CHALMERS
UNIVERSITY OF TECHNOLOGY

Enhancing Association Rule Mining for Solving the Storage Location Assignment Problem

Master's thesis in Engineering Mathematics and Computational Sciences, and Data Science and AI

JONAS BOHLIN, TOBIAS GABRIELII

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Enhancing Association Rule Mining for Solving the Storage Location Assignment Problem

JONAS BOHLIN, TOBIAS GABRIELII

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2022

Enhancing Association Rule Mining for Solving the Storage Location Assignment Problem
JONAS BOHLIN, TOBIAS GABRIELII

Company supervisor: Casper Opperud, Ongoing Warehouse
University supervisor: Divya Grover, Department of Computer Science and Engineering
Examiner: Ann-Brith Strömberg, Department of Mathematical Sciences

Master's Thesis 2022
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Image depicting an allocation of articles inside a warehouse.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Enhancing Association Rule Mining for Solving the Storage Location Assignment Problem

Jonas Bohlin, Tobias Gabrielii
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

In the era of online retailing, reducing the picking time of orders is of great importance. One way of driving the picking time down is to optimise the locations of articles within the warehouse, a problem which is referred to as the Storage Location Assignment Problem (SLAP). The SLAP is an NP-hard problem, and it is therefore desirable to find a relaxation of the problem. In this thesis a rule based approach to the problem is proposed, focusing on association rule mining and rules created from a neural network utilising distance metric learning. These rules are then used by a greedy and a genetic algorithm, to optimise the article placements. The data used to find rules and evaluate the algorithms come from an online retailer of electronic spare parts. When evaluating the genetic algorithm on this dataset, it performs worse than the baseline of storing the most frequently purchased articles closest to the picking depot. However, the greedy algorithm outperforms this baseline by up to 11%, showing that there is a lot of promise for this rule based approach.

Keywords: Storage Location Assignment Problem, Association Rule Mining, Distance Metric Learning, Genetic Algorithm, Warehouse Management.

Acknowledgements

We would like to express our gratitude to Ongoing Warehouse AB for allowing us the opportunity to write this thesis and being so hospitable towards us. Especially we want to thank our Ongoing Warehouse supervisor Casper Opperud, our Chalmers supervisor Divya Grover and our examiner Ann-Brith Strömberg for providing us with guidance during the project. We are also extremely grateful towards our fellow master thesis students Sara, Mathias and Ludwig for making sure that we took much needed breaks from time to time.

Finally we want to express a heartfelt thanks to Stig and Benny for being there to cheer us on when the future looked dark, not just during this thesis but throughout our prior studies as well.

Jonas Bohlin and Tobias Gabrielii, Gothenburg, June 2022

List of Acronyms

Below is a list of acronyms that have been used throughout this thesis listed in alphabetical order:

AR	Association Rules
ARM	Association Rule Mining
BoW	Bag of Words
EAR	Extended Association Rules
GA	Genetic Algorithm
NNR	Neural Network Rules
SLAP	Storage Location Assignment Problem
TSP	Travelling Salesperson Problem

Nomenclature

Below is the nomenclature of functions, sets, parameters, and variables that have been used throughout this thesis.

Problem description

x	A set of article placements. $x_{ik} = 1$ if article i is stored at shelf k. Otherwise, $x_{ik} = 0$.
m	Number of unique articles.
n	Number of shelves.
$\mathcal{U}$	The set of future, unknown orders.
$\mathcal{O}$	The set of known orders.
S	Number of unknown orders.
h	Capacity (height) of each shelf.
$f(u, x)$	A function returning the picking distance of an order u with the article placements x.
$p_{\mathcal{O}}$, $p_{\mathcal{U}}$	The distributions of known and unknown orders.
$\mathcal{R}$	A set of rules between articles.
s_i	Support of article i, i.e. the fraction of orders in $\mathcal{O}$ containing the article i.
d_k	Distance between shelf k and the depot.
D_{kl}	Distance between shelf k and shelf l.
$W_{ij}^{\mathcal{R}}$	Weight of the rule between article i and article j in the rule set $\mathcal{R}$.

Association rule mining

$s_{\min}$	Minimum support threshold.
$L_{\min}$	Minimum lift threshold.
w_{ij}^{AR}	Weight of the association rule between article i and article j.

Extending association rules

δ, γ	Parameters controlling how word weights for the embedding space are created.
k	Maximum number of neighbours to/from which rules will be created for each regular association rule.
r	Maximum distance between two articles to be considered neighbours in the embedding space.
C_{p}	Parameter for weighting parallel rules against nearest neighbour rules.
$w_{ij}^{\mathrm{EAR_n}}$, $w_{ij}^{\mathrm{EAR_p}}$	Weight of the nearest neighbour rule and the parallel rule between article i and article j.
w_{ij}^{EAR}	Weight of the final extended association rule between article i and article j.

Neural network rules

m	Dimension of the embedding space created by the network.
k	Maximum number of neighbours to which rules will be created from each article.
r	Maximum distance between two articles to be considered neighbours in the embedding space.
$\mathcal{L}_{\mathrm{r}}(A, P, N)$	The triplet ratio loss function.
A, P, N	Anchor, positive and negative sample articles. For a specific order, A and P is contained in that order, while N is not.
f	Embedding function.
α	Parameter controlling the importance of negative samples in the loss function.
w_{ij}^{NNR}	Weight of the neural network rule between article i and article j.

Rule weights

$C_{\mathrm{EAR}}, C_{\mathrm{NNR}}$	Parameters for weighting the different types of rules.
W_{ij}	Final weight of the rule between article i and article j.

Greedy algorithm

C_r	Parameter for weighting the terms in the articleScore.
$C_\mathrm{d}, C_\mathrm{p}$	Parameters for weighting the terms in the shelfScore.
s_i	Support of article i.
d_k	Distance between shelf k and the depot.
D_{kl}	Distance between shelf k and shelf l.
W_{ij}	Weight of the rule between article i and article j.
$\mathcal{P}$	The set of articles that are already placed in the warehouse.
$\mathcal{P}_k$	The set of articles that are already placed in shelf k.

Genetic algorithm

n_tour	Tournament size for selection.
n_pop	Number of individuals in the population.
n_swaps	Number of indices to swap in crossover.
n_gen	Number of generations.
p_m	Mutation probability.
C_d	Parameter for weighting the two terms in the fitness function.
I	An individual in the population, represented by its chromosome.
s_i	Support of article i.
d_k	Distance between shelf k and the depot.
D_{kl}	Distance between shelf k and shelf l.
W_{ij}	Weight of the rule between article i and article j.

Contents

1
Introduction

Since the beginning of the 2010s, E-commerce has grown rapidly; see [1]. This growth can only be sustained if the customer's packages are delivered on time. There are three key steps a package goes through before it can end up on a customer's doorstep: processing the order through the website, picking and packaging of the order in a warehouse, and finally delivering the package to said doorstep. All of these steps have the potential to be optimised further; this Master's thesis will focus on the picking and packaging step. When a warehouse receives an order from the website, a worker is typically sent out into the warehouse to pick the order. This step can take a very long time depending on how far away from each other the different articles of the orders are placed; see [22]. Today, most warehouses use the so called ABC-system which divides the warehouse into three sections based on the frequency by which the articles are purchased; see [8]. However, there are likely better ways to distribute the articles. This Master's thesis will investigate the possibility of optimising the placements of articles in a warehouse using different kinds of rules inspired by Association Rule Mining (ARM); see [12].

1.1 Aim

The aim of this project is to develop a model for optimising the article placements in a warehouse so that the distance travelled during the picking process is minimised. The model uses transaction data consisting of different orders, and each order in turn consists of a set of articles. From this data, an improved set of article placements in the warehouse can be proposed.

1.2 Problem description

The warehouse layout is treated as a 2D grid as in Figure 1.1, with four types of tiles: shelves, aisles, walkways, and a depot. Each shelf has a capacity, representing its height, and is only accessible from adjacent aisle tiles. The picker can only move on aisle and walkway tiles, and the distance between two aisle tiles is set to the Manhattan distance between the tiles (where the shelf tiles cannot be crossed). As an example, the distance from tile 4 to tile 11 would be 6 distance units: one unit out to the walkway, three units down and two units to the left.

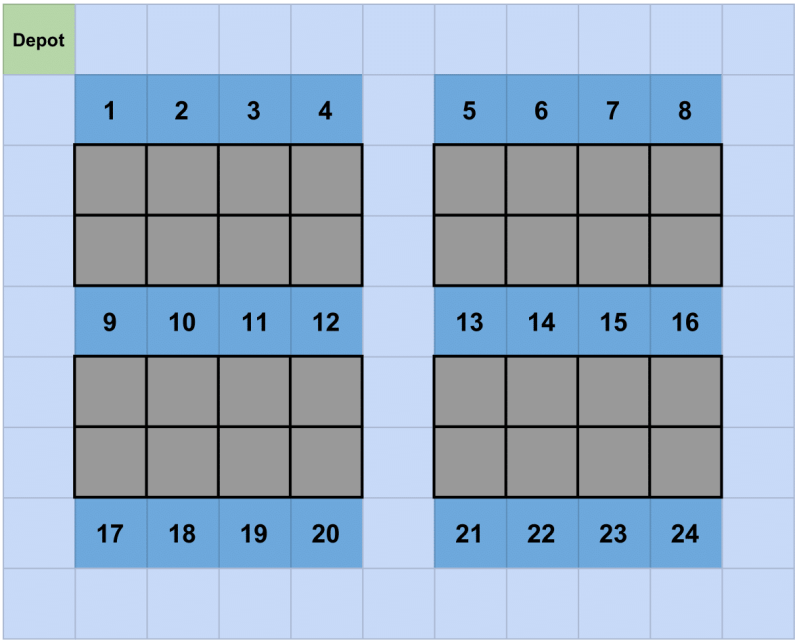


Figure 1.1: Representation of a generic warehouse layout, where the grey tiles represent shelves, accessed from adjacent aisle tiles (blue, numbered tiles). Light blue tiles represent walkways. Finally, the green depot tile in the upper left corner represents the depot, i.e. the start and end position of the picker.

The problem is a variant of the Storage Location Assignment Problem (SLAP); see [16], the aim of which is to place a set of articles at a set of locations, minimising a cost function. The SLAP belongs to the family of NP-hard optimisation problems; see [10]. In order to combat this, we intend to place the articles using rules extracted from historic order data. The problem boils down to two subproblems; finding meaningful rules in the data, and utilising these rules to improve the article placements.

More formally the problem to solve can be formulated as:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \sum_{j=1}^{S} f(\mathcal{U}_j, \mathbf{x}) \\ \text{s.t.} \quad & \sum_{k=1}^{n} x_{ik} = 1, \qquad \forall i \in \{1, 2, \ldots, m\}, \\ & \sum_{i=1}^{m} x_{ik} \leq h, \qquad \forall k \in \{1, 2, \ldots, n\}, \\ & x_{ik} \in \{0, 1\}, \ \forall i \in \{1, 2, \ldots, m\}, \ \forall k \in \{1, 2, \ldots, n\} \end{aligned} \tag{1.1}$$

where m is the number of articles, n is the number of shelves, S is the number of orders, $f(\mathcal{U}_j, \mathbf{x})$ is the picking distance of an order, $\mathcal{U}_j$, using the article placements given by the vector $\mathbf{x}$ of binary variables, where $x_{ik} = 1$ means that article i is stored at shelf k, and $x_{ik} = 0$ means that it is not. The first set of constraints ensure that every article is stored at exactly one shelf. In order to ensure that the capacity of

the shelves, h, is not exceeded, the second set of constraints is included. The picking distance, f, for an order is defined as:

$$f(\mathcal{U}, \mathbf{x}) \coloneqq \mathrm{TSP}(\mathcal{U}, \mathbf{x}). \tag{1.2}$$

That is to say, the travelling salesperson problem (TSP); see [6], consisting of starting at the depot, picking all the articles in order $\mathcal{U}$ from their shelves as defined by $\mathbf{x}$, and returning to the depot. However, this function cannot be evaluated since it requires the unknown orders $\mathcal{U}_1 \dots \mathcal{U}_S$ being known. Evaluating the objective function is also very computationally expensive since it requires solving S TSPs, where typically $S >> 1$. Thus, we assume that using a given set of rules $\mathcal{R}$ that describe the set of known (past) orders, it is possible to estimate the distribution of the unknown (future) orders. This assumption makes it possible to estimate $\mathbf{x}$ as $\mathbf{x}^{\mathcal{R}}$, which is found by the following problem:

$$\begin{aligned}
\mathbf{x}^{\mathcal{R}} = \arg\min_{\mathbf{x}} \quad & \sum_{i=1}^{m}\sum_{k=1}^{n} x_{ik}\left(C_{\mathrm{d}} s_i d_k + \sum_{j=1}^{m}\sum_{l=1}^{n} x_{jl} W_{ij}^{\mathcal{R}} D_{kl}\right) \\
\text{s.t.} \quad & \sum_{k=1}^{n} x_{ik} = 1, \qquad \forall i \in \{1, 2, \dots, m\}, \\
& \sum_{i=1}^{m} x_{ik} \leq h, \qquad \forall k \in \{1, 2, \dots, n\}, \\
& x_{ik} \in \{0, 1\}, \ \forall i \in \{1, 2, \dots, m\}, \ \forall k \in \{1, 2, \dots, n\}
\end{aligned} \tag{1.3}$$

where s_i is the support (frequency) of article i, d_k is the distance between the depot and shelf k, D_{kl} is the distance between shelf k and l, the constant $C_{\mathrm{d}} > 0$ weight the two terms against each other, and finally $W_{ij}^{\mathcal{R}}$ is the *rule weight* between articles i and j. The first term indicates that it is preferable to place commonly purchased articles close to the depot, while the second term prioritises placing articles which have rules between them close together. The rules $\mathcal{R}$ are extracted from the known order data $\mathcal{O}$, using methods that are described further in Chapters 2 and 3.

1.3 Limitations

First of all, it was assumed that each article requires exactly one unit of space, and can be stored at any shelf in the warehouse. It was also assumed that all shelves have the same capacity. These limitations were set partly in order to lower the computational complexity and partly because the data on product dimensions varied extremely, and was in some cases missing altogether. Furthermore, it was assumed that the warehouse layout was fixed, since allowing for shelves to change locations would add too much computational complexity.

Another limitation of this thesis was that only orders with 2–20 unique articles were used throughout the project. One of the reasons for excluding orders of size one was that orders consisting of one article contain less information. Another reason was

that the dataset used in the thesis contained a large fraction of orders containing only one article. Including all orders when evaluating the models would make it difficult to see any significant improvements compared to the baseline of placing the articles in order by frequency. This limitation is further motivated by the fact that most of the previous research that applies association rule mining on the SLAP uses data where order sizes are significantly larger than what is used in this project; see [2] and [19] for examples. Removing orders of size one could therefore be seen as a way of making the dataset more suitable for the task, and to make the contributions of this project more comparable with previous research. The limitations on the upper bound stems from large orders adding to the computational complexity and that the rules created from large orders are less informative; see Section 3.1.2 for a more in-depth explanation.

1.4 Literature review

This section showcases some of the more influential work that inspired the direction of this thesis. The section considers ways to create association rules, solving the SLAP and using neural networks for learning similarities between articles in transaction data.

1.4.1 Association rule mining

In this section, a selection of the research related to association rule mining in the context of solving the storage location assignment problem is presented.

In [2], the SLAP is solved using a method that extracts association rules from synthetic order history. Then, a set of article placements are proposed by maximising a fitness score that rewards articles related by rules being placed in adjacent shelves, and penalises distance to depot (weighted by article importance). The method show promising results compared to random and dedicated storage.

Another method of solving the SLAP is described in [19], where the approach is to divide all articles into 8 groups of articles that are connected through association rules. Then, each group is assigned to a zone in the warehouse. The method was evaluated using data from an existing warehouse, where it showed a 14% decrease of travelling distance to pick orders.

In [26], a method that enhances the classical ABC-system using association rule mining is proposed. Along with some other improvements, the original ABC-placements are updated by making sure that if there are two articles within the same class (A, B or C) that are strongly correlated, they are placed in adjacent shelves. The proposed method show significant improvements over the classical ABC-system.

The authors of [9] propose a method for extending association rules between terms in reviews which are similar in sentiment, using similarity in a semantic vector space. The idea is that if there is a rule between a and b, and c is very similar to a, a rule

between c and b is added. This method had a promising outcome.

1.4.2 Distance metric learning

The research in this section focus on learning similarities through the use of different embedding spaces trained by neural networks on articles and images.

In [18], an approach for embedding articles in a space that not only represent semantic similarity, but also what articles are purchased together, is proposed. The method is used within market basket analysis to suggest what articles should be bought next, given a current shopping basket. In this task, the method significantly outperforms the baselines which it is compared to.

Several other approaches to learn item embeddings is proposed in [11] where eight different embedding methods are tested against each other. In [25] adaptive triplet loss is used learn similarity between images, utilising the image itself and its product description. This methods leads to a performance increase across all metrics used.

2
Preliminaries

In this chapter, the most important algorithms and concepts needed to understand the methods in Chapter 3 are presented. First, the concepts of association rule mining (Section 2.1) and distance metric learning (Section 2.2) are described. In this thesis, they are both used for creating rules between articles. Then, the basis of a genetic algorithm is presented in Section 2.3. This algorithm is then adapted to optimise article placements based on a set of rules in Section 3.3.2.

2.1 Association rule mining

The purpose of association rule mining is to extract relationships between items from transaction data; see [21]. A common application of ARM is finding rules between articles based on order history. For example, the interpretation of the rule {jeans, socks} ⇒ {t-shirt} would be that if a pair of jeans and a pair of socks are purchased it is likely that a t-shirt is also purchased. The mining of association rules is typically divided into two main steps; finding all frequent itemsets and creating rules.

2.1.1 Finding frequent itemsets

This step serves the purpose of generating all sets of items that occur more frequently than a predefined minimum support threshold. The support of an itemset $\mathcal{X}$ is defined as the fraction of orders in the entire data set that contains all articles in $\mathcal{X}$; see (2.1). The frequent itemset extraction is typically done using methods like the Frequent Pattern (FP) Growth algorithm; see [4].

$$\text{Support}(\mathcal{X}) = P(\mathcal{X}) = \frac{\text{Number of orders containing } \mathcal{X}}{\text{Total number of orders}} \tag{2.1}$$

Using the example in Table 2.1, and setting the minimum support threshold to 0.4, an itemset need to have occurred at least twice to be considered frequent (since $\frac{2}{4} > 0.4$ but $\frac{1}{4} < 0.4$). Thus, the frequent itemsets of this example would be {jeans, socks, t-shirt}, {jeans, socks}, {jeans, t-shirt}, {socks, t-shirt}, {t-shirt, sweater}, {jeans} and {sweater}, all with a support of 0.5, and {socks} and {t-shirt}, each with the support 0.75.

Order id	jeans	socks	t-shirt	sweater
1	0	0	1	1
2	1	1	1	1
3	1	1	1	0
4	0	1	0	0

Table 2.1: An example set consisting of four orders. A value of 1 at position (i, j) means that item j is purchased in order i, while a value of 0 means that it is not.

2.1.2 Creating rules

After all frequent itemsets are extracted, the next step is to create rules within these sets; see [21]. This is (in the most naive way) done by, for each frequent itemset, creating a rule between all combinations of items within that set. Assuming the itemset {jeans, socks, t-shirt} has a support greater than the minimum support threshold, the following rules would be created:

$$\begin{aligned}
\{\text{jeans}\} &\Leftrightarrow \{\text{socks}\} \\
\{\text{jeans}\} &\Leftrightarrow \{\text{socks, t-shirt}\} \\
\{\text{jeans}\} &\Leftrightarrow \{\text{t-shirt}\} \\
\{\text{socks}\} &\Leftrightarrow \{\text{jeans, t-shirt}\} \\
\{\text{socks}\} &\Leftrightarrow \{\text{t-shirt}\} \\
\{\text{t-shirt}\} &\Leftrightarrow \{\text{jeans, socks}\},
\end{aligned}$$

where the $\Leftrightarrow$ symbol indicates that rules in both directions are created. Then, each rule is given a score and the rule is kept only if that score exceeds a pre-defined threshold. A common score to use is *lift*, which is defined as:

$$\text{Lift}(\mathcal{X} \Leftrightarrow \mathcal{Y}) = \frac{\text{Supp}(\mathcal{X} \cup \mathcal{Y})}{\text{Supp}(\mathcal{X})\text{Supp}(\mathcal{Y})}.$$

An interpretation of this score is the ratio between the support of the itemset $\mathcal{X} \cup \mathcal{Y}$ and the support that would have been observed if $\mathcal{X}$ and $\mathcal{Y}$ were independent. Thus, a score of one indicates independence, a score higher than one indicates that the items are positively correlated, and vice versa for a score below one.

2.2 Distance metric learning

Distance metric learning is a machine learning technique used to learn distances for similarity based applications; see [20]. These learned distances are often more informative than distances such as Euclidean and Manhattan. In the context of deep learning, the distance function learned is the weights of the network in question. During training the weights are guided by the loss function of the network, ensuring that the learned distance has the desired properties. A common application of distance metric learning is facial recognition; see [20]. In this case the Euclidean

distance between two matrix representations of faces would give some indication to their similarity. However, there exists several more suited distance functions for this task.

2.2.1 Triplet loss

One such way of obtaining a more appropriate distance function is to train a network using *triplet loss*; see [5], which is defined as:

$$\mathcal{L}(A, P, N) = \max\left(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha,\ 0\right). \tag{2.2}$$

In (2.2), A is the so-called anchor, P is a positive sample, N is a negative sample, f is an embedding (the output of the network), and α is a parameter determining the scale of the embedding space. Being an anchor sample means that A and P are of the same class while belonging to a different class than N. Since the loss is minimised, in the best case scenario, anchors and positive samples end up close together while the negative samples are at least α away.

After training, the weights are used in order to embed the data in the learned space, i.e applying f on every datapoint. Taking the L_2 norm on the difference between two points in this metric space will, ideally, yield a *similarity* between two data points which is more informative than the norm in $\mathbb{R}^N$.

2.3 Genetic algorithm

A genetic algorithm (GA) is a method for solving optimisation problems inspired by natural selection; see [7]. In a GA, a population of individuals, each representing a candidate solution to the problem, undergoes evolution. This process typically includes crossover, mutation and selection of individuals to breed (perform crossover). The algorithm runs for a number of generations, either a fixed number or until a certain stopping criterion is met. The procedure is illustrated in Figure 2.1.

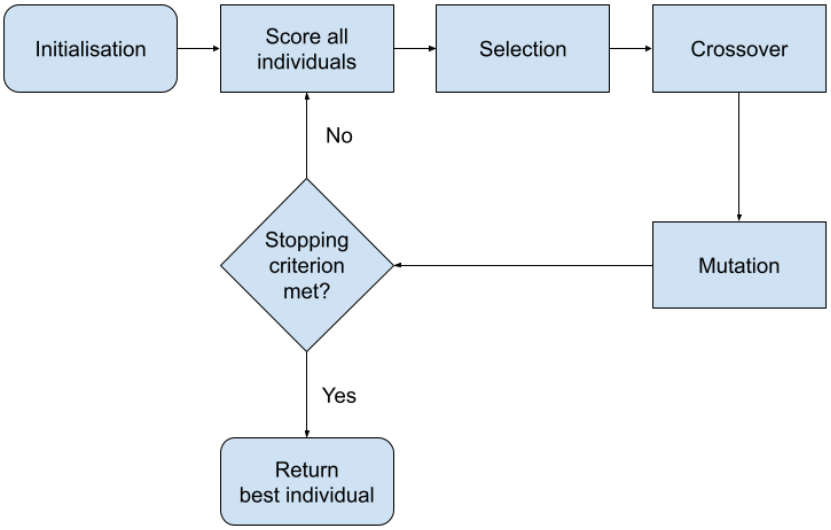


Figure 2.1: Flowchart describing a simple genetic algorithm, the components are described more in depth in the sections 2.3.1 - 2.3.6.

2.3.1 Chromosome

In a GA, an individual consists of genes forming a chromosome, which in turn represents a candidate solution to the problem. For example, if the goal is to optimise a schedule, a chromosome could represent an entire schedule with each gene representing which person works at a certain time slot. During the crossover and mutation phases described in sections 2.3.5 and 2.3.6, it is the genes that are changed between generations.

2.3.2 Population initialisation

The population of candidate solutions is typically initialised in one of two ways; cold start or warm start. During a cold start initialisation, each individual is initialised randomly somewhere within the search space. In a warm start setting on the other hand, some or all of the individuals are initialised to a solution that is known or likely to achieve a better score than random. However, warm starting increases the risk of premature convergence as it decreases diversity within the population; see [17].

2.3.3 Fitness function

Just as in nature, an individual's chance of breeding is dependent on the attributes of that individual. The analogue of this feature in a GA is the fitness score. The fitness of an individual is determined by the fitness function, which differs based on the problem at hand.

2.3.4 Selection

The main idea in selection is to give all individuals some probability of producing offspring for the next generation, with the probability being related to the fitness score. One common method for selection is *tournament selection*; see [7]. This selection procedure consists of two steps, which are executed repeatedly until the new population has reached the same size as the previous population. First, a number of different individuals are randomly sampled from the previous population. Then, these individuals are ranked by their fitness, and based on this ranking they are assigned a probability of having offspring for the next generation. Once the probabilities have been assigned, an individual is sampled from this distribution. As these steps are performed several times and individuals are chosen at random, it is likely that the same individual occurs in the new population more than once, while others may not be selected at all. The procedure is presented in Algorithm 1.

Algorithm 1 Tournament selection

```
new population ← empty list
repeat
    candidates ← n_tour randomly selected individuals from the old population
    distribution ← P(C = c) ∼ fitness(c)
    selected individual ← choice of one candidate from the distribution
    add selected individual to new population
until new population is full
```

2.3.5 Crossover

When a population of parents is selected, they should breed. This is typically done by grouping all parents into pairs and for each pair, performing crossover. When two parents are crossed, they produce two offspring. The chromosomes of the offspring should, just as in nature, be a combination of the parents' chromosomes. One simple crossover method is the one-point crossover, that picks a point in the parents' chromosomes and swaps all elements on the right side of that point to produce the two offspring.

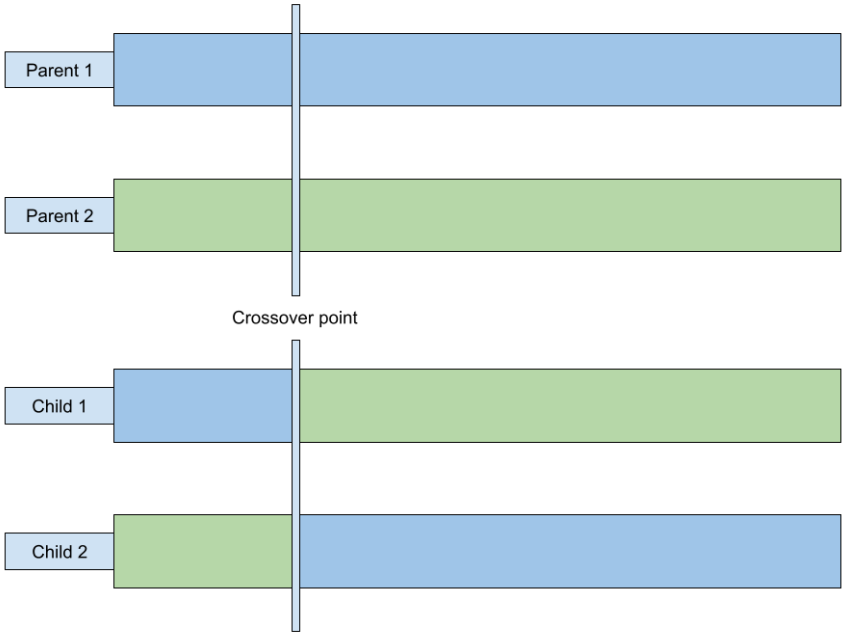


Figure 2.2: Illustration of one point crossover; the children swap values after the crossover point.

2.3.6 Mutation

When crossover has been performed on the parents, the last step of constructing a population for the next generation is to mutate the offspring. Using the example of a bit array chromosome this is commonly done by, for each bit in the array, flipping that bit with a small probability.

3
Methodology

In this chapter, the different methods used in this thesis are described. In Section 3.1, all pre-processing that was made is presented, followed by a description of the three different ways of extracting rules between articles; see Section 3.2. Finally, two algorithms for optimising article placements are described in Section 3.3.

3.1 Data pre-processing

Before running the algorithms proposed in this thesis, some pre-processing was applied to the transaction data. In this section, the different steps of pre-processing are described.

3.1.1 Warehouse layout

Unfortunately, no accurate representation of the warehouse layout was available. Thus, a simple layout was created with a total of 1440 shelves, each with capacity 14. The layout is shown in Figure 3.1.

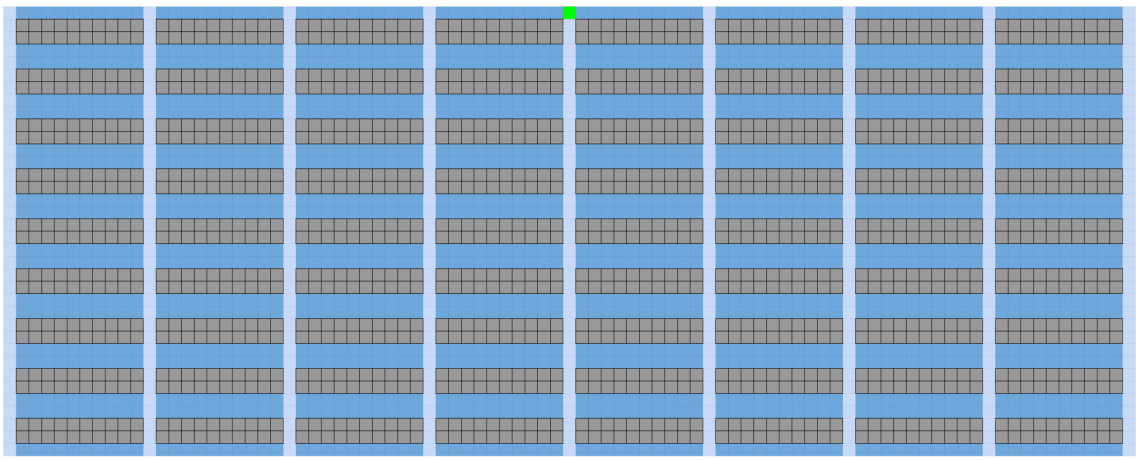


Figure 3.1: Visualisation of the warehouse layout, consisting of 1440 shelves represented by the grey bordered tiles. Between the rows of shelves are horizontal rows of dark blue tiles, representing aisles. These are placed so that each shelf has exactly one adjacent aisle tile. The vertical rows of light blue tiles are walkways, and the green tile at the top is the depot.

3.1.2 Order filtering

The majority of orders in the dataset contained only one article. To be able to extract meaningful association rules, only orders containing two or more articles were considered when constructing the model. Furthermore, orders containing more than 20 unique articles were also disregarded. The upper limit was chosen partly to reduce the computational complexity and partly so that a large company restocking wouldn't create associations between articles that in the rest of the data are unrelated. Assume, for example, that a clothing store purchased five of each available t-shirt and a couple of lighters because the store happened to be out of them at the moment. Associations between lighters and all sorts of t-shirts would be created, which might not have been purchased together any other time in the dataset. The distribution of order sizes after filtering is shown in Figure 3.2.

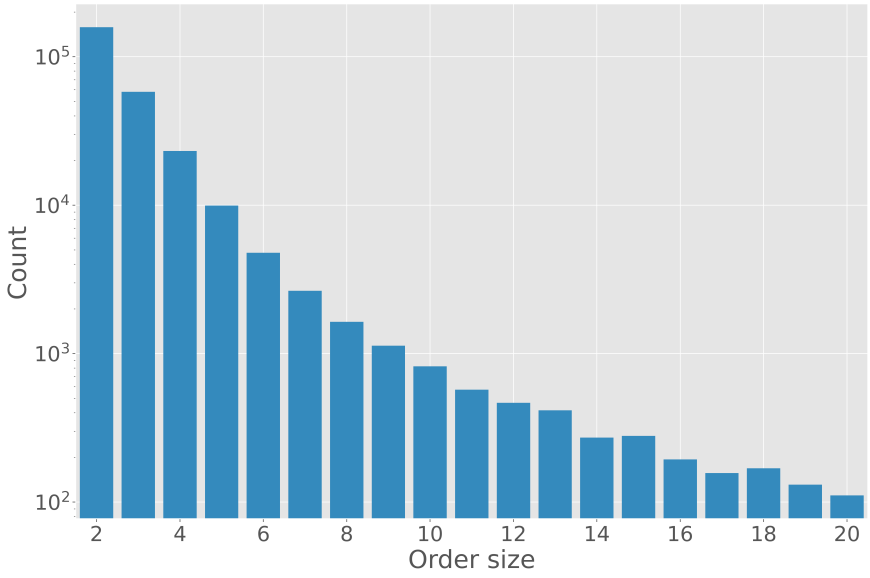


Figure 3.2: Distribution of order sizes in the final data. Note the logarithmic scale on the y-axis.

3.1.3 Article names

As the names of articles in the dataset were unstandardised, the following operations were performed on the article names:

- Numbers and units were written together (10 kWh ⇒ 10kWh).
- All letters were changed to lowercase.
- Punctuation, redundant whitespace, and other special characters were removed.
- Some words had been truncated due to a character limit and were therefore removed.

3.2 Rule mining

In this section, a number of different methods for finding rules between articles are described. The purpose of these rules is to indicate which articles would benefit from being placed closer together.

3.2.1 Association rule mining

The extraction of Association Rules (AR) was carried out in two steps; finding all frequent sets of items and creating rules. The frequent itemsets were found using the F-P growth algorithm in the python package `mlxtend`; see [15]. All itemsets of size 2 were then used to create the association rules. Using each of these sets $\{i, j\}$, the rule $(i \Leftrightarrow j)$ was then created and the lift of the rule was calculated. The lift was then compared with the minimum lift threshold and the rule was kept only if its lift exceeded the threshold. The reason for only using two-way-rules ($(i \Leftrightarrow j)$ instead of $(i \Rightarrow j)$) is that the lift is a symmetric score. This means that if the rule $(i \Rightarrow j)$ is created, so is the rule $(j \Rightarrow i)$ with equal weight.

Furthermore, the weight of the rule $(i \Leftrightarrow j)$ was defined as:

$$w_{ij}^{\mathrm{AR}} = \mathrm{Lift}(i \Leftrightarrow j) - 1.$$

If no rule was found between article i and j, w_{ij}^{AR} was set to 0.

3.2.2 Extending association rules

While the association rule mining method used above is efficient for finding rules between distinct articles in a dataset, there exist ways to extend the rules to make them even more valuable. One way to extend the rules is to take the article names into account. In this section, two methods for extending the association rules are proposed.

3.2.2.1 Nearest neighbours extension

The first method relies on the assumption that articles with similar names have similar properties. For example if there is a rule saying that the article *iPhone 12 screen protector* is related to the article *iPhone 12 lcd display white*, it may be reasonable to think that *iPhone 12 screen protector* is also related to *iPhone 12 lcd display black*. To account for this, a method inspired by the one described in [9] was developed.

First, all articles were encoded in an embedding space based on their names; see Section 3.2.2.4. For all rules $(i \Leftrightarrow j)$, new rules were then created between i and the k nearest neighbours of j, provided they lie within a radius r, and vice versa between j and the k closest articles to i. This procedure is shown in Algorithm 2

The weight of a nearest neighbour extended association rule ($\mathrm{EAR_n}$) was defined as follows:

$$w_{kl}^{\mathrm{EAR_n}} = \mathrm{sim}(i,k) \cdot \mathrm{sim}(j,l) \cdot w_{ij}^{\mathrm{AR}} \tag{3.1}$$

where $\mathrm{sim}(i,j)$ is the similarity between article i and j. The similarity was defined as:

$$\mathrm{sim}(i,j) = \frac{\beta}{\mathrm{dist}(i,j) + \beta}, \tag{3.2}$$

where $\mathrm{dist}(i,j)$ is the distance between article i and j in the embedding space provided and $\beta \geq 0$ is a small constant with the purpose to make the similarity take values between 0 and 1.

Assuming a rule has the form (i, j, weight), where i and j are the antecedent and consequent respectively, the procedure for extending association rules can be described as follows:

Algorithm 2 Nearest neighbours extension of an association rule

function NEAREST NEIGHBOURS EXTENSION($i, j, w_{ij}^{\mathrm{AR}}$)
    nearbyRules ← empty list
    $A_k \leftarrow k$ nearest neighbours of i
    $C_k \leftarrow k$ nearest neighbours of j
    for $a \in A_k$ **do**
        if $\mathrm{dist}(a,i) < r$ **then**
            Add $(a, j, \mathrm{sim}(a,i) \cdot w_{ij}^{\mathrm{AR}})$ to nearbyRules
    for $c \in C_k$ **do**
        if $\mathrm{dist}(a,i) < r$ **then**
            Add $(c, i, \mathrm{sim}(c,j) \cdot w_{ij}^{\mathrm{AR}})$ to nearbyRules
    return nearbyRules

Note that there may be more than one rule added for a specific pair of articles. When calculating the final rules, all weights of different rules between the same two articles were simply added together.

3.2.2.2 Parallel rules extension

In a similar manner as above, if the *iPhone 13* was released towards the end of the time when the data was gathered, articles related to *iPhone 13* may not have been bought enough to get appropriate rules between them. However, if there is a rule between *iPhone 12 screen protector* and *iPhone 12 lcd display white*, there should probably be a rule between *iPhone 13 screen protector* and *iPhone 13 lcd display white* as well. To create these parallel rules, Algorithm 3 was developed. This algorithm was run for all regular association rules. For parallel extended association rules ($\mathrm{EAR_p}$), the weight of a rule is simply defined as:

$$w_{kl}^{\mathrm{EAR_p}} = w_{ij}^{\mathrm{AR}} \tag{3.3}$$

Algorithm 3 Parallel rules extension of an association rule

function PARALLEL RULES EXTENSION($i, j, w_{ij}^{\mathrm{AR}}$, allArticles)
    parallelRules ← empty list
    for $a \in$ allArticles **do**
        emb ← embed(i) - embed(j) + embed(a)
        if emb is the embedding of an article b **then**
            Add $(a, b, w_{ij}^{\mathrm{AR}})$ to parallelRules
    return parallelRules

where embed(article) is a function returning the embedding of the input article in the embedding space provided.

3.2.2.3 Combining extended association rules

Once extended association rules of the two types above were extracted, they were combined using the scaling parameter C_p:

$$w_{ij}^{\mathrm{EAR}} = w_{ij}^{\mathrm{EAR_n}} + C_{\mathrm{p}} w_{ij}^{\mathrm{EAR_p}}$$

3.2.2.4 Embedding spaces

In order to compute the similarity between two products, they must first be encoded in an embedding space. In this section, three approaches for constructing this embedding space are described.

Bag of Words

In its simplest form, the embedding space for computing the distance between articles was created by converting the article names to a Bag of Words format; see [14]. In this embedding space each article is represented by a vector of dimension equal to the number of unique word in the entire dataset. The value of a bit in this vector is then set to 1 if the word it corresponds to is a part of the article's name.

The distance between two articles was then defined as the L_1-norm between their respective embeddings. An interpretation of this distance is the number of words that differ between the two article names.

Word weight by single co-occurrence

In order to improve the embedding space, each word was weighted in accordance with a score indicating importance. For example, it would be intuitive to think that the word ***iPhone*** says more about an article than the word ***and***. Thus, it is desired that the articles "*repair kit **and** tools for iPhone*" and "*repair kit **with** tools for iPhone*" are considered more similar than the articles "*repair kit and tools for **iPhone***" and "*repair kit and tools for **Samsung***". In both cases, the number of words that differ is the same, but if words like ***iPhone*** and ***Samsung*** were weighted higher, the second pair of articles would be considered less similar than the first pair.

Since words like ***and*** could be very common, just weighting the words by the frequency of which they occur would probably not give the desired effect. Thus, a more sophisticated weight was constructed and is shown in Algorithm 4. The idea of this algorithm is to relate the number of times a word occurs in two different articles within the same order to the total number of occurrences.

Algorithm 4 Single co-occurrence score

```
initialise count and matches to 0 for all words
for order in allOrders do
    for article in order do
        for word in article do
            count[word] += 1
    for all pairs (article1, article2) in order do
        for word1 in article1 do
            for word2 in article2 do
                if word1 == word2 then
                    matches[word1] ← matches[word1] + 1
for word in uniqueWords do
    weight[word] ← matches[word] / count[word]^γ
```

where $\gamma \geq 0$ is a hyperparameter.

The BoW embedding space was then scaled with the word scores. This means that if two articles differ by the word *iPhone*, weight[*iPhone*] was added to the distance between the articles, rather than just 1 as in BoW.

Word weight by pair co-occurrence

While the method above captures relationships where two articles containing the word *iPhone* are more likely to be bought together than two articles containing the word *and*, it fails on capturing others. For example, an article containing the word *display* might be likely to be bought together with one containing the word *protector* (as in the examples in the beginning of Section 3.2.2). Relationships like this could also be an indicator for word importance, even though it may not be likely that two articles both containing the word *protector* are bought together. To account for this, Algorithm 4 was modified as follows:

Algorithm 5 Pair co-occurrence score

```
initialise count, weight and matches to 0 for all words
initialise pairMatches to 0 for all possible word pairs
for order in allOrders do
    for article in order do
        for word in article do
            count[word] += 1
    for all pairs (article1, article2) in order do
        for word1 in article1 do
            for word2 in article2 do
                if word1 == word2 then
                    matches[word1] += 1
                else
                    pairMatches[word1][word2] += 1
for word in uniqueWords do
    weight[word] ← matches[word] / (count[word])^γ
    weight[word] += matches[word] / (count[word])^γ
    for word2 in uniqueWords do
        if word1 ≠ word2 then
            score ← pair_matches[word][word2] / (count[word]+count[word2])^δ
            weight[word] += score
            weight[word2] += score
```

where $\gamma \geq 0$ and $\delta \geq 0$ are hyperparameters.

3.2.3 Neural network rules

As another way of generating rules, distance metric learning was used by training a neural network to encode articles in an alternative embedding space. In this space, the idea is to not only embed semantically similar articles closely, but also articles that are frequently bought together.

3.2.3.1 Creating training samples

The training samples were created by taking the data in BoW-format and creating triplets consisting of an anchor, a positive and a negative sample. The anchor and the positive sample belongs to the same order, while the negative sample is chosen randomly from all articles not in that order; see Algorithm 6.

Algorithm 6 Generation of training samples

```
samples ← empty list
for each order in allOrders do
    for all pairs (anchor, positive) ∈ order do
        negative ← random choice among articles ∉ order
        append (anchor, positive, negative) to samples
```

3.2.3.2 Triplet ratio loss

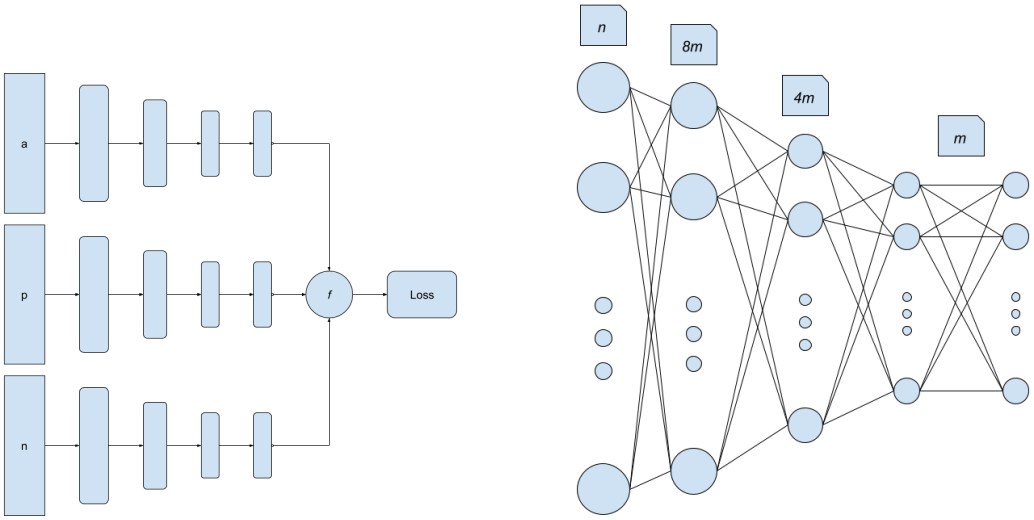
While triplet loss is quite useful when training networks for tasks such as facial identification and other similarity based applications, there are a couple of drawbacks. For example, the loss being set to 0 when the first term of Equation (2.2) is negative leads to the gradient missing information during backpropagation because it is unknown exactly how negative it was. In order to achieve good results, a large number of negative samples for each positive sample is required; see [13]. To combat these factors, a novel *triplet ratio loss* function was developed; see (3.4). This loss is based on the same idea of triplets as the *triplet loss* function in (2.2), but instead of taking the difference between the positive and negative norm they are divided. This leads to the network being rewarded more for spacing the anchor and negative sample further apart. In (3.4), A, P and N are the anchor, positive sample and negative sample respectively and α is a constant for further weighting the two norms against each other.

$$\mathcal{L}_{\mathrm{r}}(A, P, N) = \frac{\|f(A) - f(P)\|}{\|f(A) - f(N)\|^{\alpha}} \tag{3.4}$$

3.2.3.3 Network architecture

The architecture of the network was relatively simple, consisting of four fully connected layers, and can be seen in Figure 3.3. The first two layers used the leaky ReLu activation function; see [23], and the last two used linear activation. These functions were chosen so that the embedding space could include negative numbers;

if for example sigmoid activation had been used the embedding space would effectively been cut in half. After each of the first two layers, a dropout layer with a dropout rate of 0.1 was added for regularisation. The sizes of the layers were scaled with the embedding dimension m.



(a) Schematic view of the network layout.

(b) The layout of each branch in the network.

Figure 3.3: Illustration of the neural network design. Each of the three branches in subfigure 3.3a has the layout of subfigure 3.3b. Additionally, the three branches have identical weights.

3.2.3.4 Creating rules

When the model was trained, all unique articles were encoded in the model's embedding space. Then, for each article, rules were created between it and up to k of its nearest neighbours, provided that they lie within a radius r. The weight of these rules were assigned the similarity between the two articles, $w_{ij}^{\text{NNR}} = \text{sim}(i, j)$, where $\text{sim}(i, j)$ is the same as in Equation (3.2). In Algorithm 7, the procedure for creating rules from an article i to its nearby articles is shown.

Algorithm 7 Create rules between article i and the k closest articles within r

```
function NNR(i, k, r)
    nnRules ← empty list
    e_i ← f(i)                          ▷ f(i) embeds article i
    N_k ← k nearest neighbours of e_i
    for e_j ∈ N_k do
        if distance(e_i, e_j) < r then
            add (i, j, sim(i, j)) to nnRules
    return nnRules
```

3.2.4 Weighting rule types against each other

Once all rules were created, a final weight matrix representing all rules was constructed as follows:

$$W_{ij} = w_{ij}^{\mathrm{AR}} + C_{\mathrm{EAR}} w_{ij}^{\mathrm{EAR}} + C_{\mathrm{NNR}} w_{ij}^{\mathrm{NNR}},$$

where C_{EAR} and C_{NNR} are constants scaling the impact of each rule type.

3.3 Optimisation

After finding finding the set of rules, the next step was to use them and place the articles into the warehouse. Two algorithms were implemented to solve this task, one deterministic approach with a greedy algorithm and one stochastic approach with a genetic algorithm.

3.3.1 Greedy algorithm

In order to utilise the rules in a computationally efficient way, a greedy algorithm was implemented. This algorithm determines the placement of one article at a time by executing two main steps; determining which article to place, and where to place it. An overview of the greedy algorithm is shown in Algorithm 8.

Algorithm 8 Greedy algorithm

```
articles ← list of all articles
availableShelves ← list containing the shelf or shelves closest to the depot
set the score of all articles to their support
repeat
    sort articles in ascending order by score
    chosenArticle ← pop last element from articles
    chosenShelf ← best scoring shelf in availableShelves
    update article scores based on chosenArticle
    if chosenShelf is full then
        remove chosenShelf from availableShelves
        add the neighbours of chosenShelf to availableShelves
until all articles are placed
```

3.3.1.1 Choosing article

By maintaining an ordered list over all unplaced articles, the article to place is determined by simply picking the last element in the list. The list is sorted in ascending order by a score which is calculated as:

$$\text{articleScore}(i) = s_i + C_\text{r} \sum_{j \in \mathcal{P}} W_{ij}, \tag{3.5}$$

where s_i is the support for article i, $\mathcal{P}$ is the set of articles that are already placed, W_{ij} is the weight of the rule between article i and j, and C_r is a constant.

The first term in (3.5) ensures that common articles are picked earlier than uncommon ones. The second term makes the algorithm prioritise articles that have rules to other articles that are already placed, allowing for them to be placed closer together. Finally, the constant C_r determines the weighting of the two terms.

3.3.1.2 Determining location

After an article is chosen, the next step of the algorithm is choosing a shelf for storing the article. This is done by giving each shelf a score, and picking the one with the lowest score. The score for placing article i on shelf k is defined as:

$$\text{shelfScore}(i, k) = C_\text{d} s_i d_k + \sum_{j \in \mathcal{P}} W_{ij} D_{kl} + C_\text{p} \sum_{j \in \mathcal{P}_k} s_i s_j, \tag{3.6}$$

where s_i, $\mathcal{P}$ and W_{ij} are the same as in (3.5), d_k is the distance between shelf k and the depot, l is the shelf where article j is stored, D_{kl} is the distance between shelf k and shelf l, $\mathcal{P}_k$ is the set of articles that are already placed in shelf k, and C_d and C_p are constants.

The first term in (3.6) places frequently purchased articles close to the depot, while the second term favours grouping articles with rules between them together. The final term penalises choosing a shelf with other common articles. Assuming that this term is small compared to the others, it could be a tie-breaker between similarly scoring shelves to pick the one with the fewest unrelated articles. This would allow the algorithm to later place other articles related to those in $\mathcal{P}_k$ in shelf k.

To improve computational efficiency, the shelfScore is not calculated for all shelves every iteration. Instead, only the shelves closest to the depot are used in the beginning, and as these are filled up, their neighbours are added to a list of relevant shelves. Using the example layout in Figure 3.4, the first articles are automatically placed in shelf 1 until it is full. Then, shelves 2 and 9 are added to the list. Assuming shelf 2 is the next to become full, shelves 3 and 10 are then added, and when shelf 9 becomes full only shelf 17 is added (since shelf 10 is already added). Using this pattern, the shelfScore will only be calculated for a small fraction of all shelves each iteration, while still keeping the shelves that have potential of being the best one.

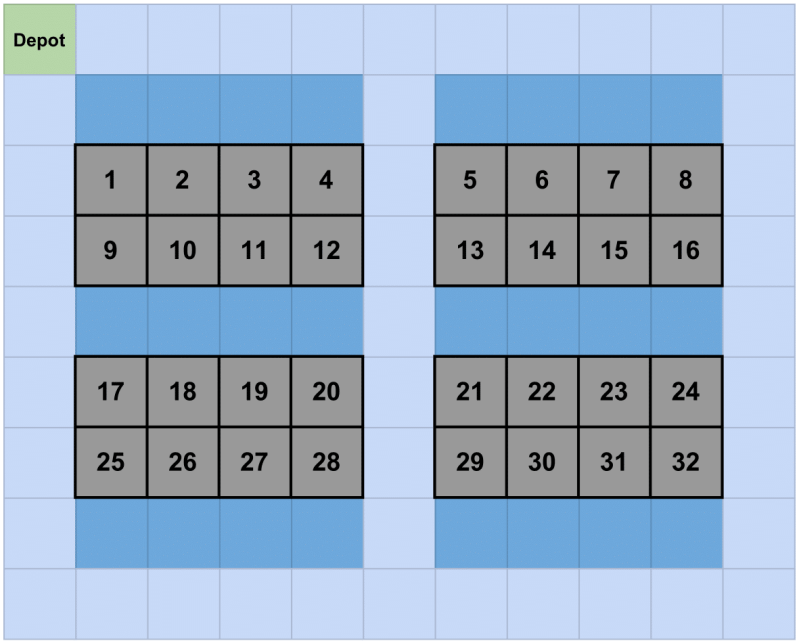


Figure 3.4: Warehouse layout for illustrating how shelves are added to a list of relevant shelves in the greedy algorithm.

3.3.2 Genetic algorithm

In addition to the greedy algorithm, a genetic algorithm was implemented. Instead of placing articles one at a time, this algorithm attempts to minimise a score that evaluates an entire set of article placements. In this section, the implementation of the different steps of the genetic algorithm is explained. These steps were executed a fixed number of times, n_{gen}.

3.3.2.1 Chromosome

The chromosome of an individual in the population was defined as a list where each index represents an article and the value at that index represents the shelf at which the article is stored. An example of a chromosome is shown in Figure 3.5.

Article	0	1	2	3	4	5	6	7	8	9
Shelf	3	1	1	3	4	2	2	0	4	0

Figure 3.5: A chromosome where ten different articles are stored at 5 different shelves, each capable of storing two articles.

3.3.2.2 Population initialisation

The population of size n_{pop} was initialised using a warm start. This meant that 95% of the individuals were randomly initialised to any feasible solution, while the final 5% were initialised to a solution where the most common articles were placed in the shelves closest to the depot.

3.3.2.3 Fitness function

The fitness of an individual I was defined as follows:

$$\text{fitness}(I) = \sum_{i=1}^{m} \left(C_{\text{d}} s_i d_{I_i} + \sum_{j=1}^{m} W_{ij} D_{I_i I_j} \right), \tag{3.7}$$

where m is the number of unique articles, s_i is the support of article i, I_i is the shelf article i is stored at, d_{I_i} is the distance between shelf I_i and the depot, W_{ij} is the weight of the rule between article i and article j, $D_{I_i I_j}$ is the distance between shelf I_i and shelf I_j, and C_{d} is a constant. Since this function is very similar to the objective function in (1.3), a low fitness score is desirable.

3.3.2.4 Selection

The selection of individuals for a new population is done using tournament selection, as described in Algorithm 1. However, the algorithm was modified so that after the candidates were scored, the best scoring one was always chosen. This modification was made in order to reduce the amount of hyperparameters.

3.3.2.5 Crossover

After selecting n_{pop} individuals, crossover is performed. The crossover method described in this section is inspired by the one proposed in [24]. When two parents are crossed, their children are first initialised as copies of their parents. A number, n_{swaps}, of indices are then selected at random and the values at these indices are swapped in the children.

An example of crossover between two parents is shown in Figure 3.6. For this example it is assumed that there are ten articles to be placed in five shelves, each one with a capacity of two articles. Furthermore, the articles are ordered so that the article with the highest support corresponds to index 0 and the article with the lowest support to index 9. The shelves are also ordered so that 0 corresponds to the shelf closest to the depot.

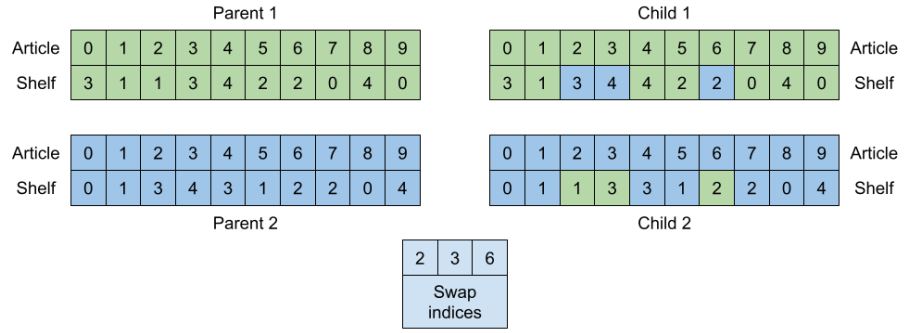


Figure 3.6: Given two parents and using $n_{\text{swaps}} = 3$ the children to the right would be created.

After these swaps have been performed one can observe that there are overfull shelves in the children; shelf 4 in child 1 and shelf 1 in child 2 each contain three articles. In order to maintain a feasible solution, some of these articles need to be moved to a different shelf, namely the closest shelf to the depot with space left over. This is illustrated in Algorithm 9.

Algorithm 9 Make solution feasible

```
for each article a do                                   ▷ in reverse order
    if shelf where a is stored is overfull then
        move a to the shelf with empty space closest to the depot
```

The reason for iterating through the articles in reverse order is that the least frequent articles should be moved. After these changes in the chromosomes have been performed they end up as the chromosomes in Figure 3.7.

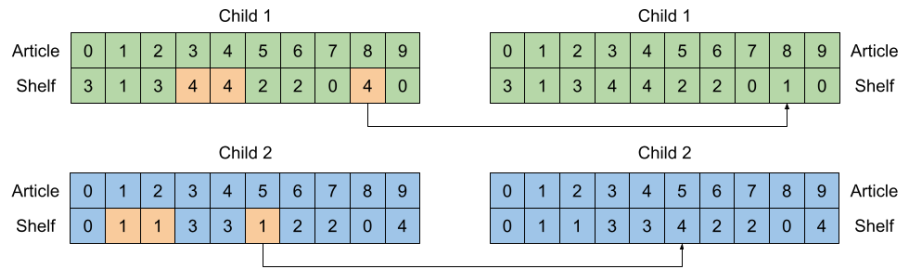


Figure 3.7: Illustration of Algorithm 9 on the children from Figure 3.6.

3.3.2.6 Mutation

The mutation step is executed for all children obtained after performing crossover. For each gene in the chromosome, a mutation is performed with probability p_{m}. The

mutation of a gene corresponds to swapping the value of that gene with the value of another randomly chosen gene. This is guaranteed to preserve feasibility since it only swaps two articles' places in the warehouse.

3.4 Evaluation

The evaluation of the models was performed on a test set consisting of orders from a separate time period. Each order was treated as a TSP where the location of all articles in the order were visited, starting and ending in the depot. Using a dynamic TSP solver from the package `python_tsp`; see [3], the minimum distance required to pick all articles in each order was then determined and the mean distance across all orders was calculated.

3.4.1 Baselines

To make the evaluation meaningful, the models were compared to two different baselines:

Random: Completely random article placements. However, if there are more shelf spaces than articles in the warehouse the articles are pushed towards the depot.

By frequency: An algorithm that orders the articles by frequency and the shelves by distance to depot, and places the most frequent articles on the shelves with the shortest distance to the depot.

4

Tests and results

This chapter presents the tests and results of the hyperparameter tuning and the evaluation of the two algorithms used to solve the SLAP. The code base produced during the project can be found at https://github.com/Stickish/EARMfStSLAP.

4.1 Choice of hyperparameters

In this section, the final choices of hyperparameters are presented. The procedure for tuning hyperparameters is described in detail in Appendix A.

Parameter	Value
$s_{\min}$	10^{-4}
$L_{\min}$	1

Table 4.1: Chosen hyperparameters for ARM, $s_{\min}$ is the minimum support threshold and $L_{\min}$ the minimum lift threshold.

Parameter	Value
k	6
r	6
β	0.6
C_{p}	1000

Table 4.2: Tuned hyperparameters for EAR using the BoW embedding space. The parameter k determines the maximum number of rules to create within the radius r, β scales the similarity between articles, and C_{p} weight the parallel and nearest neighbour rules against each other.

Parameter	Value
γ	1.6
δ	3.5
k	2
r	1
β	0.1
C_{p}	10

Table 4.3: Tuned hyperparameters for EAR using the embedding space based on the pair co-occurrence score; see Section 3.2.2.4. The parameters γ and δ scale the embedding space, and the remaining parameters are the same as in Table 4.2.

Parameter	Value
α	1
m	256
k	100
r	10^{-3}
β	10^{-4}

Table 4.4: Chosen hyperparameters for NNR. α determines the weight between positive and negative samples during training, m is the embedding dimension, and k, r and β are the same parameters as in Table 4.3.

Parameter	Value
C_{EAR}	10^{-4}
C_{NNR}	10

Table 4.5: Hyperparameters for weighting different kinds of rules together.

Rule type	C_{r}	C_{d}	C_{p}
AR	1	$5 \cdot 10^4$	0
$\mathrm{EAR}_{\mathrm{BoW}}$	1	10^8	0
EAR	1	$5 \cdot 10^5$	0
NNR	1	10^4	0
All rules	1	$5 \cdot 10^5$	0

Table 4.6: Tuned hyperparameters for the greedy algorithm when using different kinds of rules. C_{r} weight the articleScore terms in (3.5) against each other, while C_{d} and C_{p} weight the shelfScore terms in (3.6).

Parameter	Value
n_{tour}	3
n_{pop}	10
n_{swaps}	100
n_{gen}	5000
p_{m}	10^{-3}

Rule type	C_{d}
AR	10^5
EAR_{BoW}	$3 \cdot 10^8$
EAR	$5 \cdot 10^6$
NNR	10^5
All rules	$5 \cdot 10^6$

Table 4.7: Chosen hyperparameters for the genetic algorithm. The parameters in the left table are kept constant for the different rule types, while C_{d} varies depending on rule type.

4.2 Evaluation

With the hyperparameter choices in Section 4.1, the two algorithms were run using different types of rules and on training data from different time periods. The number of orders and unique articles in each time period is shown in Table 4.8. A test set consisting of 59062 orders between December 2021 and January 2022 was used for evaluation.

Time period	Orders	Unique articles
1 (dec20-feb21)	63292	15106
2 (dec21-nov21)	697987	19551
3 (sep21-nov21)	70086	14678

Table 4.8: Number of orders and unique articles in the different time periods used for training. The number of unique articles also include all articles in the test set.

4.2.1 Baselines

In Table 4.9, the average travel distance for the two baselines are shown for the different training sets.

Training period	Random	Frequency
1 (dec20-feb21)	116.07 ± 2.01	71.41
2 (dec21-nov21)	133.71 ± 2.47	54.23
3 (sep21-nov21)	112.91 ± 1.08	48.15

Table 4.9: Average distance travelled to pick all test orders for the two baseline placements. The values for the random baseline are the means and standard deviations over five randomisations.

4.2.2 Greedy algorithm

The greedy algorithm was run using training data from three different time periods. From each training set, the algorithm was run using five different sets of rules. The results are presented in Table 4.10-4.12.

Rule type	Distance	Change from baseline (%)	
		Random	Frequency
AR	70.85	-39.0	-0.8
EAR_{BoW}	70.39	-39.4	-1.4
EAR	70.51	-39.3	-1.3
NNR	68.62	-40.9	-3.9
All rules	67.09	-42.2	-6.0

Table 4.10: Average distance travelled to pick all test orders for the different types of rules using the greedy algorithm, with rules from a three month training period between December 2020 and February 2021 (period 1).

Rule type	Distance	Change from baseline (%)	
		Random	Frequency
AR	53.42	-60.0	-1.5
EAR_{BoW}	53.28	-60.2	-1.8
EAR	52.88	-60.5	-2.5
NNR	-	-	-
All rules*	48.24	-63.9	-11.0

Table 4.11: Average distance travelled to pick all test orders for the different types of rules using the greedy algorithm, and using rules from a one year training period between December 2020 and November 2021 (period 2). *The neural network rules were however not generated on this training period due to memory limitations. Thus, the results for all rules combines AR and EAR for this training period with NNR from period 3.

Rule type	Distance	Change from baseline (%)	
		Random	Frequency
AR	47.28	-58.1	-1.8
EAR_{BoW}	47.32	-58.1	-1.7
EAR	46.72	-58.6	-3.0
NNR	45.83	-59.4	-4.8
All rules	46.71	-58.6	-3.0

Table 4.12: Average distance travelled to pick all test orders for the different types of rules using the greedy algorithm. The rules were mined from a three month training period between September and November 2021 (period 3).

4.2.3 Genetic algorithm

The results of the genetic algorithm were not very promising, mainly in terms of travel distance but also in terms of computational time. Therefore, it was only run on training period 3. The results are shown in Table 4.13.

Rule type	Distance	Change from baseline (%)	
		Random	Frequency
AR	81.85	-37.5	+70.0
EAR_{BoW}	88.79	-21.4	+84.4
EAR	78.16	-23.6	+79.1
NNR*	48.15	-57.4	0
All rules*	48.15	-57.4	0

Table 4.13: Average distance travelled to pick all test orders for the different types of rules using the genetic algorithm. The rules were mined from the three month training set between September and November 2021 (period 3). *For NNR and All rules, the GA did not manage to find a better fitness score than that of the warm start solution. Thus, the results for these instances are the same as the frequency baseline.

5
Discussion

This chapter reviews and analyses the performance and implementations of the methods used. While the greedy algorithm outperformed both baselines for all rule types, its genetic counterpart only outperformed one. Thus, all results discussed in this chapter relates to the greedy algorithm, if not otherwise stated. Furthermore, the strongest performing set of rules were the combination of all three rule types, although it underperformed on the last dataset (period 3). Looking at the rule types separately, the NNR performed the best. However, they were unable to be trained on the full dataset due to a lack of computational resources.

5.1 Hyperparameter tuning

With the number of hyperparameters being large and the algorithms time-consuming to run, the hyperparameters where tuned using a simplified scheme (described in Appendix A). Thus, the chosen hyperparameter values are likely not the truly optimal values. If a more extensive hyperparameter tuning was performed, it is therefore possible that the results could be improved further.

5.2 Rule mining

In this section, the creation of different rule types are analysed from a critical perspective. Furthermore, the results from running the greedy algorithm using all rule types are, both individually and combined, are discussed.

5.2.1 Association rules

The performance of using association rules was better than both baselines for all time periods of training data. This is consistent with the improvements made by others using association rules for the SLAP; see for example [2] and [19]. However, since data sets, warehouse layout, problem formulation, and baselines differ, it is difficult to make a fair comparison to the results of these articles.

5.2.2 Extending association rules

When only using extended association rules, the average picking distance was slightly lower than with regular association rules on all training periods. This is an indication that there are relationships between the names of articles that AR fails to

uncover. Furthermore, the original association rules are not included in EAR. The full potential of EAR would therefore be expected when it is combined with other rules.

When it comes to the embedding space used for creating extended association rules, scaling the words using the measure of pair co-occurrence described in Section 3.2.2.4 was beneficial in two out of three training periods. Although the differences are quite small, it is worth noting that the largest difference comes in period 3. The difference is slightly smaller for period 2, and the smallest for period 1 where EAR_{BoW} even outperforms EAR with a small margin. While the co-occurrence score seem promising for period 2 and 3, a possible explanation for the lower performance during period 1 is that the time gap between training and testing introduces a larger number of unseen words in the test set. Thus, article names differing by unseen words would be considered more similar than those differing by other words, which might not be beneficial. A way of mitigating this problem could be to use some word embeddings that take into account similarities between words. However, for our particular dataset, the vocabulary is very different from that of any embeddings available online which makes it difficult to implement.

While no results were explicitly generated for the single co-occurrence score, one may observe that both scores are the same for $\delta \to \infty$. Thus, the single score was discarded during hyperparameter tuning of the pair co-occurrence score. However, these scores were not based on any scientific literature study and there are likely better ways of scaling the embedding space than the scores used in this thesis.

5.2.3 Neural network rules

The results of the NNR are promising, outperforming the other rule types on all training periods. However, when creating the training samples for the whole year period, even when saving them as 8bit integer vectors, they require $\approx$ 37 GB of memory. This leads to training taking extremely long and periodically crashing, meaning that it was infeasible to let the training run long enough to be confident in the embedding space learned. This is the reason why the September–November rules were used in Table 4.11. One way we could have decreased the space necessary to train on period 2 is to have performed some sort of dimensionality reduction, for example principal component analysis. This was not implemented due to time constraints and early testing did not seem promising.

A caveat with the network is that there is no way of guaranteeing that it embeds the articles in a space with the same scale each time. This affects the hyperparameter r, the radius within which articles must be to form network rules between them. In order to combat this problem some kind of normalisation could be implemented to force the embedding spaces to share a common scale, ensuring that the hyperparameters are sufficiently optimal for all training sets.

Another potential avenue of improving the performance of the NNR is to look into

more advanced language processing tools such as n-grams. With our particular data however it is unlikely that this would have yielded any improvement due to the unstandardised article names. They did not follow a uniform order and the names cut off after 50 characters, leading to several "halfwords" being included in the data since we decided against manually combing through all article names and removing these halfwords.

5.2.4 Combining rule types

The result using all rules on period 1 and 2 can be seen as an indication that combining rule types could be an efficient strategy. However, it is not always beneficial, as NNR outperform the combination of rules for period 3. A possible explanation for this is that combining rules introduces two additional hyperparameters. With the simplified tuning scheme, adding more parameters could lead to hyperparameters values further the from optimal values, leading to decreased performance. Another reason for this decrease in performance could be the absence of normalisation discussed in Section 5.2.3. The rule combination result for period 3 is very similar to that of EAR, which could be an indication that the scale of the other rule types was negligible compared to EAR.

5.3 Optimisation

This section compares the performance of the two optimisation algorithms used. Possible flaws in the two algorithms, especially flaws associated with the genetic algorithm, are also analysed.

5.3.1 Greedy algorithm

As can be seen in Tables 4.12 and 4.13, the greedy algorithm performed significantly better than its genetic counterpart. However, this is more indicative of the poor GA results than an indication of the greedy algorithm's performance. Due to the fact that the dataset used is unique to this project, it is also difficult to compare the results of the greedy algorithm to that of previous research within the area. However, it is very unlikely that the greedy algorithm manages to find the optimal solution. Thus, a more sophisticated optimisation algorithm would probably be able to showcase the full potential of the rules that were created.

Furthermore, the hyperparameter tuning gave a value of $C_{\mathrm{p}} = 0$ in (3.6). While it is possible that there might be some favourable way of penalising unrelated articles being placed close together, $C_{\mathrm{p}} = 0$ being the optimal value show that the penalty term used in this thesis was not beneficial.

5.3.2 Genetic algorithm

By comparing the results from the genetic algorithm in Table 4.13 with the frequency baseline, it is evident that the performance of the GA is unsatisfactory. The

reasons for the poor results using the GA are however unclear. Previous research applying genetic algorithms for solving the SLAP has shown promising results; see for example [24]. Thus, the overall idea should not be rejected. However, while the crossover and mutation operations of this project were inspired by [24], they had to be adjusted for our problem. As our objective was different, the fitness function also differed, and it is therefore possible that the combination of genetic operators used was simply not suitable for the problem.

Moreover, the genetic algorithm had a large number of hyperparameters and was very time-consuming to run. Therefore, the hyperparameters were tuned in a less extensive manner than those of the greedy algorithm, which could have made them further from the optimal values.

However, the most likely reason for the unsatisfactory performance is that the fitness function was somehow inappropriate. This theory comes from the fact that when the fitness score improved during training, the performance decreased. Since the fitness function builds on the same principles as in the greedy algorithm, we believe that the fitness function as a whole was not the problem, but rather that the parameter C_{d} was not tuned correctly.

5.4 Performance on different time periods

To evaluate the robustness of the different methods, they were run on three different periods of training data; a three month period starting a year before the test set (period 1), a one year period also starting a year before the test set (period 2) and a three month period ending just before the test set (period 3).

When comparing all three periods, one may observe that the results are the best for period 3 in all cases. One reason for this is likely that the dataset is temporally sensitive, with some articles becoming outdated and newer ones being launched. Thus, having recent data seem more important than the amount of data. Another possible explanation is that the model is constructed so that all articles bought at least once in either the train or test period have to be stored in the warehouse. Using period 1 and 2 will therefore keep more articles that have become outdated, increasing the average distance between articles. However, performance on period 2 is significantly better than on period 1. Since the number of articles that are never bought in the test set is larger for period 2, this indicates that recency and quantity of the data is in this case more important than number of "redundant" articles.

Furthermore, the results for period 3 is consistently better that those of period 1, even though the number of articles to place in the warehouse is very similar; see Table 4.8. This indicates that, for this particular dataset, recency in the data is more important than the time of year (as the first training period takes place at almost the same time of the year as the test set). However, this may of course differ from dataset to dataset. For example, a company selling ski wear at winter and hiking equipment at summer may benefit more from using data from a specific time

of the year.

When comparing the results with the random baseline, it is natural that the differences in performance are largest for period 2. The reason for this is that while the other algorithms may benefit from the larger amounts of training data, the random baseline cannot, and would therefore only get the negative consequences of large amounts of data. Furthermore, the improvements over the random baseline are much larger for period 3 than for period 1. This could be seen as a further indication that the recency of data is far more important than the number of outdated articles.

5.5 Evaluation

While orders are typically batched before picking, this thesis uses an evaluation scheme which simulates picking orders individually. The main reason for using this scheme is that batching the orders would add more stochasticity to the results. Using some sophisticated batching method could however simulate reality better, but would be more costly with respect to both computation and implementation time. A reasonable idea for further developing this project would therefore be to implement a more advanced evaluation method using order batching.

6
Conclusions and future work

In conclusion, the rule based approach proposed in this thesis outperforms the industry standard of storing the most frequent articles closest to the depot between 0.8% and 11%. The best result is achieved when combining the different rule types in the greedy algorithm. Looking at the rule types separately, NNR consistently outperforms both EAR and AR. The fact that the GA performed only slightly better than fully random article placements is not indicative of genetic algorithms in general, rather that the combination of genetic operators used were faulty for this task. The two biggest contributions of this thesis are the novel approaches to creating neural network rules and extending association rules, both outperforming regular association rules.

Looking forwards, there are many different ways of building upon this thesis. First of all, the optimisation algorithms used are likely far from reaching optimal solutions. Thus, it would be interesting the solve the problem with other methods, for example using mixed integer linear programming. Furthermore, both the word scores of EAR and the network architecture of NNR have potential for further improvements. Other areas suitable for development include more extensive hyperparameter tuning and a more sophisticated evaluation method.

Bibliography

[1] U.S Census Bureau. Quarterly retail e-commerce sales 3rd quarter 2021, 2021. URL https://www.census.gov/retail/mrts/www/data/pdf/ec_current.pdf.

[2] H.L. Chan, King Wah Pang, and K.W. Li. Association rule based approach for improving operation efficiency in a randomized warehouse. In *Proceedings of the 2011 International Conference on Industrial Engineering and Operations Management Kuala Lumpur, Malaysia*. IEOM Research Solutions Pty Ltd., January 22–24, 2011. ISBN 9780980825107. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1069.6416&rep=rep1&type=pdf.

[3] Fillipe Goulart. Python TSP solver. https://github.com/fillipe-gsm/python-tsp, 2020.

[4] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *ACM SIGMOD Rec.*, 29(2), May 2000. ISSN 0163-5808. doi: 10.1145/335191.335372.

[5] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network, 2018. URL https://arxiv.org/abs/1412.6622.

[6] Karla L. Hoffman, Manfred Padberg, and Giovanni Rinaldi. Traveling salesman problem. In S. I. Gass and M. C. Fu, editors, *Encyclopedia of Operations Research and Management Science*, pages 1573–1578. Springer US, Boston, MA, 2013. ISBN 978-1-4419-1153-7. doi: 10.1007/978-1-4419-1153-7_1068.

[7] John H. Holland. Genetic algorithms and adaptation. In Oliver G. Selfridge, Edwina L. Rissland, and Michael A. Arbib, editors, *Adaptive Control of Ill-Defined Systems*, pages 317–333. Springer US, Boston, MA, 1984. ISBN 978-1-4684-8941-5. doi: 10.1007/978-1-4684-8941-5_21. URL https://doi.org/10.1007/978-1-4684-8941-5_21.

[8] Milan Jemelka, Bronislav Chramcov, Pavel Kříž, and Tomas Bata. ABC analyses with recursive method for warehouse. In *2017 4th International Conference on Control, Decision and Information Technologies (CoDIT)*, 2017. doi: 10.1109/CoDIT.2017.8102722.

[9] Brian Keith Norambuena and Claudio Meneses Villegas. Extended association rules in semantic vector spaces for sentiment classification. In Álvaro Rocha, Hojjat Adeli, Luís Paulo Reis, and Sandra Costanzo, editors,

Trends and Advances in Information Systems and Technologies, pages 649–658, Cham, 2018. Springer International Publishing. ISBN 978-3-319-77712-2. URL https://doi.org/10.1007/978-3-319-77712-2_60.

[10] Monika Kofler. *Optimising the storage location assignment problem under dynamic conditions*. PhD thesis, Fachhochschule Oberösterreich, May 2015. URL https://www.researchgate.net/publication/280157450_Optimising_the_storage_location_assignment_problem_under_dynamic_conditions.

[11] Mathias Kraus and Stefan Feuerriegel. Personalized purchase prediction of market baskets with Wasserstein-based sequence matching. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, July 2019. doi: 10.1145/3292500.3330791.

[12] Trupti A Kumbhare and Santosh V Chobe. An overview of association rule mining algorithms. *International Journal of Computer Science and Information Technologies*, 5(1):927–930, 2014. ISSN 0975-9646. URL http://www.ijcsit.com/docs/Volume%205/vol5issue01/ijcsit20140501201.pdf.

[13] Binh Nguyen and Atsuhiro Takasu. Learning representations from product titles for modeling shopping transactions, 2018. URL https://arxiv.org/abs/1811.01166.

[14] Wisam A. Qader, Musa M. Ameen, and Bilal I. Ahmed. An overview of bag of words; importance, implementation, applications, and challenges. In *2019 International Engineering Conference (IEC)*, pages 200–204, 2019. doi: 10.1109/IEC47844.2019.8950616.

[15] Sebastian Raschka. MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack. *The Journal of Open Source Software*, 3(24), 2018. doi: 10.21105/joss.00638.

[16] J. J. Rojas Reyes, E. L. Solano-Charris, and J. R. Montoya-Torres. The storage location assignment problem: A literature review. *International Journal of Industrial Engineering Computations*, 10:199–224, 2019. doi: 10.5267/j.ijiec.2018.8.001.

[17] Chathurangi Shyalika. Population initialization in genetic algorithms — an insight to genetic algorithms — part II, 2019. URL \https://medium.datadriveninvestor.com/population-initialization-in-genetic-algorithms-ddb037da6773.

[18] Amila Silva, Ling Luo, Shanika Karunasekera, and Christopher Leckie. OMBA: User-guided product representations for online market basket analysis, 2020. URL https://arxiv.org/abs/2006.10396.

[19] David Sourek and Vaclav Cempirek. Optimization of skus' locations in warehouse. *13th IMHRC Proceedings*, 30, 2014. URL https://digitalcommons.georgiasouthern.edu/pmhr_2014/30.

[20] Juan Luis Suárez-Díaz, Salvador García, and Francisco Herrera. A tutorial on distance metric learning: Mathematical foundations, algorithms, exper-

imental analysis, prospects and challenges (with appendices on mathematical background and detailed algorithms explanation), 2020. URL https://arxiv.org/abs/1812.05944.

[21] Domenico Talia, Paolo Trunfio, and Fabrizio Marozzo. Chapter 1 - Introduction to data mining. In *Data Analysis in the Cloud*, Computer Science Reviews and Trends. Elsevier, Boston, 2016. ISBN 978-0-12-802881-0.

[22] James A. Tompkins, John A. White, Yavuz A. Bozer, and J. M. A. Tanchoco. *Facilities planning, fourth edition.* John Wiley & Sons, New York, NY, USA, 2010. ISBN 978-0-470-44404-7.

[23] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network, 2015. URL https://arxiv.org/abs/1505.00853.

[24] Dongwen Zhang, Yaqi Si, Zhihong Tian, Lihua Yin, Jing Qiu, and Xiaojiang Du. A genetic-algorithm based method for storage location assignments in mobile rack warehouses. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2019. URL https://ieeexplore.ieee.org/document/9013447.

[25] Xiaonan Zhao, Huan Qi, Rui Luo, and Larry Davis. A weakly supervised adaptive triplet loss for deep metric learning, 2019. URL https://arxiv.org/abs/1909.12939.

[26] Li Zhou, Lili Sun, Zhaochan Li, Weipeng Li, Ning Cao, and Russell Higgs. Study on a storage location strategy based on clustering and association algorithms. *Soft Comput*, 24:5499–5516, 2020. doi: 10.1007/s00500-018-03702-9.

A

Hyperparameter tuning

Since the number of hyperparameters was quite large and some algorithms were time-consuming to run, a full grid search was not feasible. Thus, the tuning was divided into five phases, as can be seen in Figure A.1. For a given set of parameters to be tuned, a grid search was performed on those parameters, keeping all other parameters constant.

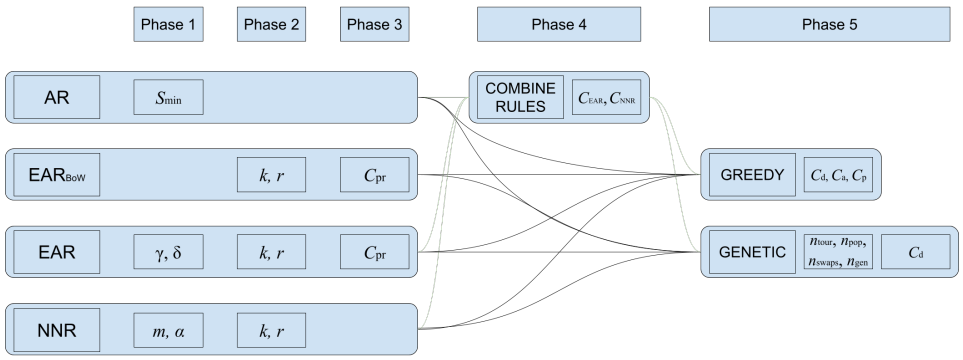


Figure A.1: Illustration of the hyperparameter tuning scheme. First, the parameters for creating the rules are tuned, followed by tuning the weights that combine the rules (these were not tuned using EAR_{BoW} as it was noticed early on that weighted EAR outperformed BoW). Finally the two optimisation algorithms are tuned.

To make the tuning feasible, it was only performed using a part of the training data consisting of 80000 orders. These orders were divided into four folds of 20000 orders each. Each set of parameters was then evaluated on each of these folds separately using 15000 orders for training and 5000 for evaluation, and the parameters with the best performance across all folds were chosen.

A.1 Rule mining

The first step was to tune the rule mining parameters, which was performed separately for each of the rule types. In order to evaluate a combination of parameters the corresponding rules were extracted. These rules were then fed to the greedy algorithm in order to find the article placements x, using the parameters $C_{\text{p}} = 0$, $C_{\text{r}} = 1$ and with the support set to 0 in equations (3.5) and (3.6). The reason for

excluding the article supports at this stage of tuning was to ensure that the greedy algorithm focuses solely on rules. Once x was found, a TSP was solved for each order in the current evaluation set. As opposed to the evaluation method described in Section 3.4, these TSPs did not include the depot in order to only evaluate the rules.

A.1.1 Association rules

For ARM, there were only two hyperparameter choices that had to be made; the minimum support threshold and the minimum lift threshold. Instead of tuning the minimum lift threshold, it was set to one, as all values greater than one indicate positive correlation between the articles in the rule.

Due to this property of the lift threshold it is desired to set $s_{\min} = 0$, since all rules with a lift over one was expected to have a positive impact on performance. However, lowering $s_{\min}$ came with a dramatic increase in run time. A compromise between the number of rules found and run time was therefore made.

A.1.2 Extending association rules

Due to the increase in hyperparameters and computational complexity compared to ARM, the tuning process was split into three steps. The first step concerned the tuning of the parameters related to the embedding space, γ and δ. They were tuned using $k = 5$, $\beta = 1$, $C_{\text{p}} = 0$ and ignoring the radius r.

Once chosen, these parameters were used to tune k and r, keeping $C_{\text{p}} = 0$. In order to reduce the number of hyperparameters to tune, β was set to $r/10$. This choice was made as the value of β depends on the distances between neighbouring articles, which is in turn controlled by r. The final step consisted of tuning C_{p} using the tuned values of the other parameters

A.1.3 Neural network rules

Just as for the EAR parameters, the NNR parameters were tuned in steps. First, the network parameters α and m were tuned using $k = 5$, $\beta = 1$ and an infinite radius r. Then, using the chosen values of α and m, k and r where tuned. Once again, $\beta = r/10$ was used.

A.2 Rule weights

After choosing all parameters for rule creation, the constants used to combine different rule types; C_{EAR} and C_{NNR}, were tuned. They were tuned by performing a grid search using the chosen rule creation parameters from above. The evaluation of each set of parameters was performed in the same way as described in Section A.1.

A.3 Optimisation

Using the parameters from above, the hyperparameters of both optimisation algorithms were tuned. Just as for the rule mining parameters, this was done by creating rules from a training set and evaluating the algorithms on a test set. However, the evaluation was done using the method from 3.4.

A.3.1 Greedy algorithm

The greedy algorithm was tuned once for each rule type, plus an additional time for the combination of all rules. This was done using a simple grid search on the three parameters $C_{\mathrm{r}}, C_{\mathrm{d}}$ and C_{p}.

A.3.2 Genetic algorithm

For the genetic algorithm, the weight in the fitness function, C_{d}, was tuned separately for each combination of rules as in Section A.3.1. However, the other parameters for the genetic algorithm were tuned only once as the optimal values of these parameters were expected to remain similar regardless of rule type.

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY