model that was solved to optimality for a real-life instance. However, due to the long computation time, a GA was implemented to approximate the solution. With faster computation time comes the advantage that changes can be done according to incoming production orders. Ene et al. (2016) also uses a GA to optimize order picking in a warehouse, but instead of optimizing with regards to typical performance measures like time or distance, they aim to find a batching and a routing for the order-picker that minimizes the energy consumption. Zhang et al. (2018) developed an evolutionary algorithm with fitness approximation to optimize a warehouse layout. To handle the problem of time-consuming evaluation of chromosomes, they trained a neural network to predict the fitness value without actually evaluating them. Furthermore, they proposed a two-layer population structure to integrate the predicted chromosomes with the actual evaluated ones. This is known as a multiple-deme parallel genetic algorithm. Park and Seo (2009) solved the Planar SLAP with a genetic algorithm with the aim of minimizing the number of obstructive object moves. For the same warehouse layout, Park and Seo (2010) compares a genetic algorithm approach and a customized PSLAP dynamic heuristic. For their particular problem, the dynamic heuristic showed the most promising results. An interesting strategy is the one of Carlo and Giraldo (2012), where they allow for rearranging the assignments during operation, a strategy they call Rearrange-While-Working (RWW). This was solved with ten heuristics for three given scenarios, and the GA approach turned out the preferred one in two of these scenarios, considering its trade-off between solution quality and computation time. Li et al. (2016) creates a greedy GA that solves the Dynamic Storage Assignment Problem, which is closely related to the standard SLAP. The greedy GA differs from a standard GA in that the initial population is generated by a greedy heuristic instead of being initialized randomly.

Simulation

Elbert and Philipp Müller (2017) does a simulation study that compares different routing policies under random storage assignments when picker maneuvers are incorporated. They use an agent-based simulator which is suited to investigate complex and weakly structured systems. The fact that pickers lose velocity by doing turns is often neglected in SLAP-papers. The same goes for the physical exposure that some maneuvers pose for the picker. Taking these factors into account can change the preferred solution, as their simulation results show. Another usage of simulation is by Pan et al. (2015). Although not used as a solution method, they create an event-based simulation model in FlexiSim to evaluate their solution. The simulator showed that the throughput gained from the genetic algorithm was statistically better than from a random policy and a first-come-first-serve policy. Glock and Grosse (2012) and Park and Seo (2010) also implemented simulator models to assess the performance of their proposed solution methods.

Pick-and-pass Solution Methods History

de Koster (1994) approximated a pick-and-pass system as a Jackson network, and assumed an exponentially distributed service time at each station and orders arriving at each station according to a Poisson process. Malmberg (1995) studied the problem of assigning products to picking zones under different zoning constraints. Petersen

(2002) showed that storage policy has a significant effect on the travel time between the zones. Jane and Lai (2005) described multiple heuristic algorithms to balance the workload among pickers along a picking line. Yu and De Koster (2008) presented an approximation method based on a G/G/m queueing network model for a pick-and-pass system. Their approximation method appears to be sufficiently accurate when used on 20 000 orders or more.

Pan and Wu (2009) showed four polynomial time algorithms which they proved to be optimal for a pick-and-pass warehouse under four sets of assumptions. Their algorithms are assigning the goods iteratively based on the order frequency, with most ordered goods first, to the center locations along the belt and expanding to the neighboring shelves with the less frequently ordered goods. For their algorithms, they consider a single conveyor belt without side belts. Later Pan et al. (2015), tested a GA algorithm for multiple parallel conveyor belts in a pick-and-pass warehouse.

4.2 Our contribution

The literature presented in Section 4.1 captures several aspects of the SLAP relevant to the studied problem in this thesis. The problem presented in Chapter 3 is a special case of SLAP that allocates goods to storage locations in a customized pick-and-pass system. Contrary to the general pick-and-pass system, Kolonial.no has a parallel conveyor belt at each picking station in addition to the main line. These are referred to as buffer zones in Chapter 3 and hereafter.

Bergland et al. (2018) formulated a Mixed-Integer-Programming model for the problem but were unable to solve the real-sized instance with an exact solution method. This model was only able to solve an instance with 4 stations and 10 boxes, wherein real-life there are 26 stations and about 20000 boxes weekly. Using the model from Bergland et al. (2018), we were also unable to solve the LP-relaxation for the real-sized instance, because of excessive memory consumption. Therefore, we applied a metaheuristic approach in this thesis. We chose to develop a genetic algorithm as the literature shows that this approach is promising for the SLAP.

Table 4.1 provides a comparative overview of our thesis and the papers reviewed in this chapter. Note that the paper of Pan et al. (2015) checks off the same boxes as our thesis. They solve SLAP for a pick-and-pass system with a genetic algorithm. In addition, they also have a warehouse layout with more than one conveyor belt. There is, however, a significant difference to our problem with regards to the layout. Where our problem has one main line with buffer zones on separate conveyor belts, Pan et al. (2015) looks into a layout with multiple main lines and no buffer zones. Another difference is in the way simulation is used in their approach. Pan et al. (2015) uses a simulation model to evaluate their solutions after solving the problem. This differs from our approach, as we instead use a simulation model in the solving process; as part of the fitness calculation in the genetic algorithm. Pan et al. (2015) uses a simpler fitness function that aims to balance the workload between stations. This is based on the hypothesis that a balanced workload gives a high throughput, but this has not

been proven for our layout. Therefore, we aim for this novel use of a genetic algorithm with a simulator. This approach has never, to the best of the authors' knowledge, been used to solve the SLAP.

Pan and Wu (2009) developed a greedy algorithm that guaranteed the optimal storage assignment for a particular pick-and-pass system. Their paper considers a single main line without buffer zones. This differs from our problem. Nevertheless, it is of interest to implement a similar greedy algorithm to assess its performance against a genetic algorithm.

Table 4.1: *Comparative overview of our thesis and the papers reviewed in this chapter in chronological order. In the comment column we have commented the solution method used, if it is not a domain-knowledge based heuristic or a GA.*

Abbreviations:

SLAP - Storage Location Assignment Problem

MB - More than one conveyor belt

P&P - Pick-and-pass warehouse layout

E - Exact model representation of the problem

GA - Genetic Algorithm

H - Heuristic

Sim - Simulator

MIP - Mixed Integer Programming

PSO - Particle Swarm Optimization

DSAP - Dynamic Assignment Problem

Paper	SLAP	P&P	MB	E	GA	H	Sim	Comment
S. Fraser (1957)								
Kay et al. (1968)								
Holland (1975)								
Hausman et al. (1976)								
Abdel-Hamid and Borndörfer (1994)								
de Koster (1994)								Queue network
Malmborg (1995)								
Berg and Zijm (1999)								
Petersen (2002)								
Jane and Laih (2005)								
Gu et al. (2007)								
de Koster et al. (2007)								
Yu and De Koster (2008)								Queue network
Park and Seo (2009)								
Pan and Wu (2009)								
Bindi et al. (2009)								
Xiao and Zheng (2010)								
Chen and Lu (2010)								Two stage MIP
Chen et al. (2010)								Tabu search
Park and Seo (2010)								
Chiang et al. (2011)								
Ene and Öztürk (2012)								
Glock and Grosse (2012)								
Carlo and Giraldo (2012)								
Pan et al. (2015)								
Franzke et al. (2016)								
Hui et al. (2016)								
Kuo et al. (2016)								PSO
Ene et al. (2016)								
Li et al. (2016)								DSAP
Elbert and Philipp Müller (2017)								
Reyes et al. (2018)								
Zhang et al. (2018)								
This thesis								

Assembly Line Routing Model

In this chapter, a mathematical formulation of the problem is presented. The mathematical model is called the Assembly Line Routing Model, hereafter referred to as the Routing model. The Routing model considers both the assignment of goods and the routing of boxes through the assembly line, following all the rules and considerations stated in the problem description. The Routing model was first described in Bergland et al. (2018), but in this chapter, some modifications have been made. The duty to yield for passing boxes at the Reunion node is added. The constraints for preferred good locations are removed because the restriction does not exist in reality. Also, the capacity constraint for a maximum number of goods is replaced with a width constraint for each station.

5.1 Assumptions

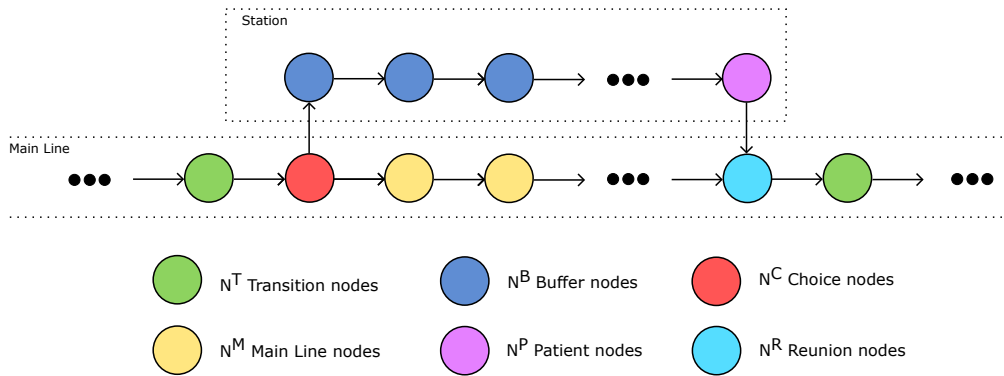


Figure 5.1: The figure displays the node sets. The sets are used to separate constraints that are viable for specific box positions.

The model represents the assembly line as a directed acyclic graph. One node indicates a possible position for a box. At most nodes, there is only one outgoing arc. The exception is at the Choice nodes, see Figure 5.1, where the box may or may not go to a station. Each station is represented with a number of nodes. The Buffer nodes are positions where the boxes wait to be processed at the station. The purpose of multiple nodes for a station is to enable for the buffer positions while also enforcing

a maximum of one box at a node at a given time. The Patient node is where an operator is processing the box. After being processed, the box is pushed back onto the main line and specifically onto the Reunion node. This node is called a Reunion node because boxes may arrive at this node from either a station or directly from the main line. In addition, some nodes are classified as Transition nodes. These are nodes that every box visits and where the next node is given. The last classification of nodes is the Main Line nodes. These are nodes adjacent to a station that are not Choice nodes nor Reunion nodes.

Figure 5.2 shows the node ordering in the graph. N_s indicates the number of nodes at station s .

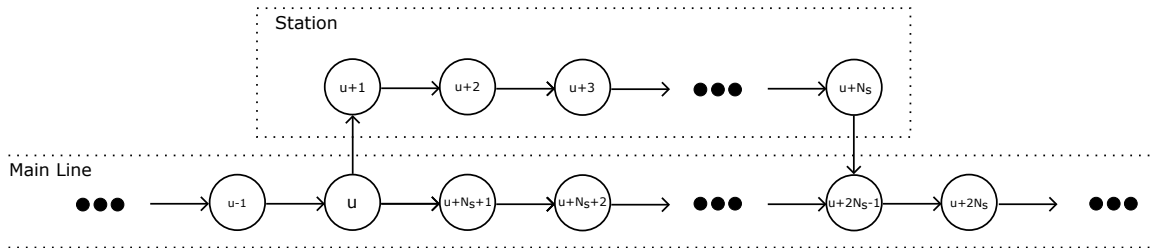


Figure 5.2: The figure displays node numbering in the graph used for this model.

The model assumes deterministic picking times of goods. Further, it is assumed that box positions on the assembly line may be discretely divided into locations represented by nodes.

The model uses continuous time with big-M formulations to model box movements on the assembly line. This was chosen over time indexing due to the number of binary variables that would be introduced using a time-indexed model (over 7 billion binary variables for 2500 boxes, 26 stations, and 1-second discretization step). Lamorgese et al. (2016) discusses this modeling choice in the context of the Real-Time Train Dispatching Problem and shows that the time indexing is an ineffective modeling choice for their problem as the number of time periods grows too large and the choice of time step becomes complicated. The same applies to this problem. This justifies the choice of continuous time with big-M formulations.

5.2 Notation

Set	Definition
$\mathcal{S}$	Stations
$\mathcal{B}$	Boxes
$\mathcal{G}$	Goods
$\mathcal{G}_b \subseteq \mathcal{G}$	Goods to be placed into box b
$\mathcal{N}$	Nodes
$\mathcal{N}^{\mathcal{B}} \subset \mathcal{N}$	Buffer nodes - Nodes at station that are not the station's Patient node.
$\mathcal{N}^{\mathcal{C}} \subset \mathcal{N}$	Choice nodes - Nodes where boxes are faced with the choice to continue on the main line or stop by the adjacent station
$\mathcal{N}^{\mathcal{M}} \subset \mathcal{N}$	Main Line nodes - Nodes adjacent to a station that are not a Choice node nor Reunion node
$\mathcal{N}^{\mathcal{P}} \subset \mathcal{N}$	Patient nodes - Nodes where goods may be placed into boxes
$\mathcal{N}^{\mathcal{R}} \subset \mathcal{N}$	Reunion nodes - Nodes where boxes may arrive from the main line or station
$\mathcal{N}^{\mathcal{T}} \subset \mathcal{N}$	Transition nodes - All boxes must pass through all nodes in this set and the node thereafter

Parameter	Definition
$N_s \in \mathbb{N}$	The number of (Buffer and Patient) nodes at station s
$N(b, g) \in \mathbb{N} \cup \{\emptyset\}$	A function returning the number of items needed for box b of good g
$S(u) \in \mathbb{N} \cup \{\emptyset\}$	A function returning the station associated with node u
$\underline{T}$	The minimum waiting time before next box is released onto the assembly line
$\bar{T} \in \mathbb{R}_{\geq 0}$	A number larger than the total duration of one work period
$T_{g,n} \in \mathbb{R}_{\geq 0}$	The time it takes to collect n units of good g
$T_{u,v} \in \mathbb{R}_{\geq 0}$	The time it takes from node u to a subsequent node v
$W_g \in \mathbb{R}_{\geq 0}$	The distribution unit width of good g
$W_s \in \mathbb{R}_{\geq 0}$	The total width allowed on station s
$Z \in \mathbb{R}_{\geq 0}$	A (small) weight used for the minimisation for departure time objective term.
$\epsilon \in \mathbb{R}_{\geq 0}$	The minimum time difference a box from main line has to departure in order to not yield for a box departing from patient node.

Variable	Definition
$a_{b,u,v} \in \{0, 1\}$	1 if box b will traverse arc (u, v) , 0 else
$p_{b,b',u} \in \{0, 1\}$	1 if box b arrives at node u before box b' , 0 else
$t \in \mathbb{R}_{\geq 0}$	The depart time for the last box at the final node
$t_{b,u}^A \in \mathbb{R}_{\geq 0}$	The arrival time of box b at node u
$t_{b,u}^D \in \mathbb{R}_{\geq 0}$	The departure time of box b from node u . Note $t_{b,u}^D = 0$ if box b never visits node u
$x_{b,s} \in \{0, 1\}$	1 if box b visits station s , 0 else
$y_{g,s} \in \{0, 1\}$	1 if good g is positioned at station s , 0 else
$\delta_{b,b',u} \in \{0, 1\}$	1 if box b from main line does not need to yield for box b' from patient node at the reunion node u , 0 else

5.3 Objective function

The main objective is to minimize the time spent from when the first box enters the assembly line until the last box leaves the assembly line, see objective part (5.1)

$$\min (t - t_{1,1}^A) \quad (5.1)$$

$$+ Z \sum_{u \in \mathcal{N}} \sum_{b \in \mathcal{B}} t_{b,u}^D \quad (5.2)$$

The objective part (5.2) minimizes the departure time for each box at every node in order to send each box quickly through the system. It is given a penalty weight Z in order to minimally affect the main objective.

5.4 Constraints

This section presents the constraints of the Assembly Line Routing Model.

5.4.1 Goods assignment constraints

All goods have to be placed at one station

$$\sum_{s \in \mathcal{S}} y_{g,s} = 1 \quad \forall g \in \mathcal{G}. \quad (5.3)$$

Each station has a maximum shelf width of W_s

$$\sum_{g \in \mathcal{G}} W_g y_{g,s} \leq W_s \quad \forall s \in \mathcal{S}. \quad (5.4)$$

5.4.2 Box and Goods relationship constraints

Constraints (5.5) enforces $x_{b,s}$ to 0 if no goods are to be picked at station s . Constraints (5.6) enforces $x_{b,s}$ to 1 when at least one good is to be picked at station s .

$$\sum_{g \in \mathcal{G}_b} y_{g,s} \geq x_{b,s} \quad \forall s \in S, \forall b \in \mathcal{B} \quad (5.5)$$

$$x_{b,s} \geq \frac{1}{|\mathcal{G}_b|} \sum_{g \in \mathcal{G}_b} y_{g,s} \quad \forall s \in S, \forall b \in \mathcal{B} \quad (5.6)$$

5.4.3 Routing constraints

The following constraints ensure that the boxes travel the correct arcs, see Figures 5.2 and 5.1.

Every box b should travel the arc exiting a Transition node

$$a_{b,u,u+1} = 1 \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^T. \quad (5.7)$$

A box b should travel the arc into a station s iff it visits that station s

$$a_{b,u,u+1} = x_{b,S(u)} \quad \forall u \in \mathcal{N}^C, b \in \mathcal{B}. \quad (5.8)$$

A box b should travel the arc to continue on the main line iff it does not visit the adjacent station s

$$a_{b,u,u+N_{S(u)}+1} = (1 - x_{b,S(u)}) \quad \forall u \in \mathcal{N}^C, b \in \mathcal{B}. \quad (5.9)$$

A box b should travel the arcs between the Buffer nodes of a station s iff it visits that station s

$$a_{b,u,u+1} = x_{b,S(u)} \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^B. \quad (5.10)$$

A box b should travel the arc from a station s to the main line iff it visits that station s

$$a_{b,u,u+N_{S(u)}-1} = x_{b,S(u)} \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^P. \quad (5.11)$$

A box b should come from the main line to the meeting node of station s iff it does not visit that station s

$$a_{b,u,u+1} = (1 - x_{b,S(u)}) \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^M. \quad (5.12)$$

5.4.4 Time constraints - Feeding the assembly line

The total work duration is not finished before the last box is done at the final station

$$t \geq t_{b,|\mathcal{N}|}^D \quad \forall b \in \mathcal{B}. \quad (5.13)$$

The work starts when the first box enters the assembly line

$$t_{1,1}^A = 0. \quad (5.14)$$

The boxes are fed to the assembly line with at least $\underline{T}$ time distance

$$t_{b+1,1}^A \geq t_{b,1}^A + \underline{T} \quad \forall b \in \mathcal{B} \setminus \{|\mathcal{B}|\}. \quad (5.15)$$

There may not be more than M boxes on the assembly line at the same time

$$t_{b+M,1}^A \geq t_{b,|\mathcal{N}|}^D \quad \forall b \in \mathcal{B}. \quad (5.16)$$

5.4.5 Time constraints - Time spent at a node

A box may not depart a node before it has arrived it

$$t_{b,u}^A \leq t_{b,u}^D \quad \forall b \in \mathcal{B}, u \in \mathcal{N}. \quad (5.17)$$

At a Patient node $u \in \mathcal{N}^P$, a box b may not depart before it has been filled with goods

$$t_{b,u}^A + \sum_{g \in \mathcal{G}_b} T_{g,N(g,b)} y_{g,S(u)} \leq t_{b,u}^D \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^P. \quad (5.18)$$

The departure time is set to 0 for a box b at a node u it never visits

$$t_{b,u}^D \leq \bar{T} x_{b,S(u)} \quad \forall b \in \mathcal{B}, u \in (\mathcal{N}^B \cup \mathcal{N}^P). \quad (5.19)$$

$$t_{b,u}^D \leq \bar{T}(1 - x_{b,S(u)}) \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^M. \quad (5.20)$$

5.4.6 Time constraints - Travelling from node to node

Constraints (5.21), (5.22) and (5.23) ensures that if a box b travels directly from node u to v , it will not arrive at node v before it has departed u and travelled the distance taking $T_{u,v}$ time.

Specifically, for all nodes except the Patient nodes and the end node, a box b may go from node u to $u + 1$. If so, the arrival time at $u + 1$ must be greater or equal to the departure time at u added the time travel distance $T_{u,u+1}$

$$t_{b,u}^D + T_{u,u+1} \leq t_{b,u+1}^A + \bar{T}(1 - a_{b,u,u+1}) \quad \forall b \in \mathcal{B}, u \in \mathcal{N} \setminus \{|\mathcal{N}|, \mathcal{N}^P\}. \quad (5.21)$$

If a box b visits a Patient node $u \in \mathcal{N}^P$, the arrival time at $u + N_{S(u)} - 1$ must be greater or equal to the departure time at u added the time travel distance $T_{u,u+N_{S(u)}-1}$

$$t_{b,u}^D + T_{u,u+N_{S(u)}-1} \leq t_{b,u+N_{S(u)}-1}^A + \bar{T}(1 - a_{b,u,u+N_{S(u)}-1}) \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^P. \quad (5.22)$$

At the Choice nodes, a box b may continue on the main line. If so, the arrival time at $u + N_{S(u)} + 1$ must be greater or equal to the departure time at u added the time travel distance $T_{u,u+N_{S(u)}+1}$

$$t_{b,u}^D + T_{u,u+N_{S(u)}+1} \leq t_{b,u+N_{S(u)}+1}^A + \bar{T}(1 - a_{b,u,u+N_{S(u)}+1}) \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^C. \quad (5.23)$$

5.4.7 Overtaking constraints - One box is in front of the other

In order to control that the boxes do not pass each other on an arc, the variable $p_{b,b',u}$ was introduced and is controlled by constraints (5.24) - (5.31)

$$p_{b,b',u} + p_{b',b,u} \leq 1 \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}, \quad (5.24)$$

$$p_{b,b',u} + p_{b',b,u} \leq \sum_{v \in \mathcal{N}} (a_{b,v,u} + a_{b',v,u}) \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}, \quad (5.25)$$

$$p_{b,b',u} + p_{b',b,u} \geq \frac{1}{2} \sum_{v \in \mathcal{N}} (a_{b,v,u} + a_{b',v,u}) \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}. \quad (5.26)$$

Constraints (5.24), (5.25) and (5.26) ensure $p_{b,b',u} + p_{b',b,u} = 1$ if both box b and box b' visit node u , and do not care otherwise.

5.4.8 Overtaking Constraints - The box in front remains in front if both boxes travel the same arc

If both box b and box b' travel the same arc, $a_{b,u,v} = a_{b',u,v} = 1$, then they must have the same precedence at the end point of the arc, at v , as they had at the start point of the arc, at u , i.e. $p_{b,b',v} = p_{b,b',u}$, otherwise do not care, see constraint (5.27) and (5.28)

$$p_{b,b',v} \geq p_{b,b',u} + a_{b,u,v} + a_{b',u,v} - 2 \quad \forall b \in \mathcal{B}, b' \in \mathcal{B}, u \in \mathcal{N}, v \in \mathcal{B}, \quad (5.27)$$

$$p_{b,b',v} \leq p_{b,b',u} + 2 - a_{b,u,v} - a_{b',u,v} \quad \forall b \in \mathcal{B}, b' \in \mathcal{B}, u \in \mathcal{N}, v \in \mathcal{B}. \quad (5.28)$$

5.4.9 Overtaking constraints and time

Constraints (5.30), (5.29) and (5.31) link the departure time and the box precedence variables.

A box b visits node u before box b' ($p_{b,b',u} = 1$) iff box b departs node u before box b'

$$t_{b,u}^D - t_{b',u}^D \leq \bar{T}(1 - p_{b,b',u}) \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}. \quad (5.29)$$

A box b' visits node u before box b ($p_{b,b',u} = 0$) iff box b' departs node u before box b

$$t_{b',u}^D - t_{b,u}^D \leq \bar{T}p_{b,b',u} \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}. \quad (5.30)$$

At a reunion node u , a box may come from a station or main line, either way the departure time will be 0 from the node it did not come from. Hence, a box b visits node u before b' ($p_{b,b',u} = 1$) iff box b departs nodes $u - 1$ before box b' departs nodes $u - 1$ or box b departs nodes $u - N_{S(u)} + 1$ before box b' departs nodes $u - N_{S(u)} + 1$.

$$(t_{b,u-N_{S(u)}+1}^D + t_{b,u-1}^D) - (t_{b',u-N_{S(u)}+1}^D + t_{b',u-1}^D) \leq \bar{T}(1 - p_{b,b',u}) \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}^R \quad (5.31)$$

5.4.10 Duty to yield at reunion node constraints

Constraints (5.32), (5.33) and (5.34) ensure the duty to yield for meeting boxes at the reunion nodes.

A box b has to arrive at reunion node u at least ϵ after box b' iff it has the duty to yield ($\delta_{b,b',u} = 0$)

$$-\bar{T}\delta_{b,b',u} + t_{b',u}^A \leq t_{b,u}^A - \epsilon \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}^R, \quad (5.32)$$

$$(t_{b,u-1}^D + t_{b,u-N_{S(u)}+1}^D + \epsilon) - (t_{b',u-N_{S(u)}+1}^D + t_{b',u-1}^D) \leq \bar{T}(1 - \delta_{b,b',u}) \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}^R, \quad (5.33)$$

$$(t_{b',u-N_{S(u)}+1}^D + t_{b',u-1}^D) - (t_{b,u}^D + t_{b,u-N_{S(u)}+1}^D + \epsilon) \leq \bar{T}\delta_{b,b',u} \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}^R. \quad (5.34)$$

Constraints (5.33) and (5.34) ensure the correct value for $\delta_{b,b',u}$.

5.4.11 Non-negativity and binary constraints

$$t \geq 0 \quad (5.35)$$

$$a_{b,u,v} \in \{0, 1\} \quad \forall b \in \mathcal{B}, u \in \mathcal{N}, v \in \mathcal{N}, u \neq v \quad (5.36)$$

$$p_{b,b',v} \in \{0, 1\} \quad \forall b, b' \in \mathcal{B}, b \neq b', v \in \mathcal{N} \quad (5.37)$$

$$t_{b,u}^A, t_{b,u}^A \geq 0 \quad \forall b \in \mathcal{B}, u \in \mathcal{N} \quad (5.38)$$

$$y_{g,s} \in \{0, 1\} \quad \forall g \in \mathcal{G}, s \in \mathcal{S} \quad (5.39)$$

$$\delta_{b,b',u} \in \{0, 1\} \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}^R \quad (5.40)$$

Genetic algorithm with an adaptive penalty for the SLAP

The Routing Model presented in Chapter 5 was formulated and implemented by Bergland et al. (2018). They found that the Routing Model only managed to solve instances of 4 stations and 10 boxes. The computation time grew exponentially for an increased instance-size. This encourages the use of heuristic and metaheuristic methods. This chapter presents a genetic algorithm with an adaptive penalty. As justified in Section 4.2, a genetic algorithm can be a promising algorithm for the SLAP considering that it scales well and finds satisfactory solutions relatively quick.

6.1 Overview

The genetic algorithm (GA) is a metaheuristic inspired by Darwin's theory of evolution. Through iterations called generations, sets of possible solutions to the problem are evaluated and recombined into new solutions. According to Eiben and Smith (2015), GA is the most widely known algorithm in the class of Evolutionary Algorithms.

As shown in Figure 6.1, the algorithm initially creates a set of random solutions to the problem. This set is called a population of solutions. A solution to the problem is an assignment of all the goods, as specified in Chapter 3. One solution is called a chromosome. All the chromosomes in the population are then evaluated and assigned a fitness value. The evaluation of the population is explained in detail in section 6.3. As long as the stopping criteria is not reached, the population will evolve and generate a new generation of chromosomes. In line with Darwin's theory of "Survival of the Fittest", the chromosomes with the highest fitness value are most likely to become parents to the next generation. Parents are mated with each other to generate child chromosomes. This process is called crossover. The children are then mutated and passed on to the population that forms the new generation. In addition, the best chromosome in each generation is passed directly on to the next generation. This operation is called elitism and is done in order not to lose the best solution. Elitism and selection aim to increase the quality of the solutions through generations, while crossover and mutation aim to create and maintain diversity in the populations.

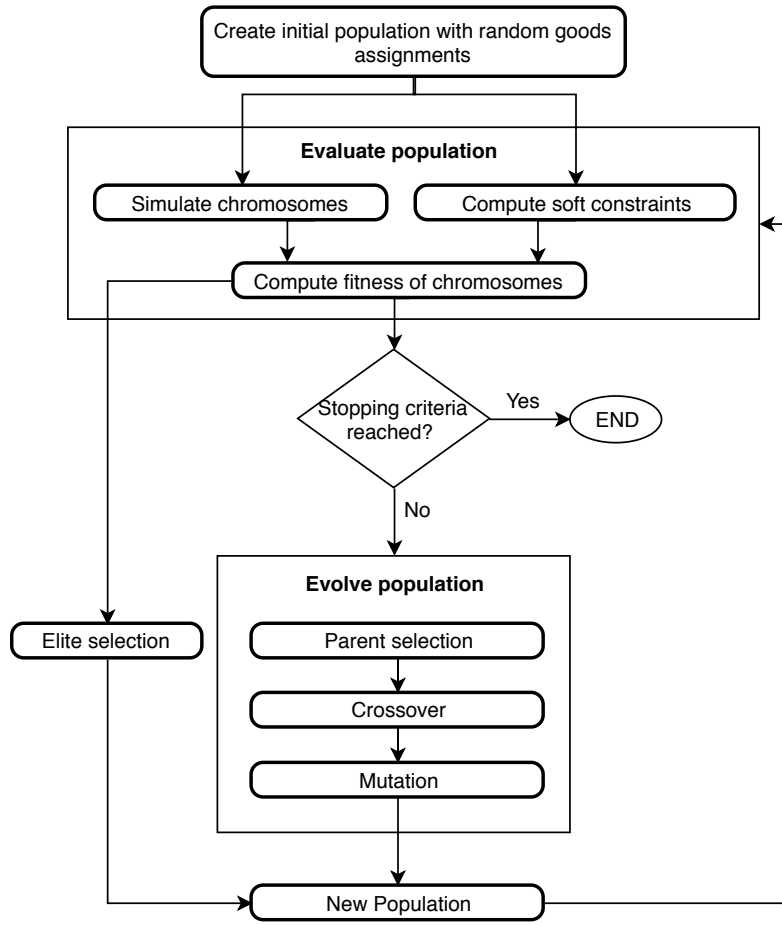


Figure 6.1: Flowchart of the genetic algorithm. The figure shows the two major steps in the algorithm; evaluation and evolution.

6.2 Chromosome Representation

A chromosome is a representation of a solution. The representation can be a decisive factor for the performance of the genetic algorithm. For the problem studied in this thesis, a binary representation is a natural choice. For each good, we need to represent which station to place it on. Figure 6.2 illustrates the representation used. Each row represents a station. Each column represents a good. The order of the goods is random. A cell has a value of 1 if the good is placed at the given station and 0 otherwise.

A requirement for a solution is that each good is placed at exactly one station. This requirement is enforced through the chromosome representation. When the initial random population is generated, it is made sure that each column has exactly one cell with a value of 1. This property is unaffected by the crossover and mutation operators as these are only done along the column axis.

The number of unique and possible chromosome representations define the search

space. There are n^k different solutions, where n is the number of stations and k is the number of goods. With 26 stations and 1985 goods, there are 26^{1985} possible goods assignments. Of course, not all these solutions are feasible, but as mentioned, also the infeasible area is explored by having an adaptive penalty that starts low.

An equivalent representation would be to merge all the rows into one row. Instead of a one-hot-vector along the column-axis, each cell would be a number between 0-25 representing the station of the good. This is a denser representation than the binary matrix, but it would not affect the solution quality of the genetic algorithm because all the genetic operators do the same in the two representations. The binary matrix is chosen because it is more general and may be used even if it is decided to allow for having a good at several stations.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
Station ID	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	
2	0	0	1	0	0	0	1	0	0	0	0	1	0	0	1	0	
3	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	
...																	
26	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	
25	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	

Figure 6.2: The figure shows how a chromosome is represented. Rows are Station-IDs and columns are good-IDs.

6.3 Evaluation of chromosomes

Each chromosome is evaluated and assigned a fitness value. As seen in Figure 6.1, the evaluation of a chromosome is composed of two parts: Simulation and soft constraints. The simulation is used to get a precise idea of how good the chromosome is. Given the assignment of the goods, the simulator routes a given set of boxes through the assembly line and computes the makespan. The makespan is the objective we want to minimize. Chapter 7 describes how the simulator works. The soft constraint is a penalty function for violating a constraint that neither the simulator nor the chromosome can handle. The constraint calculates the goods width exceedance at each station.

$$f = ((t - t_{1,1}^A) + \lambda \alpha e)^{-1} \quad (6.1)$$

Equation 6.1 describes the fitness function used. $(t - t_{1,1}^A)$ is the makespan that is returned by the simulator. $\lambda \alpha e$ is the soft-constraint. A small makespan and a small penalty should yield a good fitness value, hence the inverse. In the soft constraint, e is the solution's total width exceedance in centimeters. Equation 6.2 shows how it is calculated. For each station, the total width used is subtracted from the available width at the station.

$$e = \max\{0, \sum_{s \in S} (\sum_{g \in G} W_g y_{g,s} - W_s)\} \quad (6.2)$$

The coefficients λ and α are used to adjust the size of the width exceedance. The λ -coefficient is used to transform the width exceedance e to about the same size as the makespan value, see Equation 6.3. When these two values are in the same range, they are equally important for the fitness. The λ -coefficient is calculated after the first generation and is constant for the entire genetic algorithm.

$$\lambda = \frac{\min\{t - t_{1,1}^A\}}{\max\{e\}} \quad (6.3)$$

The α -coefficient is what makes the penalty function adaptive. In structural design optimization, the optimal solution often lies on or close to the boundary of the feasible area (Nanakorn and Meesomklin, 2001). Therefore, an infeasible solution might contain building blocks to the optimal solution. Too big a penalty would exclude this solution from being able to mate. On the other hand, we don't want a lot of infeasible solutions in the final population. This justifies the use of an adaptive penalty function, that starts with a low penalty that increases during the solution process. Crossley et al. (1997) found that a coefficient that grows linearly with the generations produced the best results for some test problems. Consequently, alpha takes the value of the current generation, but is scaled down to make sure that the penalty value does not become dominant.

6.4 Termination criteria

Our genetic algorithm uses a disjunctive termination criteria. The first trigger for termination is a given number of generations without finding a new best solution. The other part of the termination criteria is a pre-defined timelimit that sets a maximum allowed computation time. Because of the complexity of the problem, we won't have an idea of how close we are to an optimal solution, so this is not an applicable termination criterion.

6.5 Parent Selection

Parent selection is the first step in evolving a population of chromosomes. The best-fitted chromosomes have the best chances of being selected as parents for the new generation. The selection method used in this thesis is the tournament selection. Figure 6.3 illustrates how it works. When selecting a parent, k competitors are chosen at random from the population. These are assigned a probability of winning the tournament based on their fitness value. A competitor wins the tournament with the probability $p_t(1 - p_t)^{i-1}$ where p_t is a pre-defined probability value and i is determined based on the chromosome's rank among the competitors. The best chromosome wins with a probability p_t , the second best wins with a probability of $p_t(1 - p_t)$ and so on. With a $p_t < 1$, every individual has a strictly positive probability of winning. With $p_t = 1$, the $k - 1$ weakest competitors never have a chance to win the tournament. The parameter k decides the size of the tournament. If k is set high, there is an

increased chance that the best chromosome in the population will get a spot in the tournament and go on to get selected as a parent. A smaller k makes it possible for lesser fit chromosomes to win the tournament and get selected as a parent. Increasing k increases the selection pressure, which means that fitter individuals are more likely to survive.

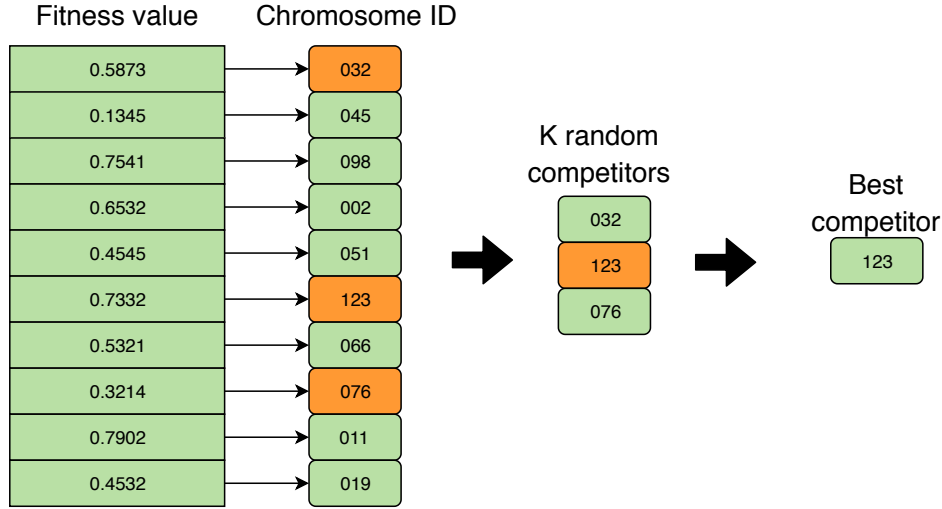


Figure 6.3: An illustrative example of how Tournament Selection works.

6.6 Crossover

Crossover is the second step in evolving the population. Crossover is a variation operator that takes two parents and recombines them into a new solution, often called a child. For a pair of parents, a crossover is performed with a probability of p_c . Otherwise, the parents are passed on unchanged to the mutation step. Two crossover operators are tested in this thesis. The first one is the one-point crossover. This operator is one of the standards for a binary chromosome representation. Figure 6.4 illustrates how it works. Both parents are split in two at a random cutting point along the column axis, and the children are created by exchanging the tails of the parents. A problem with the one-point crossover, however, is that it tends to keep together goods that are close to each other in the representation. Similarly, goods on opposite ends of the representation tend to be separated. This adequacy is called a positional bias in the literature (Eiben and Smith, 2015). To avoid positional bias, a uniform crossover can be used. This operator does not exhibit any positional bias. Figure 6.5 illustrates how it works. For each column, it is decided randomly if this is to be inherited from the first or the second parent. The uniform crossover does, however, exhibit a distributional bias. This means that the child is often created from 50 % of each parent instead of more uneven influences which may occur in the one-point crossover. Consequently, both operators have to be tested before knowing which one is the best.

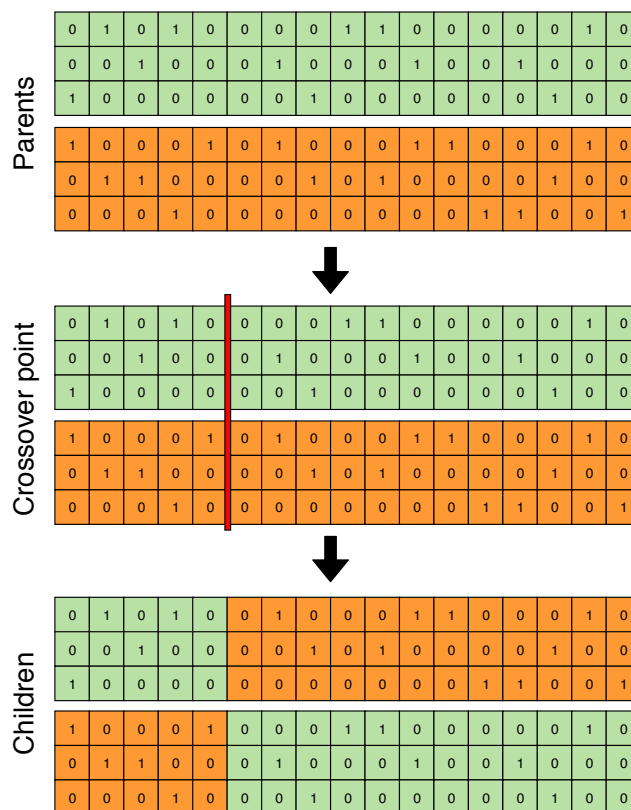


Figure 6.4: An illustrative example of how One-point Crossover works.

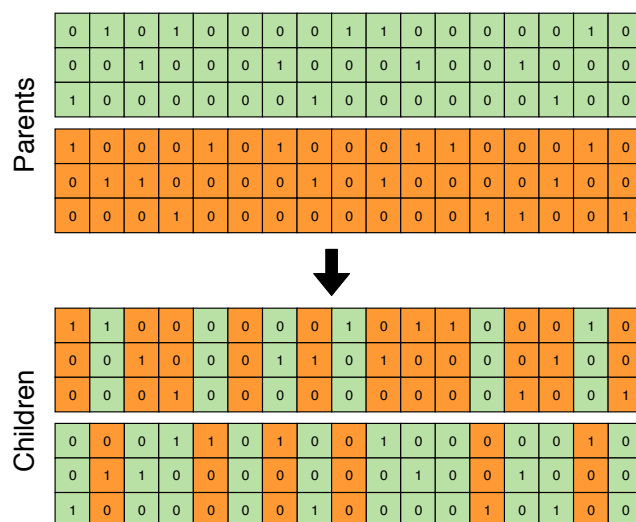


Figure 6.5: An illustrative example of how Uniform Crossover works.

6.7 Mutation

A mutation is the third step in evolving a population. It is a variation operator that contributes to the exploration part of the genetic algorithm. With a probability

p_m , the mutation is performed on a child chromosome, otherwise, the child is unchanged. The choice of the mutation rate p_m depends on the desired outcome. If we need a population where all chromosomes have high fitness, a low mutation rate should be used to avoid disrupting good solutions. If it is enough that one good chromosome is found, a higher mutation rate may be used, because we can sacrifice local optimums to get better coverage of the search space (Eiben and Smith, 2015).

There are a number of possible mutation variants. For the problem at hand, swap mutation was found to be the most fitting. It preserves the chromosome traits and has an intuitive meaning: Two goods swap stations. One swap is that two random columns are swapped with each other. Figure 6.6 illustrates how it works. A parameter n_m decides the number of swaps that should be done on each chromosome.

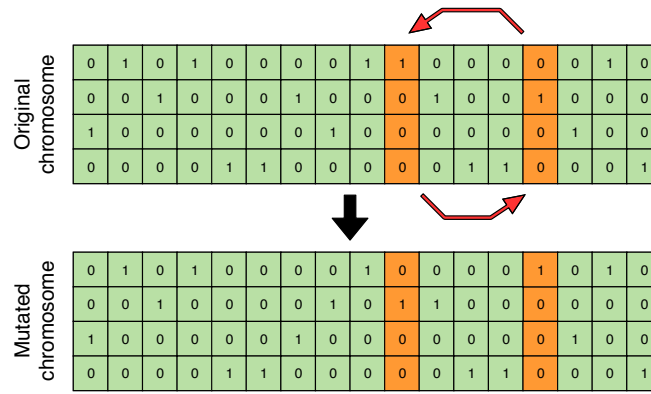


Figure 6.6: An illustrative example of how Swap Mutation works.

6.8 Algorithm Complexity

The genetic algorithm has $O(xp[Simulator + |S||G| + \log(p)])$ where x is the number of generations, p is the population size, *Simulator* is the big-O of the simulator, S is the set of all stations and G is the set of goods. Here is a short explanation.

Each generation consists of two main parts; the *Evaluate population* and the *Evolve population*. Evaluating the population is done in $O(pSimulator)$. The big-O of the simulator is explained in Section 7.3. The computation of the soft constraints is done in $O(p|S||G|)$. In evolving the population, parent selection is done in $O(p \log(p))$ because the population is being sorted. There is an additional $O(pk \log(k))$ where k is the tournament size, but since k is considered a constant, this is equivalent to $O(p)$. The big-O of the crossover operation depends on the crossover-method used. For each chromosome, one-point crossover is done in $O(1)$ and uniform crossover is done in $O(|G|)$. Hence, the crossover operation is always bounded by $O(p|G|)$. Lastly, mutation of the population is done in $O(p)$. Since all steps happen sequentially, the big-O of one generation is $O(pSimulator + p|S||G| + p \log(p))$, where terms that are dominated are omitted. Consequently the big-O for all generations is $O(xp[Simulator + |S||G| + \log(p)])$.

Chapter 7

Simulator

To evaluate a goods assignment, a simulator has been created and implemented. The simulator creates an environment of the warehouse based on a goods assignment and then routes a given sequence of boxes through the assembly line. The assembly line is approximated as a node graph, as shown in Figure 5.1. The rules and restrictions defined in Chapter 3 apply. The simulation computes the makespan of processing the boxes. To inspect the behavior of the simulator, a Graphical User Interface was implemented. A screenshot of this is illustrated in Figure 7.1. Simulating the same sequence of boxes for different goods assignments gives a basis for comparison. This is why the simulator is used to compute the fitness of individuals in our genetic algorithm. Also, the simulator is well suited for comparing different solution methods.



Figure 7.1: A Graphical User Interface used to inspect the simulation. This screenshot only portrays the first five stations. The rectangles indicate nodes. The bottom line of nodes is the main line. The top line of nodes indicate nodes of the buffer zones. The numbers inside the nodes are the box-id of the occupying box at the current time. Some nodes have a red background color. These are nodes in which boxes are currently being filled with goods.

This chapter first presents the properties of the simulator in Section 7.1. Then, the simulator algorithm is presented in Section 7.2. The algorithmic complexity is presented in Section 7.3. The chapter is concluded with a walkthrough example in Section 7.4.

7.1 Simulator properties

The simulator is a dynamic simulator with next-event time progression. The next-event time progression was chosen over the fixed-increment time progression because fixed time increments would require highly fine-grained timesteps that would cause the simulator to run significantly slower than the event-based alternative. In addition, the

simulator is characterized as a deterministic simulator, as the incoming boxes and the picking times are given. A given sequence of boxes, under the same goods assignment, always returns the same makespan.

7.2 The Simulator Algorithm

This section describes how the simulator computes the makespan from a given set of inputs.

7.2.1 An Overview

The simulator contains three events. Together, they take care of all the actions necessary to route the boxes through the assembly line. A thorough description of each event is given in Subsection 7.2.3, but the following gives a brief idea. The *Box-Arrived* event moves a box to the next node. The *Box-Ready* event tells when a box is ready to be moved from a node. Lastly, the *Node-Empty* event gives a signal that a node is free. Each event triggers new events. These are stored in an event list that contains all future events. The list is sorted in a specific way to make sure that the events are executed in the correct order.

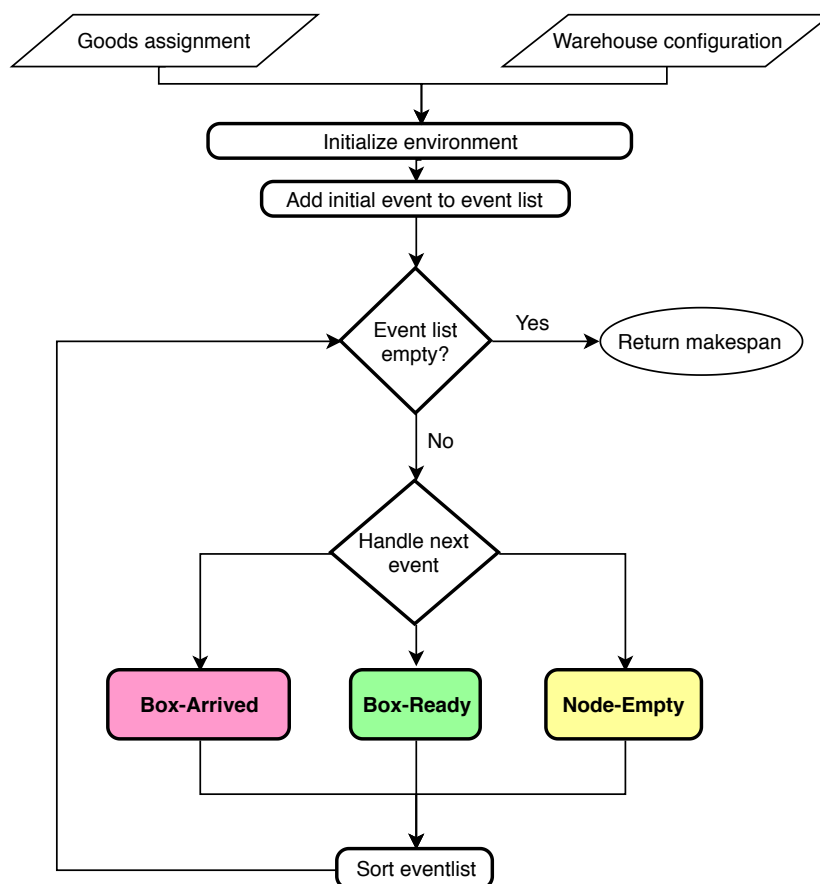


Figure 7.2: *The flowchart shows the simulator's algorithm.*

7.2.2 Detailed Description

As seen in Figure 7.2, the simulator takes a goods assignment as input. This is used together with the input of the warehouse configuration to initialize the assembly line environment. The environment contains all information about the assembly line and its state. This includes all nodes, stations, and boxes. Each station holds the goods according to the goods assignment input. For each box, information about its current position, whether it is ready to move or not and it's customer order is kept track of. Each node holds information on whether or not a box is present. The nodes are also classified in the same way as in Figure 5.1. For instance, processing of a box only happens at the patient node.

When the environment is initialized, the routing of the boxes start. This is done by handling events from the event list. Each event updates the state of the environment and adds a new event to the event list. Initially, the event list contains a *Node-Empty* event. The handling of this event will schedule a *Box-Arrived* event that pulls the first box onto the assembly line. Subsequently, a chain of events will follow. Section 7.4 gives a concrete example. The event list is non-empty until the last box leaves the assembly line. Then, the environment time is the total makespan of processing the boxes. This is the return value of the simulator.

To make sure that events are executed in the correct order, the event list is sorted when new events are added. Sorting is done on several attributes and prioritized with respect to time, event type, duty to yield and node-id.

7.2.3 The events

The three types of events contain the same attributes, see Table 7.1. *Time* contains the value of when the event happens. This is the time elapsed since the first box enters the assembly line. *Node-ID* tells on which node the event happens and *Box-ID* indicates the affected box. The *Event-Type* describes the type of the event, and *DTY* tells if the box has a duty to yield for other boxes. Most events create new events that are added to the event list. This is illustrated with the blue boxes in the flowcharts. In the following, each event is explained in detail.

Table 7.1: *These are the attributes of an event.*

Time	Node-ID	Box-ID	Event-Type	DTY
------	---------	--------	------------	-----

Box-Arrived Event

The *Box-Arrived* event takes care of moving a box to the next node. The box only moves if the node it is going to is empty. The time variable makes sure that the box moves at the correct time.

Figure 7.3 shows the process of the *Box-Arrived* event. It also illustrates how the event triggers other events. If the arrival node is a patient node, a *Box-Ready* event is created and added to the event list. The time variable for this event equals the current makespan plus the time it takes to pick the box's goods at that station. If the arrival node is not a patient node, a *Box-Ready* event is made to set the box status to ready when the box arrives at the node. Lastly, as long as the arrival node isn't the first node on the assembly line, a *Node-Empty* event is scheduled to signal that the previous node is free.

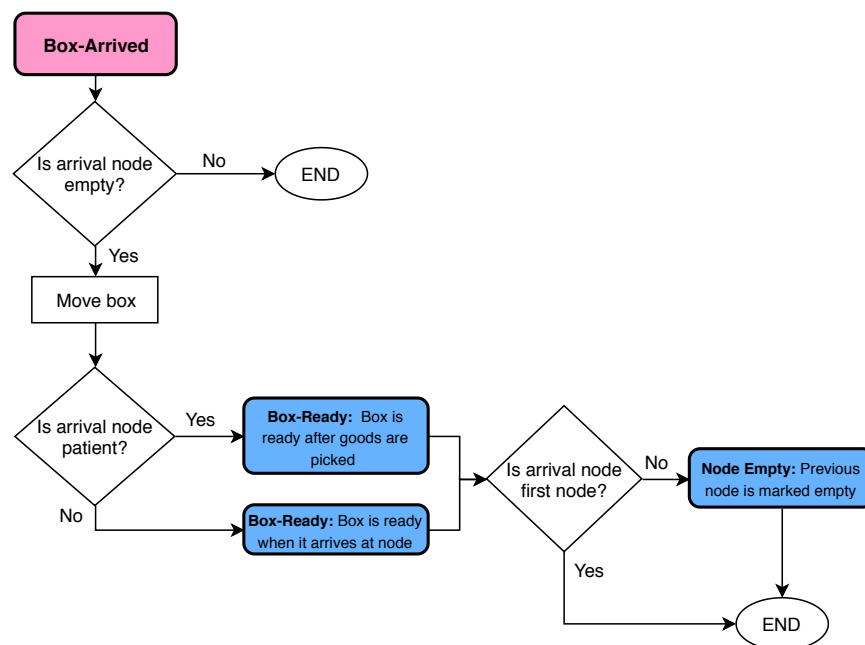


Figure 7.3: The flowchart shows the *Box-Arrived* event.

Box-Ready Event

The *Box-Ready* event updates the status of the box to ready at the correct time. Figure 7.4 shows that if a box is ready at the last node, it removes the box from the assembly line and adds a *Node-Empty* event of that last node to the event list. If the box isn't at the last node and the next node is empty, a *Box-Arrived* event for that box is created and added to the event list.

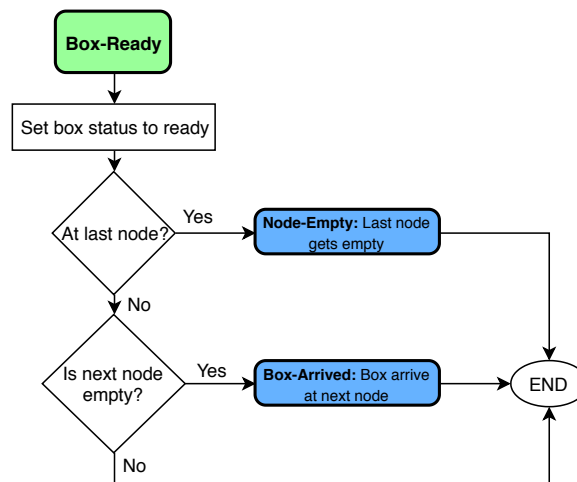


Figure 7.4: The flowchart shows the *Box-Ready* event.

Node-Empty Event

The *Node-Empty* event sets the node status of a node to empty. This is needed as a separate event to make sure that queued boxes move when the box in front moves. Specifically, a *Node-Empty* event triggers a *Box-Arrived* event if a box is ready at the previous node. Figure 7.5 illustrates the process of the event. After setting the node status to empty, the event must figure out which event to trigger next. First, it is checked if the node of the event is the first node. If so, a *Box-Arrived* event is created that will place the next box onto the assembly line. If the node is not the first node, but rather a reunion node, it is checked if there is a box at the patient node that is ready. If so, this box has the right of way to enter the main line, and a *Box-Arrived* event will be created for it. In any other case, it is checked if there is a box ready at the previous node. If so, a *Box-Arrived* event is created for that box.

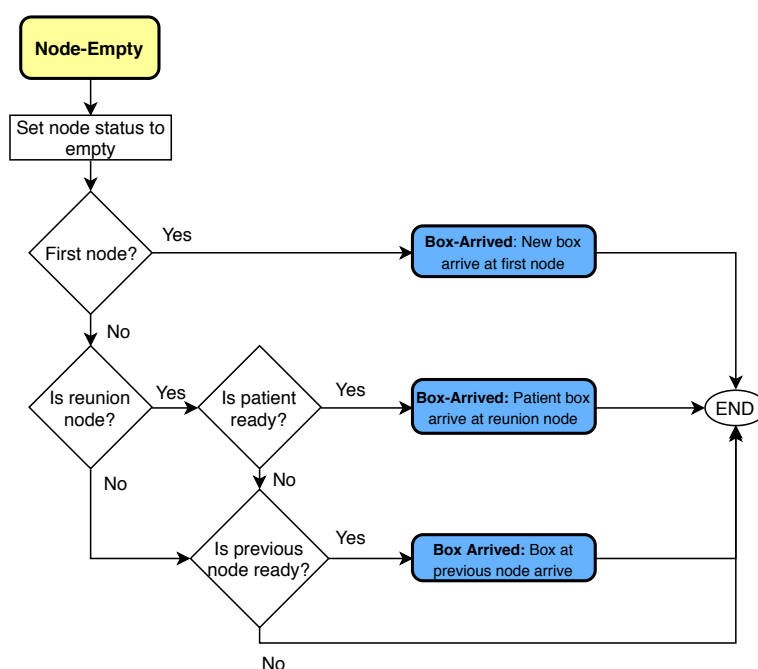


Figure 7.5: The flowchart shows the *Node-Empty* event.

7.3 Algorithm Complexity

The simulator has $O(|N||B||G| + |N|^2|B| \log(|N|))$ where N is the set of all nodes, G is the set of all goods and B is the set of all boxes. Here is an explanation.

In the flowchart, seen in Figure 7.2, each operational cell happen in $O(1)$, except the initialize environment step, *Box-Ready* event step and sort event list step. The initialize environment step happens once and in $O(|N| + |B|)$. The *Box-Ready* event step may need to calculate the picking time of a box at a station. Such a computation is done in $O(|G|)$. Based on any standard sorting algorithm, such as 'quicksort' or 'mergesort,' the sorting of the event list is done in $O(m \log(m))$, where m is the length of the event list. The event list can never be longer than $3|N|$, because there are at most 3 events that can be scheduled for any node at any time. The sorting of the event list can be called at most $3|N||B|$ times. Hence, the Big-O of the looping procedure of the simulator is $O(3|N||B| \times (|G| + 3|N| \log(3|N|))) = O(|N||B||G| + |N|^2|B| \log(|N|))$. The looping procedure is subsequent to the initialization procedure, and its Big-O is larger than that of the initialization procedure. The initialization procedure and the looping procedure are the only procedures of the simulator. The Big-O of the simulator is therefore the one of the looping procedure, i.e. $O(|N||B||G| + |N|^2|B| \log(|N|))$.

7.4 Walkthrough Example

In this section, a walkthrough example with illustrations is provided. The example shows the first movements of three boxes on the assembly line, see Figure 7.6 and 7.7.

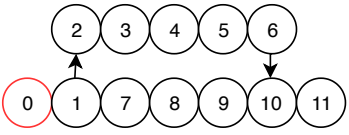
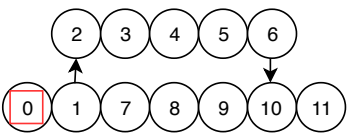
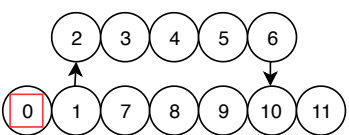
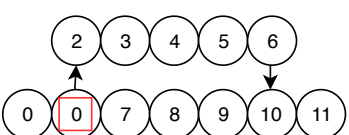
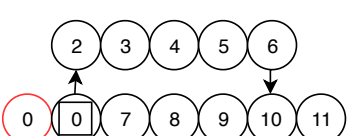
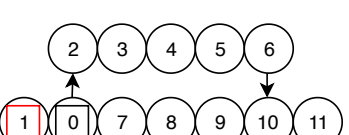
Figure		Eventlist					Explanation
Step	Makespan: 00:00	Time	Node ID	Box ID	Event type	DTY	
Step 1		00:00	0	None	Node Empty	Yes	A node-empty event will trigger the initial movement of boxes by creating a box-arrived event for box 0 in node 0.
Step	Makespan: 00:00	Time	Node ID	Box ID	Event type	DTY	
Step 2		00:00	0	0	Box Arrived	Yes	By handling the box-arrived event for the first node, a box-ready event will be created. Since this is not a patient node, the box will be ready at the same time it arrives the node.
Step	Makespan: 00:00	Time	Node ID	Box ID	Event type	DTY	
Step 3		00:00	0	0	Box Ready	Yes	When handling the box-ready event, the simulator checks if the next node in the box's future node list is empty. In this case it is and a box-arrived event is created for box 0 in node 1. The box will arrive in the next node at the current warehouse time + traverse time.
Step	Makespan: 00:01	Time	Node ID	Box ID	Event type	DTY	
Step 4		00:01	1	0	Box arrived	Yes	The simulator checks if node 1 is empty when box 0 is supposed to arrive. If so, it moves the box to node 1. After the box is being moved, 2 new event are appended: a box-ready event for box 0 in node 1 and a node-empty event for node 0.
Step	Makespan: 00:01	Time	Node ID	Box ID	Event type	DTY	
Step 5		00:01	0	None	Node Empty	Yes	Now we have more than one event in the event list. The list is sorted based on several attributes. First priority is ascending time, then DTY, then event type (node-empty first, box-ready last) and lastly descending Node ID. The node-empty- event will trigger that the next box arrive in the first node.
		00:01	1	0	Box Ready	Yes	
Step	Makespan 00:01	Time	Node ID	Box ID	Event type	DTY	
Step 6		00:01	0	1	Box Arrived	Yes	When the box-arrived event is appended, the event list is sorted. The box-arrived event comes out on top because this event type is prioritised before Box Ready. There is no traverse time when entering on to the first node.
		00:01	1	0	Box Ready	Yes	

Figure 7.6: Walkthrough example of the simulator, part 1

Step	Makespan	Time	Node ID	Box ID	Event type	DTY	
Step 7	Makespan: 00:01	00:01	1	0	Box Ready	Yes	With the same event time, and event type, the event that has the highest Node ID will get on top of the event list. The simulator checks if node 2 is empty, which is the next node for box 0 according to the box's future node list, then creates a box-arrived event for box 0 at node 2. Box 0 needs goods that is located at current station.
		00:01	0	1	Box Ready	Yes	
Step 8	Makespan: 00:01	00:01	0	1	Box Ready	Yes	The box-ready event for box 1 is handled. This creates a box-arrived event for box 1 at node 1.
		00:02	2	0	Box Arrived	Yes	
Step 9	Makespan: 00:02	00:02	2	0	Box Arrived	Yes	Check if node 2 is empty and then move box 0 to node 2. A node-empty event is created for node 1 and a box-ready event is created for box 0.
		00:02	1	1	Box Arrived	Yes	
Step 10	Makespan: 00:02	00:02	1	None	Node Empty	Yes	The simulator now handles the node-empty event. The simulator checks if there is a box ready in the previous node, which it is. A new box-arrived event is normally created and added to the event list, but this event for box 1 already exists, so no new event is added.
		00:02	1	1	Box Arrived	Yes	
		00:02	2	0	Box Ready	Yes	
Step 11	Makespan: 00:02	00:02	1	1	Box Arrived	Yes	Before moving box 1 to node 1, the simulator check if the node is empty. This is the case, thus, it creates a node-empty event for node 0 and a box-ready event for box 1.
		00:02	2	0	Box Ready	Yes	
Step 12	Makespan: 00:02	00:02	0	None	Node Empty	Yes	The node-empty event on the first node triggers new box onto the assembly line.
		00:02	2	0	Box Ready	Yes	
		00:02	1	1	Box Ready	Yes	
Step 13	Makespan: 00:02	00:02	0	2	Box Arrived	Yes	Due to the sorting prioritising, the new box-arrived event is handled first. Now there are 3 boxes on the assembly line and the process continues like this until all boxes has arrived on the assembly line.
		00:02	2	0	Box Ready	Yes	
		00:02	1	1	Box Ready	Yes	

Figure 7.7: Walkthrough example of the simulator, part 2

Greedy approach to the SLAP

To assess the performance of the genetic algorithm, it is of interest to compare it to a simple greedy algorithm. Algorithm 1 shows the pseudocode of the greedy algorithm for assigning goods to stations. This algorithm is inspired by Pan and Wu (2009), reviewed in Chapter 4. The greedy choice is to place a good at the station with the least workload where there is available space. The goods are placed one at a time, going from the most frequent to the least frequent. If several stations have the same workload, the latest of these stations is chosen. The algorithm assumes that there is always at least one station with enough space to place the next good. This simplification was done due to the excess station capacity space in the provided data, see Section 9.4. The complexity of the algorithm is explained in Section 8.1.

Algorithm 1 Greedy algorithm for assigning goods to stations**Inputs**

goods, stations	▷ Set of all goods and stations
$y_{g,s}$	▷ Empty goods assignment matrix
T_g	▷ Total picking time per good g
T_s	▷ Workload at station s , zeros initially
W_g	▷ Distribution unit width per good g
W_s	▷ Station capacity per station s

Output

$y_{g,s}$	▷ Goods assignment of all goods
-----------	---------------------------------

```

1: function GREEDY(goods, stations, yg,s, Tg, Ts, Wg, Ws)
2:   sort goods from most frequent to least frequent
3:   for  $g$  in goods do
4:      $S^{MIN} \leftarrow None$ 
5:      $T^{MIN} \leftarrow \infty$ 
6:     for  $s$  in stations do
7:       if  $T_s \leq T^{MIN} \wedge W_s \geq W_g$  then
8:          $S^{MIN} \leftarrow s$ 
9:          $T^{MIN} \leftarrow T_s$ 
10:     $T_{S^{MIN}} \leftarrow T_{S^{MIN}} + T_g$ 
11:     $W_{S^{MIN}} \leftarrow W_{S^{MIN}} - W_g$ 
12:     $y_{g,S^{MIN}} \leftarrow 1$ 
13:   return  $y_{g,s}$ 

```

8.1 Algorithm Complexity

The greedy algorithm has algorithmic complexity of $O(|G| \log(|G|) + |G||S|)$, where G is the set of all goods and S is the set of all stations. Here is an explanation.

Step 2 of the algorithm is done in $O(|G| \log(|G|))$ with any standard sorting algorithm such as 'quicksort' or 'mergesort'. The subsequent remainder steps are done in $O(|G||S|)$. The algorithm is therefore $O(|G| \log(|G|) + |G||S|)$.

It should be noted that there is a significant computation time of the algorithm's input, most notably T_g . The calculation of T_g is done in $O(|B||G|)$ where B is the set of all boxes.

Computational Study

In this chapter, the solution methods presented are subjects to a performance evaluation. The performance of the genetic algorithm from Chapter 6 and the greedy algorithm from Chapter 8 are evaluated against the assignment of the goods of Kolonial.no. The goods assignment are compared using the simulator described in Chapter 7.

The genetic algorithm, the simulator, and the greedy were all implemented in Python. Although supporting packages for both simulator models and genetic algorithms exist, our models were implemented from scratch. This was to ensure complete control over the code. The algorithm was run on the shared computer cluster of the Department of Industrial Economics and Technology Management called the Solstorm Cluster. To make use of the multiple cores in the cluster, the fitness evaluation part of the GA was parallelized so that multiple chromosomes could be simulated simultaneously. The technical specifications of the hardware and software used are shown in Table 9.1.

Table 9.1: *Hardware and software specifications used*

Processor	2x2.4GHz Intel Xeon Gold 5115	2x2.3GHz Intel E5-2670v3	4x2.2GHz AMD Opteron 6274
RAM	96 GB	64 GB	128 GB
Solstorm Rack	5	4	1
Available nodes	24	48	24
Cores per node	20	12	32
Python version	3.5.4	3.5.4	3.5.4
Numpy version	1.16.2	1.16.2	1.16.2
Pandas version	0.24.1	0.24.1	0.24.1
Scipy version	1.2.1	1.2.1	1.2.1

The chapter is organized in ten sections. First, in Section 9.1 the dataset is described. Section 9.2 describes how the data was split into *optimization data* and *unseen data*. Next, the parameters used in the simulator are described in Section 9.3, before an extensive review of the calibration of the genetic algorithm parameters follow in Section 9.4. Section 9.5 discuss the significance of the instance-size and Section 9.6 assess the performance of the algorithms. Analyzes follow with a comparison of solutions in

Section 9.7 and a study of the uniqueness of the best solution in Section 9.8. Lastly, we discuss the value of shorter processing times in Section 9.9 and start some initial analyzes on further improvements in Section 9.10.

9.1 Description of the provided data

To train and test the models, Kolonial.no delivered data containing details of orders processed in a timespan of 14 days between Oct 08, 2018 and Oct 21, 2018. They were delivered as four csv-files, see Table 9.2.

Table 9.2: *The data files provided by Kolonial.no.*

product-data.csv	Product details of every good
order-units.csv	Details about every good
order-units-products.csv	Connecting a good and a box
product-picking-seconds.csv	Picking times for every good

There are 41432 boxes and 1985 goods in the dataset. Figure 9.1 shows how the number of goods and boxes vary per day. The figure indicates a weekly cyclic trend concerning both the number of boxes and goods. In terms of boxes, Mondays are the busiest. Saturdays and Sundays are by far the quietest. In terms of goods, nearly all the goods in the product range are picked during weekdays.

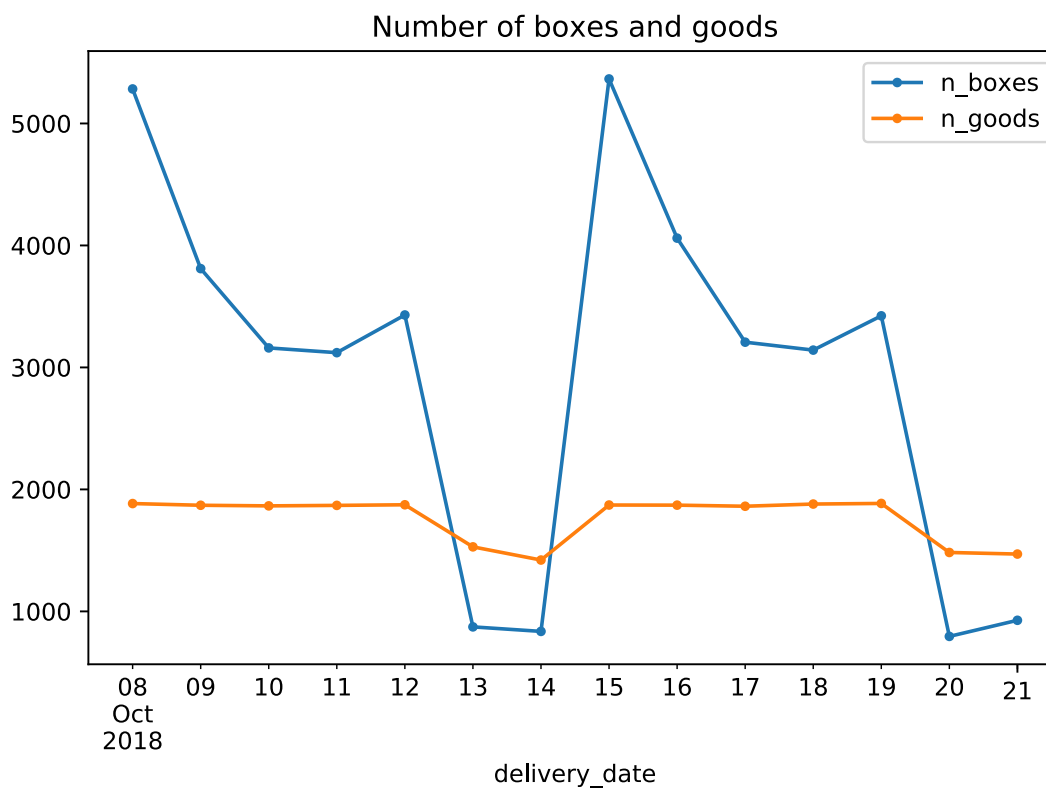


Figure 9.1: *The plot shows the number of boxes and goods per day. Oct. 08 and Oct. 15 are Mondays.*

9.2 Data split into Opt Data and Unseen Data

We have chosen to split the data into two, the first week and the second week. We have named these *optimization data*, hereafter abbreviated as *opt data*, and *unseen data*. The *opt data* is used to tune the parameters of the genetic algorithm. The *opt data* is also used to see how well the algorithms perform on a known set of boxes. We wish to emphasize that this is the studied problem, finding a goods assignment to minimize makespan on a known set of boxes. The *unseen data* is used solely out of curiosity to see how well the found goods assignments extrapolate to the subsequent week's boxes, i.e., boxes that the goods assignments are not optimized for. From an operational standpoint, it is of interest to see how the previous week's goods assignment performs on the next week's set of boxes, because this goods assignment serves as a basis before any relocations are done to fit the coming week. Figure 9.1 at the beginning of this chapter showed how cyclic the number of boxes and goods are for the two weeks. Figure 9.2 shows the good frequency of the boxes in the two weeks. The figure portrays that there is also a weekly cyclic trend in goods demanded. With this split there are 20513 boxes in the *opt data* and 20919 in the *unseen data* and more than 98 % of the goods are present in the boxes of the two datasets.

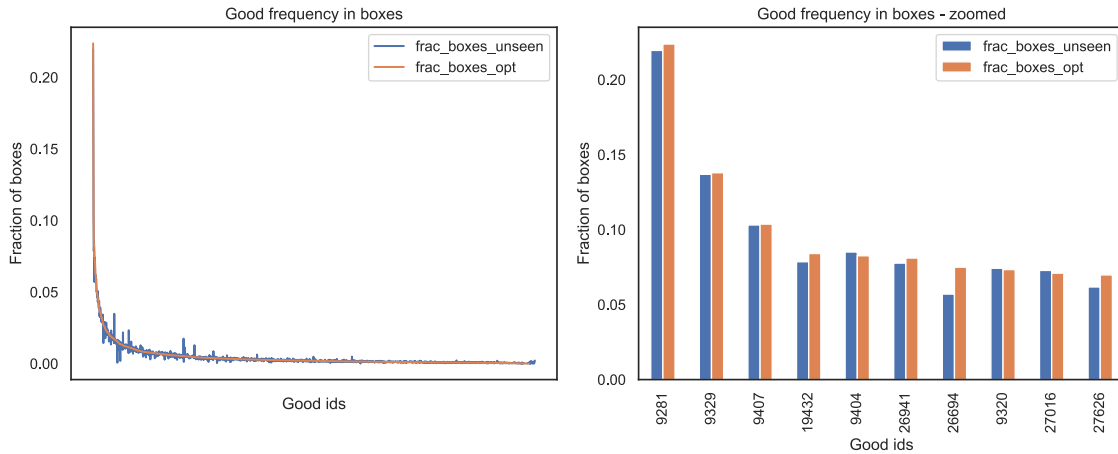


Figure 9.2: The plots show the frequency of a good in *opt data* and *unseen data*. The goods are ordered from most common goods to the least common goods in *opt data*. The frequency of a good in one dataset is divided by the number of boxes in that dataset. This is done, rather presenting the number of times a good is requested, since there are not equally many boxes in the *opt set* and the *unseen set*. The plot on the right is a zoom in on the left part of the plot on the right, i.e., the top 10 most frequent goods in the *opt set*.

9.3 Estimation of simulator parameters

This section explains the parameter values used in the simulator, i.e., the picking times and the warehouse layout parameters.

Picking times

The picking times for goods were approximated to a deterministic value. However, in reality, the picking times are stochastic. They depend on the workers' experience, the time of the day, and possibly other factors. Kolonial.no made the deterministic approximation of the picking times. An important consideration done during the approximation is worth mentioning. The distribution of picking times for a given good is often right-skewed. This happens because, in reality, a worker sometimes has responsibility for more than one station. Consequently, the worker may not have been present at the station when the picking time counter began for a good. Therefore, the deterministic values were approximated using the mean of the picking times calculated from the quickest 30% of the distribution.

The picking times do not increase linearly with the quantity of goods to be picked. Table 9.3 shows an example of this, see good 9281. The table also shows the picking time of good 26500. The picking time of this good is unexpectedly large. This has been controlled with Kolonial.no. According to them, instead of 1451.4 seconds, less than 25 seconds is an expected picking time for this good and quantity. Such extreme picking times are a consequence of few and inaccurate recordings, cf. picker absence described above. Preliminary testing showed that these outliers had a major effect on the overall makespan and goods assignment. These were thus corrected, and max picking times were introduced, see Table 9.4. The max picking times were found for each quantity column. The max values set were the max picking time per quantity column with a count higher than 9.

Table 9.3: *The table shows two examples of provided picking times. The table shows the picking times of a good and number of units. The table also shows the number of times a picking time is recorded, i.e., count columns. The picking times were provided with float precision, and are rounded in this table. Good 9281 is "Miljøgartneriet Agurk, Stor Norsk 1.00 stk" and good 26500 is "Freia Regia Kakao 2 x 1kg 2.00 kg".*

id	1 qty	count	2 qty	count	3 qty	count	4 qty	count	≤ 5 qty	count
9281	5.3 s	9445	6.6 s	2239	8.3 s	494	9.2 s	228	13.6 s	281
26500	1451 s	1	NaN	0	NaN	0	NaN	0	NaN	0

Table 9.4: *The table shows the max picking times used.*

1 qty	2 qty	3 qty	4 qty	≤ 5 qty
25	35	35	63	67

Figure 9.3 show the distribution of the picking times after corrections. The plot shows how similar the picking times are and how little they vary despite being different goods and quantities.

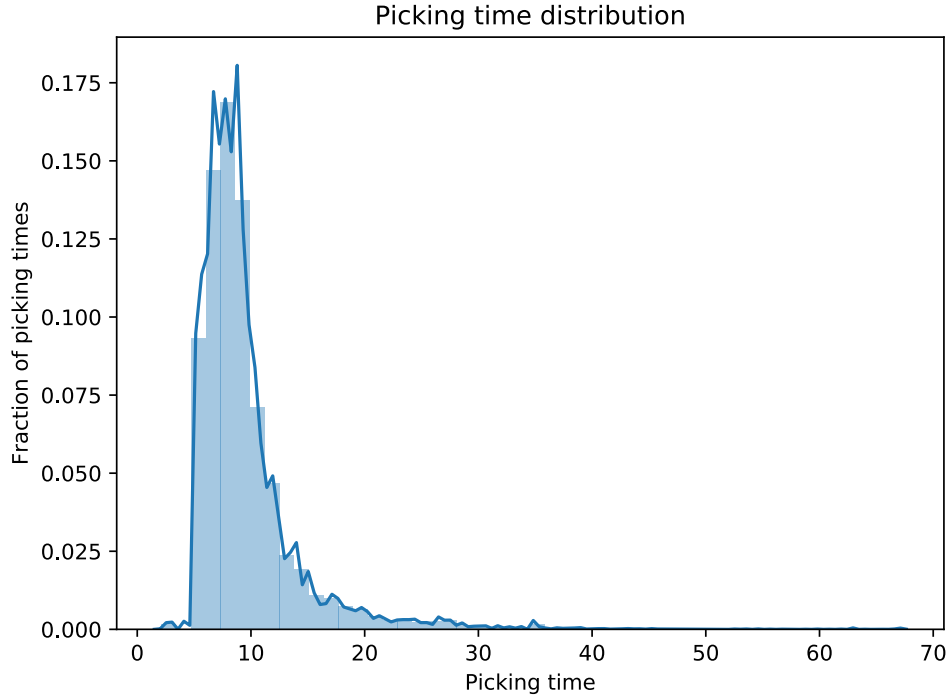


Figure 9.3: *The plot shows distribution of picking times of the picks of the boxes. Picking of k units of a good for a box, is addressed as one pick.*

Warehouse layout

The warehouse has 26 stations. There are 4 or 5 buffer nodes at each station. There is 1 node between each station, except between station 13 and station 14, where there are 40 nodes. The traverse time between two nodes is 1 second. The stations have width capacities of either 2160 cm or 2628 cm. The total width of the goods is about 64% of the total width capacities of the stations.

9.4 Calibration of genetic algorithm parameters

The genetic algorithm presented in Chapter 6 uses a set of input parameters. To get the best performance of the genetic algorithm, these parameters must be calibrated. The goal is to find a set of parameters that has a good trade-off between exploration and exploitation, such that good solutions may be found while avoiding premature convergence. Subsection 9.4.1 presents the different parameters and their final values, while Subsection 9.4.2 presents the calibration process.

9.4.1 The parameters of the genetic algorithm

The genetic algorithm consists of 10 parameters and is listed in Table 9.5 with their calibrated or chosen values. The population size ω determines the number of chromosomes in the population. An increase in the population size will increase the computation time for the algorithm stepwise linearly, as the parallelized implementation allows for evaluating several chromosomes simultaneously. The mutation rate p_m , the

number of swaps π , and the crossover rate p_c control the degree of exploration. The crossover operator δ can either be uniform or one-point, but the right operator for this problem is difficult to determine without test runs. The tournament P p_t and the tournament size τ_k adjusts the selection operator. Together with the elite number μ , they modify the selection pressure or the exploitation of good solutions. γ is a scaling factor used in the adaptive penalty function, as described in Section 6.3. Lastly, the algorithm has two termination criterias; a termination threshold ψ and a maximum run time.

Table 9.5: *The table shows the notation for each parameter, its final calibrated/chosen value and a brief description the parameters*

Parameter	Value	Name	Description
ω	100	Population size	Number of chromosomes in every generation
p_m	65%	Mutation rate	The probability that the algorithm will conduct a swap mutation on a chromosome
π	2	Number of Swaps	Number of genes that swap position during a swap mutation on a chromosome
p_c	85%	Crossover rate	The probability that crossover is performed on two parent chromosomes
δ	Uniform	Crossover Operator	The method in which the crossover is happening on the chromosomes
μ	1	Elite number	Number of best solutions that are passed on to the next generation
p_t	95%	Tournament P	The probability that the best competitor in a tournament wins
τ_k	3	Tournament size	Number of chromosomes randomly picked to compete in the tournament selection
γ	100000	Generation scaling	The degree of linear increase in penalty for exceeding width capacity
ψ	300	Termination threshold	Number of generations after best-found solution before the algorithm is terminated

9.4.2 Parameter calibration

The following subsection presents the calibration process of the parameters. The parameters are tested systematically through 166 tests. Due to limited computational resources with the same hardware and software specifications, computers with 3 different specifications were used in the calibration. The specification details are described at the beginning of this chapter. All calibration test runs that are compared with each other are run on computers with the same specifications. We did not test all combinations of parameters, as this would demand too much time and computational resources. As an example, testing all combinations of 5 parameters, each with 4 different values requires $4^5 = 1024$ runs. Besides, several runs of each combination are required because the GA is non-deterministic.

Instance size for parameter calibration

When conducting the parameter calibration, it is desirable to use a small test instance as possible. This can reduce the computation time of the GA and allow evaluations of more parameter combinations. However, if the test instance is too small, the calibrated parameter values may not be scaled to larger instances.

Figure 9.4 shows the total makespan divided by the number of boxes used for Kolonial.no's goods assignment. The plot illustrates that the relative number, makespan divided by the number of boxes, converges at around 685 boxes. Three different random goods assignments showed the same trend. Figure 9.5 shows that an instance of 685 boxes represents around 70% of all goods in the dataset.

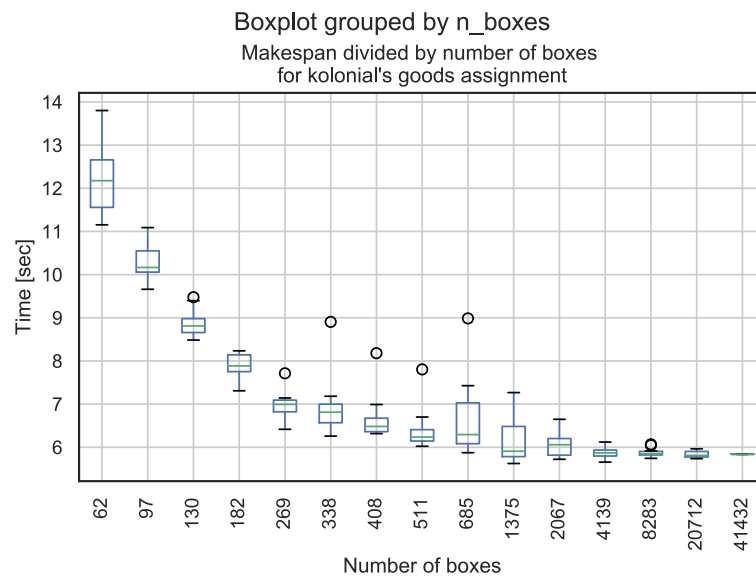


Figure 9.4: The plot shows the makespan time (warehouse time) divided by number of boxes for Kolonial's goods. For every number of boxes, ten different box samples were made.

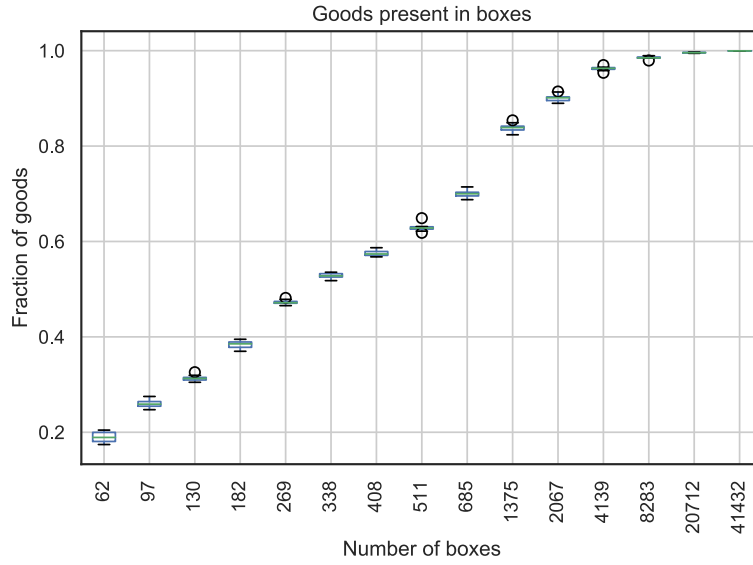


Figure 9.5: *The plot shows the fraction of unique goods in boxes. For every number of boxes, ten different box samples were drawn.*

Based on these findings, it is believed that it is enough to use 685 boxes for the parameter calibration. This is also backed by the fact that preliminary testing indicates that the calibrated parameter values on 685 boxes also performs well on larger instances. To operate with more rounded sizes, an instance size of 700 was then chosen for the calibration.

Range testing

To get a sense of the range in which to test the different parameters, a series of range tests were done on the instance size of 700 boxes. The test runs varied the values of ω , p_m , π and p_c . The rest of the parameters was fixed. Table 9.6 shows the parameter values and results.

Table 9.6: *The table shows the parameter values and results for all the preliminary test runs*

Makespan [s]	ω	p_m	π	p_c	δ	μ	p_t	τ_k	γ	ψ
3503	100	0.60	8	0.70	One-point	1	0.95	3	5K	10
3531		0.75								
3550		0.95								
3457		0.60		0.85						
3493		0.75								
3559		0.95								
3292	200	0.60	2							
3264		0.75								
3384		0.95								
3197		0.60								
3270		0.75		0.85						
3320		0.95								
3329	300	0.60	4	0.70						
3392		0.75								
3382		0.95								
3337		0.60		0.85						
3383		0.75								
3394		0.95								
3387	400	0.60	6	0.70						
3456		0.75								
3485		0.95								
3372		0.60		0.85						
3440		0.75								
3497		0.95								

The preliminary range test shows that the genetic algorithm with number of swaps $\pi = 2$ finds solutions with makespan time that are on average 2.4%, 4.4% and 6.5% better than $\pi = 4$, $\pi = 6$ and $\pi = 8$, respectively. This indicates that a lower number of swaps should be tested in the calibration runs. Furthermore we observe that runs with $p_m = 0.60$ gives better solutions compared to similar runs where just p_m is different. Runs with $p_m = 0.60$ gives on average 1.3% and 2.5% better solutions compared to $p_m = 0.75$ and $p_m = 0.95$, respectively. The effect of different crossover rates p_c are not very distinguishable, but the algorithm that found the best solution had a $p_c = 0.85$. Different population sizes were chosen in the preliminary range testing to see if we could observe any significant improvement in performance related to the increase in population size. Larger population size should, in theory, weakly improve performance. In these test runs, however, the number of swaps π seemed to dominate the overall impact on the algorithm's performance. No ideal range for population size calibration was therefore detected in these test runs.

Calibrating population size

Due to restrictions in computational resources, performance relative to computational time is a critical measurement when selecting the right population size. Roeva et al. (2013) conducted a research for a parameter identification problem in case of Cultivation Process Modelling. They tested different population values and measured the performance with respect to computational time. They tested population sizes between 5 and 200 chromosomes and found that the optimal population size was 100 chromosomes. Based on the insight from this research, we chose to evaluate the performance of the GA with both 100 and 200 in population size. Nine different combinations of parameter values were tested for each of the different population sizes. Table 9.7 shows in detail the test runs for the population size calibration.

Table 9.7: *The table shows the parameter values that was used to evaluate the performance of population size 100 and 200*

ω	p_m	π	p_c	δ	μ	p_t	τ_k	γ	ψ
100/200	0.50	1	0.85	Uniform	1	0.95	3	100K	200
	0.65								
	0.80								
	0.50	2							
	0.65								
	0.80								
	0.50	3							
	0.65								
	0.80								

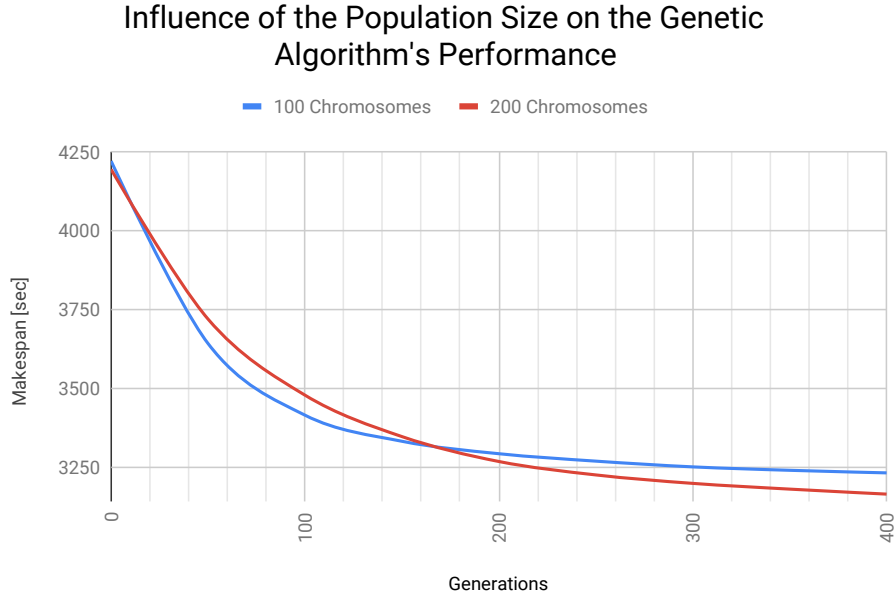


Figure 9.6: *The plot shows the performance of the algorithm with a population of 100 and 200 chromosomes*

The plot in Figure 9.6 displays the average best solution for the nine different test runs at each generation. The results indicate that the algorithm most likely will find better solutions with a population size of 200. However, computation time has to be considered. For the test runs, the algorithm uses on average 51.5 seconds per generation with $\omega = 100$ and 97.1 seconds per generation with $\omega = 200$. The computation time used when running the algorithm goes mostly to evaluate the chromosomes through the simulator. That is why the computational time per generation is almost twice as long for $\omega = 200$ compare to $\omega = 100$.

When choosing the best population size to solve the problem for larger instances, the maximum run time of the algorithm needs to be taken into account. We see from the figure that $\omega = 200$ gives strictly better results than $\omega = 100$ after about 170 generations. If maximum runtime allows the GA to go beyond 170 generations with 200 in population size, then $\omega = 200$ is the right parameter value. Preliminary test runs show that the computation time per generation is about 1300 seconds for a full week instance using $\omega = 100$. The same test run for $\omega = 200$ takes about 2400 seconds. When conducting the main runs for the genetic algorithm, resource restrictions gives a maximum run time of 302400 seconds. This implies that the algorithm would not reach more than 127 generations on a full week instance before termination, making $\omega = 100$ the best choice for the main test runs.

Calibration of mutation rate and number of swaps

To calibrate the mutation rate and the number of swaps, 3 different values of mutation rate was tested in combination with 3 different values of swap number. These values are based on the insight from the range testing before the calibration. Since the algorithm is non-deterministic, 10 runs of each parameter combination were tested

Table 9.8 shows all the parameter values tested and the average results of each parameter combination. The best results found was with a $p_m = 0.65$ and with $\pi = 2$. However this shows only marginally better results on average compared to $p_m = 0.80$ and $\pi = 1$. Since values close to $p_m = 0.65$ and $\pi = 2$ gave significantly better results in the range testing, these parameter values was chosen for the main runs.

Table 9.8: *The table shows the parameter values that was used to calibrate mutation rate and number of swaps*

Makespan [s]	ω	p_m	π	p_c	δ	μ	p_t	τ_k	γ	ψ
3414	100	0.50	1							
3272		0.65								
3250		0.80								
3298		0.50								
3249	100	0.65	2	0.85	One-point	1	0.95	3	10K	300
3284		0.80	3							
3269		0.50								
3298		0.65								
3329		0.80								

Calibrating crossover rate

When calibrating crossover rate, the calibrated population size $\omega = 100$, mutation rate $p_m = 0.65$ and number of swaps $\pi = 2$ was used. 3 values for the crossover rate were tested. Due to the non- deterministic algorithm, 10 test runs of each parameter combination was conducted. Table 9.9 displays the average result for each parameter combination. The calibration implies that a crossover rate $p_c = 0.85$ would lead to better results for the algorithm.

Table 9.9: *The table shows the parameter values that was used to calibrate crossover rate*

Makespan [s]	ω	p_m	π	p_c	δ	μ	p_t	τ_k	γ	ψ
3341	100	0.65	2	0.45	One-point	1	0.95	3	10K	300
3286				0.65						
3249				0.85						

Calibrating crossover operator

To test how the two different crossover operators performed, runs with a variety of parameter combinations were conducted. Table 9.10 shows in detail the parameter combinations tested along with the algorithms results. The best solution found with one-point crossover was a makespan of 3286.00 seconds while the best solution using uniform crossover was 3156.65 seconds. On average for all the parameter combinations, uniform crossover performed 4.24% better than one-point crossover. Hence, uniform crossover was chosen for the main test runs. This is a finding consistent with the suggestion of Eiben and Smith (2015); that uniform will give the best results when the order of the genes in the representation are random.

Table 9.10: *The table shows the parameter values that was used to find the best crossover operator*

Makespan [s]	ω	p_m	π	p_c	δ	μ	p_t	τ_k	γ	ψ
3531	100	0.50	1	0.85	One-point	1	0.95	3	100K	200
3425		0.65								
3291		0.80								
3322		0.50	2							
3286		0.65								
3305		0.80								
3289		0.50	3							
3296		0.65								
3342		0.80								
3214		0.50	1							
3185		0.65								
3194		0.80								
3191		0.50	2							
3169		0.65								
3209		0.80								
3156		0.50	3							
3216		0.65								
3274		0.80								

Calibration of Generation scaling

Table 9.11 shows in detail the parameter values used for the calibration of generation scaling. To calibrate the generation scaling value γ , it is essential to see how the penalty weight reduces solutions with width exceedance over the course of many generations. The goal is to find a γ -value that slowly decreases the number of width-exceeding solutions as the generations increase. It is not favorable that the number of width exceeding solutions decreases too fast.

Table 9.11: *The table shows the parameter values that was used to evaluate the penalty weight of generation scaling*

ω	p_m	π	p_c	δ	μ	p_t	τ_k	γ	ψ
30	0.65	2	0.85	Uniform	1	0.95	3	5K	200
								10K	
								20K	
								50K	
								100K	

In section 6.3 it is explained why it is essential for the algorithm to search outside the feasible area, and we want this to happen throughout the whole run of the algorithm. Ideally, the generation scaling will allow the algorithm to evaluate solutions with width exceedance even in the last hundred generations before termination. Figure 9.7 shows the number of solutions that are penalized because of width exceedance for every generation. We note that only $\gamma = 100K$ will allow for infeasible penalized solutions beyond generation 11 and even as far as to generation 966. This suggests that γ of at least 100K is the only viable option. One reason why very few chromosomes get penalized after generation 11 is that GA tends to distribute goods among stations evenly. Because of this even distribution, very few stations get their capacities overstepped, and the soft constraint is very often unbounded.

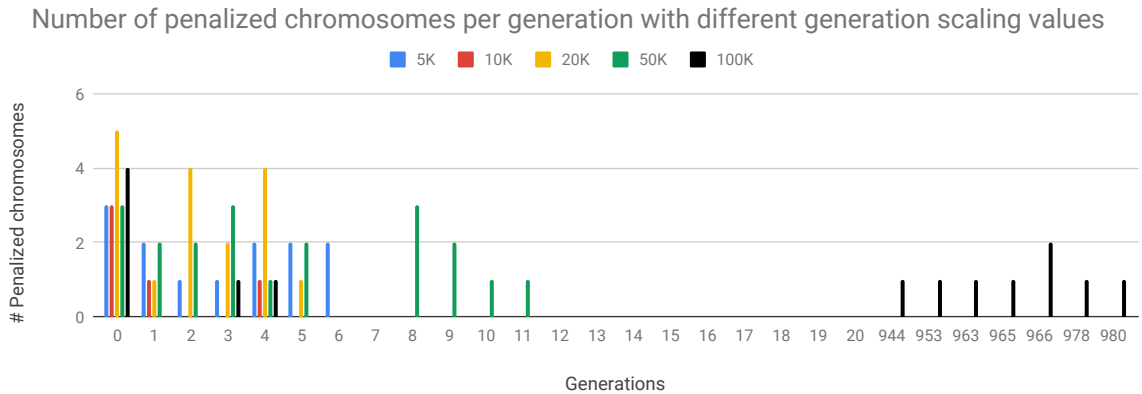


Figure 9.7: *The plot shows how many chromosomes that are penalized for every generation.*

Choosing Elite number

An elite number $\mu = 1$ was chosen at the early stages of the calibration testing. The reason for this was that the problem at hand only requires one great solution, and it was preferable to pass this on to the next generation. By keeping μ low, we also allow for more exploration. With more computational resources available, one could test the influence on the algorithm's performance with a larger elite number.

Choosing tournament selection parameters

Eiben and Smith (2015) mentions that p_t usually has the value of 1. The probability that the fittest chromosome in the tournament is selected can also be less than 1 as well. For this research, 0.95 has been chosen in particular to give fewer fit chromosomes a chance of being selected. τ_k determines the tournament size, and the selection pressure increases with larger tournament sizes. Preliminary runs show that $\tau_k = 3$ works well for the genetic algorithm, and was chosen for the main testing.

Choosing termination threshold

Previous calibration runs indicate that a termination after 200 generations without new best solution might be sufficient. However, as Figure 9.8 shows, some runs benefit from a larger termination threshold. The plot showing the elite development to the left, has a significant improvement after more than 270 generations without a new elite solution. Termination threshold of 300 was chosen for the main test runs.

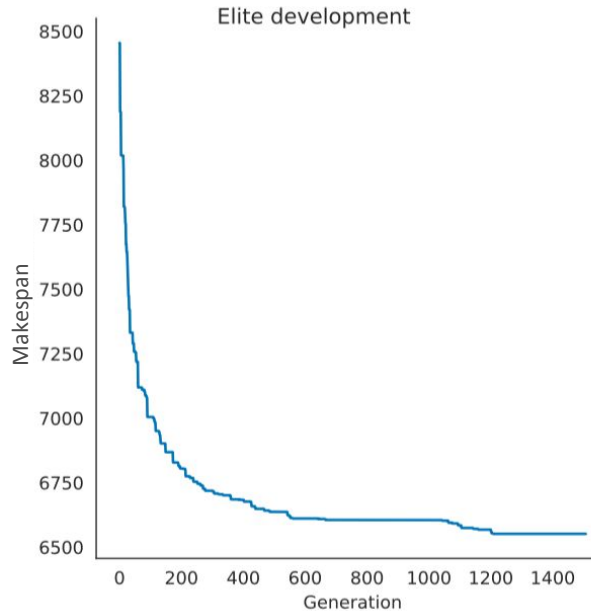


Figure 9.8: The figures show that there can be improvement of elite solution after more than 200 generations with new elite

9.5 Significance of the instance size

After the calibration of the GA parameters, the best goods assignment possible for the *opt data* was found. Different subsets of the *opt data*'s boxes were used. The different subsets had different cardinalities. The cardinalities were chosen based on the average number of boxes per shift; 1 day, 2 days, 1/2 week, 3/4 week, and 1 week. The makespans of the best goods assignments found can be seen in Table 9.12. The table shows that, on average, a lower makespan can be found using subsets of higher cardinality. The table also shows that goods assignments that are preferred on the *opt data* are often better on the *unseen data*, but not strictly better. The columns *Number of Gen.* and *Comp. Time* illustrate that box subsets of lower cardinality finish earlier than those of higher cardinality.

Table 9.12: *The table shows the simulated makespan times with the best goods assignments found using a different number of boxes. The table also shows the number of generations and computation time. * symbolizes that the computation time limit terminated the GA run, and not by the generation threshold ψ . The hardware and software specifications for each rack is described at the beginning of this chapter.*

Number of boxes	Opt Data [s]	Unseen Data [s]	Number of Gen.	Comp. Time [s]	Rack used
1465	105099	107546	2156	205465	4
1465	103381	107537	1050	93632	4
1465	105981	109731	1146	103573	4
1465	102714	105699	2535	251202	4
1465	102445	105746	1020	90432	4
2930	104138	105937	1829	302533*	4
2930	103679	105706	1819	302432*	4
2930	103313	106836	1172	189643	4
2930	102662	104928	1842	302406*	4
2930	103131	106264	1831	302415*	4
5861	102211	106555	1007	302428*	4
5861	101140	106136	1015	302703*	4
5861	101402	104693	1026	302589*	4
5861	101749	106055	1023	302650*	4
5861	101252	105060	1020	302512*	4
10258	99750	104621	710	302742*	5
10258	98712	104069	707	302604*	5
10258	99899	105313	701	302745*	5
10258	100450	104692	697	302630*	5
10258	99026	103560	698	302680*	5
15386	98286	105673	479	302801*	5
15386	98921	104165	482	303011*	5
15386	97352	104242	484	302693*	5
15386	97269	104365	483	302511*	5
20513	96030	104815	238	302594*	1
20513	96015	104277	241	303522*	1
20513	95219	103800	233	303459*	1

9.6 Performance of the algorithms

To evaluate the performance of both algorithms, they are compared with the current goods assignment at Kolonial.no. The measurement used for comparison is makespan, which we want to minimize. To get the makespan, all goods assignments are run through the simulator. The simulator evaluates the solutions using the same sequence of boxes. Table 9.13 shows the best results of both algorithms for the *opt. data* and the *unseen data*. The table clearly states that both presented algorithms outperform Kolonial.no's current solution. The GA provides the best goods assignment. The GA's solution provides an improvement of 14.8% and 9.6% over Kolonial.no's solution on the *opt. data* and *unseen data* respectively. The greedy algorithm's solution provides an improvement of 6.0% and 6.0% respectively.

Table 9.13: *The table shows the simulated makespan using the goods assignments from the different algorithms. The makespan times have been rounded to the nearest second. The table also shows the computation time of the algorithms.*

	Comp.time [s]	Makespan(Opt. data) [s]	Makespan(Unseen data) [s]
Kolonial.no	NaN	111751	114839
Greedy	0.04	105084	107990
GA	302400	95219	103799

Figure 9.9 shows that for the *opt. data*, the GA on average, outperforms the greedy algorithm after a computation time of 9.5 hours. For all three runs, the computation time needed to outperform the greedy algorithm varies from 7.5 to 10 hours. In comparison, the greedy algorithm is done in 0.04 seconds. This means that if you can allow a computation time of more than 9.5 hours, then the GA is the preferred algorithm, otherwise use the greedy algorithm. Compared with Kolonial.no's solution today, the GA on average finds solutions with lower makespan than Kolonial.no's after about an hour.

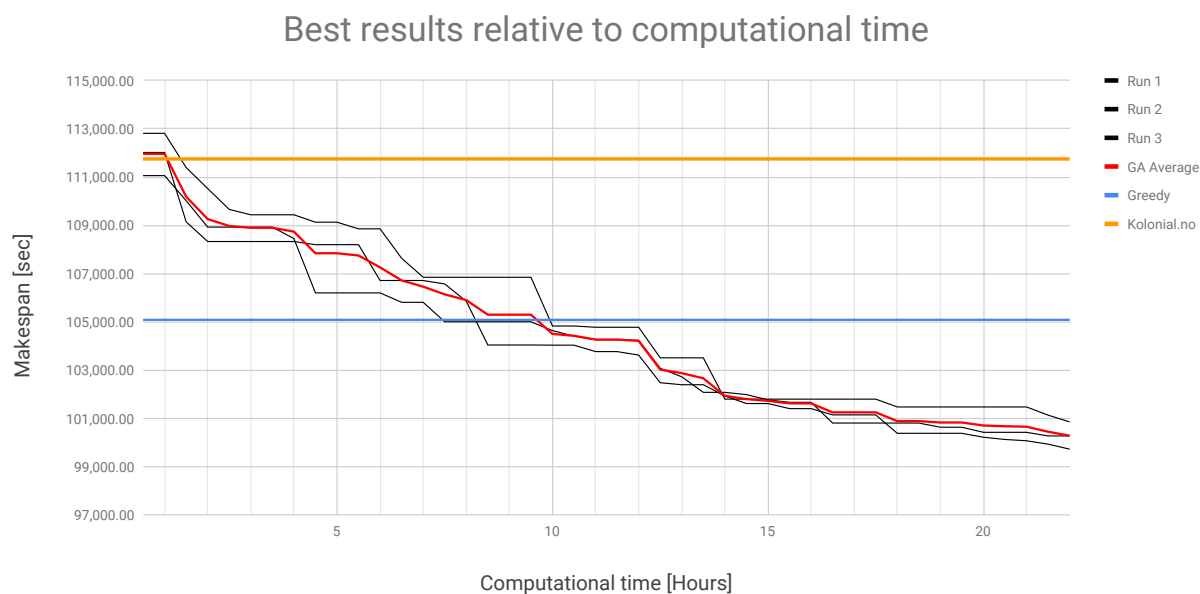


Figure 9.9: The plot shows the best solution for 3 runs of GA relative to computational time. The instance size is 1 week.

Figure 9.10 shows that the GA needs on average 27 generations to outperform the greedy algorithm. Compared with Kolonial.no's solution, one needs only a couple of generations with the GA to surpass the company's current solution. Furthermore, new solutions are continuously found throughout all three runs. This indicates that the GA does not fall short for premature convergence.

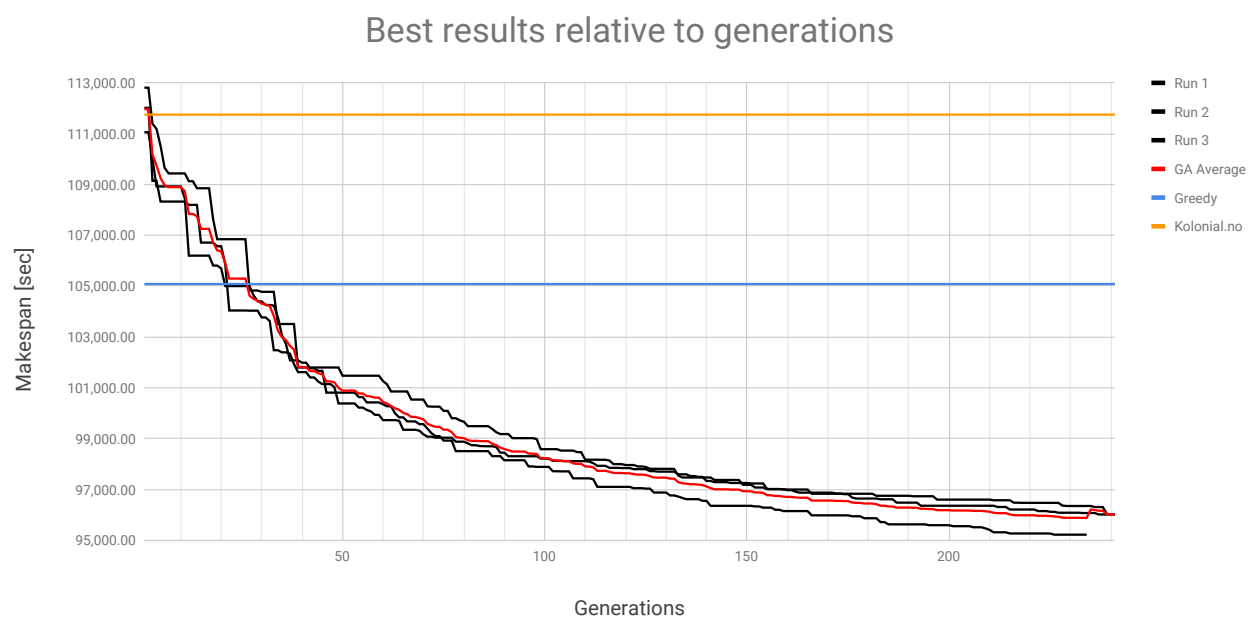


Figure 9.10: The plot shows the best solution for 3 runs of GA relative to generations. The instance size is 1 week.

9.7 Comparing Solutions

This section compares the best solution of the genetic algorithm with the best solution from the greedy and Kolonial.no's current goods assignment. Comparison is done with respect to balancing of the workloads and the queues along the assembly line.

9.7.1 Balance of the workloads

Figure 9.11 shows the total workload per station for different solutions. The solution viewed is from the *opt data*. The orange line shows the distribution of the assignment of the goods that Kolonial.no uses today. It is evident that the workload varies greatly between the stations. The picker at station 13 has about twice the workload as the picker at the first station. On the other hand, the stations are perfectly balanced in the assignment that the greedy algorithm found. This is as expected as the greedy choice is to balance the workloads. The best solution from the genetic algorithm is shown in green in Figure 9.11. This is not as balanced as the greedy solution, but significantly more balanced than the current solution of Kolonial.no. It is interesting to note the difference in workloads and the total makespan. A picker is working on average 67510 seconds during the processing of the boxes in the *opt data*. With Kolonial.no's current goods assignment, the processing of all the boxes takes 111751 seconds. This means that the average picker was idle 39.6% of the time. With the best solution from the genetic algorithm, the average idle time is reduced to 29.1%. However, there is still a significant proportion of idle time on the assembly line. Note also that the average picking time per station works as a lower bound for a given set of boxes. It is not possible to finish before the work at all stations is done. The workload at the station with the most workload can never be lower than the average workload of the stations.

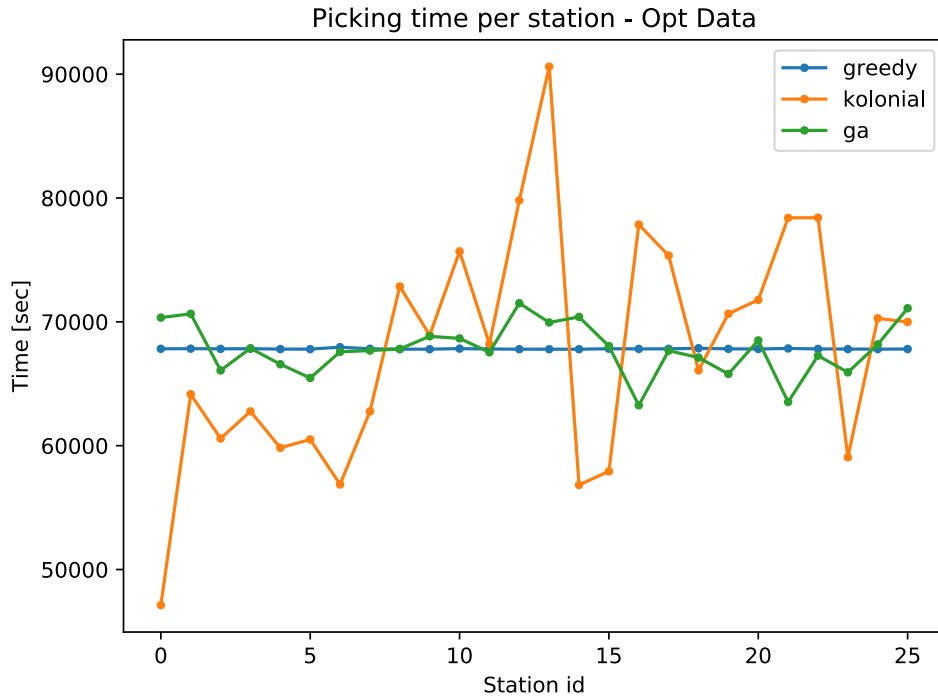


Figure 9.11: Picking time per station on the training set

Another interesting observation from Figure 9.11 is that the best solution is not the most balanced. Pan et al. (2015) used workload balancing as the objective in their genetic algorithm, but this doesn't seem to be the best for our problem, as our genetic algorithm gives better results than the greedy algorithm. The problem of aiming for a total balancing is that the workload might not be balanced at all underway in the process. Figure 9.12 shows the distribution of workloads for the first 100 boxes on the assembly line. The Figure illustrates how the workloads are not at all balanced for these boxes. This indicates that if balancing is to be used as an objective, the balance should be measured on boxes that are on the assembly line at the same time, instead of a total balance. Further, it might be necessary to change the box sequence or allow goods at multiple stations to further balance the workload among boxes together on the belt. Consider the edge case where a sequence of boxes requests solely one good. It is not possible to balance the workload among stations with goods only located at a single station. This would require changes to the box sequence or goods at multiple stations to distribute the workload among stations.

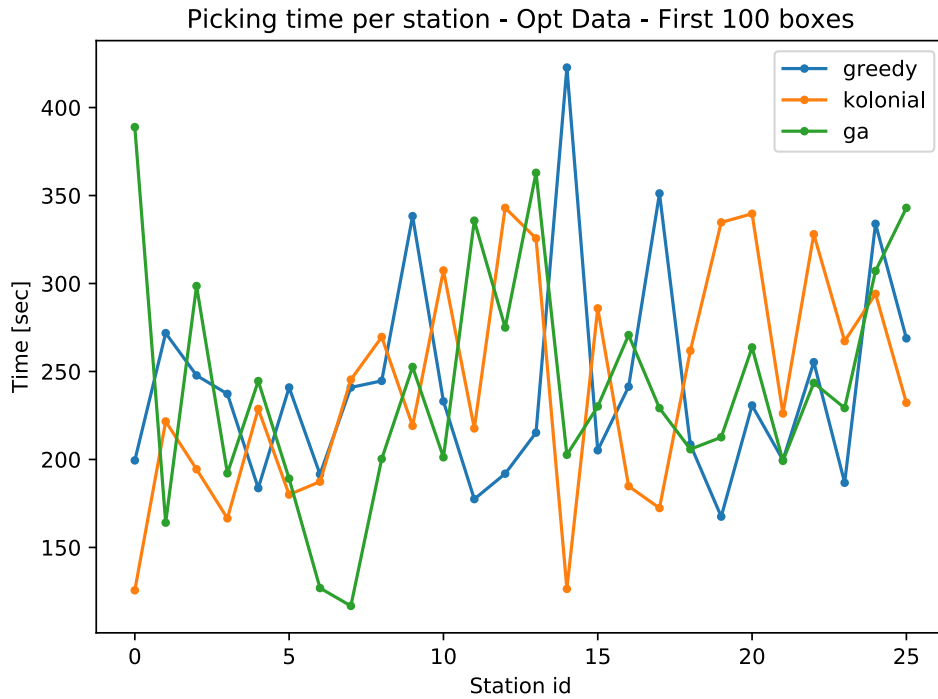


Figure 9.12: *Picking time per station on the training set*

Figure 9.13 shows the total workload per station on the *unseen data*. Most notably, the greedy algorithm now has uneven workloads. The solution of the genetic algorithm also fluctuates more than before, but it is still well balanced in comparison with the current solution.

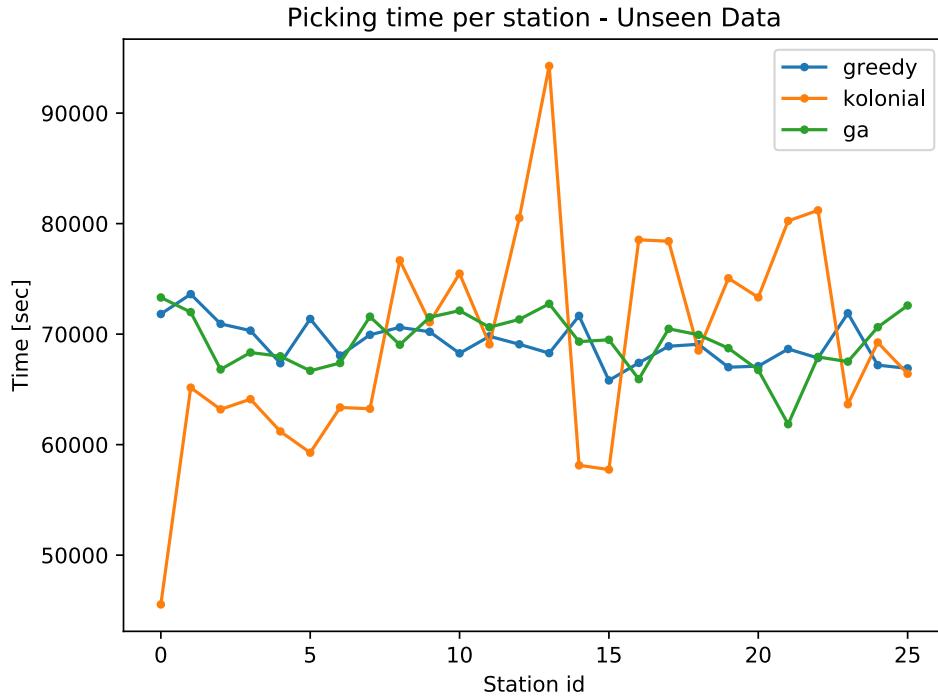


Figure 9.13: *Picking time per station on the test set*

9.7.2 Queues along the assembly line

As previously observed, the pickers still experience significant idle times. This has to do with problems with the flow along the main line. Therefore, it is of interest to investigate the queues that occur. Figure 9.14 illustrates the queues for the different solutions of GA, greedy, and Kolonial.no. The figures on the left show the total time spent in queue at the choice nodes. We remind the reader that a choice node is just outside a station, so a congestion is either because of queue on the main line or because the buffer zone is full. Where Kolonial.no's assignment looks more disordered, there seems to be a trend in the assignments of the GA and the greedy. There is a lot of queue at the first station of each aisle on the assembly line, i.e., node 1 and node 187, and the queue declines along the aisle. The spike in queue at the first station might be because new boxes are fed onto the assembly line as soon as there is available space. Therefore, because queues propagate backwards, there is an accumulation of queue at the first station.

The figures on the right show the total time boxes spent waiting at each node. The spikes show the queue time at the patient nodes, i.e., the nodes where the goods are picked. Observe that the spikes are higher than the total workload at each station, see also Figure 9.11. This means that boxes occupy the patient node even when they are not being processed. This happens when there is a full stop on the main line outside the station. The box is then prevented from leaving the station, and the picker is prevented from starting to process the new box in the buffer zone. This is part of the reason for the high idle-times. Particularly, for the first station in the GA-solution, this is the main reason for the idle time. The total time boxes spend

waiting at this station's patient node is 87045 seconds. The workload at this station is around 70342 seconds. Considering that the total makespan for this solution is 95219 seconds, the blockage makes for 67% of the idle time and 17% of the time at work.

The highest spikes are at patient nodes of the first station for the GA and greedy solution and at the patient node of the second station for Kolonial.no's solution. The highest spikes are respectively 87045, 91770 and 100821 seconds. Note that the makespan can never be lower than the value of the highest spikes. The algorithm is consequently also lowering the value of the highest spike when it is searching.

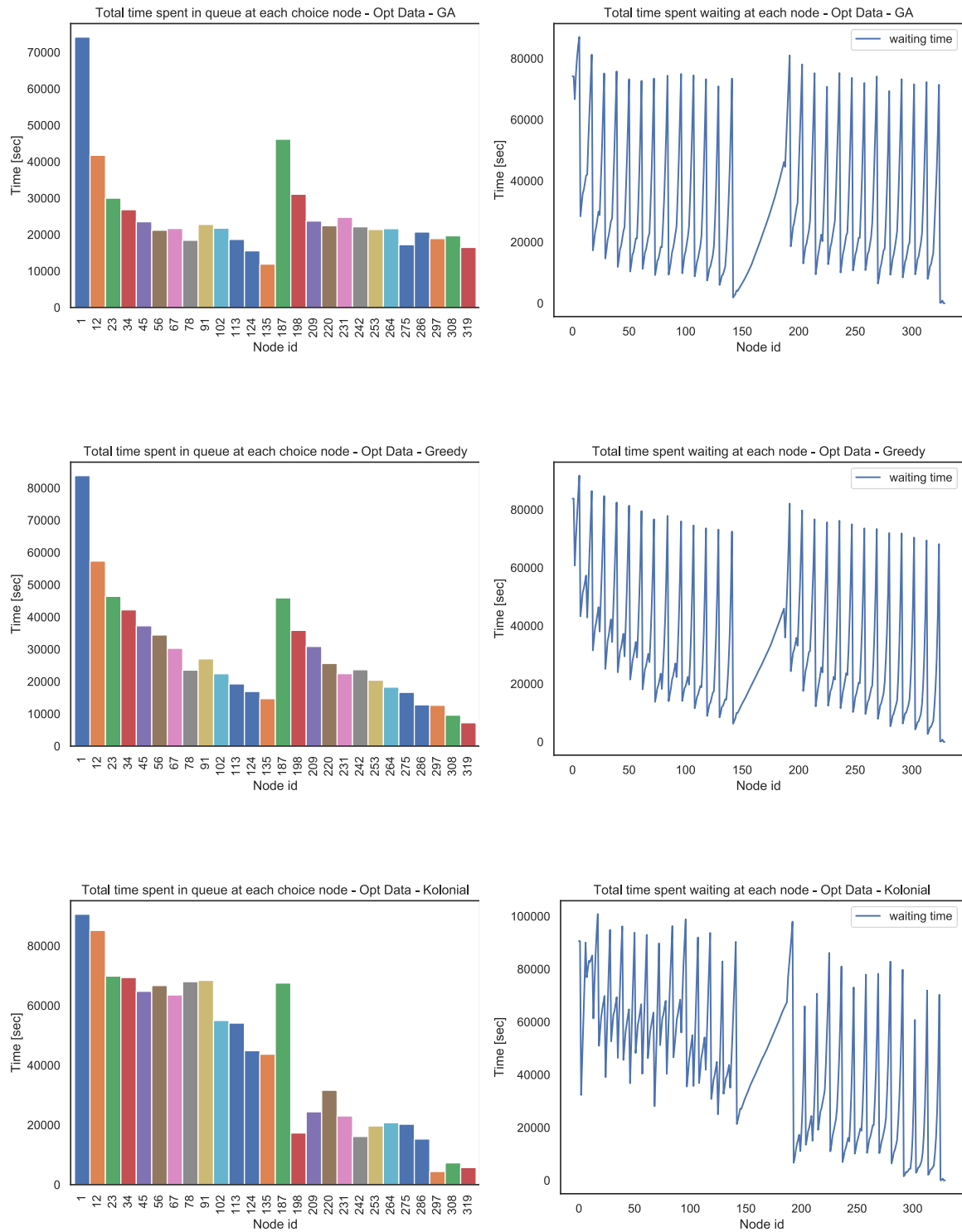


Figure 9.14: The left plots shows the total queue experienced on the choice nodes. See Figure 5.1. The plots on the right illustrate how long boxes spent standing still at a node.

9.7.3 Distribution of goods

Figure 9.15 shows that about 60% of the goods describe 90% of the total picks of the boxes, and 10% of the goods is sufficient to describe 50% of the picks. This indicates that a few goods dominate in the dataset. This is also seen in Figure 9.2. From this, a hypothesis was formed: In a good solution, the most frequent goods are not assigned to the same stations, in order to avoid queues on the main line. We'll, therefore, look into if this is the case for our solutions.

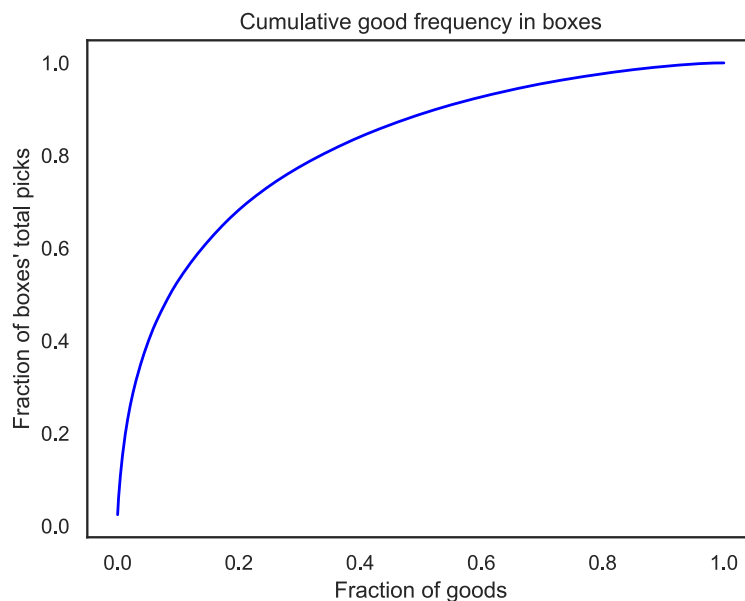


Figure 9.15: *The plot shows the cumulative frequency of the goods in the boxes, i.e. how much of the goods is needed to describe the boxes' total picks. The goods are ordered from most frequent goods to least frequent goods.*

Table 9.14 lists the 26 most frequent goods in descending order. The three rightmost columns show where they are placed in the solutions of the greedy, the GA, and Kolonial.no's current solution. In the greedy solution, none of these goods are placed on the same station. This is because the greedy algorithm places goods out based on their frequency. For Kolonial.no's current solution, 15 of the 26 most common goods share a station with another good in the top 26. For the GA-solution, this number is 12. This is a considerable number that means that the hypothesis is incorrect for this particular solution.

Table 9.14: *The table shows where the top most frequent goods, of the training set, are located in the different solutions.*

Good id	Frequency train	Total picking time - train [s]	Greedy location	GA location	Kolonial location
9281	4591	26429	25	25	15
9329	2830	29299	24	11	12
9407	2127	14222	23	23	22
19432	1725	13183	22	20	16
9404	1694	11110	21	22	18
26941	1663	13314	20	17	13
26694	1538	8973	19	20	25
9320	1505	8180	18	9	19
27016	1456	12390	17	2	20
27626	1432	10514	16	19	24
27622	1302	10195	15	8	12
27248	1283	7381	14	8	17
27241	1265	13764	13	12	25
1035	1179	13236	12	3	16
1768	1124	16032	11	7	17
24854	1074	11257	10	15	21
11596	1044	7402	9	3	11
632	1024	7387	8	22	10
7831	1007	12153	7	14	1
23319	983	11209	6	14	2
27145	950	9276	5	0	23
2971	945	10881	4	24	16
16284	932	8134	3	10	11
19708	896	5654	2	5	14
8488	894	5056	1	6	20
16281	835	12755	0	0	10

9.8 Uniqueness of the best solution

The following section investigates whether there exist several different goods assignments that are good or if there is one particular goods assignment that is superior.

Within the same run of the genetic algorithm, there are many similar solutions. This is because they stem from the same ancestors. However, it is of interest to see if solutions from different runs of the GA are similar or not. Consequently, we examined the best solution from each of the three runs of the genetic algorithm for a full-week instance. These solutions are pretty close with regards to makespan; only 0.8% distinguish the best from the worst. Table 9.15 shows the comparisons. The number of equal goods at equal stations are at best 85. This equals only 4.3% of the goods. This indicates that there might not be one optimal location for a good. However, there might still be collections of goods at a station that are superior to others. The rightmost column of Table 9.15 shows the number of common goods at the two most similar stations. At best, two stations have 10 goods in common. This is on average 12.3% of the goods at a station. Consequently, this examination indicates that the

best solution is not unique, but that several different goods assignments can perform equally good.

Table 9.15: *The table compares three solutions with regards to similarity. The three solutions are from different runs of the GA. The solutions 1,2 and 3 have a makespan of 96030, 96015 and 95219 respectively.*

Solutions	Number of equal goods at equal stations	Most similar stations	Number of common goods at most similar stations
(1,2)	80	11 and 20	10
(1,3)	71	24 and 25	10
(2,3)	85	6 and 16	10

It is also interesting to investigate whether the order of the stations matter. If this is of little importance, there is a high degree of symmetry in the solution space. The test is performed by switching all the goods at a station with the goods at another station and see how much it affects the makespan. This is a compelling case to examine, as it gives an indication of the uniqueness of a solution, but also on whether it is the content of the stations or their order that matter most.

Table 9.16: *The table shows the relative increase in makespan when manipulating the order of the stations.*

Manipulation method	Relative increase in makespan
Switch station 13 and 14	0.80%
Switch station 1 and 26	1.99%
Reverse	3.64%
Sort by workload	4.19%

Table 9.16 shows the results of some manipulations done on a goods assignment. *Switch station 13 and 14* switches the goods content at station 13 and 14 and keeps everything else unchanged. The particular station numbers are chosen at random. By simulating this new goods assignment, it appears that it is 0.80% slower than the original goods assignment. This amounts to less than 2 minutes every day. Switching the first and last station, however, gives a relative increase of 1.99%, which is about 4 and a half minutes every day. We also note that switching more than two stations has a larger effect on the makespan. *Reverse* keeps the goods at each station unchanged but reverses the order of the stations. This gives a 3.64% increase in the makespan, which is a daily increase of about 8 minutes. *Sort by workload* sorts the stations by their workload so that station 1 has the least workload and station 26 has the highest workload. It may be that it is better to have the highest workload late in the assembly line because potential bottlenecks won't affect the subsequent stations. However, this gave a relative increase in makespan of 4.19%, which amount to almost 10 minutes extra each day. Consequently, it doesn't seem like it is beneficial to keep the busiest stations last on the assembly line. To sum up, it seems that the order of the stations does not cause a high degree of symmetry in the solution space, as the order of the stations appears to have an effect.

9.9 The value of shorter processing times

As mentioned in Chapter 2, Kolonial.no needs to scale their operations to achieve positive profits in the future. Shorter processing times can make this scaling process easier as more boxes can be processed in the same amount of time. This makes Kolonial.no better equipped to meet increased demand.

In addition to increased service levels, a better goods assignment can lead to significant cost reductions. An important cost driver in the operation of the assembly line is the labor costs of the pickers. Kolonial.no estimates a cost per picker of 300 NOK per working hour. This cost includes the picker's wage, taxes and fees. Considering the fact that 26 pickers operate the assembly line, an hour saved equals 7800 NOK. By using the assignment of the goods proposed by the genetic algorithm, it is possible to save 4.59 hours a week. In practice, this means a scheduled work time for each picker of 4.5 hours less each week. This amounts to about 1.8 million NOK in saved labor costs a year.

Savings of 1.8 million NOK amount to 1.13% of Kolonial.no's negative profits in 2017. Considering that this is a change done in one link of the supply chain and only by changing the assignment of the goods, the savings are considerable. In addition, as Kolonial.no has two other assembly lines, there is a reason to believe that it is possible to reach the same savings for these.

It is worth mentioning that the labor costs associated with changing the assignment of the goods every week are not considered. These costs might be substantial since changes in the shelves' configuration are needed along with an update in the routing software. Operational impracticalities due to weekly changes in the assignment of the goods are also not considered. Some goods that come in large distribution units, such as pallets of soda, are not easy to reassign. Pickers get used to their station's composition of goods, and frequent changes might reduce the picker's effectiveness. Lastly, in hectic periods, the resources used to conduct the change in goods assignment might be needed elsewhere, thus, ending up delaying other critical processes.

Nevertheless, should Kolonial.no choose to keep a certain goods assignment for a more extended period to avoid costs of reassignments, our results also show an improvement on the *unseen data*. This indicates that the solution still might reduce their costs.

9.10 Further improvements

The simulator enables us to do tweaks to the warehouse and see how this affects the makespan. Therefore, we'll look into some adjustments that can be made to improve the makespan further or reduce costs. These adjustments are, however, slacking the constraints given in the Problem Description in Chapter 3 or changing the original warehouse layout. This section aims to advocate directions for further research.

9.10.1 Extending the buffer zones to reduce makespan

If the capacity at the buffer zones is increased, this may reduce the queue along the assembly line. Consequently, this can reduce the makespan. A larger capacity enables more boxes to wait at a station without causing a queue on the main line. We tested the effect of doubling the capacity at each station. This reduces the makespan of the greedy solution by 9 % on the *Opt Data*. This indicates that Kolonial.no should investigate expanding their buffer zones and do further research on different buffer zone set-ups.

9.10.2 Goods at multiple stations to reduce makespan

The queues and high idle times described in Subsections 9.7.1 and 9.7.2, illustrate that boxes are not arriving at the picker or not departing the picker because of queue. One scenario of particular interest is when several boxes in a sequence are demanding the same good. If enough boxes in a sequence are demanding the same good, then this will cause a queue that will propagate onto the main line. In order to counter this behavior, slacking the problem description to allow the same goods to be placed at several stations was investigated. Specifically, the top 5, top 25 and top 250 most frequent goods were allowed to be placed at two stations. A simple approach to test this effect was used; the frequent goods were duplicated and given separate good-IDs. In practice, this means that they were handled as separate goods. Every other box is assigned the duplicate good instead of the original. The idea is that the workload of a good is distributed between two stations, instead of one, to reduce the queue on the main line. The resulting makespans of the greedy solutions can be seen in Table 9.17. Letting the 250 most frequent goods be placed at two stations, improves the makespan of 3%. These results show significant improvement in makespan even with this primitive procedure. The matter should, therefore, be further investigated. Moreover, placing goods at several stations opens up the opportunity for boxes to be dynamically rerouted in the events of gridlock, full buffer zones, or sudden unexpected station shutdown which was not modelled in this paper.

Table 9.17: *The table show the makespan when goods are duplicated and placed using the greedy algorithm.*

Number of duplicates	Makespan [s] - <i>Opt Data</i>
0	105084
5	103912
25	102070
250	101650

9.10.3 Fewer pickers at work to reduce costs

An observation from the solutions was that the pickers spend a significant part of their shifts idle. For Kolonial.no's solution workers at some stations where up to 50 % idle. This is an inefficient use of labor. It is therefore of interest to investigate the effect of having fewer pickers at work. Moreover, increasing the buffer zone size of each station showed promising results. Halving the number of pickers and doubling the buffer zone sizes is practically accomplishable. Both adjustments can be accomplished by merging station s and $s + 1$, where $s \in \{0, 2, \dots, 24\}$. Now each picker is responsible for a station twice the size. It is assumed that the pickers still pick at the same speed as earlier, although their station size is larger. The makespan found for this layout with the greedy solution is 171074 seconds, see Table 9.18. This is an increase in makespan of 62 %. However, the average idle time per picker is reduced from 36 % to 21 %. The operation cost is reduced with 19 % under the same cost assumptions per picker as earlier.

Table 9.18: *The table illustrates the cost improvement of halving the number of pickers. The comparison is done using the greedy algorithm.*

Number of pickers	Makespan [s] - <i>Opt Data</i>	Avg. picking time per station [s]	Cost per week [NOK]
26	105084	67510	185 000
13	171074	135521	229 000

Conclusion

Kolonial.no is operating in a tight price-sensitive market with low margins on products. Therefore it is essential to have cost-effective processes. Considering that the costs of warehouse operations are about 2/3 of their operational costs, excluding the cost of goods, there can be significant savings to gain in the warehouse. Additionally, Kolonial.no expect to grow in the coming years, which makes it essential to have a warehouse process that can meet the increased demand.

In this thesis, we are inspired by the suggestion that the storage assignment has a significant impact on the throughput of the assembly line. Therefore, we have investigated the Storage Location Assignment Problem (SLAP) for Kolonial.no. In short, we assign goods to workstations to minimize the processing time of a given sequence of boxes. Previous work with this particular problem done by Bergland et al. (2018) indicated that the complexity of the problem makes an exact solution method improper. Therefore, a heuristic and metaheuristic approach was the focus of this thesis. A simple greedy approach was able to solve the problem in 0.04 seconds. The results show an improvement of 6.0% on a known set of boxes, the *opt data*, compared to Kolonial.no's current solution. On the *unseen data*, there was an improvement of 6.0%. The main contribution of this thesis, however, is the simulation-based genetic algorithm. This is a novel approach to the SLAP. It was computationally expensive to run, but produced great solutions. After an hour it beat Kolonial.no's solution, and after nine and a half hours it beat the greedy solution. The final solution was found by letting the algorithm run for 84 hours. On the *opt data*, this approach shows an improvement of 14.9% compared to the current goods assignment. This can amount to a scheduled work time for each picker of 4.5 hours less each week. Yearly, this means savings of 1.8 million NOK. The faster processing of boxes also means that Kolonial.no are better able to handle the increased demand they expect in the years to come. In addition, the results performed well on the *unseen data*, here with a 9.9% improvement. This indicates that the solution can be useful even when it is kept unchanged for an extended period.

The results achieved in this thesis satisfies the purpose of finding a solution that is significantly better than Kolonial.no's current solution. It is evident that significant improvements may come from only changing the goods assignment. In addition, our

analysis shows interesting insights into the characteristics of a good solution. The best solution was not the most overall balanced. Besides, our results show that instead of being one superior solution, several different goods assignments can perform equally good. Working on the problem has also shown different aspects of the assembly line that can greatly affect the makespan. By having a simulator of the assembly line, it is simple to test modifications on the warehouse and investigate their effect. One discovery was the idle-times along the assembly line. Although our best solution reduces the average idle time of a picker from 39.6% to 29.1%, there is still a significant proportion of idle time on the assembly line. We observed that a significant amount of idle-time was due to a blockage: When there is a full stop on the main line outside of a station, finished boxes are prevented from leaving the station. This prevents the picker from starting to process the next box in the buffer zone. To further reduce the idle-time, actions beyond the goods assignment might be needed.

Further research

Our results show that the GA is a viable algorithm for this problem. However, fitness evaluation using the simulator presented in this thesis is computationally expensive. It can therefore be of interest to look at other approaches that may find equally good solutions faster. The aforementioned paper of Zhang et al. (2018) uses an evolutionary algorithm with a fitness approximation. Developing a neural network to predict the fitness values could reduce the number of simulations needed and make the algorithm more effective.

The real-life problem presented in Chapter 3 has a level of detail that can be further modeled mathematically and implemented in the genetic algorithm. This includes constraints that maintain preferred locations for some specific goods and stochastic occurrences such as the absence of pickers, malfunction of station hardware, etc. In addition, section 9.10 shows that it could be interesting to investigate capacity extensions on certain stations. For example, Figure 9.14 shows an uneven distribution of queue time at stations. An increase in buffer zones or a different distribution in shelf capacity might give a solution with lower makespan.

Alternative searching algorithms to the GA can be studied. A compelling solution method is the particle swarm algorithm (PSO). Kuo et al. (2016) shows that the PSO has the potential to find better solutions for SLAP than the GA, especially for bigger instances. Additionally, Figure 4.4 show that the Taboo search has been as frequently used as GA to solve SLAP and is therefore worth investigating.

The objective of reducing the makespan has in this thesis, been approached by studying the assignment of goods. However, our study indicates that numerous factors affect the total makespan. Section 9.10 does a small initial investigation on further measures that can be taken beyond changing the goods assignment. Extending the buffer zones might be beneficial if the warehouse layout allows it, as our results show an improvement of 9% by doubling the buffer capacities. In addition, having half the pickers at work to reduce costs can be done without halving the efficiency. It might also be worthwhile to investigate the effect of having the most frequent goods at multiple stations. Our data

show that some goods dominate the customer orders, and distributing this workload over several stations is favorable in our tests. Moreover, placing goods at several stations opens up the opportunity for boxes to be dynamically rerouted in the events of gridlock, full buffer zones, or sudden unexpected station shutdown. Lastly, it would be of value to also take the order of the boxes into consideration. Solving the problem by also optimizing the sequence of the boxes might have a significant impact on the makespan.

Bibliography

- Abdel-Hamid, A. A.-A. and Borndörfer, R. (1994). On the complexity of storage assignment problems.
- Berg, J. D. and Zijm, W. (1999). Models for warehouse management: Classification and examples. *International Journal of Production Economics*, 59(1):519–528.
- Bergland, L. E., Støre, I. R., and Vale, S. W. (2018). Modeling the storage location assignment problem for an assembly line production.
- Bindi, F., Manzini, R., Pareschi, A., and Regattieri, A. (2009). Similarity-based storage allocation rules in an order picking system: an application to the food service industry. *International Journal of Logistics Research and Applications*, 12(4):233–247.
- Carlo, H. J. and Giraldo, G. E. (2012). Toward perpetually organized unit-load warehouses. *Computers & Industrial Engineering*, 63(4):1003–1012.
- Chen, L., Langevin, A., and Riopel, D. (2010). The storage location assignment and interleaving problem in an automated storage/retrieval system with shared storage. *International Journal of Production Research*, 48(4):991–1011.
- Chen, L. and Lu, Z. (2010). The storage location assignment problem for outbound containers in a maritime terminal. *International Journal of Production Economics*, 135(1).
- Chen, M. and Wu, H. (205). An association based clustering approach to order batching considering customer demand patterns. *Omega*, 33:333–343.
- Chiang, D. M.-H., Lin, C.-P., and Chen, M.-C. (2011). The adaptive approach for storage assignment by mining data of warehouse management system for distribution centres. *Enterprise Information Systems*, 5(2):219–234.
- Crossley, W., Williams, E., Crossley, W., and Williams, E. (1997). *A study of adaptive penalty functions for constrained genetic algorithm-based optimization*.
- de Koster, R. (1994). Performance approximation of pick-to-belt orderpicking systems. *European Journal of Operational Research*, 72(3):558 – 573.

- 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.
- Eiben, A. E. and Smith, J. E. (2015). *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd edition.
- Elbert, R. and Philipp Müller, J. (2017). Manoeuvres in manual person-to-parts order picking systems: Is there a significant influence on travel time and metabolic rates? *IFAC PapersOnLine*, 50(1):6894–6899.
- Ene, S., Küçükoğlu, , Aksoy, A., and Öztürk, N. (2016). A genetic algorithm for minimizing energy consumption in warehouses. *Energy*, 114:973–980.
- Ene, S. and Öztürk, N. (2012). Storage location assignment and order picking optimization in the automotive industry. *The International Journal of Advanced Manufacturing Technology*, 60(5):787–797.
- Franzke, T., Grosse, E., Glock, C., and Elbert, R. (2016). An investigation of the effects of storage assignment and picker routing on the occurrence of picker blocking in manual picker-to-parts warehouses. *The International Journal of Logistics Management*, 28.
- Glock, C. H. and Grosse, E. H. (2012). Storage policies and order picking strategies in u-shaped order-picking systems with a movable base. *International Journal of Production Research*, 50(16):4344–4357.
- Gu, J., Goetschalckx, M., and McGinnis, L. F. (2007). Research on warehouse operation: A comprehensive review. *European Journal of Operational Research*, 177(1):1–21.
- Hagen, G. A. (2008). Derfor floppet lidl. Accessed 12. December 2018.
- Hausman, W. H., Schwarz, L. B., and Graves, S. C. (1976). Optimal storage assignment in automatic warehousing systems. *Management Science*, 22(6):629–638.
- Hogseth, M. H. and Lorch-Falch, S. (2018). Marked.no legger ned driften. Retrieved from <https://e24.no/naeringsliv/marked-no-legger-ned-driften/24228335>. Accessed 12. December 2018.
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI. second edition, 1992.
- Hopland, S. (2019). Ny rapport: Kolonial.no må omsette for to milliarder for å tjene penger. Retrieved from <https://e24.no/naeringsliv/netthandel/ny-rapport-kolonial-no-maa-omsette-for-to-milliard-er-for-aa-tjene-penger/24581467>. Accessed 25. Mars 2019.
- Hui, Y. Y., Choy, K., Ho, G., Leung, K., and Lam, H. (2016). A cloud-based location assignment system for packaged food allocation in e-fulfillment warehouse. *International Journal of Engineering Business Management*, 8.

- Høgseth, M. H. (2018). Vekst på 17 prosent fra fjoråret. Retrieved from <https://e24.no/naeringsliv/netthandel/fersk-undersokelse-nordmenn-bruker-144-milliarder-paa-netthandel/24450737>. Accessed 13. December 2018.
- Jane, C.-C. and Lai, Y.-W. (2005). A clustering algorithm for item assignment in a synchronized zone order picking system. *European Journal of Operational Research*, 166(2):489 – 496.
- Kay, E., Brown, R., Chandler, G., and Davis, W. A. (1968). *A Mathematical Model for Handling in a Warehouse: The Commonwealth and International Library: Social Administration, Training, Economics and Production Division*. Pergamon.
- Kuo, R., Kuo, P., Chen, Y. R., and Zulvia, F. (2016). Application of metaheuristics-based clustering algorithm to item assignment in a synchronized zone order picking system. *Applied Soft Computing*, 46:143–150.
- Lamorgese, L., Mannino, C., and Piacentini, M. (2016). Optimal train dispatching by benders'-like reformulation. *Transportation Science*, 50(3):910–925.
- Li, J., Moghaddam, M., and Nof, S. Y. (2016). Dynamic storage assignment with product affinity and abc classification—a case study. *The International Journal of Advanced Manufacturing Technology*, 84(9):2179–2194.
- Lorch-Falch, S. (2018). Vi har aldri sagt at det ville bli lett eller billig. Retrieved from <https://e24.no/naeringsliv/dagligvarebransjen/konkurrenten-fikk-nok-mens-kolonial-no-naer-doblet-omsetningen/24228398>. Accessed 12. December 2018.
- Malmberg, C. J. (1995). Optimization of cube-per-order index warehouse layouts with zoning constraints. *International Journal of Production Research*, 33(2):465–482.
- Myrset, B., Hortling, B., and Maric, S. (2016). From waiting in-line to going online: Norway's grocery market disruption. Retrieved from <https://www.accenture.com/no-en/insight-waiting-in-line-to-going-online>. Accessed 21. November 2018.
- Nanakorn, P. and Meesomklin, K. (2001). An adaptive penalty function in genetic algorithms for structural design optimization. *Computers & Structures*, 79(29):2527 – 2539.
- Pan, J. C.-H., Shih, P.-H., Wu, M.-H., and Lin, J.-H. (2015). A storage assignment heuristic method based on genetic algorithm for a pick-and-pass warehousing system. *Computers & Industrial Engineering*, 81:1–13.
- Pan, J. C.-H. and Wu, M.-H. (2009). A study of storage assignment problem for an order picking line in a pick-and-pass warehousing system. *Computers & Industrial Engineering*, 57(1):261 – 268. Collaborative e-Work Networks in Industrial Engineering.
- Park, C. and Seo, J. (2009). Mathematical modeling and solving procedure of the planar storage location assignment problem. *Computers & Industrial Engineering*, 57(3):1062 – 1071.

- Park, C. and Seo, J. (2010). Comparing heuristic algorithms of the planar storage location assignment problem. *Transportation Research Part E*, 46(1):171–185.
- Petersen, C. and Gerald, A. (2002). Considerations in order picking zone configuration. *International Journal of Operations & Production Management*, 27:793–805.
- Petersen, C. G. (2002). Considerations in order picking zone configuration. *International Journal of Operations & Production Management*, 22(7):793–805.
- Proff.no (2017). Retrieved from <https://www.proff.no/selskap/kolonial.no-as/oslo/postordre-internetthandel-annet/IF34Y9A10KA/>. Accessed 11. April 2019.
- Reyes, J. J. R., Solano-Charris, E. L., and Montoya-Torres, J. R. (2018). The storage location assignment problem: A literature review. *International Journal of Industrial Engineering Computations*, 10(2):199–224.
- Roeva, O., Fidanova, S., and Paprzycki, M. (2013). Influence of the population size on the genetic algorithm performance in case of cultivation process modelling. *Federated Conference on Computer Science and Information Systems*, page 371–376.
- S. Fraser, A. (1957). Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Sciences*, 13.
- Skaar, J. (2017). Etableringshindringer i dagligvaresektoren.
- Sundberg, J. D. and Wig, K. (2018). Brodboksen.no slår seg konkurs. Retrieved from <https://e24.no/naeringsliv/dagligvarebransjen/broedboksen-no-avvikler-driften/24239671>. Accessed 12. December 2018.
- Valvik, M. E. (2017). Godtlevant.no og adams matkasse slår seg sammen. Retrieved from <https://www.aftenposten.no/okonomi/i/PQGnR/Godtlevantno-og-Adams-matkasse-slar-seg-sammen>. Accessed 26. Mars 2019.
- Weldeghebriel, L. (2018). Kraftig ut mot kolonial.no: –ikke for å være eplekjekk, men matvarer på nett vil aldri kunne lønne seg. det er faktisk helt umulig! Retrieved from <https://shifter.no/kraftig-ut-mot-kolonial-no-ikke-for-a-vaere-eplekjemmen-matvarer-pa-nett-vil-aldri-kunne-lonne-seg-det-er-faktisk-helt-umulig/>. Accessed 14. December 2018.
- Wifstad, K., Bull Jenssen, T., Stemland Eide, L., A. Grünfeld, L., and Skogli, E. (2018). Konkurransen i dagligvaremarkedet.
- Xiao, J. and Zheng, L. (2010). A correlated storage location assignment problem in a single-block-multi-aisles warehouse considering bom information. *International Journal of Production Research*, 48(5):1321–1338.
- Yu, M. and De Koster, R. (2008). Performance approximation and design of pick-and-pass order picking systems. *IIE Transactions*, 40(11):1054–1069.
- Zhang, H., Guo, Z., Cai, H., Wang, C., Zhang, W., Yu, Y., Li, W., and Wang, J. (2018). Layout design for intelligent warehouse by evolution with fitness approximation.

Appendix

A

A.1 Assembly Line Routing Model

Set		Definition
$\mathcal{S}$		Stations
$\mathcal{B}$		Boxes
$\mathcal{G}$		Goods
$\mathcal{G}_b$	$\subseteq \mathcal{G}$	Goods to be placed into box b
$\mathcal{N}$		Nodes
$\mathcal{N}^{\mathcal{B}}$	$\subset \mathcal{N}$	Buffer nodes - Nodes at station that are not the station's Patient node.
$\mathcal{N}^{\mathcal{C}}$	$\subset \mathcal{N}$	Choice nodes - Nodes where boxes are faced with the choice to continue on the main line or stop by the adjacent station
$\mathcal{N}^{\mathcal{M}}$	$\subset \mathcal{N}$	Main Line nodes - Nodes adjacent to a station that are not a Choice node nor Reunion node
$\mathcal{N}^{\mathcal{P}}$	$\subset \mathcal{N}$	Patient nodes - Nodes where goods may be placed into boxes
$\mathcal{N}^{\mathcal{R}}$	$\subset \mathcal{N}$	Reunion nodes - Nodes where boxes may arrive from the main line or station
$\mathcal{N}^{\mathcal{T}}$	$\subset \mathcal{N}$	Transition nodes - All boxes must pass through all nodes in this set and the node thereafter

Parameter	Definition
$N_s \in \mathbb{N}$	The number of (Buffer and Patient) nodes at station s
$N(b, g) \in \mathbb{N} \cup \{\emptyset\}$	A function returning the number of items needed for box b of good g
$S(u) \in \mathbb{N} \cup \{\emptyset\}$	A function returning the station associated with node u
$\underline{T}$	The minimum waiting time before next box is released onto the assembly line
$\bar{T} \in \mathbb{R}_{\geq 0}$	A number larger than the total duration of one work period
$T_{g,n} \in \mathbb{R}_{\geq 0}$	The time it takes to collect n units of good g
$T_{u,v} \in \mathbb{R}_{\geq 0}$	The time it takes from node u to a subsequent node v
$W_g \in \mathbb{R}_{\geq 0}$	The distribution unit width of good g
$W_s \in \mathbb{R}_{\geq 0}$	The total width allowed on station s
$Z \in \mathbb{R}_{\geq 0}$	A (small) weight used for the minimisation for departure time objective term.
$\epsilon \in \mathbb{R}_{\geq 0}$	The minimum time difference a box from main line has to departure in order to not yield for a box departing from patient node.

Variable	Definition
$a_{b,u,v} \in \{0, 1\}$	1 if box b will traverse arc (u, v) , 0 else
$p_{b,b',u} \in \{0, 1\}$	1 if box b arrives at node u before box b' , 0 else
$t \in \mathbb{R}_{\geq 0}$	The depart time for the last box at the final node
$t_{b,u}^A \in \mathbb{R}_{\geq 0}$	The arrival time of box b at node u
$t_{b,u}^D \in \mathbb{R}_{\geq 0}$	The departure time of box b from node u . Note $t_{b,u}^D = 0$ if box b never visits node u
$x_{b,s} \in \{0, 1\}$	1 if box b visits station s , 0 else
$y_{g,s} \in \{0, 1\}$	1 if good g is positioned at station s , 0 else
$\delta_{b,b',u} \in \{0, 1\}$	1 if box b from main line does not need to yield for box b' from patient node at the reunion node u , 0 else

$$\min (t - t_{1,1}^A) \quad (\text{A.1})$$

$$+ Z \sum_{u \in \mathcal{N}} \sum_{b \in \mathcal{B}} t_{b,u}^D \quad (\text{A.2})$$

$$\sum_{s \in \mathcal{S}} y_{g,s} = 1 \quad \forall g \in \mathcal{G}. \quad (\text{A.3})$$

$$\sum_{g \in \mathcal{G}} W_g y_{g,s} \leq W_s \quad \forall s \in \mathcal{S}. \quad (\text{A.4})$$

$$\sum_{g \in \mathcal{G}_b} y_{g,s} \geq x_{b,s} \quad \forall s \in \mathcal{S}, \forall b \in \mathcal{B} \quad (\text{A.5})$$

$$x_{b,s} \geq \frac{1}{|\mathcal{G}_b|} \sum_{g \in \mathcal{G}_b} y_{g,s} \quad \forall s \in S, \forall b \in \mathcal{B} \quad (\text{A.6})$$

$$a_{b,u,u+1} = 1 \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^T. \quad (\text{A.7})$$

$$a_{b,u,u+1} = x_{b,S(u)} \quad \forall u \in \mathcal{N}^C, b \in \mathcal{B}. \quad (\text{A.8})$$

$$a_{b,u,u+N_{S(u)}+1} = (1 - x_{b,S(u)}) \quad \forall u \in \mathcal{N}^C, b \in \mathcal{B}. \quad (\text{A.9})$$

$$a_{b,u,u+1} = x_{b,S(u)} \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^B. \quad (\text{A.10})$$

$$a_{b,u,u+N_{S(u)}-1} = x_{b,S(u)} \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^P. \quad (\text{A.11})$$

$$a_{b,u,u+1} = (1 - x_{b,S(u)}) \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^M. \quad (\text{A.12})$$

$$t \geq t_{b,|\mathcal{N}|}^D \quad \forall b \in \mathcal{B}. \quad (\text{A.13})$$

$$t_{1,1}^A = 0. \quad (\text{A.14})$$

$$t_{b+1,1}^A \geq t_{b,1}^A + \underline{T} \quad \forall b \in \mathcal{B} \setminus \{|\mathcal{B}|\}. \quad (\text{A.15})$$

$$t_{b+M,1}^A \geq t_{b,|\mathcal{N}|}^D \quad \forall b \in \mathcal{B}. \quad (\text{A.16})$$

$$t_{b,u}^A \leq t_{b,u}^D \quad \forall b \in \mathcal{B}, u \in \mathcal{N}. \quad (\text{A.17})$$

$$t_{b,u}^A + \sum_{g \in \mathcal{G}_b} T_{g,N(g,b)} y_{g,S(u)} \leq t_{b,u}^D \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^P. \quad (\text{A.18})$$

$$t_{b,u}^D \leq \bar{T} x_{b,S(u)} \quad \forall b \in \mathcal{B}, u \in (\mathcal{N}^B \cup \mathcal{N}^P). \quad (\text{A.19})$$

$$t_{b,u}^D \leq \bar{T}(1 - x_{b,S(u)}) \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^M. \quad (\text{A.20})$$

$$t_{b,u}^D + T_{u,u+1} \leq t_{b,u+1}^A + \bar{T}(1 - a_{b,u,u+1}) \quad \forall b \in \mathcal{B}, u \in \mathcal{N} \setminus \{|\mathcal{N}|, \mathcal{N}^P\}. \quad (\text{A.21})$$

$$t_{b,u}^D + T_{u,u+N_{S(u)}-1} \leq t_{b,u+N_{S(u)}-1}^A + \bar{T}(1 - a_{b,u,u+N_{S(u)}-1}) \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^P. \quad (\text{A.22})$$

$$t_{b,u}^D + T_{u,u+N_{S(u)}+1} \leq t_{b,u+N_{S(u)}+1}^A + \bar{T}(1 - a_{b,u,u+N_{S(u)}+1}) \quad \forall b \in \mathcal{B}, u \in \mathcal{N}^C \quad (\text{A.23})$$

$$p_{b,b',u} + p_{b',b,u} \leq 1 \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}, \quad (\text{A.24})$$

$$p_{b,b',u} + p_{b',b,u} \leq \sum_{v \in \mathcal{N}} (a_{b,v,u} + a_{b',v,u}) \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}, \quad (\text{A.25})$$

$$p_{b,b',u} + p_{b',b,u} \geq \frac{1}{2} \sum_{v \in \mathcal{N}} (a_{b,v,u} + a_{b',v,u}) \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}. \quad (\text{A.26})$$

$$p_{b,b',v} \geq p_{b,b',u} + a_{b,u,v} + a_{b',u,v} - 2 \quad \forall b \in \mathcal{B}, b' \in \mathcal{B}, u \in \mathcal{N}, v \in \mathcal{B} \quad (\text{A.27})$$

$$p_{b,b',v} \leq p_{b,b',u} + 2 - a_{b,u,v} - a_{b',u,v} \quad \forall b \in \mathcal{B}, b' \in \mathcal{B}, u \in \mathcal{N}, v \in \mathcal{B} \quad (\text{A.28})$$

$$t_{b,u}^D - t_{b',u}^D \leq \bar{T}(1 - p_{b,b',u}) \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}. \quad (\text{A.29})$$

$$t_{b',u}^D - t_{b,u}^D \leq \bar{T}p_{b,b',u} \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}. \quad (\text{A.30})$$

$$(t_{b,u-N_{S(u)}+1}^D + t_{b,u-1}^D) - (t_{b',u-N_{S(u)}+1}^D + t_{b',u-1}^D) \leq \bar{T}(1 - p_{b,b',u}) \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}^{\mathcal{R}} \quad (\text{A.31})$$

$$-\bar{T}\delta_{b,b',u} + t_{b',u}^A \leq t_{b,u}^A - \epsilon \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}^{\mathcal{R}} \quad (\text{A.32})$$

$$(t_{b,u-1}^D + t_{b,u-N_{S(u)}+1}^D + \epsilon) - (t_{b',u-N_{S(u)}+1}^D + t_{b',u-1}^D) \leq \bar{T}(1 - \delta_{b,b',u}) \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}^{\mathcal{R}} \quad (\text{A.33})$$

$$(t_{b',u-N_{S(u)}+1}^D + t_{b',u-1}^D) - (t_{b,u}^D + t_{b,u-N_{S(u)}+1}^D + \epsilon) \leq \bar{T}\delta_{b,b',u} \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}^{\mathcal{R}} \quad (\text{A.34})$$

$$t \geq 0 \quad (\text{A.35})$$

$$a_{b,u,v} \in \{0, 1\} \quad \forall b \in \mathcal{B}, u \in \mathcal{N}, v \in \mathcal{N}, u \neq v \quad (\text{A.36})$$

$$p_{b,b',v} \in \{0, 1\} \quad \forall b, b' \in \mathcal{B}, b \neq b', v \in \mathcal{N} \quad (\text{A.37})$$

$$t_{b,u}^A, t_{b,u}^A \geq 0 \quad \forall b \in \mathcal{B}, u \in \mathcal{N} \quad (\text{A.38})$$

$$y_{g,s} \in \{0, 1\} \quad \forall g \in \mathcal{G}, s \in \mathcal{S} \quad (\text{A.39})$$

$$\delta_{b,b',u} \in \{0, 1\} \quad \forall b, b' \in \mathcal{B}, b \neq b', u \in \mathcal{N}^{\mathcal{R}} \quad (\text{A.40})$$

B.1 Additional plots

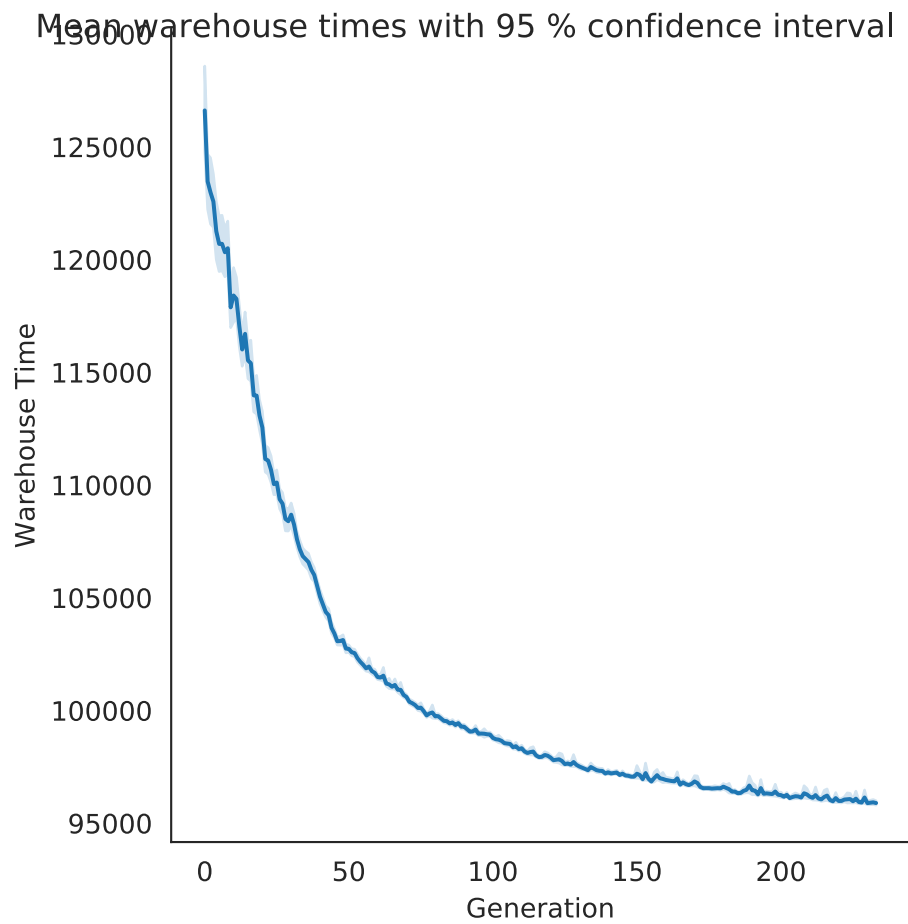


Figure B.1: *The plot shows the mean warehouse time(makespan) for all the chromosomes in the population with 95% confidence interval over the course of all generations. These results are taken from the best run with 1 week instance size.*

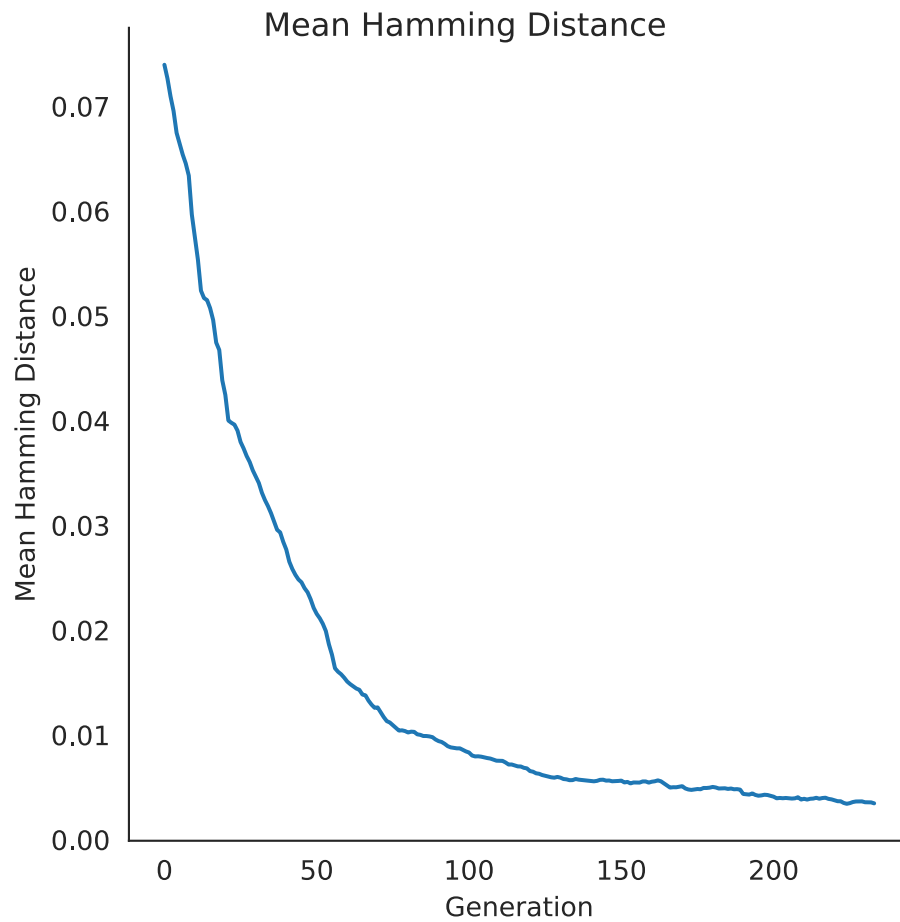


Figure B.2: *The plot shows the development of hamming distance as the generations increase. These results are taken from the best run with 1 week instance size.*

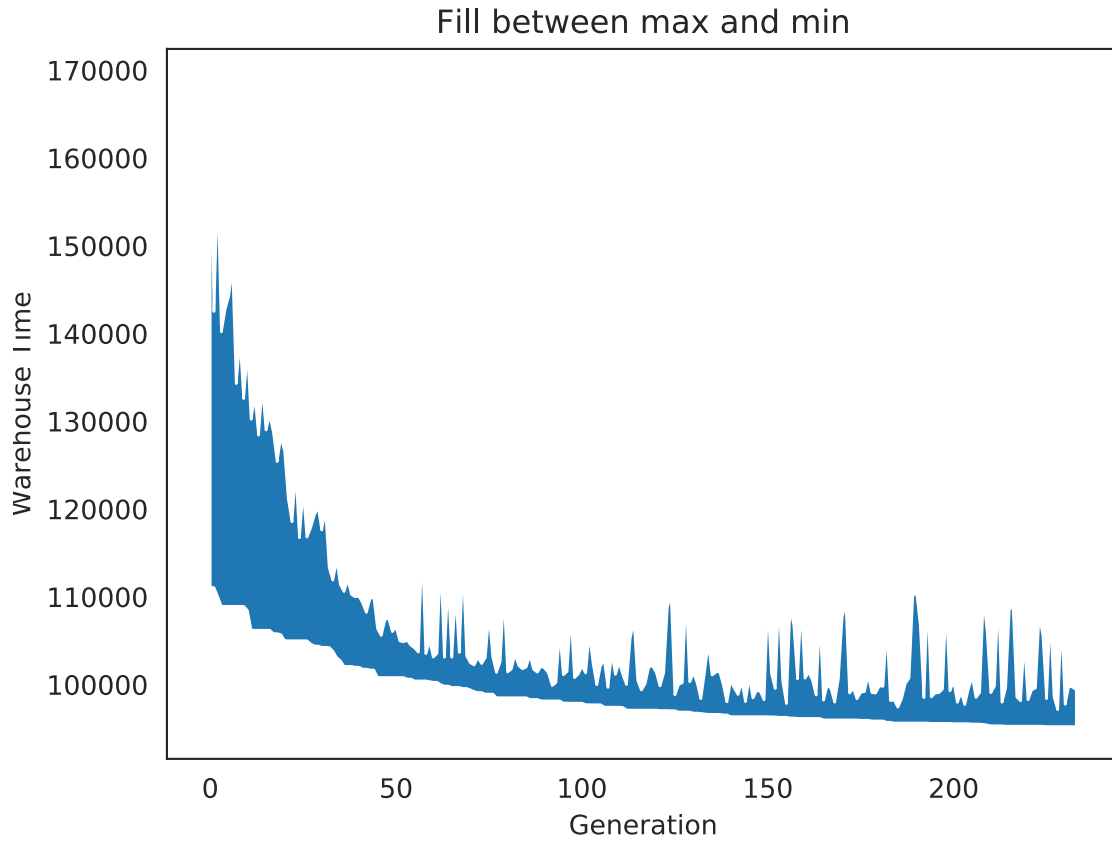


Figure B.3: *This plot shows the max and min warehouse time(makespan) as the generations increase. One can see that the gap between max and min solutions decrease during the first 50 generations. These results are taken from the best run with 1 week instance size.*

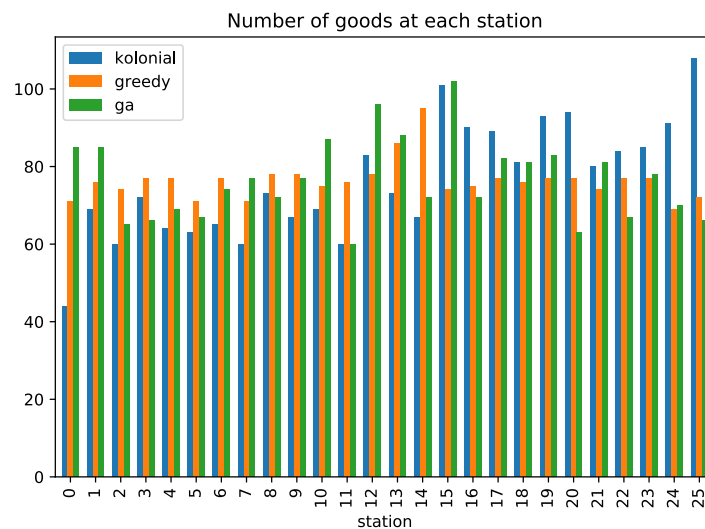


Figure B.4: *The plot shows the number of goods at each station for different solutions.*

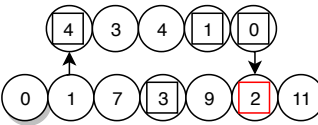
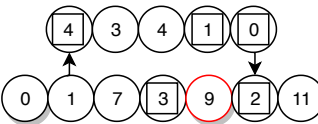
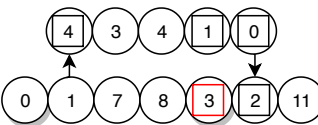
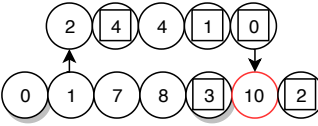
Step 101	Makespan: 00:06	Time	Node ID	Box ID	Event type	DTY	The events are sorted as earlier described. The sorting on DTY is now decisive. The box-arrived event for box 2 will be handled first. In this example, box 0 is not ready before warehouse time 00:07, so box 2 can freely arrive at the reunion node 00:06
		00:06	10	2	Box Arrived	Yes	
		00:06	8	3	Box Ready	Yes	
		00:06	5	1	Box Ready	Yes	
		00:06	2	4	Box Ready	Yes	
		00:07	6	0	Box Ready	No	
Step 102	Makespan: 00:06	Time	Node ID	Box ID	Event type	DTY	Box arrived creates a node-empty event in box 2's previous node.
		00:06	9	none	Node Empty	Yes	
		00:06	8	3	Box Ready	Yes	
		00:06	5	1	Box Ready	Yes	
		00:06	2	4	Box Ready	Yes	
		00:07	6	0	Box Ready	No	
Step 102	Makespan: 00:06	Time	Node ID	Box ID	Event type	DTY	The node-empty event creates a box-arrived event for box 3 at node 9
		00:06	9	3	Box Arrived	Yes	
		00:06	8	3	Box Ready	Yes	
		00:06	5	1	Box Ready	Yes	
		00:06	2	4	Box Ready	Yes	
		00:07	6	0	Box Ready	No	
●							
●							
●							
Step 103	Makespan: 00:07	Time	Node ID	Box ID	Event type	DTY	Moving forward to the situation where there is a node-empty event in reunion node. The sorting on DTY is now decisive. The box-ready event for box 0 is handled before box 3 since it does not have a duty to yield in the reunion node.
		00:07	10	None	Node Empty	Yes	
		00:07	6	0	Box Ready	No	
		00:07	9	3	Box Ready	Yes	
		00:07	3	4	Box Ready	Yes	

Figure B.5: This figure displays a simulator walkthrough were the duty to yield is enforced for one of the boxes coming from main line to the reunion node.

Appendix C

C.1 Run summary

Table C.1: *The table shows all the runs done for range testing, calibration and selection of parameters.*

Makespan [s]	Num Boxes	Population size	Mutation rate	Num swap	Crossover rate	Num elite	Gen_scaling	Tourn- ament P	Tourn- ament K	Crossover method	Term tresh- hold
3503	700	100	60%	8	70%	1	5 000	95%	3	One-Point	100
3531	700	100	75%	8	70%	1	5 000	95%	3	One-Point	100
3550	700	100	95%	8	70%	1	5 000	95%	3	One-Point	100
3292	700	200	60%	2	70%	1	5 000	95%	3	One-Point	100
3264	700	200	75%	2	70%	1	5 000	95%	3	One-Point	100
3384	700	200	95%	2	70%	1	5 000	95%	3	One-Point	100
3330	700	300	60%	4	70%	1	5 000	95%	3	One-Point	100
3393	700	300	75%	4	70%	1	5 000	95%	3	One-Point	100
3383	700	300	95%	4	70%	1	5 000	95%	3	One-Point	100
3373	700	400	60%	6	70%	1	5 000	95%	3	One-Point	100
3456	700	400	75%	6	70%	1	5 000	95%	3	One-Point	100
3497	700	400	95%	6	70%	1	5 000	95%	3	One-Point	100
3457	700	100	60%	8	85%	1	5 000	95%	3	One-Point	100
3494	700	100	75%	8	85%	1	5 000	95%	3	One-Point	100
3560	700	100	95%	8	85%	1	5 000	95%	3	One-Point	100
3198	700	200	60%	2	85%	1	5 000	95%	3	One-Point	100
3270	700	200	75%	2	85%	1	5 000	95%	3	One-Point	100
3320	700	200	95%	2	85%	1	5 000	95%	3	One-Point	100
3337	700	300	60%	4	85%	1	5 000	95%	3	One-Point	100
3383	700	300	75%	4	85%	1	5 000	95%	3	One-Point	100
3395	700	300	95%	4	85%	1	5 000	95%	3	One-Point	100

3373	700	400	60%	6	85%	1	5 000	95%	3	One-Point	100
3440	700	400	75%	6	85%	1	5 000	95%	3	One-Point	100
3497	700	400	95%	6	85%	1	5 000	95%	3	One-Point	100
3492	700	100	50%	1	85	1	10 000	95%	3	One-Point	300
3306	700	100	50%	1	85	1	10 000	95%	3	One-Point	300
3440	700	100	50%	1	85	1	10 000	95%	3	One-Point	300
3302	700	100	50%	1	85	1	10 000	95%	3	One-Point	300
3397	700	100	50%	1	85	1	10 000	95%	3	One-Point	300
3531	700	100	50%	1	85	1	10 000	95%	3	One-Point	300
3405	700	100	50%	1	85	1	10 000	95%	3	One-Point	300
3497	700	100	50%	1	85	1	10 000	95%	3	One-Point	300
3364	700	100	50%	1	85	1	10 000	95%	3	One-Point	300
3347	700	100	65%	1	85	1	10 000	95%	3	One-Point	300
3271	700	100	65%	1	85	1	10 000	95%	3	One-Point	300
3249	700	100	65%	1	85	1	10 000	95%	3	One-Point	300
3231	700	100	65%	1	85	1	10 000	95%	3	One-Point	300
3269	700	100	65%	1	85	1	10 000	95%	3	One-Point	300
3285	700	100	65%	1	85	1	10 000	95%	3	One-Point	300
3232	700	100	65%	1	85	1	10 000	95%	3	One-Point	300
3256	700	100	65%	1	85	1	10 000	95%	3	One-Point	300
3291	700	100	65%	1	85	1	10 000	95%	3	One-Point	300
3295	700	100	65%	1	85	1	10 000	95%	3	One-Point	300
3253	700	100	80%	1	85	1	10 000	95%	3	One-Point	300
3253	700	100	80%	1	85	1	10 000	95%	3	One-Point	300
3275	700	100	80%	1	85	1	10 000	95%	3	One-Point	300
3215	700	100	80%	1	85	1	10 000	95%	3	One-Point	300
3247	700	100	80%	1	85	1	10 000	95%	3	One-Point	300
3268	700	100	80%	1	85	1	10 000	95%	3	One-Point	300
3258	700	100	80%	1	85	1	10 000	95%	3	One-Point	300
3253	700	100	80%	1	85	1	10 000	95%	3	One-Point	300
3254	700	100	80%	1	85	1	10 000	95%	3	One-Point	300
3225	700	100	80%	1	85	1	10 000	95%	3	One-Point	300
3261	700	100	50%	2	85	1	10 000	95%	3	One-Point	300
3343	700	100	50%	2	85	1	10 000	95%	3	One-Point	300
3295	700	100	50%	2	85	1	10 000	95%	3	One-Point	300
3253	700	100	50%	2	85	1	10 000	95%	3	One-Point	300

3371	700	100	50%	2	85	1	10 000	95%	3	One-Point	300
3250	700	100	50%	2	85	1	10 000	95%	3	One-Point	300
3301	700	100	50%	2	85	1	10 000	95%	3	One-Point	300
3258	700	100	50%	2	85	1	10 000	95%	3	One-Point	300
3358	700	100	50%	2	85	1	10 000	95%	3	One-Point	300
3293	700	100	50%	2	85	1	10 000	95%	3	One-Point	300
3245	700	100	65%	2	85	1	10 000	95%	3	One-Point	300
3235	700	100	65%	2	85	1	10 000	95%	3	One-Point	300
3312	700	100	65%	2	85	1	10 000	95%	3	One-Point	300
3244	700	100	65%	2	85	1	10 000	95%	3	One-Point	300
3296	700	100	65%	2	85	1	10 000	95%	3	One-Point	300
3261	700	100	65%	2	85	1	10 000	95%	3	One-Point	300
3223	700	100	65%	2	85	1	10 000	95%	3	One-Point	300
3257	700	100	65%	2	85	1	10 000	95%	3	One-Point	300
3228	700	100	65%	2	85	1	10 000	95%	3	One-Point	300
3199	700	100	65%	2	85	1	10 000	95%	3	One-Point	300
3286	700	100	80%	2	85	1	10 000	95%	3	One-Point	300
3312	700	100	80%	2	85	1	10 000	95%	3	One-Point	300
3302	700	100	80%	2	85	1	10 000	95%	3	One-Point	300
3292	700	100	80%	2	85	1	10 000	95%	3	One-Point	300
3322	700	100	80%	2	85	1	10 000	95%	3	One-Point	300
3291	700	100	80%	2	85	1	10 000	95%	3	One-Point	300
3268	700	100	80%	2	85	1	10 000	95%	3	One-Point	300
3263	700	100	80%	2	85	1	10 000	95%	3	One-Point	300
3253	700	100	80%	2	85	1	10 000	95%	3	One-Point	300
3256	700	100	80%	2	85	1	10 000	95%	3	One-Point	300
3327	700	100	50%	3	85	1	10 000	95%	3	One-Point	300
3272	700	100	50%	3	85	1	10 000	95%	3	One-Point	300
3310	700	100	50%	3	85	1	10 000	95%	3	One-Point	300
3275	700	100	50%	3	85	1	10 000	95%	3	One-Point	300
3281	700	100	50%	3	85	1	10 000	95%	3	One-Point	300
3242	700	100	50%	3	85	1	10 000	95%	3	One-Point	300
3225	700	100	50%	3	85	1	10 000	95%	3	One-Point	300
3224	700	100	50%	3	85	1	10 000	95%	3	One-Point	300
3350	700	100	65%	3	85	1	10 000	95%	3	One-Point	300
3283	700	100	65%	3	85	1	10 000	95%	3	One-Point	300

3309	700	100	65%	3	85	1	10 000	95%	3	One-Point	300
3297	700	100	65%	3	85	1	10 000	95%	3	One-Point	300
3287	700	100	65%	3	85	1	10 000	95%	3	One-Point	300
3318	700	100	65%	3	85	1	10 000	95%	3	One-Point	300
3278	700	100	65%	3	85	1	10 000	95%	3	One-Point	300
3302	700	100	65%	3	85	1	10 000	95%	3	One-Point	300
3283	700	100	65%	3	85	1	10 000	95%	3	One-Point	300
3280	700	100	65%	3	85	1	10 000	95%	3	One-Point	300
3370	700	100	80%	3	85	1	10 000	95%	3	One-Point	300
3331	700	100	80%	3	85	1	10 000	95%	3	One-Point	300
3345	700	100	80%	3	85	1	10 000	95%	3	One-Point	300
3347	700	100	80%	3	85	1	10 000	95%	3	One-Point	300
3316	700	100	80%	3	85	1	10 000	95%	3	One-Point	300
3320	700	100	80%	3	85	1	10 000	95%	3	One-Point	300
3337	700	100	80%	3	85	1	10 000	95%	3	One-Point	300
3300	700	100	80%	3	85	1	10 000	95%	3	One-Point	300
3322	700	100	80%	3	85	1	10 000	95%	3	One-Point	300
3311	700	100	80%	3	85	1	10 000	95%	3	One-Point	300
3358	700	100	65%	2	45	1	10 000	95%	3	One-Point	300
3342	700	100	65%	2	45	1	10 000	95%	3	One-Point	300
3348	700	100	65%	2	45	1	10 000	95%	3	One-Point	300
3371	700	100	65%	2	45	1	10 000	95%	3	One-Point	300
3376	700	100	65%	2	45	1	10 000	95%	3	One-Point	300
3336	700	100	65%	2	45	1	10 000	95%	3	One-Point	300
3310	700	100	65%	2	45	1	10 000	95%	3	One-Point	300
3315	700	100	65%	2	45	1	10 000	95%	3	One-Point	300
3355	700	100	65%	2	45	1	10 000	95%	3	One-Point	300
3309	700	100	65%	2	45	1	10 000	95%	3	One-Point	300
3315	700	100	65%	2	65	1	10 000	95%	3	One-Point	300
3275	700	100	65%	2	65	1	10 000	95%	3	One-Point	300
3280	700	100	65%	2	65	1	10 000	95%	3	One-Point	300
3302	700	100	65%	2	65	1	10 000	95%	3	One-Point	300
3303	700	100	65%	2	65	1	10 000	95%	3	One-Point	300
3279	700	100	65%	2	65	1	10 000	95%	3	One-Point	300
3265	700	100	65%	2	65	1	10 000	95%	3	One-Point	300
3302	700	100	65%	2	65	1	10 000	95%	3	One-Point	300

3261	700	100	65%	2	65	1	10 000	95%	3	One-Point	300
3289	700	100	65%	2	65	1	10 000	95%	3	One-Point	300
3431	700	30	65%	2	85	1	5 000	95%	3	One-Point	300
3553	700	30	65%	2	85	1	10 000	95%	3	One-Point	300
3453	700	30	65%	2	85	1	20 000	95%	3	One-Point	300
3452	700	30	65%	2	85	1	50 000	95%	3	One-Point	300
3370	700	30	65%	2	85	1	100 000	95%	3	One-Point	300
3214	700	100	50%	1	85	1	100 000	95%	3	Uni-form	200
3186	700	100	65%	1	85	1	100 000	95%	3	Uni-form	200
3195	700	100	80%	1	85	1	100 000	95%	3	Uni-form	200
3191	700	100	50%	2	85	1	100 000	95%	3	Uni-form	200
3169	700	100	65%	2	85	1	100 000	95%	3	Uni-form	200
3210	700	100	80%	2	85	1	100 000	95%	3	Uni-form	200
3157	700	100	50%	3	85	1	100 000	95%	3	Uni-form	200
3217	700	100	65%	3	85	1	100 000	95%	3	Uni-form	200
3275	700	100	80%	3	85	1	100 000	95%	3	Uni-form	200
3085	700	200	50%	1	85	1	100 000	95%	3	Uni-form	200
3100	700	200	65%	1	85	1	100 000	95%	3	Uni-form	200
3129	700	200	80%	1	85	1	100 000	95%	3	Uni-form	200
3110	700	200	50%	2	85	1	100 000	95%	3	Uni-form	200
3132	700	200	65%	2	85	1	100 000	95%	3	Uni-form	200
3160	700	200	80%	2	85	1	100 000	95%	3	Uni-form	200
3107	700	200	50%	3	85	1	100 000	95%	3	Uni-form	200
3205	700	200	65%	3	85	1	100 000	95%	3	Uni-form	200
3240	700	200	80%	3	85	1	100 000	95%	3	Uni-form	200
3532	700	100	50%	1	85	1	100 000	95%	3	One-Point	200
3425	700	100	65%	1	85	1	100 000	95%	3	One-Point	200
3292	700	100	80%	1	85	1	100 000	95%	3	One-Point	200
3322	700	100	50%	2	85	1	100 000	95%	3	One-Point	200
3287	700	100	65%	2	85	1	100 000	95%	3	One-Point	200
3305	700	100	80%	2	85	1	100 000	95%	3	One-Point	200
3289	700	100	50%	3	85	1	100 000	95%	3	One-Point	200
3296	700	100	65%	3	85	1	100 000	95%	3	One-Point	200
3342	700	100	80%	3	85	1	100 000	95%	3	One-Point	200

