SAARLAND UNIVERSITY

Faculty of Natural Sciences and Technology I
Department of Computer Science
Bachelor's thesis

FrAPP - Framework for Automated Product Placement

Nico Herbig
Bachelor's Program in Computer Science
February 2014

Advisors
Gerrit Kahl, German Research Center for Artificial Intelligence,
Saarbrücken, Germany

Frederic Raber, German Research Center for Artificial Intelligence,
Saarbrücken, Germany

Reviewers
Prof. Dr. Antonio Krüger, German Research Center for Artificial Intelligence,
Saarbrücken, Germany

Prof. Dr. Peter Loos, German Research Center for Artificial Intelligence,
Saarbrücken, Germany

Submitted
26th of February, 2014

Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science
Campus - Building E1.1
66123 Saarbrücken
Germany

Statement in Lieu of an Oath:

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Saarbrücken, 26th of February, 2014

Declaration of Consent:

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, 26th of February, 2014

Acknowledgments

I sincerely thank Prof. Dr. Antonio Krüger and Prof. Dr. Peter Loos for reviewing my thesis. Also I would like to express my gratitude to my adviser Gerrit Kahl for his continuous support and feedback. Furthermore, I want to thank Frederic Raber for the useful remarks and comments. Also, I would like to thank GLOBUS SB-Warenhaus Holding in St. Wendel for providing the data which enabled me to perform realistic evaluations. I also want to express my gratitude to Dr. Michael Piotrowski who allowed me to use his LaTeX template. Lastly, I want to thank my family and friends who have supported me throughout the entire process.

Abstract

The main objective of this thesis is to find optimal solutions for the shelf space allocation problem in retailing: how much space should be assigned to the individual products and where should they be placed in order to maximize the category profit? Therefore, a mathematical optimization model was formalized and an efficient algorithm for solving this model was determined. The implemented system called **FrAPP**, **Fr**amework for **A**utomated **P**roduct **P**lacement, offers an interface for selecting boards and products and allows to specify further criteria such as minimum facing amounts and stacking limits. Furthermore, to fulfill merchandising specifications, users can manually place products and let FrAPP optimize the empty space around them. It is also possible to enter discounts achieved by purchasing a specific amount of products from the producer or wholesaler merchant, which will then be considered in the computation. The output solution is placed directly into a 3D model of a supermarket through which the user can navigate. FrAPP also offers the possibility to export a planogram of the found solution as a web page for easy accessibility.

For formalizing the optimization model a function capturing the demand in a given placement is needed. To make the model practical, the demand function aims to only use available data and at the same time reflect reality as closely as possible. It contains known parameters as the facing area, the space elasticity and location values, however, trends and the so called cross space location elasticity are also introduced. Here, the trends reflect the expected sales or losses due to factors outside of the store such as advertisement. The cross space location elasticity is an improvement to the well known cross space elasticity and combines the effects of space and location of one product on the demand of a substitute product. In order to express the fact that some products are more popular than others even if placed equally, the so called worst allocation demand is used. That is, the demand a product would achieve if it had one single facing on the lowest board. This worst allocation demand can be computed from the previous sales and placements and even incorporates regional differences in popularity.

The evaluation function of the optimization model is the overall profit achieved by the placement. All valid solutions must fulfill the following requirements: first, given upper and lower bounds on the facing amounts and stacking limits have to be satisfied. Secondly, decreases in demand due to disappearing facings as well as stock-outs are prohibited as they would result in financial penalties. Finally, all instances of a product must be connected to achieve a minimum level of sorting and therefore avoid irritating solutions.

As the search space is too large for simple brute force algorithms, FrAPP uses a simulated annealing based hyper-heuristic as an optimization algorithm. This hyper-heuristic approach uses a set of heuristics to generate neighbors instead of one single neighboring function and can therefore adapt to different problem

instances easily. The heuristics were created such that they fulfill the above requirements with high probability. To improve customer satisfaction, the found solutions are sorted by brand or type while keeping the quality of the placement intact. Apart from a single threaded version three different parallelization modes were implemented. An evaluation with data provided by a German retailer showed that all algorithms are able to find the optimal solution within a small amount of time. However, some algorithms outperform the others depending on the allowed computation time.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

This thesis addresses the shelf space allocation problem: how much space should a retailer assign to a particular product in a category and where should he place the product instances on the shelf in order to maximize the category profit?

1.1 Motivation

With today's huge product diversity it is impossible to assign each product on the market a sufficiently large amount of space. Therefore, the retailer has to select a set of products he wants to allocate to the shelves first and afterwards choose the amount of space and location within the shelf for each product. It is important to note that the retailer's and the manufacturer's goals differ: the retailer wants to maximize the overall category profit while in contrast the manufacturer's goal is to maximize the individual profits of his products (Murray et al. [1]). Hereby, products should have enough space to avoid out-of-stocks which result in lost sales but at the same time assigning too much space to a product might be less profitable than using the scarce space for another product. However, finding the right amount of space for each product is only part of the problem. The decision about the assigned space has to be performed jointly with the decision about where to place the product, as some space (e.g. eye level) is more valuable than other space. Drèze et al. [2] found that the location is even more important than the amount of space assigned to each product as long as the amount of space is sufficient to avoid out-of-stocks. One also has to consider that products have different sizes and therefore get noticed more or less easily. Furthermore, interrelations between products exist such that assigning more space to one product might reduce the demand of another product.

Since nowadays self-service in supermarkets is common, the challenge of convincing the customers to buy more profitable products has moved from the salesman towards the shelf itself (Anderson and Amato [3]). Therefore, the factors concerning the amount of space and the location of products mentioned above have become more important in the last decades, since obviously a product having assigned a lot of space on eye level gets more attention by the customers than a product with only little space on floor level.

Unfortunately, finding the best possible placement for all products manually consumes much time and only results in subjectively optimal solutions, since it is only possible to examine a hand full of different possible solutions. However, optimizing automatically is challenging as well because abstracting from reality while both keeping it realistic and using available data only is a difficult task. Furthermore, creating an efficient and precise algorithm is ambitious as the problem is NP-hard.

In the following I will present **FrAPP**, a **Fr**amework for **A**utomated **P**roduct **P**lacement, which finds close to optimal solutions within an acceptable amount of time and simulates demand in reality using available data only. Therefore, an optimization model is defined and solved using a simulated annealing based hyper-heuristic approach. Here, a set of low-level heuristics modifying shelf allocations in place is used by a high-level simulated annealing procedure. A simple user interface allows to specify further requirements for the solution. Additionally, stock-outs get avoided and solutions are sorted as much as possible to improve customer satisfaction. Last, the user can directly export the found solution and give the result to the shelf-stocking staff.

1.2 Terminology

In this section some frequently used terms and concepts of this thesis and the related literature are introduced. It is intended as a reference and should provide a better understanding of the context.

Facing

A "facing" describes a product instance located directly on a board being visible from the front. However, stacking in either height or depth is not considered a "facing". E.g. if no products are stacked, a facing amount of three simply means that three product instances are visible to the customer.

Planogram

According to the *Oxford Dictionary* a planogram is "a diagram or model that indicates the placement of retail products on shelves in order to maximize sales"

[4]. In practice planograms have two purposes: first, during space planning they enable the retailer to see the whole shelf in a clearly arranged way without having to actually place products in the real world. Second, it intuitively describes how the shelf is supposed to look and therefore planograms are used by companies to pass information about shelves from the planning employee towards the employee responsible for actually filling the shelf. Apart from a virtual image additional information like the EAN (European Article Number), the producer, the physical dimensions and the amount of stacking are included in tabular form.

Space Elasticity

The term space elasticity or own space elasticity can be defined as "the ratio of relative change in unit sales to relative change in shelf space" (Curhan [5]) or as "the sensitivity of the customer to the inventory displayed in terms of the quantity bought" (Hariga et al. [6]). It describes how much the unit sales of a product get affected by a change in the space allocated to the product. The easiest way to explain this phenomena is that a customer is more likely to notice a product if it has more space assigned. Another explanation for the phenomena of space elasticity "is that when some consumers see one product having more displayed inventory on the shelf than the competing products, they think that this product must be more popular and should provide less risk of disappointment, so they decide to buy this product" (Reyes and Frazier [7]). Here, different products might be more or less elastic than others. For example fruit and vegetables have a high value, while textile has a low value (Desmet and Renaudin [8]).

Mostly, when using space elasticity to describe the demand of a product mathematically it is used in an exponential form: $space^{space\_elasticity}$. Since the different space elasticity values are always between 0 and 1, this exponential form ensures diminishing returns: placing a product twice instead of once leads to a bigger increase in demand than placing it 15 instead of 14 times, since a product with 14 facings should be noticed by the customer anyway. Generally, the more space is already assigned to a product, the more likely a customer notices this specific product and therefore the less a further increase in assigned space affects the demand.

Cross Space Elasticity

While an increase in space allocated to a product affects its demand positively, it might also affect the demand of other products. Therefore, cross space elasticity describes the influence of the space assigned to one product on the unit sales of another product. This influence can happen in different ways: it can influence a product positively, negatively or not at all. Complementary products affect each others sales positively, e.g. if cereals get bought more often because they get more space assigned, so will milk. If products are substitutes they affect each other negatively, for example if milk of brand A gets more space and therefore its

demand increases the demand of the milk of brand B will decrease. The cross space elasticity value equals 0 if the products are not related at all, as probably with toilet paper and pizza.

It should be noted that the cross space elasticity is not always symmetric: as Coskun [9] points out "an increase in camera sales may affect the battery sales positively but not vice versa".

In most related literature (e.g. Hwang et al. [10]) significantly smaller values are used for cross space elasticities than for own space elasticities as they seem to have less impact on sales. Nevertheless, they express an intuitively understandable aspect of customers' purchase behavior and are therefore not negligible.

Shelf vs. Board

Since "shelf" in the English language describes both a collection of levels on which products are allocated as well as a single unit within such a collection, I will avoid confusion by using the term "shelf" only to describe the whole collection and "board" to describe a single level in the following.

Chapter 2
Related Work

This chapter will discuss the different approaches used in the related literature on the product allocation problem. It is broadly divided into two sub-problems: first, finding a mathematical optimization model that describes reality formally, and second, creating an algorithm that solves this optimization model with close to optimal solutions within a reasonable amount of time.

2.1 Optimization Models

2.1.1 General Approach

Almost all papers on the shelf space allocation problem have a similar overall approach:

First, a function called *demand function* is developed, which receives as input all data about the placement of a product (e.g. amount of facings, location of placement) and computes the demand for that product in the given placement. Basically the demand function evaluates how often a product would be sold if it was placed in the given way. Naturally, the related literature differs strongly in the concrete design of such a demand function.

Second, an optimization model is designed which uses this demand function. Therefore, an evaluation function is developed which in most cases is the maximization of the category gross or net profit, or of the category return on investment. For example the objective function for the maximization of the category gross profit would be $max \sum_{i \in I}(sp_i - pc_i)D_i$ where I is the set of all products in the category, sp_i and pc_i are the selling price and purchasing cost of product i, and D_i is the demand of product i. Furthermore, a set of constraints is defined

which has to be satisfied by all valid solutions. An example constraint would be the physical limitations of a board which may not be exceeded by the products.

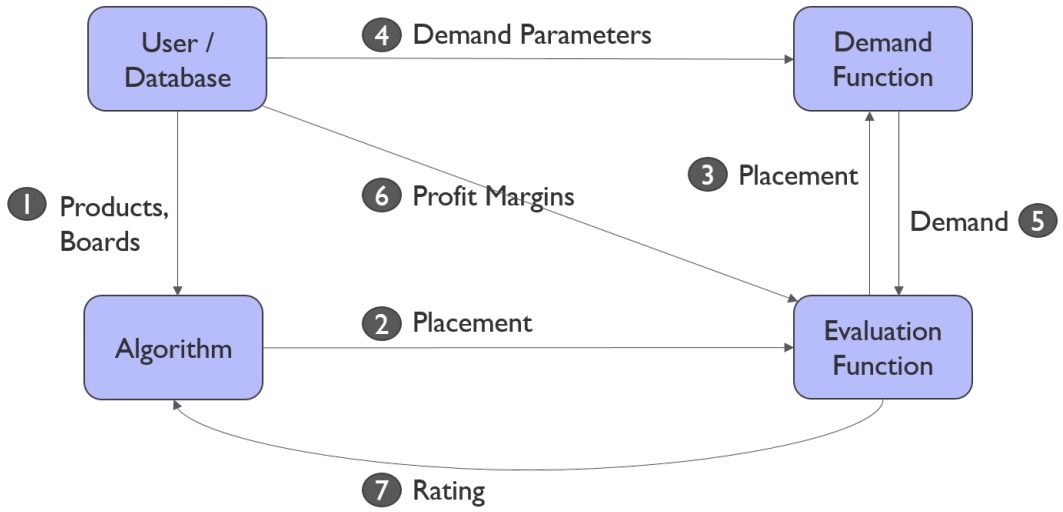

Figure 2.1: The communication between database, solving procedure, evaluation function and demand function.

Figure 2.1 gives an overview of how solutions are created and rated. First, the user selects the products and boards specifying the problem instance. An algorithm is invoked upon these parameters and generates a placement. For estimating the quality of the solution the placement is given to the evaluation function. In order to rate an allocation the demand and profit margin of each product are needed. For retrieving the demand the demand function is queried which gets the placement as a parameter and queries all needed parameters from the database or user input. Then the profit margins for all products are retrieved from the database and a rating is computed and returned to the algorithm. The algorithm then creates the next placement and queries an evaluation value in the above manner.

2.1.2 A Simple Demand Model Using Space Only

The authors of the paper "An Investigation of Automated Planograms Using a Simulated Annealing Based Hyper-Heuristic" [11], Bai and Kendall, probably have one of the easiest optimization models. As a measure of space assigned to a product they simply use the amount of facings. Furthermore Bai and Kendall incorporate the space elasticity as an exponent of the amount of facings. Finally an unspecified scale parameter is used. Taken together the demand function is $D_i = \alpha_i x_i^{\beta_i}$ where x_i is the amount of facings, $\alpha_i > 0$ is the scale parameter and $0 < \beta_i < 1$ is the space elasticity for product i.

Since the space elasticity parameter is limited to $]0, 1[$ this demand function has the property of diminishing returns which the authors describe as the decreasing increase in demand while the allocated space of the product increases. This property is close to reality for two reasons: first, the percentage gain in space by adding a further facing decreases and second, the more space a product already has, the more customers notice it and therefore the less extra demand a further increase will gain.

The objective function then is $MaxP = \sum_{i=1}^{N}(p_i D_i)$, where N denotes the amount of products, p_i is the unit profit of product i and D_i is the demand of product i in the current placement as defined above. So basically the profit achieved by a product is simply the profit of one unit times the demand for the product (which is the expected sales amount for that product), and the overall profit is the sum over all product profits.

As constraints they used the physical limitation in width, i.e. is the sum of product widths may not exceed the board width, that the facing amount is always an integer and that given lower and upper bounds on the amount of facings per product are satisfied. As Murray et al. [1] point out lower bounds can be used to express and fulfill contracts with manufacturers. Often they pay retailers for more shelf space as it will increase their profit. Furthermore, according to Yang [12] upper bounds give the retailer the possibility to phase out products.

It should be noted that Bai and Kendall ignore the depth and height of boards and products completely. Furthermore, they make the assumption that "retailers prevent out-of-stock occurrences" and they do not enforce each chosen product to be displayed which means their model performs product selection.

2.1.3 Considering Location and Inventory Level Effects of Demand

The demand function proposed by Hwang et al. [10] is much more complex than the one previously presented: it includes a scale parameter, the amount of facings and the space elasticity in a similar way as the demand function of Bai and Kendall [11]. However, they also incorporate cross space elasticity and location effects. This is an important improvement since Drèze et al. [2] found in an experimental field study that "location had a large impact on sales, whereas changes in the number of facings allocated to a brand had much less impact as long as a minimum threshold (to avoid out-of-stocks) was maintained". For the location effect the authors assigned each board a location value: the board on the eye level gets the largest value while the worst board, namely the bottom or top board gets the lowest value of 1. For products being displayed on multiple levels they computed the location value as the weighted average over all location values. E.g. a product being displayed twice on a board with location value 2 and once on a board with value 3 gets the location value $\frac{2*2+1*3}{3} = \frac{7}{3}$. Overall the demand function is defined as

$$D_i = \alpha_i x_i^{\beta_i} \prod_{k \neq i}^{N} x_k^{\beta_{ik}} loc_i \tag{2.1}$$

where N is the amount of products, x_i is the amount of facings, $\alpha_i > 0$ is the scale parameter, $0 < \beta_i < 1$ is the space elasticity for product i and β_{ik} is the cross space elasticity between products i and k. loc_i describes the location parameter and is defined by

$$loc_i = \frac{\sum_{j=1}^{M} x_{ij} Loc(j)}{x_i} \tag{2.2}$$

where M is the amount of boards, x_{ij} is the amount of facings of product i on board j and $Loc(j)$ is the given location value of board j.

As profit the authors use the gross margin subtracted by the holding costs for the facings and the backroom, the display expense and the ordering costs. The objective function then is the maximized overall profit defined as above under the constraints that the board width is not exceeded, the amount of facings for each product is within the given minimum and maximum and that the amount of facings and order quantity are non-negative. However, the number of facings is not required to be an integer, which is not feasible in practice.

By purchasing a product costumers remove facings, which in turn decreases demand, as it depends on the amount of facings according to the demand function. Therefore, Hwang et al. make the reasonable assumption that an employee reorders the products in a way that the shelf still looks fully stocked and the missing product is in the back instead, which keeps the demand constant as long as enough products for the front row are left. The authors call this policy "full-shelf merchandising". Further assumptions of the authors are that every product has to be placed which means that no product selection is performed and that there are "no joint replenishments", instead each product gets replenished individually.

Apart from the amount of facings and the locations of the facings being decision variables they also determine the order quantity. Generally, they put a lot of effort into investigating the ordering process. However, since the authors' assumption that the order is delivered instantaneous when the whole backroom is depleted, which in my eyes is a false assumption, I will not go into much detail here.

2.1.4 Joint Optimization of Price, Orientation and Shelf-Space Allocation

Murray et al. [1] use one of the most complex demand functions. As a basis they use an unspecified scaling parameter is used. Instead of using facings as a measure of space the authors use the facing area instead, which is more realistic since naturally a bigger product gets noticed more often by customers than a small one. Furthermore, for aesthetic reasons and better fitting their model allows products to be placed in different orientations, that is all sides can be used as the front. Also, the location and space elasticity are considered. A further interesting aspect of their model is that prices are not assumed fixed, but instead get computed jointly with the location, orientation and shelf-space allocation.

Therefore, own price elasticity and cross price elasticity, which are the price equivalent to own space and cross space elasticity, are considered as well.

Since the orientation is kept flexible, they use the respective measures of the current orientation for the computation of the facing area and a so called "shelf-location-orientation quality-adjustment weight". This is the analogon to the location value in Hwang et al.'s model [10] with the extension that the orientation is considered as well.

Murray et al. "assume that good logistics are sufficient to eliminate [its] out-of-stock occurrences". Furthermore they do not select the product assortment but assume that it has already been determined. Since the authors consider width and height in their space measurement, stacking is performed. Here, the authors always stack as many products as possible in the same orientation as the facing on top of it. However, this is unrealistic since many products cannot be stacked or are only stacked within limited bounds for aesthetic reasons. In order to allow products of non-rectangular shape to be stacked and oriented, Murray et al. use the "least-area rectangular contour that can fit the non-rectangular contour" instead.

The objective function maximizes the gross profit of the whole category. Valid solutions must fulfill constraints as the physical limitations in width, depth and height for all products in the orientation of the placement. The amount of facings and the price must satisfy given lower and upper bounds and the number of facings must be an integer.

2.1.5 Further Ideas

Location vs. Space

For the paper "Shelf Management and Space Elasticity" [2], Dréze et al. performed a long time field study to evaluate the effects on demand caused by shelf space or location in reality. As a result of the high costs of field experiments in this topic there are only very few studies like Drèze et al.'s and therefore it is perhaps the most cited paper of the shelf space allocation topic.

The authors found that location has a much bigger impact on demand than the amount of space allocated to a product. However, even though the horizontal position was statistically important, the best horizontal location differed between categories: some categories preferred the edges and others the center of the boards. On the vertical axis positions in or slightly below eye level turned out to be the most valuable. The vertical position seems to be the way more important one as here an increase in demand between the worst and best position of 39% could be achieved whereas the horizontal position only lead to a difference of 15% according to the authors.

Base Level Allocation

Reyes and Frazier's model [7] differs from most demand models in some key aspects:

As a basis for their demand function they use a not further specified parameter called "base level demand". That is, the space the product would achieve if all space was assigned in a demand-dependent way. The authors then compute the space elasticity by dividing the space allocated to a product by the base level allocation. This quotient then gets modified by a product independent exponent. The price sensitivity is computed in a similar way: the average price gets divided by the price of the product and the ratio again gets altered by a product independent value. So one can clearly see that the authors use one space elasticity value and one price elasticity value for all products instead of product-dependent values. Instead, the product dependent effects of the sensitivity factors are computed using the deviation towards the corresponding average value.

Another interesting aspect is their objective function which has two goals: maximizing profit and minimizing the deviation in space allocated compared to the base level allocation. The minimization is performed because the authors argue that the base level allocation provides the most customer service since it avoids out-of-stocks as much as possible.

Staple Products

Cox [13] made an experimental study from which he concluded that staple products like salt and pepper are independent of the shelf space assigned to them. Therefore, retailers should only assign them as much space as needed to avoid stockouts.

Block Allocation

Zufryden [14] is one of the only researchers who tried to allocate multiple facings of products as blocks of products for aesthetic reasons. To achieve this goal and to be able to perform dynamic programming, which will be explained the next section, he divided the shelf into a set of slots. Therefore, the author assumes that the physical dimensions of each product are integer multiples of the dimensions of the slot.

Maintaining the Grouping of Product Families

The authors Russel and Urban [15] put a lot of consideration into "maintaining the grouping of product families". For example one could require a sorting by brand or variety. Therefore, Russel and Urban's approach keeps "product families in uniform and complete columns". The authors allow categories to "span several

shelves as long as they maintain a rectangular physical presence, with small deviations allowed". Furthermore, they consider both horizontal and vertical effects on demand.

Random and Preferenced Demand

Anderson and Amato [3] divide the demand into different categories: random demand and preference demand. Here, random demand arises from customers without any preference who appear to choose their products randomly. On the other hand, preference demand emerges from customers preferring a specific set of products. According to the authors these customers can again be split in two groups, namely those being loyal towards their preference and would never buy an alternative product and those who are not as loyal and would switch under appropriate circumstances.

Unmodified, Modified, Acquired and Stockout Demand

Borin et al. [16] separated demand into four categories: unmodified, modified, acquired and stockout demand. Unmodified demand is kind of a base demand that a product would achieve if all products "received identical retail support". Modified demand describes the effects of space, advertisement, etc. on the sales of a product. Furthermore, acquired demand describes the demand a product receives because other products are not in the assortment and therefore customers switch to this product. Finally, stockout demand is the demand products "receive from the items which have temporarily stocked out".

Reordering

Urban [17] investigated the reordering process. He was the first researcher who considered the problem that facings deplete as products are sold and therefore the demand decreases. In his study he considers both backroom and showroom and assumed that the showroom is kept fully stocked as long as it can be refilled from the backroom. Since he wanted to optimize the net profit, part of his study is to find the perfect reordering point as to reduce ordering costs while not loosing too much demand.

Flexible Shelf Height

Apart from offering an excellent overview of the related literature on the shelf space allocation problem, Coskun's [9] optimization model allows flexible board heights "to maximize the shelf space utilization". In his evaluation it turned out that boards with high location values tend to become bigger due to the adjustments of board heights.

Freshness

Bai and Kendall [18] argue that for fresh products the demand is not constant over time, but instead decreases as products loose their freshness. Therefore, they develop a demand function considering the amount of facings and the freshness condition of the product.

Showroom and Backroom Inventory

Hariga et al.'s [6] approach is interesting as they make a clear distinction between showroom and backroom inventory and investigate products being displayed in multiple locations within the store. Furthermore, their model performs product selection and inventory replenishments. They maximize the net profit which incorporates investment, storage and display cost.

Advertisement and Distribution

Urban's approach [19] is interesting as he includes advertisement and distribution in his demand function. He even incorporates own and cross distribution and advertisement elasticities. Although these parameters undeniably represent aspects of reality, it seems unfeasible to access such data.

2.2 Solving Procedures

This section describes the different approaches used to solve the above optimization models. It is important to note that due to the NP-hardness of the shelf space allocation problem it is not possible to brute force solutions for medium to big-sized problems. Therefore, the following algorithms try to find a close to optimal solution within a reasonable amount of time without guaranteeing optimality.

2.2.1 MINLP-Solver

After determining their optimization model as described in 2.1.4, Murray et al. [1] noted that the objective function is nonlinear and that there exist both integer and continuous variables (the amount of facings is constrained to be an integer, while e.g. the prices and physical units are continuous variables). Therefore, it can be classified as a mixed integer nonlinear programming problem (MINLP). Furthermore, the objective function as the authors defined it is non-convex, which leads to a big problem: according to Bussieck and Pruessner [20] no method is known which guarantees to solve non-convex MINLPs to optimality.

However, even though optimality is not guaranteed solvers for non-convex MINLPs exist. The authors used the "Basic Open Source Nonlinear Mixed IN-

teger programming" (BONMIN) C++ open source code which is available from the COIN-OR (COmputational INfrastructure for Operations Research) website (`https://projects.coin-or.org/Bonmin`). Due to the non-convexity of the problem the authors argued that the diverse BONMIN algorithms serve as a heuristic only. However, computational features are included which "significantly improve the quality of solutions even for non-convex problems" [1].

BONMIN offers a set of algorithms for minimizing MINLPs:

"B-BB" is a **branch and bound** algorithm which, as the name suggests, iteratively branches the problem in sub-problems and achieves new bounds. Specifically, it solves the continuous relaxations of the problem, which itself is only a non-linear programming problem (NLP) and therefore easier to solve. Whenever a solution to the NLP is also a solution for the original MINLP (this means the integer variables are indeed integers), the solution is compared to the current best solution and if it is smaller it becomes the new lower bound. On the contrary, if the solution of the NLP is not a solution of the MINLP, there are two further cases: first, if the solution has a lower value than the current lower bound, it cannot be the optimal solution since a solution of the NLP is a upper bound on the MINLP, and therefore the node can be skipped. Second, if the solution is not bigger than the current upper bound the search-node gets split into two sub-nodes, one which requires that the integer variables are at most as high as the rounded down value of the corresponding continuous variable and one which requires the integer variable to be at least as big as the rounded up integer variable. At these new nodes the corresponding NLPs are solved and the process starts anew. Here, each sub-node has smaller allowed intervals than his parent. For more details on the implemented branch and bound technique the reader is referred to Bonami et al. (2011) [21].

"B-OA" is an **outer approximation** algorithm, which solves the problem by alternatively solving two easier sub-problems: if the integer variables of the MINLP get fixed, it reduces to a nonlinear programming problem (NLP) since there are only continuous variables. Solving this NLP results in an upper bound x, since it is a feasible solution for the MINLP as well. Furthermore, linearizing all functions of the original MINLP around the solution x results in a mixed integer linear programming problem (MILP) since the functions are linear. Solutions of this easier MILP can be seen as a lower bound for convex functions. Iterating these steps until the lower and upper bounds coincide results in optimized solutions for the original MINLP. For more information on the implemented algorithm in BONMIN the reader is referred to Bonami et al. (2008) [22].

Apart from the above two BONMIN offers many other algorithms, e.g. the "B-Hyb" which is a hybrid of and outer-approximation and a branch and-cut-algorithm, however I will not explain every algorithm in detail here.

Even though the creators of BONMIN "strongly recommend using B-BB (the outer-approximation algorithms have not been tailored to treat nonconvex problems at this point)" in their users' manual [23], Murray et al. [1] used not only

the "B-BB" algorithm, but also two of the outer approximation based algorithms "B-OA" and "B-Hyb". However, in their numerical experiments both outer approximation based algorithms outperformed the branch and bound algorithm since they were way more effective.

Apart from Murray et al. many other researchers on the space allocation problem used available solvers to evaluate their models. For example Coskun [9] used an optimization modeling language called GAMS and experimented with three solvers, the above mentioned BONMIN, BARON and KNITRO; Hariga et al. [6] solved their MINLP using the LINGO software and Russel and Urban [15] modeled their discrete model in the ILOG OPL modeling language and solved using the CPLEX 10.1 solver.

2.2.2 Dynamic Programming

As mentioned in 2.1.5, Zufryden [14] divides the shelves into equally sized "slots" in order to allocate space in a dynamic programming approach. Therefore, he assumes that the width of each product is an integer multiple of the width of a slot and analogously that the height of a product is an integer multiple of the height of a slot. In his approach the placement of each product item is considered as a "stage". For the dynamic programming the recursive relation used states that the best way to allocate a particular product is the maximum possible sum of the profit received by allocating x slots to the product and the best way to allocate the remaining slots by all products before the current one (recursion). To explain this recursive relation better I will pseudo-formalize it in a simple way by not introducing too many variables and abstracting from unneeded detail. However, the reader is referred to the original paper for a more precise and detailed formalization.

All products are numbered: $j = 1, 2, ..., N$, where N is the amount of products that should be placed. The recursive relation used in the author's dynamic programming approach is:
$BestAllocation_j(slotsLeft) =$
$Max_x[profitByAllocatingXSlotsToProdJ + BestAllocation_{j-1}(slotsLeft - x)]$,
where $slotsLeft$ is the amount of slots being empty. Naturally, the base case is $BaseAllocation_0(y) = 0 \quad \forall y$, which means that allocating the nonexistent product 0 results in no profit, no matter how many slots are left. In the above equation the maximization is only performed over valid solutions, meaning that constraint violating allocations do not count.

This recursive relation can now be used by starting with the last product N and the amount of overall slots available as $slotsLeft$. Naturally, the intermediate results have to be memorized to achieve dynamic programming and therefore reduce computation time dramatically.

Even though Zufryden's approach works for his specific problem it has serious limitations: first, the assumption that all products must be sized as an integer

multiple of the slot size is only applicable if one defines slots as 1mm x 1mm, since in reality products are not integer multiples in size of each other. Making the slots too tiny however results in enormous computation time which destroys the practicability of the approach. Second, the approach in the current form does not work for more complex demand functions. For example this recursive relation cannot handle locations and I doubt that any recursive relation for simultaneously considering locations and space can be found.

2.2.3 Genetic Algorithm

Hwang et al. [10] proposed a genetic algorithm to solve their model as defined in 2.1.3 by simulating the evolution process. Here, the first step is to define an encoding for a fully filled shelf, which defines a chromosome. Then the authors start with the initialization of the algorithm: an initial population of chromosomes is generated randomly as can be seen in figure 2.2.

Figure 2.2: An exemplary initial population for the genetic algorithm.[1]

The next step is called crossover: here, two chromosomes (which are the encoding of shelves) have to be combined to one new offspring shelf. Therefore, the authors used the "one-cut-point" technique which simply uses a random cut-point at which the two chromosomes are split and then combines the left half of one chromosome with the right part of the other. This basically relates to taking a point in the shelf and merging the products in the top-left part of that point from one parent shelf and the products from the bottom-right part of the cut-point from the other parent shelf together. In their algorithm "25% of chromosomes undergo crossover". Figure 2.3 shows a simple example where the left and right halves of the shelves are merged.

[1]Here and in the following the image of the empty shelf is taken from `http://www.photosinbox.com/download/empty-wooden-shelf.jpg`

Figure 2.3: Crossover in the genetic algorithm by concatenating left and right halves of the parent shelves.

Apart from crossover also mutation is used to produce offspring. Here, genes are simply changed within some chromosomes. This can be illustrated as e.g. changing a facing in the shelf. For an example see the facings at the red arrows in figure 2.4 in comparison to figure 2.3. In the authors' algorithm 1% of genes get altered. Generally, mutation is used to gain more diversity.

Figure 2.4: Mutation in the genetic algorithm by changing random facings.

Naturally, the parent generation dies as evolution is simulated. This can be seen in figure 2.5.

Figure 2.5: Parent generation dies in the genetic algorithm.

One problem Hwang et al. had to face is that their model defines a constrained optimization. However, since the generation of the initial population as well as the production of offspring by crossover or mutation is using a lot of randomness, it can easily happen that chromosomes violate these constraints. Now one could remove all such constraint-violating chromosomes from the population, however, the authors decided to use a "penalizing strategy" instead. That is, depending on the degree of constraint violation the chromosomes get penalized more or less in the following fitness test. If we assume that the maximum facing amount of the red product in the example is three, the right offspring shelf in figure 2.6 would receive a penalty.

Figure 2.6: Constraint violating solutions get a penalty in the genetic algorithm. Assume for example the maximum facing amount of the red product is three.

To perform evolution, survival of the fittest has to be simulated. Therefore, a fitness function is defined which simply is the total profit achieved by the shelf. As mentioned above, constraint violating chromosomes get a penalty here, meaning that the chromosome is treated as if its corresponding shelf would result in less profit.

The new generation is build by choosing offsprings with a probability which is proportional to the fitness value (namely the profit). It should be noted that in order to "diversify the population" the fitness values have to be scaled: the authors determined the scaled fitness value by dividing the normal fitness value by the sum of the fitness values of all chromosomes (and including a predetermined scaling factor). An exemplary survival of the fittest process can be seen in figure 2.7.

Figure 2.7: Survival of the fittest in the genetic algorithm: bad solutions are removed with high probability.

Now this whole procedure is repeated with the new generation instead of the initial population. The algorithm terminates after some stopping criteria (e.g. amount of iterations) and returns the best encountered solution.

2.2.4 Simulated Annealing Based Hyper-Heuristic

Bai and Kendall's approach [11] to optimize their model as defined in 2.1.2 is called "simulated annealing based hyper-heuristic". Here, I will first explain the advantages of hyper-heuristics and afterwards come to the algorithm itself.

According to the authors meta heuristics can be applied on many problems and perform well on specific problems. However, "once the problem changes (even slightly), the performance of the already developed meta-heuristic may decrease dramatically for the new problem". In order to adapt to the new problem parameter tuning is needed. Furthermore, according to the "No Free Lunch

Theorem" there cannot be an algorithm outperforming all others in all problems [24]. If an algorithm outperforms all others on a specific problem, it will be beaten on another problem. Therefore, Bai and Kendall argue that "a good way to raise the generality of meta-heuristics is to apply different (meta-)heuristics at different times of the search", which is called "hyper-heuristics" and "broadly describes the process of using (meta-)heuristics to choose (meta-)heuristics to solve the problem in hand" [25]. To put it in a nutshell, as Bai and Kendall explain it the hyper-heuristic approach starts from an initial soluiton and uses a so called high level heuristic to guide the search and select amongst a set of low level heuristics which then modify the solution. Here, the low level heuristics have to be designed for the specific problem, while the high level heuristic can be used for different problems as it only "searches over the heuristic space" [11]. Therefore, exchanging the low level heuristics is sufficient to reuse the approach for another problem.

As a high level heuristic Bai and Kendall use a simulated annealing algorithm. Simulated annealing is a simulation of a cooling process in physics. The algorithm starts with an initial but probably bad solution which will be optimized. Therefore, neighbors of the solution get generated repeatedly and compared to the current solution. If the neighbors objective value is higher than the objective value of the current solution the algorithm transfers to it. However, to avoid getting stuck in local optima it is possible to transfer to worse solutions as well. Therefore, a parameter called temperature is introduced: since the algorithm is called simulated annealing it has an initially high value and gets reduced according to a temperature function in each iteration. Now this temperature is used if the objective value of the neighbor is worse than the value of the current solution. Here, Bai and Kendall used the Metropolis probability defined as $exp(-\delta/t)$ where δ is the difference in objective value between the two solutions and t is the temperature. This means that weather or not the algorithm transfers to a worse solution depends on two factors: how much worse is the solution and how low is the temperature. Since the temperature gets decreased over and over it is more likely to deteriorate in the beginning of the search. The cooler the temperature gets the less likely it is to worsen. Furthermore, it is always more likely to transfer to a worse solution if its difference in objective value towards the current optimum is small.

Simulated annealing based hyper-heuristic now uses the above simulated annealing as a high level heuristic. However, instead of generating a neighbor in the same way in each iteration simulated annealing based hyper-heuristic chooses a heuristic out of a set of low level heuristics which all create neighbors in a different way. This allows the algorithm to adapt to the specific problem. Overall Bai and Kendall implemented 12 different low level heuristics such as adding a random facing, deleting a random facing, the deletion of one facing of a product and filling the gap with as many facings as fit of another product, etc. The pseudo code of the simulated annealing based hyper-heuristic can be seen in algorithm 1.

Algorithm 1 Simulated Annealing Based Hyper-Heuristic [11]

```
Select an initial solution s_0;
repeat
  Randomly select a heuristic h ∈ H;
  iteration_count = 0;
  repeat
    iteration_count++;
    Applying h to s_0, get a new solution s_1;
    δ = f(s_1) − f(s_0);
    if (δ ≥ 0) then
      s_0 = s_1;
    else
      Generate a random x uniformly in the range (0, 1);
      if x < exp(δ/t) then s_0 = s_1;
    end if
  until iteration_count = nrep;
  set t = temperatureFunction(t);
until the stopping criteria is satisfied
```

Here, $f(s)$ is the objective function, which in the case of the shelf space allocation problem mostly is the overall achieved profit. Furthermore, $nrep$ simply is a predefined number for the amount of iterations the algorithm should perform before updating the temperature.

As a temperature function the authors used Lundy and Mees's [26] cooling schedule which is defined as $t \rightarrow t/(1 + \beta t)$. Bai and Kendall defined $\beta = \frac{(t_s - t_f)T_{average}}{T_{allowed}t_s t_f}$ where t_s and t_f are the starting and final temperature and $T_{average}$ and $T_{allowed}$ are the average time needed for one iteration and the time allowed for the whole search.

Bai and Kendall experimented with two different ways of determining the initial temperature. The first was to set it to 0.3 times the objective value of the initial solution while the other was a lot more complicated: from the initial solution neighbors were generated and the maximum difference δ_{max} in objective value between the initial solution and the neighbors was computed. Then in order to accept 85% of bad moves in the beginning the starting temperature was set to $-\delta_{max}/ln(0.85)$. However, as their evaluation showed, the algorithm is insensitive towards the initial temperature and therefore it did not matter which approach was used to determine t_s.

Simulated annealing algorithms normally terminate when the temperature becomes too low. Here, the authors used 0.1 as the final temperature.

2.3 Comparison with FrAPP

Table 2.1 shows the factors included in the three most related papers in comparison to FrAPP's demand function.

Model	Basis	Space Measurement	Price	Space Elast.
Bai and Kendall [11]	Scale Parameter	Amount of Facings	No	Yes
Hwang et al. [10]	Scale Parameter	Amount of Facings	No	Yes
Murray et al. [1]	Scale Parameter	Facing Area	Yes	Yes
FrAPP	Worst Allocation Demand	Facing Area	No	Yes

Model	Cross Effects	Vert. Pos.	Orientation	Trends	Stacking
Bai and Kendall [11]	No	No	No	No	No
Hwang et al. [10]	Cross Space El.	Yes	No	No	Depth
Murray et al. [1]	No	Yes	Yes	No	Height
FrAPP	Cr. Sp. Loc. El.	Yes	No	Yes	Both

Table 2.1: Tables comparing the demand factors of the three most related papers and FrAPP.

A basis is needed to express the fact that some products are sold more often than others even if they are placed badly. Therefore, all related works I presented here included a scale parameter. However, it was not clearly defined where this scale factor comes from and if it is a fair measurement. Therefore, FrAPP uses the so called worst allocation demand which will be described in the next chapter.

As the amount of facings ignores the fact that bigger products are more likely to be seen than smaller products, Murray et al. [1] used the facing area as a measurement of space. This is a clear improvement compared to the other two presented approaches which is why FrAPP also uses the facing area.

Murray et al. [1] also had the price as a decision variable which of course offered more flexibility. However, in chain stores the price should not differ from store to store but the allocation might be different due to the physical constraints of the different buildings. Therefore, it seems unrealistic to determine the price depending on the product allocation.

All three papers included the space elasticity, while only Hwang et al. [10] used the cross space elasticity. Even though cross space effects might not be as important as other factors, like the vertical position, they reflect an intuitively understandable aspect of reality. Therefore, FrAPP uses an extension of the cross space elasticity which also includes interrelations of products due to better positioning and is therefore superior to the simple cross space elasticity.

Dréze et al. [2] showed that the vertical position is more important than the amount of space assigned to a product. However, Bai and Kendall's model [11] does not include location effects which is the biggest disadvantage of their model compared to the other two.

Murray et al. [1] used the orientation of products as a decision variable. However, since products are designed with a clearly defined front, FrAPP and the other two presented approaches do not include different orientations in their respective models.

In FrAPP users can also enter expected trends due to out-of-store effects into the model. This will be described in more detail in the next chapter.

While Bai and Kendall [11] did not perform any stacking, Hwang et al. [10] stacked products in depth as a storage to refill from when facings get sold and to be able to compute the perfect reordering point. Murray et al. [1] always stacked as many products as fit on top of each other to further increase the facing area. Even though this is not done for all product categories in reality for aesthetic reasons, it is a good approach to increase the demand. FrAPP performs both, stacking in depth and height, to compute the facing area and avoid stock-outs.

Detailed explanations leading to this choice of demand factors and their formalization can be found in the next chapter.

Chapter 3
Mathematical Model

In this chapter a mathematical model is developed, with the purpose to capture the sales of a shelf as realistic as possible. However, at the same time the model should only use parameters or data which can be achieved or at least approximated.

3.1 Demand Function

The first step in computing the overall profit of a shelf is to be able to capture demand. That is, given a placement, compute how much each product would be sold if the market would use the placement in reality. Figure 3.1 gives an overview over the factors influencing demand: the position a product is placed in as well as the amount of space it covers are the key aspects of demand as they determine how easily a customer notices the product. Furthermore, elasticities such as the space elasticity are used to describe intuitive aspects of reality. Last, a scale parameter is needed to express the fact that some products are more popular than others even if placed identically.

3.1.1 Demand Parameters

In this section FrAPP's concrete specifications of the factors of figure 3.1 will be explained and motivated. Furthermore, it is explained which of these parameters are adopted from related works and where a different approach is used.

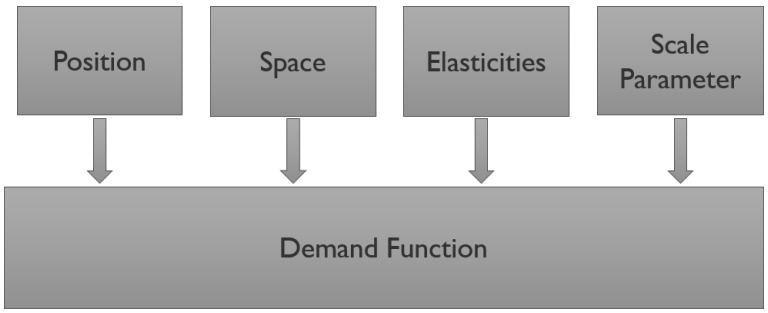

Figure 3.1: Overview on the factors influencing demand.

Position

Dréze et al. [2] pointed out that the vertical position of a product is even more important than the space allocated to a product, therefore, I will use this factor in my demand function. Like the authors I assign location values to the boards depending on their distance to eye level: the lowest board gets a value of one, then the values are increasing until eye level is reached and in the following location values are decreasing again. If a product is allocated on multiple boards the weighted average over all boards it is placed on is used (analogously to Hwang et al. [10]). As an example consider a product being placed once on the lowest board with location value one and twice on a board with location value 1.5 as can be seen in figure 3.2. For this specific scenario the location value is $\frac{1*1+2*1.5}{3} = \frac{4}{3}$.

Figure 3.2: Example for computing the location value for products located on multiple boards.

Even though the vertical position is integrated in FrAPP's demand function, the horizontal aspect is ignored because the best horizontal position differs from category to category. In some categories the middle position and in others the border positions achieved more demand (Dréze et al. [2]). Therefore, unless data for each category exists, the horizontal position cannot be included in the demand function. Furthermore, since the horizontal position influences demand significantly less than the vertical position does (Dréze et al. [2]) the more important aspect of reality is captured in FrAPP's demand function.

Amount of Space: Area

Instead of simply using the amount of facings as a measurement of space I use Murray et al.'s [1] approach which is the area facing front. Naturally, a bigger product gets noticed more easily than a small product, which is ignored by the amount of facings. However, since products cannot be expected to be equally sized even if they are in the same category (e.g. chips and peanuts) the facing area is the superior measurement.

Elasticities

Naturally, the space elasticity is incorporated into FrAPP's demand function as it represents diminishing returns (see section 1.2.3) and is therefore important to automatically prevent products from being displayed exaggeratedly often.

Even though cross space elasticity is hard to capture and influencing demand less than space elasticity, it still describes an important aspect of reality and should therefore be integrated into the demand function. Without cross space elasticity, the highly competitive interrelations would be ignored completely which abstracts too far away from reality.

Scale Parameter

Some products are more popular than others and get sold more often even if they are placed in exactly the same way. For this reason almost every related paper incorporated a product dependent scale parameter in their demand function. However, no one specified how this scale parameter is achieved.

I will use the demand in the worst possible placement as a basis. That is, the product has exactly one facing on the lowest board directly above the ground. This worst possible allocation is a good scale parameter as it treats all products equally and ignores unbalances like differently sized products or better positioning of products. In contrast, if the market share was taken as a scale parameter, it is not taken into account that more popular items often already have better locations than less popular ones which additionally increases the demand. Therefore, the market share achieved by better positioning and higher amount of space should not influence the scale parameter.

The reason I chose the worst possible placement as a scale parameter instead of any other fair measurement having the same amount of space on each board is that one can simply multiply a factor bigger than one if the product is placed on a better level or if the product is placed more often. Furthermore, one should note that the worst allocation demand considers local differences in sales of products, which is another advantage over the market share.

Trends

There exist many factors influencing demand outside the store, like advertisement, price change, weather (influencing the sales of grill meat or umbrellas), etc. However, those are not incorporated directly as this thesis' focus is on the influences of indoor product allocation. Also, these factors are extremely hard to capture if not completely unpredictable. Nevertheless, a factor called trend is incorporated in FrAPP's demand function to allow retailers expecting e.g. a 5% increase in demand for a specific product to insert this expectation into the program as another scale parameter for the demand of the product.

3.1.2 Formalization

I will now formalize the demand function taking into account all previously mentioned demand parameters.

The following notation will be used:

- v_i worst allocation demand
- α_i space elasticity of product i, value between 0 and 1
- β_{ji} cross space elasticity from product j to product i, value between -1 and 1
- D_i demand of product i
- I set of all products i
- B set of all boards b
- w_i width of product i
- W_b width of board b
- h_i height of product i
- H_b height of board b
- d_i depth of product i
- D_b depth of board b
- f_{ib} facings of product i on board b
- Loc_b rating of the location of board b
- $Trend_i$ known change in sells of product i independent of the product allocation (given as a float, not in %)

The demand function is designed as follows:

The maximum possible stacking for product i on board b is

$$\lfloor \frac{H_b}{h_i} \rfloor \tag{3.1}$$

which is the height of the board divided by the height of the product. It gets rounded down such that the stacking is always an integer and fits on the board.

The total amount of products facing front of product i on a board b is given by the multiplication of facings on the board and their stacking. The overall amount of product instances in the front for product i is then simply defined by the sum over the products facing front on all individual boards:

$$F_i = \sum_{b \in B} f_{ib} \lfloor \frac{H_b}{h_i} \rfloor \tag{3.2}$$

Naturally, instead of using the maximum possible stacking users can also enter a product-specific stacking amount.

As mentioned above, the location value for product i gets computed as the weighted average of the location values on which it is placed:

$$LocVal_i = \frac{\sum\limits_{b \in B} f_{ib} \lfloor \frac{H_b}{h_i} \rfloor Loc_b}{F_i} \tag{3.3}$$

Note that the above formula uses the products facing front and not simply the amount of facings on the individual boards. This is logical as products being stacked in height are more conspicuous than non-stacked products and therefore achieve a higher visual weight.

The facing area of product i gets computed by multiplying the amount of products facing front with the width and height of an individual product:

$$Area_i = w_i h_i F_i \tag{3.4}$$

Including the space elasticity leads to

$$Area_i^{\alpha_i} = (w_i h_i F_i)^{\alpha_i} \tag{3.5}$$

The cross space elasticity from product j to product i is the size of the facing area of product j to the power of the cross space elasticity value from j to i. Therefore, the overall effect of all products on product i can be expressed as

$$\prod_{\substack{j \in I \\ j \neq i}} Area_j^{\beta_{ji}} = \prod_{\substack{j \in I \\ j \neq i}} (w_j h_j F_j)^{\beta_{ji}} \tag{3.6}$$

Taken together, the overall demand function is

$$D_i = v_i * Trend_i * Area_i^{\alpha_i} * LocVal_i * \prod_{\substack{j \in I \\ j \neq i}} Area_j^{\beta_{ji}} \tag{3.7}$$

Capturing v_i is now pretty simple: if one takes the current placement and the current sales for each product as demand, everything except v_i in the above formula is known. Therefore, it can simply be solved for v_i which shows that no additional information is needed to get the worst allocation value.

With the computed worst allocation value, given price, trend as well as product and board dimensions, it is now possible to compute the demand for any product allocation.

3.1.3 Improved Demand Function

It seems logical that not only the space assigned to one product influences the demand of another product, but also the position. E.g. a facing of fruit tea of brand A on eye level should influence the demand of fruit tea of brand B more negatively than one on ground level. Since the position has a positive effect on the sales of a product [2], better positions of related products should also have a negative impact on demand as it leads to a higher probability that the worse positioned product gets substituted by the other. Therefore, one should incorporate this fact in above demand function 3.7. Since both space and location assigned to one product influence the sales of another product, a parameter called "cross space location elasticity" between two products is needed. Even though this fact was ignored completely in the related literature, it should be even more important than the simple cross space elasticity as the position of a product has a higher impact on sales than the space assigned to a product [2].

Therefore, it is incorporated in the above demand function in the following way: Instead of multiplying the demand of a product by the space of each other product to the power of the cross space elasticity it should now be multiplied by the product of space and location of each other product to the power of this new cross space location elasticity parameter between the two products. Formally:

$$D_i' = v_i * Trend_i * Area_i^{\alpha_i} * LocVal_i * \prod_{\substack{j \in I \\ j \neq i}} (Area_j * LocVal_j)^{\gamma_{ji}} \tag{3.8}$$

where γ_{ji} is the cross space location elasticity from product j to product i.

It should be noted that this cross space location elasticity is not symmetric meaning that $\gamma_{ji} = \gamma_{ij}$ does not necessarily hold for the same reasons as cross space elasticity is not symmetric.

Even though it sounds completely plausible that the position of one product can influence the demand of another, the existence of the cross space location

elasticity is not proven by a field study yet. Nevertheless, due to reasonable arguments as the above, it is included into FrAPP's demand function.

It should be noted that the cross space location elasticity can in theory be both positive and negative, using the same arguments as for cross space elasticity: positive values describe scenarios like sales in milk leading to an increase in the sales of muesli, while negative values describe the negative impact on sales on fruit tea of brand A caused by a better position and more assigned space to fruit tea of brand B.

3.2 Optimization Model

Using demand function 3.8 it is now possible to define the optimization model.

3.2.1 Profit

FrAPP maximizes the profit defined as selling price - purchase price - taxes, which is basically the gross profit minus taxes. The taxes must be included since in Germany taxes differ between product categories. Staple food like meat have only 7% while drinks have 19% taxes.

Naturally, one could maximize the net profit as many related papers do. However, data like cleaning costs, transportation costs, etc. is needed for the net profit but was not available for my thesis. Furthermore, and more importantly, it would only lead to different solutions than the gross profit if the cost aspects divide differently among the products. Otherwise, the overall net profit would be less than the gross profit but since the costs would split equally among the products the proportions would stay the same and therefore the placement would not be influenced. For this reason, considering the net profit is only superior if the costs get split differently between products (e.g. transportation costs, carrying costs).

3.2.2 Requirements

Valid solutions must naturally fulfill some requirements:

Bounds on Facing Amount

The retailer may define lower and upper bounds on the facing amount of each product (like e.g. Coskun [9]). Minimal facing amounts can for example be used to ensure that the own brand is placed at least x times even if this is not lucrative. However, since the retailer might gain from the production of these products as well, it could still be cost-efficient which is why this possibility should be given.

No Stock-Outs

Furthermore, FrAPP prohibits solutions where stock-outs are possible, since they result in lost sales. Even solutions where products in the front row (facings + stacking) disappear are eliminated. Since I assume full-space merchandising (like Hwang et al. [10]) products in the front row can only disappear if all products behind were already sold, otherwise a salesmen would refill the front. However, since the facing area is part of the demand function, disappearing front row products lead to a decrease in demand and should therefore be avoided. FrAPP assumes that products are delivered in fixed intervals (e.g. once a week). For the above reasons, only solutions having at least as many products behind the front as the overall demand in that replenishment interval are allowed.

Connection Between Products

The optimization model requires of all solutions to fulfill the following product connection criteria: all instances of a product must be connected within a shelf. This does not enforce uniform columns or blocks, however, it rules out irritating solutions where product instances can be found in multiple locations in a shelf without those being connected. Figure 3.3 shows an example to clarify the product connectivity criteria.

Figure 3.3: The left image shows a valid solution because all product instances are connected for both products, while this is not the case in the right placement.

In the left image the brownies span over multiple boards. However, since for both products it holds that all product instances are connected, it is a valid solution. On the right the "QKies" are valid as well, since all instances build one connected component. However, since the two bottom left brownie instances are not connected to the three top right instances the solution as a whole becomes invalid. Placements as the right one are prohibited to avoid customer irritation and an untidy overall appearance of the store.

A stronger limitation like enforcing products to build complete blocks reduces the solution space tremendously, which might lead to less profitable overall solutions. In contrast, allowing all combinations leads to a messy store appearance which is not customer friendly. Therefore, this approach is a good compromise between eliminating possibly optimal solutions and producing confusing product allocations.

Integer Valued Facing Amount

Last, the amount of facings must be a positive integer, otherwise it would not be applicable in practice. The requirement that it must be positive indicates that FrAPP does not perform product selection, since each product must be placed. However, since I assume that the retailer already selected all products in advance it makes sense to display each product at least once.

Comparison to Related Papers

Model	Profit	Stock-Outs	Facing Amount	Product Selection
Bai and Kendall [11]	Net P.	Not Considered	Integer	Yes
Hwang et al. [10]	Net P.	Partly Considered	Continuous	No
Murray et al. [1]	Gross P.	Not Considered	Integer	No
FrAPP	Gross P. - Taxes	Considered	Integer	No

Table 3.1: Comparison of FrAPP's optimization model to the three most related papers.

As table 3.1 shows, Hwang et al. [10] "partly considered" stock-outs, since they computed the best reordering point to avoid stock-outs or even a reduction in demand due to the depletion of front row products. However, since the authors make the unrealistic assumption that replenishment is instantaneous, it can only be considered a "partly" avoidance of stock-outs.

Bai and Kendall's model [11] allowed products to have a facing amount of zero, which enables their solver to perform product selection. However, in general product selection is not as simple since excluding a product from the assortment might transfer its sales to a substitute product within the assortment. Furthermore, simply allowing the amount of facings to be zero might lead to solutions where whole categories are not placed due to their unprofitability. Even though in this way the shelf might produce more income, missing a whole product category can motivate customers to switch the store [27]. For the above reasons, the planning staff is expected to perform product selection instead of the planning tool.

Coskun [9] used flexible board heights in his optimization model. Even though his results were quite interesting, usually shelf boards are not completely flexible in height for aesthetic reasons. Having differently arranged boards in each shelf leads to an irritating overall impression. Therefore, FrAPP assumes fixed board heights.

Zufryden [14] incorporated product blocks in his optimization model to allow e.g. bottles to be aligned as six-packs. However, a grouping of product is usually considered as an individual product with an own article number in the databases, which makes Zufryden's consideration redundant.

Furthermore, even though Cox [13] showed that positioning and space assignment of staple products does not influence the demand, FrAPP does not consider them separately. First, the data provided does not support easy distinction of staple products and second, treating them differently would always lead to solutions having salt, etc. placed on the lowest board and having assigned just enough space to not stock-out. However, this seems highly customer-unfriendly and should therefore be avoided.

Last, it should be noted that the model does not explicitly minimize empty showroom. However, a maximization of profit will always minimize empty areas because they do not produce income.

3.2.3 Formalization

Apart from the above notation some further abbreviations are needed to formalize the optimization model:

- sp_i selling price of product i
- pp_i purchasing cost of product i
- t_i taxes on product i
- r_i replenishment interval for product i
- $F_{i_{min}}$ lower bound on facings of product i
- $F_{i_{max}}$ upper bound on facings of product i
- F_i total amount of facings of product i

The optimization model can now be defined as:

$$max \sum_{i \in I} (sp_i - pp_i - t_i) D'_i \tag{3.9}$$

such that the following holds:

$$\sum_{i \in I} w_i f_{ib} \leq W_b \qquad \forall b \in B \tag{3.10}$$

$$F_{i_{min}} \leq F_i \leq F_{i_{max}} \qquad \forall i \in I \tag{3.11}$$

$$D'_i r_i \leq \sum_{b \in B} f_{ib} \lfloor \frac{H_b}{h_i} \rfloor \lfloor \frac{T_b}{t_i} \rfloor - f_{ib} * \lfloor \frac{H_b}{h_i} \rfloor \qquad \forall i \in I \tag{3.12}$$

$$productsConnected(i) = true \qquad \forall i \in I \tag{3.13}$$

$$F_i = \sum_{b \in B} f_{ib} \in \mathbb{N}^+ \qquad \forall i \in I \tag{3.14}$$

Here, equation 3.10 requires the summed up product width to be less or equal to the board width. Naturally, the depth and height of boards should not be exceeded either, however, since only as many products as fit are stacked in depth or height these requirements are met automatically. Naturally, the model assumes that the dimensions of a board are large enough to accommodate at least one product instance, which is reasonable as in the supermarket environment boards usually are way larger than products. Formula 3.11 enforces the amount of facings per product to be in the given bounds. Equation 3.12 prohibits stock-outs or even lower sales because of disappearing facings. That is, the expected sales in a replenishment interval, namely the demand multiplied by the amount of days until the shelf is refilled, cannot extend all stocked products minus the front row products as discussed above. Furthermore, 3.13 requires that the product connectivity criteria is fulfilled for all products. As formalizing this properly involves building a graph of products and therefore leads to a tremendous amount of new variables which in term complicates the optimization model, a boolean oracle called "productsConnected" is introduced instead. The algorithm representing this oracle by checking the product connectivity is described in detail in section 4.4.3. Last, equation 3.14 simply enforces that the amount of facings per product is a positive integer.

Chapter 4
Optimization Algorithm

4.1 General Algorithm

As an optimization algorithm I chose a modified version of Bai and Kendall's [11] simulated annealing based hyper-heuristic. The main reason for this is the product connectivity. E.g. the genetic algorithm creates new solutions by pairing existing solutions. However, pairing two placements such that the resulting placement satisfies the product connectivity criteria with high probability seems unfeasible. As another example performing mathematical optimization using MINLP solvers requires that all conditions that have to be satisfied can be expressed mathematically. However, this formalization is hard as it requires introducing a huge amount of new variables needed for the product graph upon which the connectivity check is performed. Furthermore, meta heuristics cannot find optimal solutions on all problem instances as the factors differ too strongly depending on the products included in the problem or e.g. the relation between the amount of space available and the amount of products to place varies strongly. While on a specific problem instance a meta-heuristic leading to high quality solutions might be developed, the same meta-heuristic might fail upon other instances. This can be explained using the "No Free Lunch Theorem" which states that there cannot be an algorithm outperforming all others in all problems [24]. Therefore, Bai and Kendall [11] argue that a hyper-heuristic should be used upon such changing problems which "operates over the solution space indirectly by searching the heuristic space" and can therefore adapt to the specific problem instance.

As explained in section 2.2.4, Bai and Kendall [11] used a simulated annealing algorithm as a high level heuristic choosing from a set of low level heuristics

which define transitions from one solution to neighboring solutions. Even though their algorithm was proven efficient and delivers high quality solutions in their optimization model, it again does not support the product connectivity requirement in its current form. However, by exchanging the low-level heuristics to heuristics keeping the product connectivity intact with high probability, their fundamental idea can be adapted and reused even for this more complex optimization model. This reusability of hyper-heuristics by exchanging the low level heuristic to fit the specific problem specification is a huge advantage compared to other optimization algorithms.

The used low level heuristics are described in section 4.3 where the problem of Bai and Kendall's [11] heuristics, namely the violation of product connectivity, is explained in further detail. Even though the solutions output by these new heuristics produce valid solutions with high probability, it might still happen that a generated neighbor violates a requirement. Therefore, algorithm 1 was modified to check if the generated neighbor fulfills all requirements and skips it if this check fails. Furthermore, instead of returning only one solution the algorithm is modified to output a set of solutions from which the user can choose. The modified version of the algorithm is outlined in algorithm 2.

As already explained in the related work section 2.2.4 the main feature of simulated annealing algorithms is that they can escape local optima. This comes from the fact that depending on the temperature and the difference in rating to the current placement the algorithm can transition to worse neighbors. While this is likely in the beginning when the temperature is high it happens rarely towards the end of the algorithm when the temperature is low.

The initial solution is determined as described in the next section. For outputting a set of the best solutions, a priority queue ordered by the evaluation value obtained by $f(s)$ was used. This queue only memorizes the best $resultSize$ results, as saving all would quickly exceed memory capacity. Naturally, only solutions not already contained are added to the queue. This is important as the algorithm might visit the same solution multiple times. For the amount of iterations $nrep$ until the cooling schedule is invoked a value of 3 was used, which was achieved by experimental tests of the algorithm's performance. The temperature function is $t \rightarrow t/(1 + \beta t)$ as defined by Lundy and Mees's [26] cooling schedule. This is the same schedule Bai and Kendall [11] applied who defined $\beta = \frac{(t_s - t_f)T_{average}}{T_{allowed}t_s t_f}$ where t_s and t_f are the starting and final temperature and $T_{average}$ and $T_{allowed}$ are the average time needed for one iteration and the time allowed for the whole search. $T_{allowed}$ is given by the user through the UI while $T_{average}$ is computed by an initial procedure applied upon startup. This procedure simply samples $T_{allowed} * 10$ iterations of the algorithm and computes the average time needed for one iteration. The value 10 was used as it seemed to be a good compromise between wasting too much time before the algorithm starts and getting a good approximation of the average iteration time in experiments. As t_f I used 0.01 while the starting temperature is set dynamically depending

Algorithm 2 Modified Simulated Annealing Based Hyper-Heuristic

```
Select an initial solution s_0;
Insert s_0 into a priority queue Q limited to size resultSize;
repeat
  Randomly select a heuristic h ∈ H;
  iteration_count = 0;
  repeat
    iteration_count++;
    Applying h to s_0, get a new solution s_1;
    if (isConsistent(s_0)) then
      δ = f(s_1) − f(s_0);
      if (δ ≥ 0) then
        s_0 = s_1;
        enqueue s_0 into Q;
      else
        Generate a random x uniformly in the range (0, 1);
        if (x < exp(δ/t)) then
          s_0 = s_1;
          enqueue s_0 into Q;
        end if
      end if
    end if
  until iteration_count = nrep;
  set t = temperatureFunction(t);
until the stopping criteria is satisfied
return Q;
```

on the concrete problem instance: $T_{allowed} * 10$ neighbors are generated from the initial solution and the biggest possible gain δ_{max} in evaluation value is computed. Now analogously to Bai and Kendall [11] the starting temperature is set to $t_s = -\delta_{max}/ln(0.85)$ in order to allow 85% of bad moves in the beginning of the algorithm.

One should note that starting the algorithm with a computation time of 1000 but interrupting it after 20 seconds and outputting the current best solution can not be considered equivalent to starting the algorithm with an allowed computation time of 20. This comes from the fact that the temperature function uses $T_{allowed}$ and therefore the temperature cools off at a different speed in both cases. Therefore, the worse neighbors are accepted more easily in one case than the other.

Figure 4.1 offers an overview over the entities needed for the optimization algorithm. The algorithm used for finding initial solutions is outlined in section 4.2. From this initial solution the algorithm generates neighbors by applying any of the heuristics described in section 4.3. Even though these heuristics produce valid

neighbors with high probability it might still happen that a neighbor violates a requirement defined in section 3.2.2. Therefore, a consistency checker is needed which is described in section 4.4. In order to compare the current state to the neighbor the evaluation function defined in section 3.2 is used which in turn utilizes the demand function 3.7. In the end found solutions are sorted without adulterating the quality of the placement as is described in section 4.5.

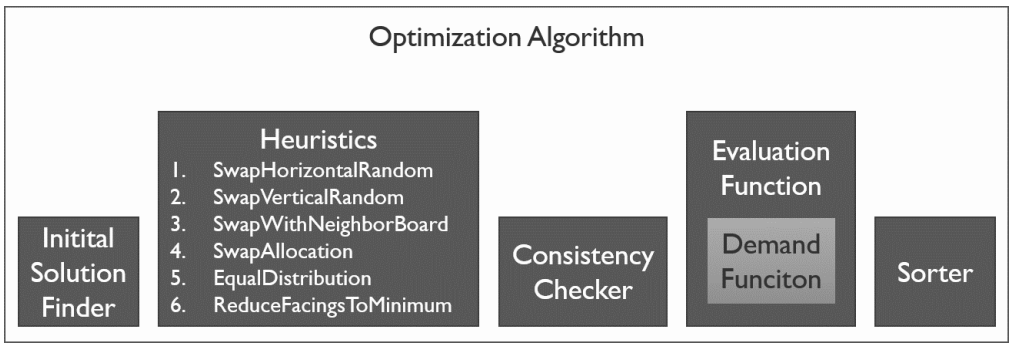

Figure 4.1: Overview over the needed entities for the optimization algorithm.

4.2 Finding an Initial Solution

For placements with a high density, meaning that there is barely enough space to accommodate the lower facing bounds for all products, finding an initial solution which satisfies all requirements is not as easy as it might seem. Depending on the combinations of products chosen for each board there can be more or less unused space which can be critical if almost all available space is needed to satisfy the lower facing bounds. Therefore, a greedy algorithm was developed, which works well in practice. Nevertheless, as it is a greedy procedure it is not complete. The algorithm is outlined in the following:

1. Sort products by the biggest amount of space needed in descending order.
2. In that order of products: try to find a board, where the whole minimum facing amount fits. If no such board exists try to allocate the minimum facing amount on neighboring or unrelated boards without destroying the connectivity of products.
3. Fill up the used boards. Therefore first try to add facings to existing products, second add up with other products while not loosing connectivity. Here, products which will stock-out according to the demand function are prioritized.
4. Fill up unused boards. Therefore first check if it has a used neighbor which can get copied or extended (vertical or horizontal). Take the neighbor filling

up the most space and and do so. Afterwards or if no neighbor exists do a consistent random add up.

The biggest amount of space needed simply describes the width needed to allocate the minimum amount of facings for each product. Two boards are unrelated if they are located in different shelves. The consistent random add up simply chooses a random product and places it on the board. If this new placement violates product connectivity the newly placed product is removed again and the next product is placed instead.

Naturally, this greedy procedure cannot be complete. Therefore, if no valid solution is obtained by the above algorithm, the user is asked if he wants to perform a brute force search. However, it happens only very rarely that the greedy algorithm does not find a solution, and in the case it fails, it might be that there exists none. It might be the case that the overall space needed is smaller than the available space, however, due to the physical dimensions of the products there does not exist a solution. E.g. consider a shelf consisting of 3 boards each of size 4 and 4 products of size 3. Basically, the products need a total space of 12 and the boards have a total space of 12, however on each board fits exactly one product and the last product would have to be split up in 3 equal parts in order to fit. In this scenario the above construction would not find a solution, however, the brute force algorithm would not either. Still there could be cases where the greedy algorithm fails but the brute force algorithm finds a solution. Therefore, upon a fail of the greedy algorithm the user can choose to brute force the solution, however, as this might take a long time as there potentially exists no solution the program suggests to change the settings instead.

4.3 Heuristics

Bai and Kendall [11] used extremely random heuristics like randomly selecting a facing of a product and replacing it with as many as possible facings of another randomly chosen product or, as further examples, deleting a random facing or adding a random product to a random board. This tremendous amount of randomness was fine within their model as they had only very few constraints and especially no product connectivity constraint. However, in a more constrained model as FrAPP's it leads to a huge problem: highly random heuristics like swapping random facings of random products violates the product connectivity property with high probability. One must be extremely lucky to swap two random facings in a way that both newly placed products are still connected to the old ones. Therefore, the neighbor state is probably invalid and can therefore not be considered any further.

Another problem of Bai and Kendall's [11] heuristics is that one third of their heuristics deletes facings which always results in less demand and therefore less profit. Unfortunately, as the temperature cools off it becomes less and less

likely to transition to worse neighbors which means that even though the delete heuristics get applied and use computation time, it is extremely unlikely that the changes they made actually get applied. This in turn makes the add heuristics, which are another third of all heuristics, superfluous since adding is impossible if the shelves are fully stocked. Taken together this means that at the end of the simulated annealing algorithm almost two thirds of the heuristics are executed without having any effect on the final result.

Due to those limitations the heuristics used by FrAPP's simulated annealing algorithm try to achieve two things: leading to valid solutions with high probability and not becoming superfluous during the algorithm. The used heuristics will be briefly explained in the following:

Swap1FacingHorizontalRandom: This heuristic chooses a random board and a random position between neighboring products on the board. Then facings of one product get deleted until the resulting gap is large enough to accommodate at least one facing of the other product. This gap then gets filled with as many facings of the other product as fit. A simple example can be seen in figure 4.2. Naturally, the swap is only performed if the product loosing facings still has more facings than the minimum amount afterwards and the product gaining facings does not exceed the upper facing bound after the swap. Otherwise the swap is not performed. If a swap on one board fails, the next randomly chosen board is used until either the swap was successful or failed upon all boards.

Figure 4.2: A simple example for the Swap1FacingHorizontalRandom heuristic.

Swap1FacingVerticalRandom: First, this heuristic tries to find two vertically neighboring boards at random. Then a horizontal position is randomly chosen and facings of one of the products located at this position upon one of the vertically neighboring boards are deleted until at least one facing of the product from the other board at this position fits into the gap. For an example see figure 4.3. Naturally, the swap is only performed if the products at the random position differ. Again, facing bounds have to be satisfied after the swap.

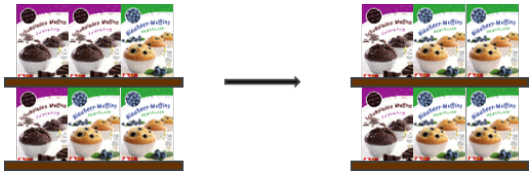

Figure 4.3: A simple example for the Swap1FacingVerticalRandom heuristic.

Swap1FacingWithNeighborBoard: This heuristic tries to find two horizontally neighboring boards and does the same as the above heuristic with the two

products at the border of the boards. Naturally, the boards are chosen as random as possible. An exemplary application can be seen in picture 4.4.

Figure 4.4: A simple example for the Swap1FacingWithNeighborBoard heuristic.

SwapAllocation: This heuristic chooses two products at random and places as many facings of product A where product B was and the other other way around. In the whole, allocations are swapped. Figure 4.5 shows an exemplary neighbor generation using this heuristic.

Figure 4.5: A simple example for the SwapAllocation heuristic.

EqualDistribution: This heuristic balances the amount of space allocated to two products as equally as possible by removing and adding facings from each board on which both are placed. As an illustration see figure 4.6.

Figure 4.6: A simple example for the EqualDistribution heuristic.

ReduceFacingsToMinimum: This heuristic chooses a product randomly and reduces its facing to the minimum allowed amount. Then the resulting gaps on the different boards get filled with facings of the horizontal and vertical neighbors. Figure 4.7 shows a really simple example. Again, this heuristic is only applied to products where one has more than the allowed minimal amount of facings and the other has less than the allowed maximum.

Figure 4.7: A simple example for the ReduceFacingsToMinimum heuristic.

One should note that even though the probability that these heuristics violate constraints is significantly less than Bai and Kendall's [11] heuristics in the context of my constraints, it is not impossible for them to violate constraints. For example, if product instances were only connected because this product had a great amount of facings on a board, the EqualDistribution heuristic might destroy connectivity by removing facings from this specific board. Nevertheless, the above heuristics

will lead to valid solutions with significantly higher probability Bai and Kendall's [11].

Furthermore, it is important to see the difference between the randomness in the above heuristics and the randomness in Bai and Kendall's [11] heuristics. E.g. the swap facing heuristics look similar at first glance, however, the above heuristic swaps only between neighboring products and considers minimum and maximum facing bounds while Bai and Kendall's [11] swap heuristic swaps random facings of randomly chosen products. This especially means that the disappearing facings might be in the middle of the space allocated by a product and that it might be filled using a product which is located at a completely different spot in the shelf.

4.4 Satisfying Requirements

This section shows how the fulfillment of the requirements defined in section 3.2.2 is ensured. As soon as any of the following verifications fails, the current placement is considered inapplicable and the simulated annealing algorithm proceeds from the previous state again instead of translating to the new neighbor. This approach ensures that only valid solutions are returned. Naturally, one could also transfer to invalid neighbors, as an invalid neighbor might become valid again at some point and only exclude the inconsistent state from the solution set. Nevertheless, FrAPP uses the first approach since it is almost impossible that all possible neighbors are invalid, while on the other hand it might take a long time until an inconsistent state becomes consistent again: consider for example a scenario with 50 products where a product P has a minimal facing amount of 2 and a heuristic reduces the amount of facings of P to 1. Now first of all a heuristic which could in theory add the missing facing needs to be chosen (not all heuristics can do this depending on the concrete placement), second the heuristic must choose this product as the one getting more space assigned (probability 1 over 50), and third the assignment has to be applicable without violating any other constraint.

4.4.1 Facing Bounds

Checking if the facing bounds given by the user are satisfied is easy: FrAPP simply counts the amount of facings assigned to each product in the current state and checks if given bounds are satisfied or violated.

4.4.2 Prohibit Stock-Outs

Deciding weather or not a given placement leads to stock-outs, is done exactly as formulated in the mathematical model: as argued in section 3.2.2 it is not even allowed that product instances in the front (facings or their stackings) disappear.

Therefore, as formula 3.12 suggests the expected sales for each product (defined by the demand function) should be less or equal to the product instances behind the front which can be trivially be computed.

4.4.3 Product Connectivity

The optimization model requires of all valid solutions to satisfy the connection criteria. That is, all product instances of a product must be connected within a shelf. In section 3.2.2 this requirement was described and an oracle called "$productsConnected(i)$" was used in the model thereafter. However, for keeping the amount of variables manageable this limitation of solutions was not defined mathematically. In the following the implemented algorithm for checking the above criteria is explained.

The first step of the algorithm is to transform the placement into a graph. Here, each product is represented by a node and an edge between two nodes exists if and only if both nodes represent the same product and the two products are connected directly. That is, the two products are either direct horizontal neighbors or they are connected vertically meaning that the intervals in space covered by the products overlap. Example 4.8 show two placements in the shelf with the connectivity graphs as an overlays.

Figure 4.8: The left picture shows the graph of a valid solution as all product instances are connected, while the right image shows the graph upon a more complex and forbidden placement where products are arranged in an offset pattern.

One main observation can be made from these examples: in legal placements all product instances of a particular product form a connected component, while illegal placements have product instances distributed over at least two connected components. As an example consider the right image of figure 4.8, where Brownies and QKies each form one single connected component, while the shower gel forms two separate components each of size two. More formally, a placement is valid if and only if for each product in the placement it holds that the amount of product instances equals the size of a connected component for this product. This basically requires that for each product there exists exactly one connected component holding all product instances.

With this observation the algorithm checking for product connectivity simply compares the amount of each product with the size of a connected component for this product and returns false if they do not match. The size of a connected component gets computed using a breadth-first search starting at a node representing the product and counting the visited nodes.

Using the graph described as above where edges can only exist between nodes of the same product instead of the complete neighboring graph improves the running time of the algorithm significantly. For one, building the graph is faster as many edges are superfluous and more importantly the algorithm computing the size of connected components runs faster since all edges are valid and tremendously less edges have to be examined. It is important to improve this connection checking algorithm as much as possible since it is called thousands of times in the simulated annealing algorithm.

4.5 Sorting

The resulting placement can be sorted without changing the evaluation value as long as products stay on the same levels of height. As argued in the previous chapter, demand is not influenced by the horizontal position or at least no overall preferred horizontal position could be found [2]. Therefore, the horizontal position is not influencing demand in FrAPP's model, which is why products can be translated along the horizontal axis without changing the profit achieved by the placement. As better sorted shelves are more customer friendly, a horizontal sorting should be performed. Of course, one could also constrain the optimization model more strictly to automatically yield perfectly sorted shelves, however, since this would rule out a huge amount of solutions it probably leads to less profitable solutions which itself should be avoided. One should note that product connectivity is already a constraint which yields a base level of sorting. However, even though this base sorting is already acceptable, stronger sorting is preferred. For this reason when the final solution is found, it gets sorted by horizontally swapping the product order. However, there exist cases where sorting is not possible because e.g. a category is located on the topmost and bottom most board, but not on the middle board. If this is the case, the placement is returned in its current form, that is, the product connectivity criteria is fulfilled, but no additional sorting is performed.

Two different orders are supported: sorting by brand and sorting by type. Of course any other order can be easily used as long as a function dividing the products into categories is given. If sorted by brand this function simply puts products of the same brand into one category and if sorted by type products belonging to the same parent category are taken as a unit. If the data is specific enough, one could for example divide muesli into chocolate muesli, fruit muesli etc. An example with overlays for sorting by type can be seen in figure 4.9. Image 4.10 shows another solution for the same problem instance being sorted by brand.

Figure 4.9: Boards sorted by type with an overlay for each category.

Figure 4.10: Boards sorted by brand with an overlay for each producer.

Given these categories of products the sorting process is started. It consists of two independent sorting procedures, one trying to find category allocations that match and another aiming for product connectivity within such a category connected allocation. By splitting the sorting procedure into the above two subroutines the search space is reduced dramatically in the first step which speeds up finding overall solutions in the second step.

Retrieving consistent category orderings is done in the following way: the sizes of all categories on each board are computed by simply summing up the product widths for each category. Then starting from an initial category ordering the order per board gets permuted in every possible way leading to a set of possible category orderings per board. Using simple combinatorics, shelves are created using all possible combinations of entries in the category orderings per board. Naturally, only shelves where all category instances are connected are memorized. This category connection criteria is analyzed by building up a graph and checking connectivity therein. Overall this verification is done analogously to the product connectivity check which is explained in further detail in section 4.4.3.

Now that all category orderings are found, allocating products within this order is realized by trying all permutations within each category consistent shelf until an ordering satisfying the product connectivity criteria is found. If one such category

consistent shelf has no product connectivity satisfying product allocation, the algorithm proceeds with the next category consistent shelf. Again, the product connectivity gets checked by building up a graph as explained in more detail in section 4.4.3.

4.6 Concurrency Modes

Apart from the single threaded version of the simulated annealing algorithm, which can be seen in figure 4.11, three different concurrency modes were implemented. The amount of threads spawned in each mode is limited by the amount of cores the target machine supports.

The **parallel runs** mode executes the complete algorithm in parallel. For a graphical overview see figure 4.12. The result sets of the different threads then get merged with duplicate elimination and ordered by their evaluation value. This concurrency mode is basically the same as running the algorithm multiple times using the same settings and taking the best results. Different algorithm runs might lead to different results as each run can develop into a completely different direction depending on the selected heuristics, their internal randomness and which worse solutions get accepted.

As the name suggests, concurrency mode **parallel heuristics** executes multiple heuristics in parallel as can be seen in figure 4.13. That is, whenever the simulated annealing procedure normally chooses one heuristic and generates a neighbor, it now creates multiple neighbors using different heuristics. As in the single threaded version the algorithm performs $nrep$ iterations with each heuristic, that is, $nrep$ times a neighbor is generated by the heuristic and the procedure transfers to the neighbor if it is better or the algorithm is lucky. The best neighbor according to the evaluation function is then chosen and the algorithm continues normally. If it is better than the current placement the algorithm transitions to this best neighbor, if it is worse, the algorithm transitions to it only with a probability depending on the temperature.

The last concurrency mode is called **parallel neighbors** and splits after selecting an heuristic, which is depicted in figure 4.14. Using this selected heuristic multiple neighbors are generated. Here, one should note that all heuristics contain a randomness and therefore executing the same heuristic multiple times yields different neighbors. These neighbors are then compared and the best is chosen. Again, using this best neighbor the algorithm continues normally.

Previous two concurrency modes consider multiple neighbors in each iteration thereby leading the algorithm in better directions. However, taking the most promising direction might not always be the path to the optimum. The evaluation should determine which of the above concurrency modes yields the best solutions and which is more efficient.

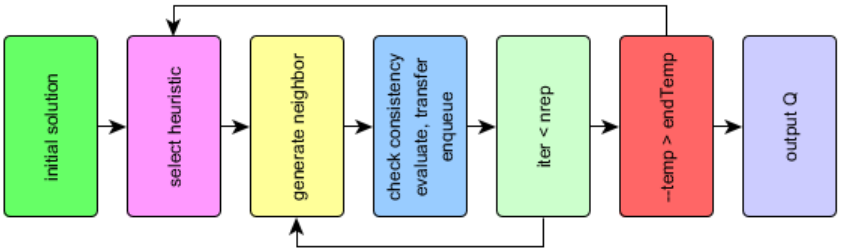

Figure 4.11: Single threaded version of the simulated annealing based hyper-heuristic.

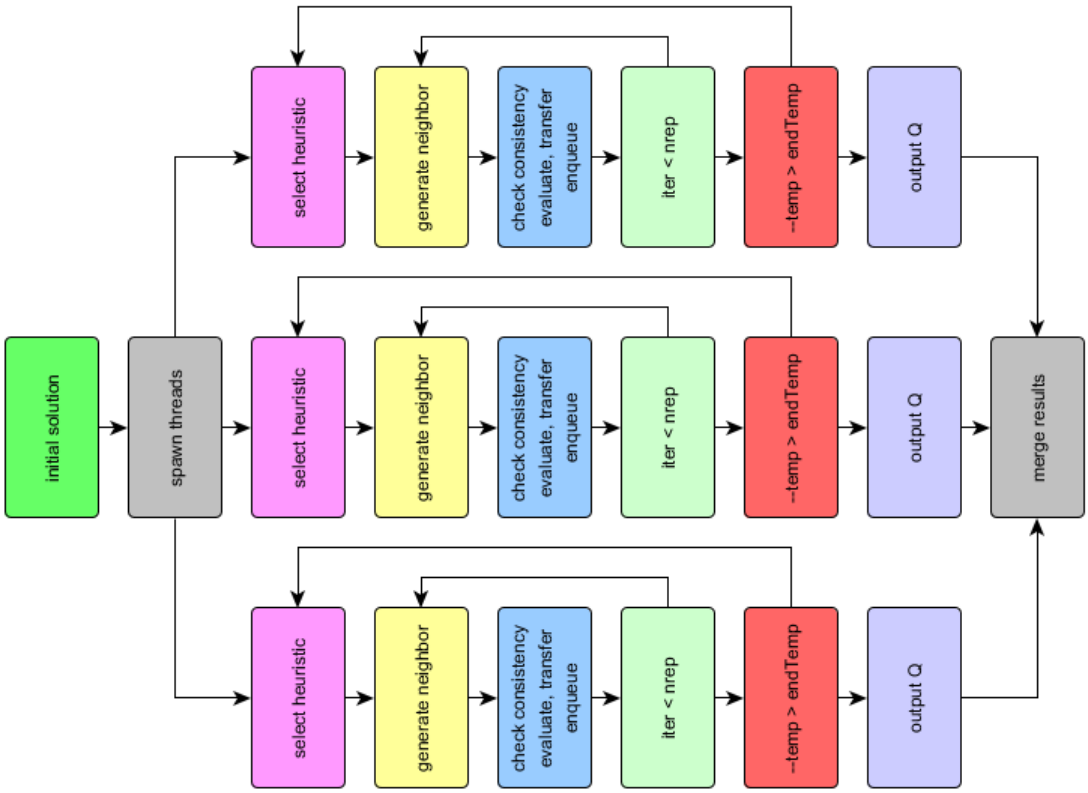

Figure 4.12: Parallelization mode parallel runs.

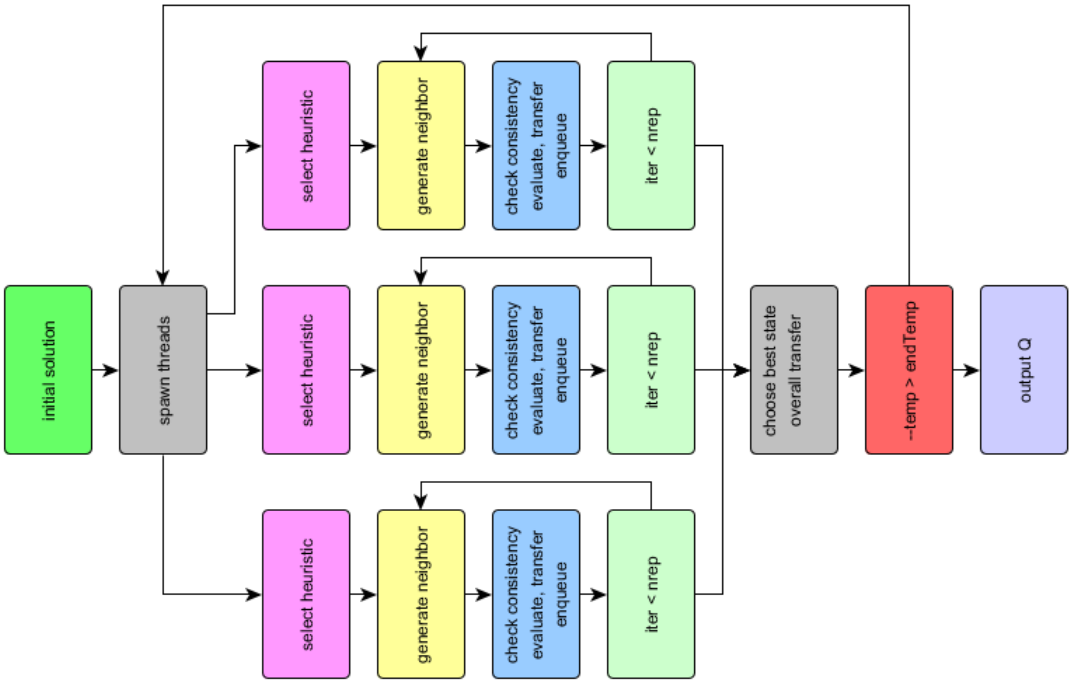

Figure 4.13: Parallelization mode parallel heuristics.

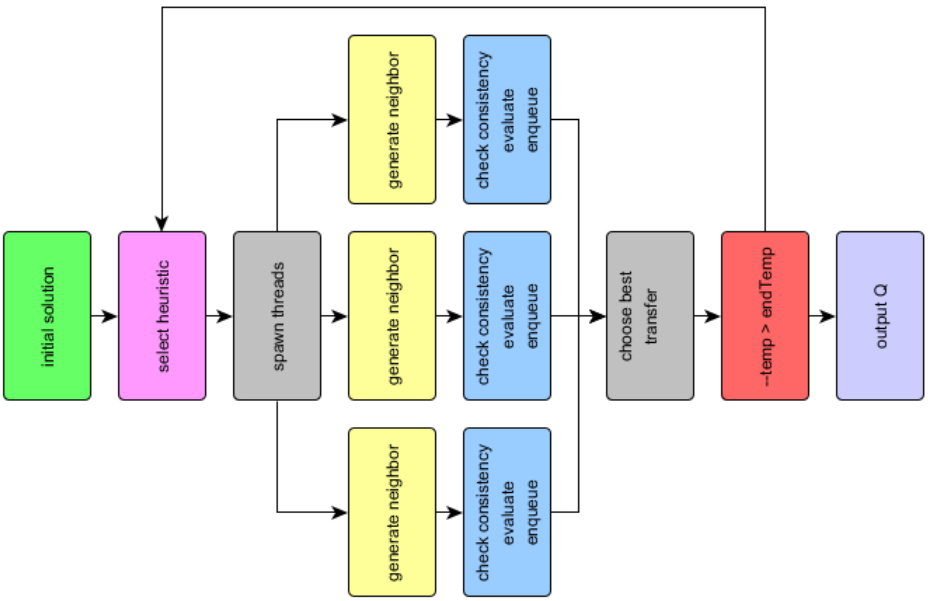

Figure 4.14: Parallelization mode parallel neighbors.

Chapter 5
Implementation

5.1 Implementation Details

The program was implemented as part of the "Innovative Retail Laboratory" (IRL) software, a research lab of the DFKI (German Research Center for Artificial Intelligence) in cooperation with the German retailer GLOBUS SB-Warenhaus Holding in St. Wendel [28]. This software already offered the possibility to walk through 3D models of supermarkets existing in reality.

As a programming language Java with Java3D for rendering the scenes and products was used.

Real data was provided by Globus which allows realistic scenarios. It contained almost all needed information like product assortment, prices, physical dimensions of products and boards, previous sells, previous locations, etc. This data even enabled the program to distinguish stackable products from unstackable ones (e.g. Muesli is stackable while wine is not) and compute their demand differently. Even such detailed data as the stacking behavior is important as e.g. assigning vertically large boards to unstackable products might be less profitable than assigning it to stackable products even if the unstackable product itself is more lucrative and demanded. All data from the database was fetched in a initialization phase and then cached to limit the network traffic to a minimum and thereby increase the computation speed dramatically.

Unfortunately, even though a lot data was at hand, some other data needed for the optimization model was missing, namely the amount of taxes and the purchasing price of different products. Therefore, instead of optimizing the gross profit subtracted by the taxes it is simply assumed that the profit of each product is 5% of its selling price. This basically sets the purchasing costs to 95% of the

selling price and ignores the taxes. Of course having the required data would lead to higher quality solutions, however, most of the important data was provided by Globus which already yields good and realistic solutions.

5.2 Parameter Estimation

For the demand function described in 3.1.3 the following parameters are needed: the location values of the different boards, the space elasticity, the cross space location elasticity and the worst allocation demand.

Location Values

The location values of boards get computed in a really simple manner: depending on the distance to the floor, boards get assigned different location value. Here, the worst positions get a value of one (analogously to Dréze et al. [2]) and the closer a board is to eye level, the higher the value gets. The eye level is the height in which the eyes rest assuming an average human. Since the average German is 172 cm tall [29] and the eyes are approximately 7 cm below, the eye height is around 165 cm above the floor. Humans prefer viewing slightly towards the ground which naturally influences the eye level. Analogously to Dréze et al. [2] the computation uses a 15 degrees angle of view. Assuming the customer is standing 30 cm away from the shelf, the difference between eye height and the actual point of rest can be computed using the geometry of an orthogonal triangle:

$$tan(\alpha) = \frac{oppositeLeg}{adjacentLeg} \leftrightarrow tan(15\,^\circ) = \frac{oppositeLeg}{30cm} \longrightarrow oppositeLeg = 8cm \tag{5.1}$$

Therefore, the middle of the eye level should be defined as $172cm - 7cm - 8cm = 157cm$. The section containing the eye level can then be defined as the interval between 150 and 175 cm. As Dréze et al.'s [2] field study showed that a 39% difference in sales exists between the best and the worst vertical position, the eye level section gets a value of 1.39, meaning that 39% more products get sold if they are placed on eye level compared to floor level. Table 5.1 shows the concrete values used for the different sections.

0 - 50	50 - 80	80 - 110	110 - 140	140 - 170	170 - 190	> 190
1.0	1.1	1.2	1.3	1.39	1.25	1.0

Table 5.1: Table defining the location values corresponding to different heights.

If the data dividing products into children and adult products was at hand it would easily be possible to determine the location value depending on the average height of the target audience. This would lead to better results as children's

products should be located lower in order to attract attention than adult's products. However, since this data is not available the current implementation uses 157 cm as the best height.

Space Elasticity

A lot of research has been done to get good estimates of the space elasticity. Curhan [5] found an average value of 0.212 in his experiments, while others like Desmet and Renaudin [8] found an average value of 0.2138. Generally, Hwang et al. [10] and Reyes and Frazier [7] offer great overviews over the space elasticity parameters achieved by different experiments. Of course, it would be best to have different space elasticities for different products, however since this data does not exist this implementation uses the same value for all products as an approximation. Despite arguments for all of the above values, FrAPP uses 0.17 as space elasticity which is the average value Eisend [30] found in the latest research on this topic.

Cross Space Location Elasticity

Even though in theory cross space elasticities and therefore also cross space location elasticity can be positive as well as negative, automatically determining positive values is almost impossible because positive values describe scenarios like sells in milk leading to an increase in the sales of muesli. However, negative interrelations can be approximated using the given data in the following way: the products are given in a category graph where each category can have parent and child categories. The assumption is that products influence each other more negatively the closer the earliest common ancestor is. This is logical as for example chocolate mueslis have a higher competition than mueslis in general. Therefore, the algorithm determining the cross space location elasticity computes the closest ancestor between two products and retrieves the amount of categories between the product itself and this common ancestor. If it is the direct parent of one of the two products the cross space location elasticity is high (-0.03), if it is the grandfather of a product the value is already lower (-0.005) and if the common ancestor is even further away there is no interdependence anymore (value 0). Products without a common ancestor are of course assumed unrelated and therefore have a cross space location elasticity value of 0. The above values are taken from Coskun [9], however, like most of the related work, Coskun simply chose a random value in an interval for product pairs instead of using the category graph. Even though Coskun designed those values for cross space elasticity instead of cross space location elasticity, they should apply here as well, since the only difference is that the amount of space assigned to a product is multiplied by a location factor in the range of [1, 1.39] before it is expotentiated by the cross space location elasticity value. Therefore, it is potentially larger than the cross space elasticity which is fine as the combined effect of space and location is

larger than the effect of space as well. Even though using this value should serve its purpose, finding precise values in a field study would improve the results of FrAPP. Furthermore, the approach using the category dependencies is still only an approximation and requires the dependency data, however it is closer to reality than most of the related work approaches.

For both space elasticity and cross space location elasticity the measure of space is extremely important: if, abstracting from the unit, space falls below 1, potentiating it to a value in]0,1[is larger than the space, while expotentiating a value bigger than 1 to the same cross space location elasticity value is larger than 1, but smaller than the unit space itself. Analog problems occur with negative exponents for cross space location elasticity. As it is always required that the product values for space elasticity are bigger or equal 1 and smaller equal 1 for cross space elasticity, it is implied that the space covered by a product should never be below 1. Since square meters or decimeters cannot guarantee this, and smaller units lead to an extreme blow up in the space units covered by a product, FrAPP uses the area of the smallest product as the unit of space and computes all areas as multiples thereof. This ensures that the area is always equal or bigger to 1 while avoiding gigantic space amounts as far as possible.

Worst Allocation Demand

The worst allocation demand can be computed using the above parameters, the previous demand which is equivalent to previous sales, and the previous placement which consists of the locations and amount of space in these locations. Using the demand function and inserting the previous placement and previous demand one can now solve the equation for the only missing variable, namely the worst allocation demand.

Previous sales are part of the data provided by Globus which also contains a mapping between products and the boards they were placed upon together with the amount of facings on that board. The previous sales value was averaged over the last three months to remove daily variation. Naturally, some products might be sold disproportionately often on some days of the week, however, as shelves are not reallocated on daily basis these effects should be ignored. Unfortunately, for some products the previous demand or the previous placement was missing. If this is the case, the average worst allocation demand gets used instead.

5.3 Program Interaction

5.3.1 GUI

The three relevant parts of the GUI can be seen in figure 5.1: the top left part is the main menu of the IRL software where a button starting FrAPP's problem specifying GUI is located. This GUI can be seen in the right part and is explained

in further detail in section 5.3.1. The bottom left part allows the user to navigate through 3D models of supermarkets existing in reality. As this possibility was given, the user interface was designed such that it interacts with the 3D scene: boards get selected by clicking on them and the resulting solution is directly placed in the 3D model.

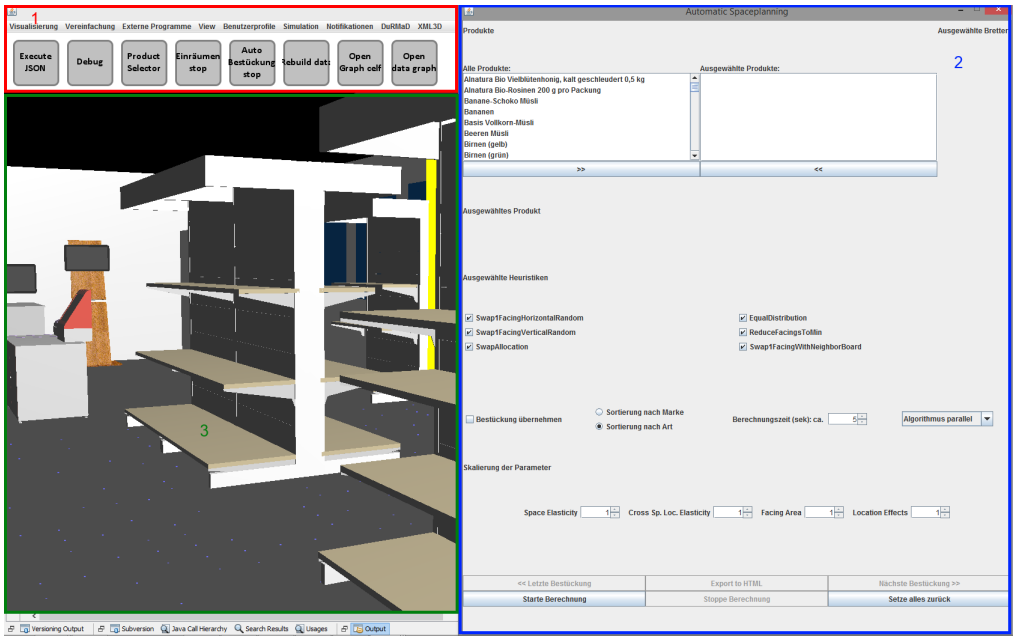

Figure 5.1: FrAPP's GUI included in the IRL software with overlays for the different sections.

User Specifications

Figure 5.2 shows the GUI for specifying the requirements of the solution. At first, the user can select the products that should be placed out of a catalog of all available items (see 1). As already mentioned the boards which should be filled get selected by simply clicking on the virtual representation in the 3D supermarket. All selected boards are then displayed in section 2 of figure 5.2.

Even though products and boards are already sufficiently defining a problem instance the user has the possibility to specify the requirements of a solution requirements in more depth.

Figure 5.2: The UI for specifying the concrete shelf space allocation problem.

In section 3 of the GUI the user can specify product dependent requirements. For one he can set lower and upper bounds on the amount of facings as well as an upper bound on the vertical stacking. As mentioned earlier these facing bounds can have different reasons like for example contracts with manufacturers to place their products at least x times [12]. Each intermediate solution will then be checked to fulfill these given bounds in the algorithm. The stacking bounds can be used for aesthetic reasons, since some products should not be stacked more than e.g. twice. Naturally, products will then be stacked at most as often as specified and the specific stacking gets used in the facing area computation

which directly influences the demand. If no stacking bound is given, products get stacked as often as they fit in the height of the board. Furthermore, a trend can be defined for each product which can be used to insert expected sales due to out-of-store factors. If e.g. the person planning the shelf expects an increase in sales because a lot of promotion was made or, as another example, if the Easter season is starting it is known that the sales in eggs rise for a specific percentage, this can be inserted into the program and will be considered in all further computations. Furthermore, the user can specify discounts the retailer gets if a given amount of products is purchased. Therefore, he simply enters the amount of products needed and the percentage discount. Upon rating a placement the algorithm then checks if the products behind the front are at least as many as the volume needed and if this is the case deducts the discount from the purchasing costs. As already argued products in the front should never disappear since this would result in a smaller demand and therefore lost sales. Therefore, the maximum the retailer can purchase at a time is the amount behind the front which is the reason for the above computation. Due to such volume discount specifications the program is able to detect not only which placement gains the most income, but also considers that one placement might cost less than another because the retailer gains more discounts.

Despite specifying product-specific parameters, the algorithm can also be trimmed generally. The heuristics getting applied can be activated or deactivated (see 4), the computation time can be specified, it can be chosen whether to sort by brand or type, the concurrency mode can be selected and it can be chosen whether products currently located on the boards should be removed or if the solution should be build around them (see 5) and different parameters influencing demand can be scaled differently (see 6).

Even though the user can choose the heuristics, some used heuristics can also get post-selected when the algorithm starts: it is useless to include the heuristic switching facings between horizontally neighboring boards if there is no such neighbor. Therefore, in all these cases where heuristic become useless due to the specific setting, the superfluous heuristics get deactivated automatically to use the computation time for meaningful heuristics instead.

The computation time can be changed depending on how much time the user has available. Naturally, longer computations should result in higher quality solutions, however, the resulting gains should be diminishing, meaning that 5 instead of 2 seconds improves the result way more than 110 instead of 100 seconds. The given computation time only specifies the time the simulated annealing algorithm itself can use and does not contain the initialization or sorting time. The reason for this is that e.g. the sorting time cannot be computed in advance and therefore it cannot be reserved for the end. Furthermore, only the time used by the simulated annealing procedure directly influences the quality of the result.

Furthermore, the user can place products on the shelves manually and then start the shelf planner on the remaining space. This can be useful if the retailer has merchandising standards which the shelf has to fulfill but wants to use the remaining space with the remaining products as profitable as possible. Internally, the algorithm then marks the used space as filled and only allocates products to the remaining space.

Parameters can be scaled to give the user more flexibility: if in the user's opinion the space elasticity is not influencing the placement enough, he can choose to scale it up.

Section 7 of figure 5.2 shows a button panel. At any point in time it is possible to stop the computation which allows a user noticing failures in his configuration to quickly fix them without having to wait till the end. After successful termination the program offers a solution set from which the user can choose the final result by clicking on the next and previous placement buttons. The offered solutions are the ten best solutions found in the process (in parallelization mode parallel runs the sets of solutions from all threads get merged to one solution set). However, since sorting changes the horizontal product order, solutions might coincide after they are sorted which tightens the solution set due to duplicate elimination. Once the user finally selects a solution he can export it to a HTML website as described in section 4.5. Furthermore, it is possible to reset the whole configuration to the default one and remove all placed products again.

Visualization of Results

The resulting placement is visualized by placing 3D models of the products into the shelves in the virtual supermarket. This method also allows viewing the shelves from the side and therefore see the stacking in depth which would be invisible in a 2D visualization. Often users need to consider the products in neighboring shelves when selecting the items the software should place within a particular shelf. Here, the possibility to navigate through the scene allows them to easily grasp the orderings of the neighbors by simply moving the camera in the 3D scene. Figure 5.3 shows a screen-shot of stocked boards.

Feedback on Algorithm Performance

The simulated annealing based hyper-heuristic algorithm as described in the previous chapter is highly non-deterministic as it uses lots of randomness. Therefore, a chart depicting the performance of the algorithm is shown upon termination to proof that despite all this randomness the algorithm performed as expected. Otherwise the user would simply have to believe that a close to optimal solution was found without any visualization of its performance.

The chart depicting the performance of the algorithm was created using JFreeChart, a Java library for easily creating high quality charts. Upon termination of the

Figure 5.3: A stocked shelf in 3D view.

simulated annealing algorithm a line chart is shown, having a counter of all valid solutions on the x-axis and the rating corresponding to this solution on the y-axis. Naturally, placements violating some requirements are excluded from the chart. As the rating corresponds to the expected sales, the chart basically maps placements, ordered by their appearance in the algorithm, to expected profits achieved by using these placements. Two exemplary charts can be seen in the figure 5.4.

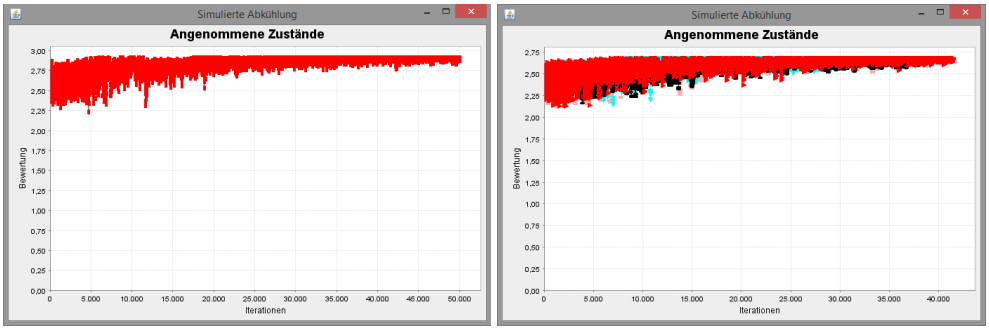

Figure 5.4: The performance graph for the simulated annealing procedure. Left image in parallelization mode SINGLE_THREADED and right image in mode PARALLEL_RUNS.

If the parallelization mode is PARALLEL_RUNS different lines for each thread are drawn as can be seen in the second image. In the simulated annealing algorithm it is likely to transition to worse neighbors in the beginning but then becoming less and less probable as the temperature controlling transitions cools off. Therefore, the graph should vary strongly in amplitude in the beginning

and then become more and more constant towards the end as it is the case in the exemplary graphs. Furthermore, since the optimization model is a maximization, the overall tendency of the graph should of course be ascending.

5.3.2 Export

Planograms are used to communicate the shelf layout between the planner who designed it and the employer who actually places the products. Therefore, an export functionality generating such planograms is a core functionality. This export consists of a picture of the visualized shelf and two tables storing additional information. These three elements get exported as a HTML website to allow employees to access the data directly from the sales area.

For generating a virtual image a camera is placed in the 3D model of the supermarket and the currently rendered frame is saved as an image. In order to find the camera location and viewing direction to see all newly placed products in one picture three steps have to be processed. First, the midpoint of the filled boards is computed. Second, the vector vertical to the shelf has to be found since the final camera location is located upon the straight line defined by the midpoint and this vertical vector. Third, the distance of the camera to the midpoint on the straight line has to be computed. Since the camera should see all newly placed products, first the bigger dimension, width or height, has to be determined. Then using the geometry of an orthogonal triangle and a 45 degree angle of view camera the distance can be computed with the following formula:

$$distance = tan^{-1}(22.5\,^{\circ}) * 0.5 * Max(Width, Height) \tag{5.2}$$

This formula can be explained using figure 5.5.

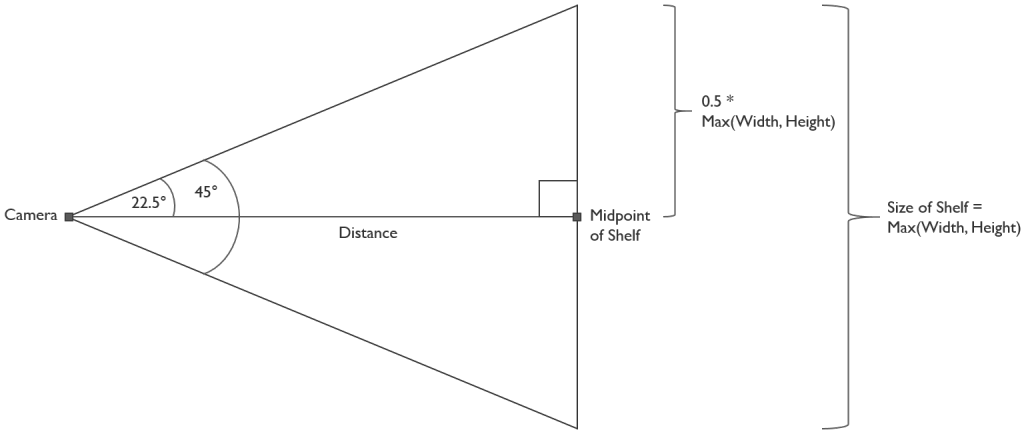

Figure 5.5: The orthogonal triangle used for the computation of the distance between shelf and camera.

Placing the camera in the above way ensures that all newly placed products can be seen while not being too far away from the shelf. An exemplary rendered image can be seen in figure 5.6.

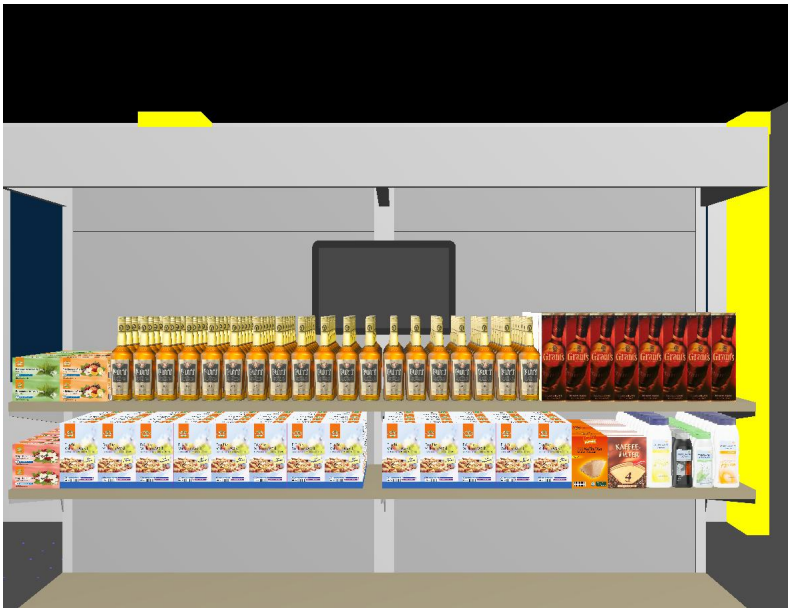

Figure 5.6: The image of the shelf used as a planogram.

Furthermore, two tables are part of the exported HTML website. They both show the information of the image like the names of the products and how often they are stacked. However, they also store additional information which the image does not offer: the EAN (European Article Number), the producer and the physical dimensions (width, height, depth). As the tables for the above example are large, only parts can be seen in figures 5.7 and 5.8. The complete tables can be found in appendix A. These two tables differ in their layout: table 5.7 is ordered according to the rendered image, that is, each row describes one board and the cells correspond to the facings. In contrast table 5.8 has one row per facing with one column for the board number and another one indicating the order position within the board. This second table layout is the one integrated in Globus' planograms at the moment.

Regal	Meter	BrettID	Facings												
6	1, 2	2	**Globus Hagebuttentee** Stacking: 2, EAN: 4304218716316, Hersteller: Ostfriesische Tee-Gesell., Breite: 0.16, Höhe: 0.074, Tiefe: 0.066	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074
6	1, 2	3	**Globus Brennnesseltee** Stacking: 2, EAN: 4304218716354, Hersteller: Ostfriesische Tee-Gesell., Breite: 0.16, Höhe: 0.074, Tiefe: 0.066	**Globus Früchtetee** Stacking: 2, EAN: 4304218716330, Hersteller: Ostfriesische Tee-Gesell., Breite: 0.16, Höhe: 0.074, Tiefe: 0.066	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077

Figure 5.7: The planogram table according to product placements in the rendered image.

Regal	Meter	BrettID	Platzierungsnummer	Name	Stacking	EAN	Hersteller	Breite	Höhe	Tiefe
6	1	2	1	Globus Hagebuttentee	2	4304218716316	Ostfriesische Tee-Gesell.	0.16	0.074	0.066
6	1	2	2	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	1	2	3	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	1	2	4	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	1	2	5	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	1	2	6	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	1	2	7	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	1	2	8	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	1	2	9	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	2	2	10	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	2	2	11	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	2	2	12	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	2	2	13	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	2	2	14	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	2	2	15	Kaffeefilter Größe 2	1	2000088958154	Kaffeefilterfabrik	0.122	0.173	0.042
6	2	2	16	Kaffeefilter Größe 4	1	2000088958178	Kaffeefilterfabrik	0.132	0.173	0.042
6	2	2	17	Natuvell Shampoo Vital 5	1	4304218603050	Otto Cosmetics Gmbh	0.09	0.235	0.045
6	2	2	18	Natuvell Pflegedusche Energy for me	1	4304218603166	Otto Cosmetics Gmbh	0.07	0.185	0.04
6	2	2	19	Natuvell Pflegedusche Aloe Vera	1	4304218603135	Otto Cosmetics Gmbh	0.07	0.185	0.04
6	2	2	20	Natuvell Shampoo Vitamin	1	4304218603043	Otto Cosmetics Gmbh	0.09	0.235	0.045
6	2	2	21	Natuvell Cremedusche Rose & Joghurt	1	4304218603210	Otto Cosmetics Gmbh	0.07	0.185	0.04
6	1	3	1	Globus Brennnesseltee	2	4304218716354	Ostfriesische Tee-Gesell.	0.16	0.074	0.066
6	1	3	2	Globus Früchtetee	2	4304218716330	Ostfriesische Tee-Gesell.	0.16	0.074	0.066
6	1	3	3	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	4	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	5	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	6	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	7	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	8	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	9	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	10	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	11	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077

Figure 5.8: The planogram table ordered as currently used by Globus.

Chapter 6
Evaluation

6.1 Overview and Setup

Overall three different problem instances were evaluated: two small problems upon which the different simulated annealing algorithms could be compared to the optimum achieved using a brute force procedure and one bigger problem, for which brute forcing was unfeasible. Nevertheless, one could easily compare the performances of the four different algorithms on this bigger problem. All executions were performed on an AMD Phenom II X4 955 Processor, having 4 cores each at 3.2 GHz. Naturally, only products with complete data were used. Unfortunately many products were missing data about their previous locations, previous sales or their prices in which case FrAPP uses average values instead. However, as I wanted to perform the evaluation on scenarios being as realistic possible, this missing data limited the possible test cases dramatically. Due to the huge amount of randomness in the simulated annealing based hyper-heuristic procedure the program was executed multiple times in each concurrency mode and given amount of time. Here one should notice that running the algorithm with an inserted computation time of 10 seconds is not equivalent to running it for 20 seconds but stopping after 10 seconds. This inequality is a result of the temperature function which includes the overall allowed time (see section 4.1). Therefore, the temperature decreases in a different speed in the above two cases which results in a different likelihood of worse neighbors being chosen. Furthermore, the brute force algorithm used for comparison was optimized such that it only considers solutions where the whole shelf is filled and therefore only needs to check consistency if this is the case. This is reasonable since less filled shelves cannot be more profitable than completely filled once.

6.2 Problem Instance 1

The first problem consisted of only one single board which had to be filled using 6 products: Whiskey, three different kinds of tea of the same brand, coffee filters and cookies. Brute forcing the solution took 96.98 seconds and resulted in the placement illustrated in figure 6.1. One can clearly see that the most valuable

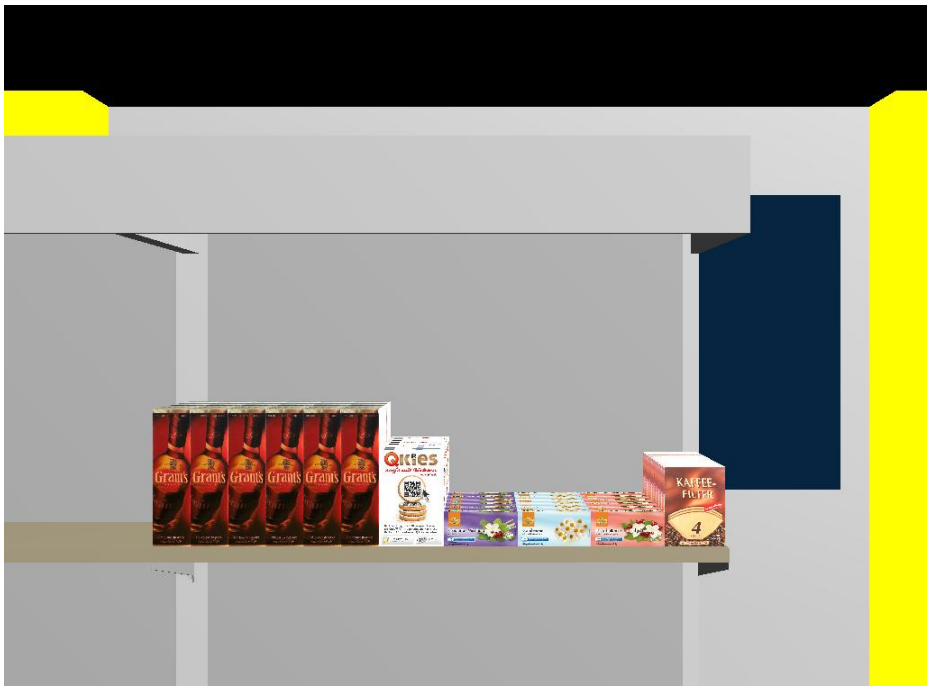

Figure 6.1: The optimal solution of problem instance 1 found using a brute force algorithm.

product, namely the whiskey, got the most space assigned while all other products only had space for exactly one facing.

In order to compare the four simulated annealing algorithms to the brute forced solution, each concurrency mode was executed multiple times with different allowed computation times: each mode was executed three times in each 2, 5 and 10 seconds of allowed time. As one can easily see in table 6.1, the single threaded simulated annealing procedure as well as the parallel runs concurrency mode outperformed the other two: within only 2 seconds both of the above algorithms achieved the maximum in three of three runs. The parallel heuristics concurrency mode only found the optimum once given 2 seconds of computation time and achieved 98.64% of the optimum in the other two runs. However, with an increased computation time of 5 seconds the optimum was found in all three test runs. The worst algorithm was the parallel neighbors concurrency mode which needed 10 seconds to achieve the optimum in three of three runs. However, even this worst algorithm already achieved 98.64% in all three runs

given only 2 seconds of computation time which is only 2% of the time the brute force algorithm needed.

Algorithm	Single	Parallel Runs	Parallel Heuristics	Parallel Neighbors
Time needed	2	2	5	10

Table 6.1: Results of problem instance 1: time needed by each algorithm to achieve the optimum in 3 of 3 runs.

The parallel heuristics and parallel neighbors modes always transfer to the best placement found by any thread. The best neighbor of all threads is with high probability better than taking a single generated neighbor. Therefore, the algorithm worsens less with one of the above parallelization modes than with the parallel runs or single threaded versions. Keeping this in mind, a possible explanation for the differences in performance between the single threaded or parallel runs and the other two modes is that one might have to worsen strongly before finding the optimum. However, this is less likely with parallel neighbors or parallel runs and therefore more time is needed to achieve the optimum. Naturally, if the single threaded version performs well the parallel runs mode also performs well as it simply executes the single threaded version in parallel.

6.3 Problem Instance 2

The second problem consisted of two vertically neighboring boards and three different kinds of tea of the same brand. Due to the exponential computation time of the brute force procedure, problem instances with more or smaller products were not testable. Even on this still pretty small problem the brute force procedure needed 744.03 seconds. The four simulated annealing procedures were each executed three times with an allowed computation time of 2, 5, 10 and 20 seconds. Every single run achieved the maximum three times in a row, no matter which algorithm or computation time was selected. This means that all algorithms were able to find the optimum in only 0.26% of the time the brute force procedure needed.

Figure 6.2 shows two optimal solutions for this problem instance. Overall six different solutions were found which all achieved the same evaluation value as the brute force procedure. One fact that all solutions have in common is that the fruit tea (the tea on the bottom right in the right image) has only one single facing. However, this is logical as fruit tea has the smallest worst allocation demand of the three teas. This can also be seen as an explanation why all algorithms were able to find the optimum in 2 seconds which was not the case in problem instance 1: the probability to find one optimum of six is of course higher than finding the only optimum. A possible reason why so many optimums exist is that three products with the same dimensions and profit margins were used.

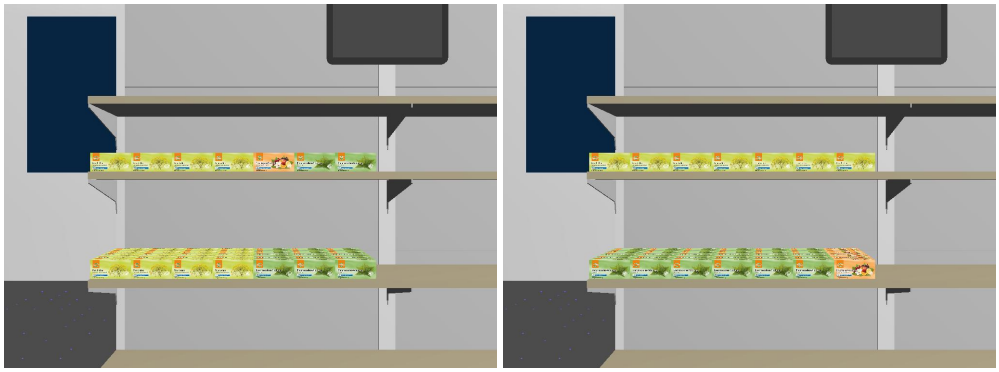

Figure 6.2: Two optimal solutions for problem instance 2.

6.4 Problem Instance 3

The third problem is significantly more complex than the previous two: it consists of 4 boards neighboring horizontally and vertically and 15 products of different categories and sizes: a muesli, Whiskey, Rum, 4 kinds of tea, 5 different shower lotions or shampoos, cookies and 2 different types of coffee filters. An exemplary placement can be seen in figure 6.3. One can clearly see that the expensive spirituous beverages are all placed on the higher board and have really many facings. Furthermore, the muesli has relatively many facings as well as it is has the highest worst allocation demand of all products and no competitor product decreasing the demand. The fennel tea (second from the left) clearly dominates the other teas as it has the highest worst allocation demand of these four.

As this problem instance is way to complex to brute force, the solutions achieved by the four simulated annealing algorithms could not be compared to a known optimum. Instead they were compared against each other. Overall 44 runs were performed, each algorithm twice with a computation time set to 600 seconds, and three times each with 5, 20 and 60 seconds of computation time. As expected, the maximum was found in a 600 seconds run, which was the longest tested amount of time. However, it was found using the parallel heuristics concurrency mode which is surprising as this algorithm performed worse than the single threaded version or the parallel runs version upon problem instance 1. On average over all runs with the same algorithm and computation time every combination achieved at least 98% of the best known solution. Furthermore, there was no single execution achieving less than 97.57% of the best known solution. Table 6.2 summarizes the results of the different algorithm-time pairs, where each cell contains the average over all runs in the corresponding configuration. The overall maximum is part of the average in the cell containing a "*".

As one can easily see, the parallel heuristic mode was the best algorithm if given lots of time, however, if only little time was at hand it was the worst. In contrast, the parallel runs concurrency mode performed best if given only a small amount

Figure 6.3: A solution for problem instance 3.

Time/Algorithm	Single	Parallel Runs	Parallel Heuristics	Parallel Neighbors
5	99.06%	**99.65%**	98.21%	99.18%
20	99.39%	**99.44%**	98.75%	99.02%
60	99.08%	99.24%	99.12%	**99.50%**
600	99.16%	99.41%	**99.98%** *	99.85%

Table 6.2: Results of problem instance 3: achieved percentage of best known solution averaged over three runs.

of time. Also, the parallel neighbors algorithm seems to perform better if given more time: it was the best algorithm for 60 seconds and the second best for 600 seconds. Not too surprisingly, the single threaded version never was the best, no matter how much computation time was selected. This seems logical as running the program in concurrency mode parallel runs on 4 cores is equivalent to taking the best result of 4 runs of the single threaded version. Furthermore, one should note that there is no clear improvement due to a bigger amount of time, which indicates that it is needless to let the algorithm run for a long time.

Nevertheless, as all these results are relatively equal no algorithm can be said to be definitely superior to all others.

6.5 Conclusion

All algorithms are able to achieve the brute forced optimum within significantly less time upon small problem instances, no matter whether only similar products are used (problem 2) or many different kinds of products (problem 1). As all algorithms were equally good on problem 2 but the single threaded and parallel runs version outperformed the others on problem 1 we have a small performance tendency towards those two algorithms. While still being relatively equal, the parallel heuristics and parallel neighbors modes seem to slightly outperform the parallel runs concurrency mode if given much time on a bigger problem. However, if given only little time the parallel runs concurrency mode slightly outperforms the other two. In general one can say that parallelizing the simulated annealing based hyper-heuristic leads to very small improvements only. Nevertheless, summing up even these tiny improvements of each board over the whole supermarket can lead to significant profit gains.

Chapter 7
Conclusion and Future Work

7.1 Summary and Conclusion

FrAPP aims to find optimal solutions for the shelf space allocation problem. In an intuitively usable GUI users can specify which products should be placed upon which boards of a supermarket. Here, minimum and maximum facing amounts, the maximum allowed stacking, volume discounts and trends can be specified for each product. For finding an optimal placement, the profit of the allocation has to be computable which, in turn, requires the knowledge of the expected demand per product in a placement.

FrAPP aims to define demand as realistic as possible while at the same time it only uses data which is easily accessible. Naturally, there is always a trade-off between those two factors, for example it seems logical that advertisement for one product might decrease the demand of a substitute product, however, getting concrete values for this effect is impossible. FrAPP's demand function contains the facing area, the space elasticity, the vertical position, the cross space location effects and a trend. As a scaling the worst allocation demand is used. Here, the facing area together with an approximated space elasticity and a location value are used analogously to the demand factors in the related works. The location value depends on the heights of the boards the product is allocated to, where the eye level of an average sized human is considered the best height. FrAPP makes the same assumption as Hwang et al. [10], namely that the staff pulls products to the front after a customer purchased a product from the front. This ensures that the shelf always looks fully stocked and therefore guarantees that the facing area stays the same. However, instead of including only the cross space elasticity into the demand function, as many researchers did, a new parameter called cross space location elasticity, which also reflects the effects of better positioning of one

product on a substitute product, is used. Furthermore, trends can be defined to insert expected gains or losses due to the weather, advertisement or the like. To reflect the effect that some products are more popular than others even if placed exactly the same way, most related works simply include a non-defined scale factor. However, one cannot simply use the market share as a scale factor as this would ignore the fact that more popular items tend to be placed more often and in better positions than less popular ones. For this reason the scale factor has to be determined fairly, meaning that it has to be neutral to the position and amount of space assigned. FrAPP's way to achieve such a fair scale parameter is to compute the demand of a product in the worst possible placement, namely being placed exactly once on the lowest board. This worst allocation demand can be computed from the previous sales and previous placement and therefore even includes regional differences.

The overall profit, defined as selling price subtracted by the purchasing costs and the taxes, is maximized. Here, valid solutions have to fulfill the following requirements: first, in order to avoid customer confusion due to random looking shelves, all instances of a product must be connected. Secondly, the overall demand of each product in the given replenishment interval must be lower than the amount of products in the shelf behind the front to avoid decreases in demand or even stock-outs. Thirdly, given minimum and maximum facing amounts must be satisfied, for example, in order to allow users to fulfill contracts with the manufacturer.

Since hyper-heuristics can adapt to changing problems such as the shelf space allocation problem, the optimization follows Bai and Kendall's [11] simulated annealing based hyper-heuristic approach. However, as FrAPP's optimization model is way more complex than Bai and Kendall's [11], some essential changes had to be made: due to the strong restriction of product connectivity, less random heuristics had to be used to prohibit solutions from being invalid all the time. Furthermore, FrAPP does not use heuristics which become superfluous as the temperature decreases since these only steal valuable computation time from other heuristics. Therefore, six heuristics were implemented: three heuristics swapping a facing with a direct neighbor, a neighbor on a horizontally or vertically neighboring board, a heuristic swapping the whole allocation between products, a heuristic distributing space equally between two products and lastly, a heuristic reducing the amount of facings of a product to the minimum and filling the gaps with the neighbors.

Additionally to a single threaded version of the simulated annealing based hyper-heuristic three different concurrency modes were implemented. Furthermore, found solutions are sorted horizontally by brand or type as this does not change the evaluation value according to the demand function but improves customer service dramatically.

In the end of the algorithm, a set of solutions is returned from which the user can select his favorite. He can then choose to export a planogram as a web page

which includes an image of the stocked shelf and two tables containing additional information about the solution.

An evaluation was performed to see if the different parallelizations of the simulated annealing based hyper-heuristic are able to find the optimum and which algorithm is the best. For easy settings it showed that all concurrency modes of the optimization algorithm are able to find the solution achieved by a brute forcing algorithm in a fraction of the computation time. Upon a more complex problem where brute forcing was unfeasible, the parallel runs mode outperformed the parallel heuristic and parallel neighbors mode if given little time, however, with more computation time these two performed slightly better than the parallel runs concurrency mode.

7.2 Future Work

As already argued in section 3.1.3 it seems logical that the cross space location elasticity should exist. However, a field study proving its existence and validating the used parameter estimations should be conducted.

Furthermore, interesting studies could be performed if more data for more products was at hand: one could compute the evaluation value of shelves in reality and compare it to the real profit achieved by the shelf. This would determine to which degree reality is reflected by FrAPP's demand model.

Another interesting aspect would be to transfer the concept of location values from vertical shelves to other forms. Freezers for example are often a simple cube from the floor to hip-height which can be opened from the top. Normally only the top side of freezers is transparent. On such kinds of product storage new location values have to be determined as there is no eye level. One should note that such freezers cannot be seen as upside-down shelves since the opaque sides reduce the vision to products placed at the borders dramatically, especially when the freezer is not completely stocked. Therefore, one could analyze which products are seen from the most places in the most pleasant angles of views and define location values. Naturally, one would also have to conduct a long-term field study analogously to Drèze et al. [2] to see not only which positions are the best but also how strong sales differ between these positions.

Lastly, one could extend the sorting which is performed after a solution is found to also be able to sort by color, size or other aesthetic criteria. It would then also be interesting to combine different sorting procedures like sorting by producer and size.

Appendix A
An Exported Placement

Regal	Meter	BrettID	Facings												
6	1, 2	2	**Globus Hagebuttentee** Stacking: 2, EAN: 4304218716316, Hersteller: Ostfriesische Tee-Gesell., Breite: 0.16, Höhe: 0.074, Tiefe: 0.066	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Früchte Vollkorn-Müsli** Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074
6	1, 2	3	**Globus Brennnesseltee** Stacking: 2, EAN: 4304218716354, Hersteller: Ostfriesische Tee-Gesell., Breite: 0.16, Höhe: 0.074, Tiefe: 0.066	**Globus Früchtetee** Stacking: 2, EAN: 4304218716330, Hersteller: Ostfriesische Tee-Gesell., Breite: 0.16, Höhe: 0.074, Tiefe: 0.066	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077

Figure A.1: The planogram table corresponding to figure 5.6 ordered according to product placements in the rendered image: part 1

Früchte Vollkorn-Müsli Stacking: 1, EAN: 4304218715357, Hersteller: Globus, Breite: 0.128, Höhe: 0.219, Tiefe: 0.074	**Kaffeefilter Größe 2** Stacking: 1, EAN: 2000088958154, Hersteller: Kaffeefilterfabrik, Breite: 0.122, Höhe: 0.173, Tiefe: 0.042	**Kaffeefilter Größe 4** Stacking: 1, EAN: 2000088958178, Hersteller: Kaffeefilterfabrik, Breite: 0.132, Höhe: 0.173, Tiefe: 0.042	**Natuvell Shampoo Vital 5** Stacking: 1, EAN: 4304218603050, Hersteller: Otto Cosmetics Gmbh, Breite: 0.09, Höhe: 0.235, Tiefe: 0.045	**Natuvell Pflegedusche Energy for me** Stacking: 1, EAN: 4304218603166, Hersteller: Otto Cosmetics Gmbh, Breite: 0.07, Höhe: 0.185, Tiefe: 0.04	**Natuvell Pflegedusche Aloe Vera** Stacking: 1, EAN: 4304218603135, Hersteller: Otto Cosmetics Gmbh, Breite: 0.07, Höhe: 0.185, Tiefe: 0.04	**Natuvell Shampoo Vitamin** Stacking: 1, EAN: 4304218603043, Hersteller: Otto Cosmetics Gmbh, Breite: 0.09, Höhe: 0.235, Tiefe: 0.045	**Natuvell Cremedusche Rose & Joghurt** Stacking: 1, EAN: 4304218603210, Hersteller: Otto Cosmetics Gmbh, Breite: 0.07, Höhe: 0.185, Tiefe: 0.04						
GUTE POTT 54% 0.70 L Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GUTE POTT 54% 0.70 L** Stacking: 1, EAN: 4001731152798, Hersteller: Henkell & Co., Breite: 0.077, Höhe: 0.285, Tiefe: 0.077	**GRANT`S WHISKY40% 0.7 L** Stacking: 1, EAN: 5010327000404, Hersteller: Diversa Spezialitaeten, Breite: 0.082, Höhe: 0.299, Tiefe: 0.073	**GRANT`S WHISKY40% 0.7 L** Stacking: 1, EAN: 5010327000404, Hersteller: Diversa Spezialitaeten, Breite: 0.082, Höhe: 0.299, Tiefe: 0.073	**GRANT`S WHISKY40% 0.7 L** Stacking: 1, EAN: 5010327000404, Hersteller: Diversa Spezialitaeten, Breite: 0.082, Höhe: 0.299, Tiefe: 0.073	**GRANT`S WHISKY40% 0.7 L** Stacking: 1, EAN: 5010327000404, Hersteller: Diversa Spezialitaeten, Breite: 0.082, Höhe: 0.299, Tiefe: 0.073	**GRANT`S WHISKY40% 0.7 L** Stacking: 1, EAN: 5010327000404, Hersteller: Diversa Spezialitaeten, Breite: 0.082, Höhe: 0.299, Tiefe: 0.073	**GRANT`S WHISKY40% 0.7 L** Stacking: 1, EAN: 5010327000404, Hersteller: Diversa Spezialitaeten, Breite: 0.082, Höhe: 0.299, Tiefe: 0.073

Figure A.2: The planogram table corresponding to figure 5.6 ordered according to product placements in the rendered image: part 2

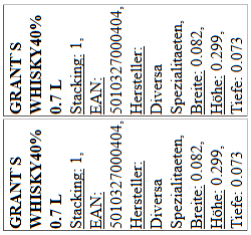

Figure A.3: The planogram table corresponding to figure 5.6 ordered according to product placements in the rendered image: part 3

Regal	Meter	BrettID	Platzierungsnummer	Name	Stacking	EAN	Hersteller	Breite	Höhe	Tiefe
6	1	2	1	Globus Hagebuttentee	2	4304218716316	Ostfriesische Tee-Gesell.	0.16	0.074	0.066
6	1	2	2	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	1	2	3	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	1	2	4	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	1	2	5	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	1	2	6	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	1	2	7	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	1	2	8	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	1	2	9	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	2	2	10	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	2	2	11	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	2	2	12	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	2	2	13	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	2	2	14	Früchte Vollkorn-Müsli	1	4304218715357	Globus	0.128	0.219	0.074
6	2	2	15	Kaffeefilter Größe 2	1	2000088958154	Kaffeefilterfabrik	0.122	0.173	0.042
6	2	2	16	Kaffeefilter Größe 4	1	2000088958178	Kaffeefilterfabrik	0.132	0.173	0.042
6	2	2	17	Natuvell Shampoo Vital 5	1	4304218603050	Otto Cosmetics Gmbh	0.09	0.235	0.045
6	2	2	18	Natuvell Pflegedusche Energy for me	1	4304218603166	Otto Cosmetics Gmbh	0.07	0.185	0.04
6	2	2	19	Natuvell Pflegedusche Aloe Vera	1	4304218603135	Otto Cosmetics Gmbh	0.07	0.185	0.04
6	2	2	20	Natuvell Shampoo Vitamin	1	4304218603043	Otto Cosmetics Gmbh	0.09	0.235	0.045
6	2	2	21	Natuvell Cremedusche Rose & Joghurt	1	4304218603210	Otto Cosmetics Gmbh	0.07	0.185	0.04
6	1	3	1	Globus Brennnesseltee	2	4304218716354	Ostfriesische Tee-Gesell.	0.16	0.074	0.066
6	1	3	2	Globus Früchtetee	2	4304218716330	Ostfriesische Tee-Gesell.	0.16	0.074	0.066
6	1	3	3	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	4	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	5	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	6	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	7	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	8	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	9	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	10	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	11	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077

Figure A.4: The planogram table corresponding to figure 5.6 in the order used by Globus: part 1

6	1	3	12	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	13	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	1	3	14	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	2	3	15	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	2	3	16	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	2	3	17	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	2	3	18	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	2	3	19	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	2	3	20	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	2	3	21	GUTE POTT 54% 0.70 L	1	4001731152798	Henkell & Co.	0.077	0.285	0.077
6	2	3	22	GRANT`S WHISKY40% 0.7 L	1	5010327000404	Diversa Spezialitaeten	0.082	0.299	0.073
6	2	3	23	GRANT`S WHISKY40% 0.7 L	1	5010327000404	Diversa Spezialitaeten	0.082	0.299	0.073
6	2	3	24	GRANT`S WHISKY40% 0.7 L	1	5010327000404	Diversa Spezialitaeten	0.082	0.299	0.073
6	2	3	25	GRANT`S WHISKY40% 0.7 L	1	5010327000404	Diversa Spezialitaeten	0.082	0.299	0.073
6	2	3	26	GRANT`S WHISKY40% 0.7 L	1	5010327000404	Diversa Spezialitaeten	0.082	0.299	0.073
6	2	3	27	GRANT`S WHISKY40% 0.7 L	1	5010327000404	Diversa Spezialitaeten	0.082	0.299	0.073
6	2	3	28	GRANT`S WHISKY40% 0.7 L	1	5010327000404	Diversa Spezialitaeten	0.082	0.299	0.073
6	2	3	29	GRANT`S WHISKY40% 0.7 L	1	5010327000404	Diversa Spezialitaeten	0.082	0.299	0.073

Figure A.5: The planogram table corresponding to figure 5.6 in the order used by Globus: part 2

Bibliography

[1] MURRAY, Chase C. ; TALUKDAR, Debabrata ; GOSAVI, Abhijit: Joint Optimization of Product Price, Display Orientation and Shelf-Space Allocation in Retail Category Management. In: *Journal of Retailing* 86 (2010), Nr. 2, 125-136. http://dx.doi.org/10.1016/j.jretai.2010.02.008. – DOI 10.1016/j.jretai.2010.02.008. – ISSN 0022–4359. – Special Issue: Modeling Retail Phenomena

[2] DRÈZE, Xavier ; HOCH, Stephen J. ; PURK, Mary E.: Shelf management and space elasticity. In: *Journal of Retailing* 70 (1994), Nr. 4, 301 - 326. http://dx.doi.org/10.1016/0022-4359(94)90002-7. – DOI 10.1016/0022–4359(94)90002–7. – ISSN 0022–4359

[3] ANDERSON, E. E. ; AMATO, H. N.: A mathematical model for simultaneously determining the optimal brand-collection and display-area allocation. In: *Operations Research* 22 (1973), Nr. 1, S. 13 – 21

[4] *Oxford Dictionary on Planogram*. http://www.oxforddictionaries.com/definition/english/planogram, . – Accessed: 2014-02-11

[5] CURHAN, Ronald C.: Shelf Space Allocation and Profit Maximization in Mass Retailing. In: *Journal of Marketing* 37 (1973), Nr. 3, pp. 54-60. http://www.jstor.org/stable/1249947. – ISSN 00222429

[6] HARIGA, Moncer A. ; AL-AHMARI, Abdulrahman ; MOHAMED, Abdel-Rahman A.: A joint optimisation model for inventory replenishment, product assortment, shelf space and display area allocation decisions. In: *European Journal of Operational Research* 181 (2007), August, Nr. 1, 239-251. http://ideas.repec.org/a/eee/ejores/v181y2007i1p239-251.html

[7] REYES, Pedro M. ; FRAZIER, Gregory V.: Goal programming model for grocery shelf space allocation. In: *European Journal of Operational Research* 181 (2007), Nr. 2, 634-644. http://EconPapers.repec.org/RePEc:eee:ejores:v:181:y:2007:i:2:p:634-644

[8] DESMET, Pierre ; RENAUDIN, Valérie: Estimation of product category sales responsiveness to allocated shelf space. In: *International Journal of Research in Marketing* 15 (1998), Nr. 5, S. 443–457

[9] COSKUN, Mehmet E.: Shelf Space Allocation: A Critical Review and a Model with Price Changes and Adjustable Shelf Heights. In: *Open Access Dissertations and Theses* (2012), January, S. 185–195

[10] HWANG, Hark ; CHOI, Bum ; LEE, Min-Jin: A model for shelf space allocation and inventory control considering location and inventory level effects on demand. In: *International Journal of Production Economics* 97 (2005), August, Nr. 2, 185-195. http://ideas.repec.org/a/eee/proeco/v97y2005i2p185-195.html

[11] BAI, Ruibin ; KENDALL, Graham: An investigation of automated planograms using a simulated annealing based hyper-heuristics. (2005), S. 87–108

[12] YANG, Ming-Hsien: An efficient algorithm to allocate shelf space. In: *European Journal of Operational Research* 131 (2001), Nr. 1, 107 - 118. http://dx.doi.org/http://dx.doi.org/10.1016/S0377-2217(99)00448-8. – DOI http://dx.doi.org/10.1016/S0377–2217(99)00448–8. – ISSN 0377–2217

[13] COX, K.: The Effect of Shelf Space Upon Sales of Branded Products. In: *Journal of Marketing Research* 7 (1970), Nr. 1

[14] ZUFRYDEN, F. S.: A dynamic programming approach for product selection and supermarket shelf-space allocation. In: *The Journal of the Operational Research Society* 37 (1986), Nr. 4, S. 413–422

[15] RUSSELL, RobertA. ; URBAN, TimothyL.: The location and allocation of products and product families on retail shelves. In: *Annals of Operations Research* 179 (2010), Nr. 1, 131-147. http://dx.doi.org/10.1007/s10479-008-0450-y. – DOI 10.1007/s10479–008–0450–y. – ISSN 0254–5330

[16] BORIN, Norm ; FARRIS, Paul W. ; FREELAND, James R.: A Model for Determining Retail Product Category Assortment and Shelf Space Allocation. In: *Decision Sciences* 25 (1994), Nr. 3, 359–384. http://dx.doi.org/10.1111/j.1540-5915.1994.tb00809.x. – DOI 10.1111/j.1540–5915.1994.tb00809.x. – ISSN 1540–5915

[17] URBAN, Timothy L.: An inventory-theoretic approach to product assortment and shelf-space allocation. In: *Journal of Retailing* 74 (1998), Nr. 1, 15-35. http://dx.doi.org/10.1016/S0022-4359(99)80086-4. – DOI 10.1016/S0022–4359(99)80086–4. – ISSN 0022–4359

[18] BAI, Ruibin ; KENDALL, Graham: A Model for Fresh Produce Shelf-Space Allocation and Inventory Management with Freshness-Condition-Dependent Demand. In: *INFORMS J. on Computing* 20 (2008), Januar, Nr. 1, 78–85. http://dx.doi.org/10.1287/ijoc.1070.0219. – DOI 10.1287/ijoc.1070.0219. – ISSN 1526–5528

[19] URBAN, Glen L.: A Mathematical Modeling Approach to Product Line Decisions. In: *Journal of Marketing Research* 6 (1969), Nr. 1, S. 40–47

[20] BUSSIECK, Michael R. ; PRUESSNER, Armin: Mixed-integer nonlinear programming. In: *SIAG/OPT Newsletter: Views & News* 14 (2003), Nr. 1, S. 19–22

[21] BONAMI, Pierre ; LEE, Jon ; LEYFFER, Sven ; WÄCHTER, Andreas: More branch-and-bound experiments in convex nonlinear integer programming. In: *Preprint ANL/MCS-P1949-0911, Argonne National Laboratory, Mathematics and Computer Science Division* (2011)

[22] BONAMI, Pierre ; BIEGLER, Lorenz T. ; CONN, Andrew R. ; CORNUÉJOLS, Gérard ; GROSSMANN, Ignacio E. ; LAIRD, Carl D. ; LEE, Jon ; LODI, Andrea ; MARGOT, François ; SAWAYA, Nicolas: An algorithmic framework for convex mixed integer nonlinear programs. In: *Discrete Optimization* 5 (2008), Mai, Nr. 2, 186–204. http://dx.doi.org/10.1016/j.disopt.2006.10.011. – DOI 10.1016/j.disopt.2006.10.011

[23] BONAMI, Pierre ; LEE, Jon: BONMIN Users' Manual (Version 1.7). (2013). https://projects.coin-or.org/Bonmin/browser/stable/1.7/Bonmin/doc/BONMIN_UsersManual.pdf?format=raw

[24] WOLPERT, D.H. ; MACREADY, W.G.: No free lunch theorems for optimization. In: *Evolutionary Computation, IEEE Transactions on* 1 (1997), Nr. 1, S. 67–82. http://dx.doi.org/10.1109/4235.585893. – DOI 10.1109/4235.585893. – ISSN 1089–778X

[25] BURKE, Edmund ; KENDALL, Graham ; NEWALL, Jim ; HART, Emma ; ROSS, Peter ; SCHULENBURG, Sonia: Hyper-Heuristics: An Emerging Direction in Modern Search Technology. 57 (2003), 457-474. http://dx.doi.org/10.1007/0-306-48056-5_16. – DOI 10.1007/0–306–48056–5_16. ISBN 978–1–4020–7263–5

[26] LUNDY, M. ; MEES, A.: Convergence of an annealing algorithm. In: *Mathematical Programming* 34 (1986), Nr. 1, 111-124. http://dx.doi.org/10.1007/BF01582166. – DOI 10.1007/BF01582166. – ISSN 0025–5610

[27] CAMPO, Katia ; GIJSBRECHTS, Els ; NISOL, Patricia: The impact of stock-outs on whether, how much and what to buy. (2000)

[28] SPASSOVA, Lübomira ; SCHÖNING, Johannes ; KAHL, Gerrit ; KRÜGER, Antonio: Innovative Retail Laboratory. In: *Roots for the Future of Ambient Intelligence. European Conference on Ambient Intelligence (AmI-09), 3rd, November 18-21, Salzburg, Austria*, o.A., 2009. – ISBN 978–3–902737–00–7

[29] STATISTISCHES BUNDESAMT: Mikrozensus - Fragen zur Gesundheit - Körpermaße der Bevölkerung. (2009). https://www.destatis.de/DE/

Publikationen/Thematisch/Gesundheit/Gesundheitszustand/Koerpermasse5239003099004.pdf?__blob=publicationFile

[30] EISEND, Martin: Shelf space elasticity: A meta-analysis. In: *Journal of Retailing* (2013), Nr. 0, -. http://dx.doi.org/http://dx.doi.org/10.1016/j.jretai.2013.03.003. – DOI http://dx.doi.org/10.1016/j.jretai.2013.03.003. – ISSN 0022–4359