A comparative study between a genetic algorithm and a simulated annealing algorithm for solving the order batching problem

EDVIN ARDÖ

JOHAN LINDHOLM
A comparative study between a genetic algorithm and a simulated annealing algorithm for solving the order batching problem

EDVIN ARDÖ
JOHAN LINDHOLM

Degree Project in Computer Science, DD143X
Date: June 17, 2019
Supervisor: Jana Tumova
Examiner: Örjan Ekeberg
School of Electrical Engineering and Computer Science
Swedish title: En jämförande studie mellan en genetisk algoritm och simulated annealing för att lösa order batching problemet
Abstract

Optimizing warehouse automation requires finding efficient routes for picking up items. Dividing the orders into batches is a realistic requirement for warehouses to have. This problem, known as the order batching problem, is an NP-hard problem. This thesis implements and compares two meta-heuristics to the order batching problem, simulated annealing (SA) and a genetic algorithm (GA).

SA was found to perform equal to or better than GA on all occasions in terms of minimizing traveling distance. The algorithms were tested on 6 different warehouses with various layouts. The algorithms performed similarly on the smallest problem size, but in the largest problem size SA managed to find 17.1% shorter solutions than GA. SA tended to find shorter solutions in a smaller amount of time as well.
Sammanfattning


SA visade sig vara lika bra eller bättre än GA vid alla tillfällen då målet är att minimera den totala färdsträckan. Algoritmen testades på 6 olika varuhus som hade olika designer. Algoritmerna kom fram till liknande lösningar för de minsta varhusen, men i det största varhuset lyckades SA hitta en lösning som var 17.1 % bättre än GA. SA tenderade även att hitta kortare lösningar givet mindre tid.
Contents

1 Introduction 1
  1.1 Purpose ...................................................... 1
  1.2 Problem statement ........................................... 2
  1.3 Scope .......................................................... 2

2 Background 3
  2.1 Order batching problem ....................................... 3
  2.2 Genetic algorithm .............................................. 4
  2.3 Simulated annealing .......................................... 4
  2.4 Related work .................................................. 5

3 Method 6
  3.1 Solution ...................................................... 6
  3.2 Test approach ............................................... 6
    3.2.1 Warehouse ............................................... 6
    3.2.2 Environment .............................................. 8
  3.3 Data set ..................................................... 8
  3.4 Genetic algorithm ........................................... 9
    3.4.1 Pseudocode ............................................... 9
    3.4.2 Selection .................................................. 9
    3.4.3 Crossover ................................................ 9
    3.4.4 Mutation ................................................. 10
    3.4.5 Reparation .............................................. 10
  3.5 Simulated annealing ....................................... 11
    3.5.1 Pseudocode ............................................... 11
    3.5.2 Neighbor function ...................................... 11
    3.5.3 Acceptance function .................................... 12
    3.5.4 Cooling schedule ...................................... 12

4 Result 13
5 Discussion
   5.1 Further research ........................................ 21
6 Conclusion ..................................................... 22
Bibliography ..................................................... 23
A Source code ................................................... 25
B Warehouses .................................................... 26
C Graphs .......................................................... 31
Chapter 1

Introduction

The need for automation in warehouses grows as the demand for cheaper and more environmentally friendly alternatives to warehouse management rises. Warehouses are essential for most supply chains and reducing the travel distance for picking up orders may save time and money [1]. Order picking is an essential part of modern warehouses. This process accounts for 55-65 % of the total operational warehouse cost according to Theys et al. [2].

The order batching problem consists of dividing a set of orders into batches. Dividing orders into batches and finding the shortest path to retrieve all items is an NP-hard problem. A common strategy for solving NP-hard problems is using some sort of heuristics. A heuristic is an approach to solving a problem using practical techniques and educated guesses. The solution found is not guaranteed to be optimal but is often used to find a satisfying solution. We seek out to compare two different algorithms to the warehouse order batching problem to find which is best at minimizing the total travel distance for retrieving a set of orders in a single-agent system. Implementing simulated annealing (SA) into a real warehouse has previously resulted in saving over 5000 kilometers of travel distance compared to their previous strategy during a 3 month period [3]. Genetic algorithms (GA) have also proved to be useful at minimizing energy consumption in warehouses [4].

1.1 Purpose

Having an efficient algorithm for finding short solutions to the order batching problem is important to reduce costs of warehouses, and since both SA and GA have been proven to be useful at this task we intend to compare them to each other. The purpose of this thesis is therefore to find out which heuristic performs best at solving the order batching problem out of SA and GA in
1.2 Problem statement

Which algorithm performs best at minimizing the total travel distance to the warehouse order batching problem out of a genetic algorithm and a simulated annealing algorithm?

1.3 Scope

We assume that the warehouse is divided into cells. Each cell is either a drop-off location, shelf, path or wall. The items are randomly placed on the shelves in the warehouse, and each item is to be retrieved exactly once. Each path cell represents a unit of distance long. The path that the retrieval agent takes is determined by the heuristics used. The carrying capacity of the retrieval agent is defined as the batch size. This thesis focuses on the order batching problem where all orders are known in advance and uses the optimal distance for each route, which is given by solving the travelling salesman problem via an exhaustive search. This is possible since the route never exceeds five destinations. Finally, this thesis does not intend to compare SA and GA in general, but instead develop and compare two specific implementations.
Chapter 2

Background

This chapter presents the order batching problem, the genetic algorithm and the simulated annealing algorithm.

2.1 Order batching problem

A warehouse is constantly receiving orders of items, which it has to retrieve from its storage in order to transport them to their customers. It is therefore desirable to optimize order picking as much as possible, in order to reduce operational costs as well as to reduce the amount of time it takes for an order to be processed and shipped. One main aspect of this optimization is minimizing the distance needed to travel to retrieve the order.

It is common in a warehouse for an agent to retrieve several orders in a single route, which is known as batching. Karasek [5] states that the performance of order picking is significantly impacted by batching, which also is an NP-hard problem. Given a specific batch there exists a minimum travelling distance to retrieve it, which is given by solving the Travelling Salesman Problem (TSP). If the batch size is small, e.g. less than 6, the optimal solution can be found by an exhaustive search. If the order batching problem is to be studied for large batch sizes a heuristic for TSP should be used, e.g. an adaptation of the Lin-Kernighan Heuristic, as used by Theys et al. [2].

The order batching problem is the problem of grouping a set of orders in a warehouse, in which a retrieval agent can fetch multiple orders at once, in order to minimize the total travel distance needed to fetch all orders. There are a lot of variable parameters to this problem, such as warehouse layout, batch maximum size as well as distribution and amount of items in the warehouse.

The batching problem can be divided into two categories, static and dynamic batching. Static batching means that all customer orders are known in
advance, whereas dynamic batching means that orders may arrive at different time points [6]. This thesis only handles the static order batching problem.

### 2.2 Genetic algorithm

Genetic algorithms are a group of optimization algorithms inspired by evolution. They optimize solutions through the use of mutation, selection and crossover operations. The algorithm starts off by creating a population of random valid solutions, called chromosomes. The populations prosperity is determined by a fitness function which evaluates each chromosome. Some of the candidates are selected to breed a new generation. Chromosomes with a lower fitness are typically more likely to be selected. Crossover is used to exploit the current chromosomes and maximize the overall fitness of the population.

To increase diversity, a chance of mutation of chromosomes is introduced, which helps the algorithm avoid getting stuck at local minimums by exploring a wider range of possible solutions. In addition to mutation the chromosomes are also mixed using a crossover algorithm. The numbers of total generations are user specified, and a higher value may result in better solutions. Genetic algorithms do not require any mathematical analysis to improve solutions to optimization problems and are effective at performing global searches [7].

### 2.3 Simulated annealing

Simulated annealing is a probabilistic algorithm for finding approximations for optimization problems. Its name comes from metallurgy [8], in which annealing is the process of heating a metal to a certain temperature and then slowly cooling it at a steady rate in order to change its properties. Simulated annealing consists of a few main steps.

First an initial temperature is set and a random state is chosen. Thereafter the looping procedure begins by choosing a neighbor of the current state by making a small change to the current state, for example swapping the order of two elements in a permutation. A decision is then made of whether the neighboring state should be chosen. This decision is made based on the change in energy between the two states as well as the current temperature. The energy is a measure of how good the state is. If the energy of the neighboring state is higher, it is chosen. If it is lower it will be chosen with a certain probability. The higher the temperature and the lower the change in energy, the more likely the neighboring state is to be chosen. Lastly the temperature is decreased and the looping procedure is repeated until a predetermined temperature has been reached.
2.4 Related work

A paper by Ene et al. [4] studies a genetic algorithm with the goal of minimizing energy consumption in warehouses. Genetic algorithms are used for both the batching and routing problem. The paper arrived at the conclusion that genetic algorithms proved useful for minimizing energy consumption in warehouses. Genetic algorithms have the advantage of adapting to any sort of warehouse and applying genetic algorithms to warehouses could provide significant energy savings worldwide.

A study by Matusiak et al. [3] claims that order picking is the most important process in distribution centers. Using simulated annealing and a data set over three months from a large warehouse they managed to reduce travel distance by 15.7%. They also got inside a 1.2% margin of error compared to the optimal solution (up to 27 orders with batch size 3). The warehouse used in this study is a standard warehouse with parallel aisles and cross aisles.

A previous degree project from Yamazaki and Pertof [9] studies the scalability of genetic algorithms that solve university course scheduling problems. The paper concluded that genetic algorithms were suited for timetable problems. Smaller input data have a tendency to quickly find a high quality solution. Bigger input data also tend to get solved but require a higher amount of generations.

Simulated annealing has also been used previously by Grosse, Glock, and Ballester-Ripoll [10] to solve the joint order batching and routing problem with a weight constraint. Instead of considering a batch size consisting of an amount of items, every item is instead assigned a weight and the batch a maximum weight.

To the best of our knowledge, there are no previous studies focusing on comparisons between genetic algorithms and simulated annealing for the order batching problem.
Chapter 3

Method

3.1 Solution

A valid solution to the order batching problem with \( n \) orders and a batch size \( m \) is represented as an array \( sol \) of length \( n \) containing integer values in the range \([0, m - 1]\). Order item \( i \) belongs to batch \( j \) if \( sol_i = j \). No batch may consist of more than \( m \) orders.

\( sol \), in equation 3.1, is an example of a solution with 9 orders and a batch size of 3 where \( sol_0 = 2 \), meaning that the first order is a part of batch 2.

\[
\text{sol} = \{2, 1, 0, 0, 1, 1, 0, 2, 2\}
\] (3.1)

3.2 Test approach

SA was run with a set of parameters, given in table 3.2, until its temperature reached \( T_{final} \). The total running time of SA was measured, whereafter the GA was allowed to run for the same amount time.

3.2.1 Warehouse

The input warehouse is a text file containing three rows of metadata and then the text representation of the warehouse layout. The first row contains the width of the warehouse. The second row contains the height of the warehouse. The third row contains the amount of order items in the warehouse. The number of rows following are equal to the height and are all of length equal to the width. Every character in this text representation of a warehouse corresponds to a single cell in the warehouse according to table 3.1.
Table 3.1 also shows how the text representation corresponds to the internal representation of the warehouse, which consists of an integer matrix. There must be exactly one drop-off location in the warehouse. Every order item location is randomly assigned a unique integer value from 0 to the order amount.

<table>
<thead>
<tr>
<th>Warehouse cell</th>
<th>Drop-off</th>
<th>Path</th>
<th>Wall</th>
<th>Order item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text representation</td>
<td>X</td>
<td>+</td>
<td>#</td>
<td>O</td>
</tr>
<tr>
<td>Internal representation</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>x ≥ 0</td>
</tr>
</tbody>
</table>

Table 3.1: Translation between input text characters and internal representation of warehouse cells.

### 3.2.1.1 Order access points

Every order item can only be accessed from a single adjacent tile. This limitation was necessary because of the simplification of turning the warehouse into a graph representation, explained in further detail in 3.2.1.2. Since the distance between any two order items is obtained from a graph, allowing multiple access points would result in inaccuracies when summing distances in a route.

During instantiation of a warehouse every order item is therefore given a single access point from an adjacent non-occupied cell, which includes paths and the drop-off location. If available, the cell above the order item will be chosen as its access point. If not, the next direction in the priority order will be tried. The entire priority order in which adjacent cells will be tried for availability is above, then below, then to the left, then to the right. If any order item location has no available access point the warehouse is considered invalid.
3.2.1.2 Warehouse graph representation

It is desirable to have a constant lookup time when accessing distance between any two order items in a warehouse. The warehouse is therefore transformed from a matrix into a graph, in which every order access point and drop-off location represent a node. A breadth first search in the warehouse matrix from every order access point is used to retrieve the distance to every other order access point and the drop-off location, which are then stored as weighted edges in the graph.

3.2.2 Environment

All algorithms and testing software were implemented in Java. The algorithms were run on a computer with an Intel Core i5-4690K 3.5 GHz CPU. No other active processes were running on the computer during this time.

3.3 Data set

In total six different warehouses were used to run tests on. These can be found in appendix B. An image representation of the three small versions are available in figure 3.1. Additionally, batch sizes used were 3, 4 and 5.

![Image Representation](image.png)

Figure 3.1: An image representation from left to right of the ordered, irregular and weird warehouse. X represents the drop-off location, gray tiles a path, orange tiles a shelf and black tiles a wall.
3.4 Genetic algorithm

3.4.1 Pseudocode

Algorithm 1 Genetic algorithm

Create initial population \textit{With random solutions}

\begin{algorithmic}
\While{\text{Time is left}}
\State Elitism \textit{Save top chromosomes}
\State Selection \textit{Select using tournament selection}
\State Crossover \textit{Single point crossover}
\State Mutate \textit{with predefined chance}
\State Repair \textit{Fix broken chromosomes}
\EndWhile
\Return{Chromosome with best fitness}
\end{algorithmic}

3.4.2 Selection

GA creates a new set of chromosomes for each generation. The implementation used in this thesis uses tournament selection and elitism selection. The tournament selection used chooses 4 chromosomes at random and then picks the one with the highest fitness. The chromosomes picked become the parents used in crossover for a child chromosome. To keep the best chromosomes, we use elitism selection, which saves the chromosomes with the highest fitness value for each generation. The amount of chromosomes saved for each generation is set to 2, which seemed to perform best out of a few arbitrarily selected tests.

3.4.3 Crossover

Crossover is the act of combining two parent chromosomes in hope of retrieving a better child. The implementation used in this thesis were single point crossover. Single point crossover combines the mother’s chromosome with the father at a randomly selected point. The child will have a combination of both the mother and father chromosome.

Step 1: Find a randomly selected index in the mother chromosome

Mother chromosome: \{2,1,0,0,1 | 1,0,2,2\}

Father Chromosome: \{2,2,1,2,1 | 0,0,0,1\}

Step 2: Select the first part of the mothers chromosome and combine it with
the second part of the father’s chromosome to create the child chromosome. 
Child chromosome: \{2,1,0,0,1,0,0,0,1\}

Step 3: The new solution is not guaranteed to be valid and might need to be repaired.

### 3.4.4 Mutation

Mutation is the act of changing a small part of a chromosome. The rate of mutation chosen for GA was 10%, which means that there’s a 10% probability of a chromosome being altered. The mutation function used in GA is identical to the neighbor function used in SA, and is described in section 3.5.2.

### 3.4.5 Reparation

The crossover function may create an invalid chromosome, which means a function which can repair these is required. Firstly, the reparation algorithm iterates through the solution and finds batches with more orders than allowed and those with capacity for more. Secondly the algorithm swaps the batch of an order in an overpopulated batch with that of an order in an available batch. It continues swapping orders as described until there are no batches that exceed the maximum batch size.
3.5 Simulated annealing

3.5.1 Pseudocode

Algorithm 2 Simulated annealing

\[
\begin{align*}
\text{sol} & \leftarrow \text{random solution} \\
\text{sol}_{\text{best}} & \leftarrow \text{sol} \\
T & = T_{\text{init}} \\
\text{while } T < T_{\text{fin}} \text{ do} \\
\quad & \text{sol}_{\text{neighbor}} \leftarrow \text{NEIGHBOR(sol)} \\
\quad & e_{\text{diff}} \leftarrow \text{DIST(sol)} - \text{DIST(sol}_{\text{neighbor}}) \\
\quad & \text{if ACCEPT}(e_{\text{diff}}, T) \text{ then} \\
\quad & \quad \text{sol} \leftarrow \text{sol}_{\text{neighbor}} \\
\quad & \text{end} \\
\quad & \text{if DIST(sol) < DIST(sol}_{\text{best}}) \text{ then} \\
\quad & \quad \text{sol}_{\text{best}} \leftarrow \text{sol} \\
\quad & \text{end} \\
\quad & T \leftarrow T \times CDR \\
\text{end} \\
\text{return sol}_{\text{best}}
\end{align*}
\]

function ACCEPT \((e_{\text{diff}}, T)\) :

\[
\begin{align*}
\quad & \text{if } e_{\text{diff}} > 0 \text{ then} \\
\quad & \quad \text{return true} \\
\quad & \text{else} \\
\quad & \quad \text{return } e_{\text{diff}} \times 0.0003 > \text{RANDOM}(0, 1) \\
\end{align*}
\]

function NEIGHBOR \((\text{sol})\) :

\[
\begin{align*}
\quad & \text{if random}(0, 1) < 0.9 \text{ then} \\
\quad & \quad \text{sol} \leftarrow \text{SWAP BATCH OF TWO ORDERS(sol)} \\
\quad & \text{else} \\
\quad & \quad \text{batch} \leftarrow \text{FIND BATCH WITH SIZE < BATCH MAXIMUM SIZE(sol)} \\
\quad & \quad \text{sol} \leftarrow \text{CHANGE BATCH OF ORDER IN BATCH(batch, sol)} \\
\quad & \text{end} \\
\text{return sol}
\end{align*}
\]

3.5.2 Neighbor function

The neighbor function is split in two separate functions, one which swaps the batch of two random orders, and one which changes the batch of one random order to a batch which contains fewer orders than the maximum batch size. The swap function is chosen 90% of the times and can be run in a constant amount.
of time. The remaining 10 % of occurrences the function which changes the batch of one order is chosen. The reason why this function is chosen less frequently than the swapping function is that it can only be run in time linear to the order amount. It is however included to allow SA to traverse the entire search space, which would not be possible without it.

### 3.5.3 Acceptance function

SA uses the acceptance function given in equation 3.2 to determine the probability that a neighbor with a lower fitness value will be chosen. It increases with a larger difference between $E_0$ and $E_1$, the fitness values of the current solution and the neighboring solution respectively. It decreases with a lower temperature $T$. $k$ is a constant determined in combination with the initial temperature $T_0$ to achieve a desired probability distribution spread over the runtime of the algorithm.

$$p(e_0, e_1, T) = e^{\frac{E_0 - E_1}{T \times k}}$$  \hspace{1cm} (3.2)

### 3.5.4 Cooling schedule

SA was run with the parameters given in table 3.2, which were chosen as the optimal out of a few manually tested sets. The cooldown rate was chosen as 0.99999 for the three small warehouses and as 0.9999 for the three large warehouses. The smaller cooldown rate was chosen for the larger warehouses to lower the amount of iterations for SA, as each iteration took more time.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_0$</td>
<td>3000</td>
</tr>
<tr>
<td>$T_{final}$</td>
<td>10</td>
</tr>
<tr>
<td>$k$</td>
<td>0.0003</td>
</tr>
</tbody>
</table>

Table 3.2: Parameter values for the SA cooling schedule.
Chapter 4

Result

The difference between the shortest solutions found from 15 test runs is shown in figure 4.1 - 4.6 for SA and GA respectively. Generally, the larger warehouses have a larger difference in total distance between SA and GA than the small warehouses.

Figure 4.1 shows that the median distance for SA and GA was 55 and 65 respectively for the weird warehouse with a batch size of 3. The median solution found by SA is therefore 15.4 % shorter than the median solution found by GA.

Figure 4.6 shows that the median distance for SA and GA was 11680 and 14105 respectively for the large ordered warehouse with a batch size of 5. The median solution found by SA is therefore 17.1 % shorter than the median solution found by GA.

Figure 4.4 shows that the median distance for SA and GA was 24680 and 26554 respectively for the large ordered warehouse with a batch size of 3. The median solution found by SA is therefore 7.0 % shorter than the median solution found by GA.

Figure 4.1 shows that the median distance for SA and GA was 258 and 260 respectively for the ordered warehouse with a batch size of 3. The median solution found by SA is therefore 0.7 % shorter than the median solution found by GA.

All warehouse layouts can be found in appendix B. Graphs showing the best solution found over time for all warehouses and batch sizes for both SA and GA can be found in appendix C.
Figure 4.1: The difference between the shortest solution found for SA and GA with 15 test runs with a batch size of 3. The tests were run on the the small warehouses.

Figure 4.2: The difference between the shortest solution found for SA and GA with 15 test runs with a batch size of 4. The tests were run on the the small warehouses.
Figure 4.3: The difference between the shortest solution found for SA and GA with 15 test runs with a batch size of 5. The tests were run on the small warehouses.

Figure 4.4: The difference between the shortest solution found for SA and GA with 15 test runs with a batch size of 3. The tests were run on the large warehouses.
Figure 4.5: The difference between the shortest solution found for SA and GA with 15 test runs with a batch size of 4. The tests were run on the large warehouses.

Figure 4.6: The difference between the shortest solution found for SA and GA with 15 test runs with a batch size of 5. The tests were run on the large warehouses.
Table 4.1 shows the standard deviation for all test runs. Note that some instances have a standard deviation of 0, which means that all of these test runs got the same result. As a general note, the standard deviations seems to increase with the batch size and the warehouse size, although inconsistently. Additionally, SA consistently has a lower standard deviation than GA.

<table>
<thead>
<tr>
<th>Warehouse Layout</th>
<th>Heuristic</th>
<th>Batch size 3</th>
<th>Batch size 4</th>
<th>Batch size 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ordered</td>
<td>SA</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>1.69</td>
<td>1.46</td>
<td>2.0</td>
</tr>
<tr>
<td>Irregular</td>
<td>SA</td>
<td>0.0</td>
<td>2.73</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>0.63</td>
<td>3.15</td>
<td>3.44</td>
</tr>
<tr>
<td>Weird</td>
<td>SA</td>
<td>1.26</td>
<td>1.37</td>
<td>5.18</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>4.77</td>
<td>5.01</td>
<td>8.69</td>
</tr>
<tr>
<td>Large Ordered</td>
<td>SA</td>
<td>35.86</td>
<td>39.21</td>
<td>37.97</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>81.60</td>
<td>134.17</td>
<td>103.62</td>
</tr>
<tr>
<td>Large Irregular</td>
<td>SA</td>
<td>36.73</td>
<td>53.03</td>
<td>49.69</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>138.60</td>
<td>151.96</td>
<td>152.70</td>
</tr>
<tr>
<td>Large Weird</td>
<td>SA</td>
<td>17.22</td>
<td>25.40</td>
<td>22.06</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>60.05</td>
<td>79.26</td>
<td>75.49</td>
</tr>
</tbody>
</table>

Table 4.1: The standard deviation for 15 test runs on all warehouses with batch size 3, 4 and 5.
Figure 4.7-4.9 show how SA and GA find shorter solutions as time progresses. Notice how the difference between SA and GA is more significant on the larger warehouses and how they continues to find shorter solutions. All data of best solutions found over time can be found in appendix C. What is presented in this section is just a selection.

![Graph showing distance over time for SA and GA solutions.](image)

Figure 4.7: Best solutions found over time with a batch size of 4 in the ordered warehouse.
Figure 4.8: Best solutions found over time with a batch size of 5 in the large ordered warehouse.

Figure 4.9: Best solutions found over time with a batch size of 5 in the weird warehouse.
Chapter 5

Discussion

Figure 4.1 shows how the solutions with a batch size of 3 found by SA and GA in the small warehouses are similar except for the weird layout. In this case, SA found a solution which had a median distance that was 15.4 % shorter than GA in the weird layout. This is interesting since the weird warehouse layout was specifically designed to have groups of shelves with large distances from each other. This means that the distance of a random solution will likely be far longer than that of the optimal solution. The reason why there is a big difference between SA and GA in this layout might be that SA converges faster to lower distances than GA, as shown in figure 4.9.

SA consistently produces solutions with a lower distance, as shown in figures 4.1-4.6. In the large ordered warehouse with batch size 5, as shown in 4.6, SA finds a median solution which is 17.1 % shorter than the one GA created. In the same warehouse with batch size 3, as shown in 4.4, SA only finds a median solution that is 7.0 % shorter than GA. Large batch sizes seem to benefit SA when compared to GA in finding short solutions in a small amount of time, as shown in figure 4.8.

The median solution found with a batch size of 3 in the large ordered warehouse for SA is 7.0 % lower than that of GA, as shown in figure 4.4. The median difference for the corresponding small warehouse, is only 0.7 %, as shown in figure 4.1. This shows that SA performs better than GA for large warehouses, with regard to minimizing traveling distance. A reason for this might be that larger warehouses have more combinations of batches and therefore require more time to converge to a near-optimal solution.

Finding the shortest route for a given batch is time consuming, even though we have limited ourselves to a batch size of 5. This is because it is the same as an exhaustive search of the travelling salesman problem, which is NP-hard. To calculate the distance for a specific solution this problem has to be solved for each batch in said solution. It therefore seems likely that an algorithm which
minimizes the amount of times it has to calculate solution distances would be able to try more solutions than one which does not.

A reason for why GA produced worse solutions than SA could be that the crossover did not account for the fact that identical batches in different solutions could be numbered differently. For example, the two chromosomes $(0, 0, 1, 2, 1, 2)$ and $(1, 1, 2, 0, 2, 0)$ are identical, even though a crossover between them with half a chromosome from each would look like $(1, 1, 2, 2, 1, 2)$, which after being repaired into a valid solution might differ quite a lot from both parents. Furthermore, whilst the crossover itself might not take much time, it seems likely that it might cause the need to repair the chromosome, which is a time consuming process. The neighbor function of SA, which is the same as the mutate function of GA, differs from the crossover in the way that it might never turn a valid solution into an invalid one.

5.1 Further research

Both SA and GA rely on a set of parameters, such as temperature and cooldown rate for SA, and tournament size, elite offset and mutation rate for GA. The optimal value for these parameters vary for different problem instances, and could be optimized in further research.

Additionally, it would be interesting to look into the order batching problem using a heuristic for finding short routes for individual batches. Doing this would allow for testing of larger batch sizes. Furthermore, a combined batching and route heuristic could be implemented using either SA or GA.

Finally, to make the research more applicable in real warehouses, where all order items do not look the same, the order batching problem could be combined with the knapsack problem by assigning a weight to each item. This has previously been done for SA [10], but not in comparison to GA or other sophisticated heuristics. Other properties assigned to the order items which could be investigated include size and priority.
Chapter 6

Conclusion

Our implementations of SA and GA show that SA performs better than GA in all of our tests with regards to finding the shortest solution overall as well as finding shorter solutions significantly faster. Batch size, warehouse size and layout are all found to have significant impact on the performance of both SA and GA, and not always uniformly. Both SA and GA can be further optimized by adjusting constants such as the temperature range for SA and the mutation rate for GA.
Bibliography


Appendix A

Source code

The Java source code can be found in the GitHub repository at https://github.com/EdvinArdo/order-batching
Appendix B

Warehouses

--- Ordered warehouse.txt ---

10 10 48
X+++++++++++++++++
+oooooooooo+
+oooooooooo+
+oooooooooo+
+oooooooooo+
+oooooooooo+
+oooooooooo+
+oooooooooo+
+++

--- Irregular warehouse.txt ---

10 10 30
X+++++++++++++++++O+O
+oooooooo+O+O
+oooooooooo+O
+OO++++++O
+OO++++++O
0++OO+++O
0++OO+++O
0++OO+++O

---
Large ordered warehouse
Large weird warehouse
Appendix C

Graphs

Figure C.1: Best solutions found over time with a batch size of 3 in the ordered warehouse.
Figure C.2: Best solutions found over time with a batch size of 3 in the irregular warehouse.

Figure C.3: Best solutions found over time with a batch size of 3 in the weird warehouse.
Figure C.4: Best solutions found over time with a batch size of 3 in the large ordered warehouse.

Figure C.5: Best solutions found over time with a batch size of 3 in the large irregular warehouse.
Figure C.6: Best solutions found over time with a batch size of 3 in the large weird warehouse.

Figure C.7: Best solutions found over time with a batch size of 4 in the ordered warehouse.
Figure C.8: Best solutions found over time with a batch size of 4 in the irregular warehouse.

Figure C.9: Best solutions found over time with a batch size of 4 in the weird warehouse.
Figure C.10: Best solutions found over time with a batch size of 4 in the large ordered warehouse.

Figure C.11: Best solutions found over time with a batch size of 4 in the large irregular warehouse.
Figure C.12: Best solutions found over time with a batch size of 4 in the large weird warehouse.

Figure C.13: Best solutions found over time with a batch size of 5 in the ordered warehouse.
Figure C.14: Best solutions found over time with a batch size of 5 in the irregular warehouse.

Figure C.15: Best solutions found over time with a batch size of 5 in the weird warehouse.
Figure C.16: Best solutions found over time with a batch size of 5 in the large ordered warehouse.

Figure C.17: Best solutions found over time with a batch size of 5 in the large irregular warehouse.
Figure C.18: Best solutions found over time with a batch size of 5 in the large weird warehouse.