Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Sami Kalkas

Simplified Shelf Problem as a Binary Linear Integer Programming Problem

Master's Thesis
Espoo, May 25, 2022

Supervisor: Senior University Lecturer Kerttu Pollari-Malmi, Aalto University
Advisor: Tuomas Viitanen D.Sc. (Tech.)

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Sami Kalkas		
Title:	Simplified Shelf Problem as a Binary Linear Integer Programming Problem		
Date:	May 25, 2022	**Pages:**	76
Major:	Computer Science	**Code:**	SCI3042
Supervisor:	Senior University Lecturer Kerttu Pollari-Malmi		
Advisor:	Tuomas Viitanen D.Sc. (Tech.)		

A planogram is a visual representation of a merchandising plan for the display of the available products in a store. Having the ability to create the most optimal planogram, based on some predefined criterion, can cut the costs generated by the manual labour related to the unpacking and placement of goods. One possible method that can be utilised for placing products into a fixture is a heuristic approach called simulated annealing. This technique does not necessarily yield true optimal results without verifying every combination, requiring vast computational efforts. Thus, some alternative solving method is required in order to provide a better planogram layout in a shorter time period. One such method utilised in this work, is called linear programming.

Linear programming is a mathematical modeling technique that is able to generate an optimal solution for a linear objective function subjected to various constraints, given that all are represented with linear relationships. Usually this approach requires polynomial computational complexity, meaning that it could generate a global optimum faster than a heuristic approach. With the use of linear programming the presented problems could be formulated in such a way, that either an optimal result or one greater compared to the heuristic approach could be obtained in considerably less time.

Many various models were tested and it was deduced that presenting the problems with solely binary variables (utilising binary linear programming) was computationally too exhaustive and was not a feasible way of solving the most complex of problems. Since planograms contain such a large number of possible configurations for product placements, an iterative algorithm was designed, relaxing the initial restrictions, which could yield up to a 30% result increase up to 70 times more efficiently compared to the simulated annealing approach. Though the achieved results were not necessarily optimal, a considerable improvement was achieved, which could provide a competitive edge for any supply chain company taking it into use.

Keywords:	linear, programming, binary, planogram, optimisation, simplex, branch-and-bound, lp, milp, blp
Language:	English

Acknowledgements

Throughout the writing of this thesis I have received a great amount of support.

First, I would like to thank Tuomas who took on the responsibility of acting as my advisor in this work.

Second, I am deeply grateful to my supervisor Kerttu that took on the responsibility of guiding me through the writing process. She took on the challenge of guiding me towards the finish line with a great deal of support. Having clear instructions on what needed to be worked on with a set schedule made it easy for me to finalise the last part of my studies.

Third, I would like to express my gratitude to Karen Pickup who, even during times when my work did not proceed as anticipated, reassured me that she is here to help and support me and always provided me with encouraging words.

Most importantly I would like to thank my wife, Siiri, who has unwaveringly supported me throughout this challenging time and who has encouraged me for the entirety of this work. Though this was a lengthy and bumpy road, I am lucky to have such a supportive spouse that no matter how anguished I felt at times, she was able to bring a smile to my face and help me take small steps in order to complete this thesis.

Espoo, May 25, 2022

Sami Kalkas

Abbreviations and Acronyms

LP	Linear Programming; A mathematical modeling technique for finding the optimal result for a linear objective function given a system of linear constraints.
MILP	Mixed Integer Linear Programming; A linear programming approach in which some or all of the variables are bound by integrality restrictions ($\in \mathbb{Z}$).
BLP	Binary Linear Programming; A linear programming approach in which some or all of the variables are bound by binary restrictions ($\in \{0, 1\}$).
NLP	Non-Linear Programming; A mathematical modeling technique for finding the optimal result for an optimisation problem in which some of the constraints or the objective function (or both) are non-linear.
B&B	Branch-And-Bound; An algorithm generally utilised for solving combinatorial optimisation problems, such as a Mixed Integer Linear Programming problem.
BFS	Breadth-First Search; A search algorithm traversing the tree-like search space one layer at a time (breadthwise). This algorithm explores all of the nodes at the current depth before proceeding to the next depth level.
DFS	Depth-First Search; A search algorithm traversing the tree-like search space by exploring as far as possible depthwise along an arbitrarily selected branch before backtracking. This algorithm explores all of the nodes in one branch before proceeding to other branches.

Contents

Chapter 1

Introduction

A display of goods offered by a vendor that are organised on a set of shelves provides a visual representation, which is referred to as a planogram. The product arrangement on a planogram can have impact in various factors, such as customer shopping experience or the generated costs for the vendor. This tool for visual merchandising exists in a three-dimensional space, consisted of width, height and depth. Presenting the placement of products in all three dimensions yields a lot of complexity, since such aspects as stacking, nesting and placement have to be taken into account. When trying to find an optimal planogram layout, a large proportion of this complexity can be ignored. This is because answers to questions such as how high a product can be stacked are determined by shelf dimensions and product configuration. Thus, this simplified shelf problem can be optimised in a single dimension by only taking into account the product and shelf widths.

Optimality is a specific (or possibly multiple) point in the solution space for a given problem. At such a point, the objective function's value cannot increase if maximised (or decrease if minimised) by modifying the problem's parameters. Hence, it is the best possible value that can be achieved. One method that can be utilised for finding the optimal result to an objective function is simulated annealing [31]. Though this metaheuristic procedure can generate promising results quickly, it has a drawback of requiring the visit to all result nodes in order to have certainty of a generated result being truly globally optimal. Depending on the problem formulation, linear programming can provide methods for solving such tasks in a smoothed polynomial [46] or exponential complexity [41]. Thus, computing every possible outcome is usually not necessary and certainty of optimality can be achieved in a shorter time than by a simulation. Any sort of improvement in both result and runtime can provide a large value for a supply chain company optimising planograms.

1.1 Problem statement

A fixture, consisting of a set of shelves, can contain a multitude of products that can be organised in a vast number of ways. More precisely, a product can be displayed multiple times on a shelf which is referred to as the number of facings placed for a specific product. Knowing which specific combination of product facings on the fixture's shelves provides the best outcome according to a defined function can result to an optimal planogram. In this work, the objective function's value comes from being able to place the ideal number of units on the shelf. The higher the objective function's result is, the less shelf breaches occur during the implementation period for a given number of units placed. A shelf breach has taken place in case some delivered product has not fitted onto the shelf and needs to be moved into storage. Thus, generating the most optimal planogram can help a company lower the required manual labour in moving products into storage, which can provide cuts in costs generated by deliveries.

The target of this thesis is to formulate a simplified shelf problem utilising linear programming. In this simplified formulation, the problem has been rendered from a three-dimensional to a one-dimensional problem where only the product positioning onto shelves is decided, thus considering only the width aspect of product placement. If this method is able to outperform the current approach of simulated annealing utilised at a company specialising in retail and supply chain optimisation in both resulting score and computational complexity, it could replace the current way of trying to build the most optimal planograms. In order to achieve this optimal result, we must find the best way of organising a set of ordered products, with an arbitrary yet restricted number of facings, on a set of shelves while conforming this process to some predefined rules.

1.2 Structure of the Thesis

In this thesis, we will first present the problem we are trying to solve in chapter 2. In the same chapter we will present different ways of solving it, containing the current approach utilising simulated annealing and the target method implemented with linear programming. For the linear programming approach, we will display various problem formulations containing variables conformed to specific restrictions, their time complexity analyses and the solving algorithm used to generate an optimal result. After expressing the background for this work, the environment and tools used for building the software capable of solving presented problems are conveyed in chapter 3.

In this chapter the programming language and all relevant packages used to apply the solving algorithm are presented. Next, in chapter 4, the data containing individual problems of varying complexities is presented followed by the experiments and results obtained by the linear programming method in comparison to the benchmark results achieved with simulated annealing. Finally, we will discuss our results in chapter 5 and compare them to the ones obtained with the simulative method followed by conclusive words in chapter 6.

Chapter 2

Background

During the past decade, business has become riskier due to shorter product life-cycles and outsourcing. In order for companies to thrive in this complex environment, cooperation with strategic partners within the supply chain is a must [53]. Supply chain management can be described as a strategic coordination of business functions and illustrates many aspects of a company, such as product and material flows [26]. Though this coordination is crucial, synchronising it can be rigorous, even within the organisation [24].

Though there are multiple areas where any supply chain company can improve their current way of operation, the focus of this thesis is in finding an enhancement within space planning. Organising shelves provides a visual representation of the products offered by a specific vendor, and this display of goods is often referred to as a planogram, which can be seen as a tool for visual merchandising. In space planning, one possible way of minimising costs is directly related to the number of delivered products. In case within a delivery there are products that do not fit onto a planogram, they need to be manually transferred into storage, which requires labour (transfer into and from storage) and augments the size of the inventory requiring additional space (such as a back-room or storage unit). Thus, minimising the number of products that are required to be placed into storage can provide a save in human labour, delivery costs and rent of a storage unit.

In this section, a short summary on the theoretical background of linear programming, which is the method utilised in this work, will be presented. This is followed by a look at the current process utilised within the company on an abstract level for solving the given issue. After understanding the simulation approach, we will see what sorts of benefits a new approach based on the Simplex algorithm could provide to the company. Finally, after understanding the benefits and drawbacks of the mixed integer linear programming method, we will take a look what the new problem formulation

looks like and why it could be considered as an alternative way of trying to find an optimal planogram.

2.1 Linear programming

Any industrial process, may it be food processing, the petroleum industry including everything from oil extraction to the distribution process or any military efforts, managing resources and finances are in a pivotal role. The common key concept of all these complex operations is the establishment of concrete actions to execute that when completed from a given initial status would move the system towards some target state as much as possible. Though the target states or complexities of a multitude of processes may widely vary, they still share similar structure when modelled in mathematical terms. Thus, in order to solve any process computationally, an algorithm capable of deducing the best possible outcome, an optimal solution, among all possible alternatives would need to be devised. [13]

Since a multitude of systems, such as industrial or financial, can be modelled or reasonably accurately approximated by a system of linear equalities and inequalities, the development of linear programming has seen a rise since its discovery and creation by George Dantzig in the late 1940s to early 1950s [12–14]. The revolutionary development of linear programming has provided the possibility to state general goals and outline a path of individual steps to perform in order to best achieve these set goals even in the most complex of situations. Thus, the three main tools for solving linear programming problems are the ability to formulate real-world problems mathematically with models, the techniques utilised in order to solve these mathematical models with carefully tailored algorithms and the possibility of executing each step of these algorithms with computers and software [13].

Prior to the invention of linear programming, there was no use of stating general goals for any planning systems due to the inability of solving them. Even though systems would have rules for restricting the objective itself, they would provide no help and only add confusion while trying to solve such problem. Thus attaining any plausible objective becomes the objective itself and the search for optimality can be discarded quickly from the model. This changed in the year 1947, when Dantzig first proposed the concept of linear programming. Ever since the proposal of this new concept, it has been widely used amongst academics. Though there has been a large amount of progress in the past decades, there still remains a lot of work to be done, especially for solving practical problems containing uncertainty. [13]

Linear optimisation is a process where the objective is to find the max-

imal (or minimal) value for some objective function denoted z, given some unknown variables $x_n, n \in \mathbb{N}$ subjected to some arbitrary constraints. A simple linear optimisation formulation can be seen in figure 2.1, where we try to maximise some function $2 * x_1 + 4 * x_2 = z$ with respect to x_1 and x_2, given four different constraints: $x_1 \geq 0$, $x_2 \geq 0$, $3 * x_1 + 2 * x_2 \leq b$ and $x_1 + x_2 \leq a$. The feasible region, highlighted in gray, contains all the valid solution to the problem that adhere to the presented constraints. The optimal solution will be at some extremity point, which is pointed out in figure 2.1 at the intersection of the objective function and two of the constraints. It is possible to move the objective function line in space in either direction by increasing or decreasing the value of z, but neither of these would be beneficial. If we lower the value for z, the result will be lower then the current optimal solution and in case it is increased, some constraints are violated making the solution non-acceptable.

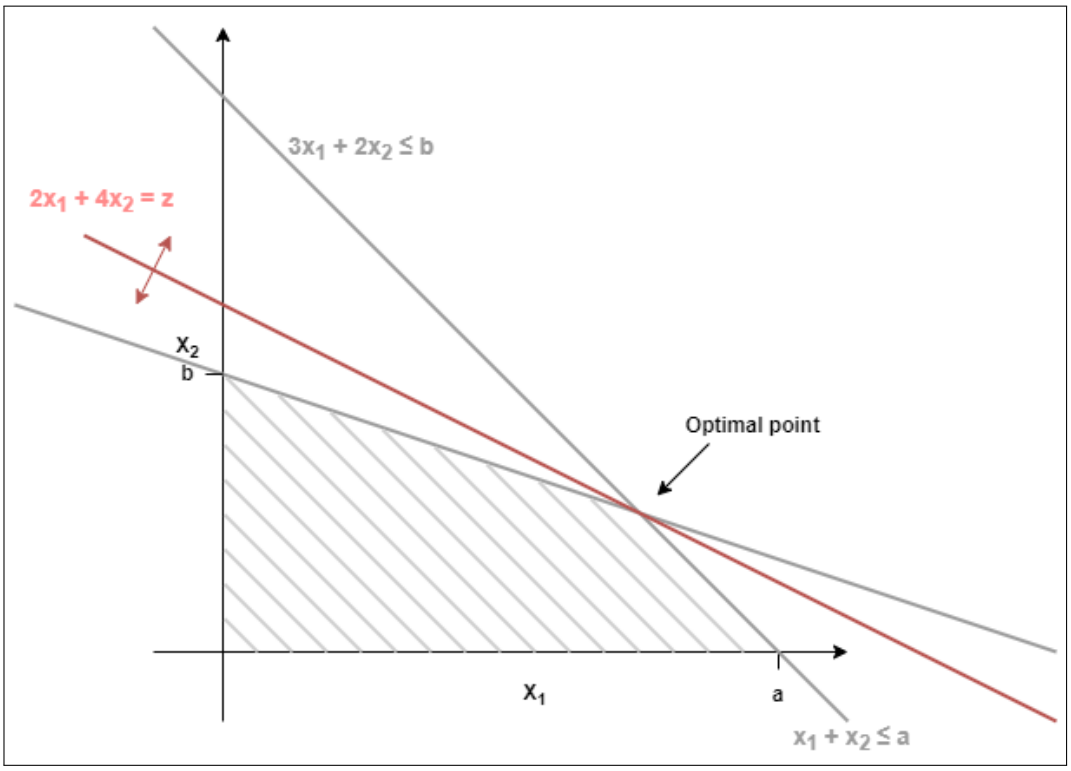

Figure 2.1: *Simple linear optimisation example*

Formally, the branch of mathematics for this sort of operations research deals with techniques that either maximise or minimise a given objective function subject to linear, non-linear and integer constraints subjected to the variables of the model. Linear programming belongs to this field of

mathematics since it targets to maximise/minimise a linear objective function where variables are subject to only linear equality and inequality constraints. [13] Though the problem statement is subjected only to linear constraints, it may and in our case will contain additional restrictions subjected to the variables, namely categorising them into mathematical sets, such as $\mathbb{R}$, $\mathbb{Z}$ or one containing only binary values, $\{0, 1\}$. The standard form of any linear programming problem can be expressed as

$$\begin{array}{ccccccccc} c_1x_1 & + & c_2x_2 & + & \cdots & + & c_nx_n & = & z\ (Max) \\ a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & \leq & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & \leq & b_2 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & \leq & b_m \end{array} \tag{2.1}$$

where the target is to find non-negative values $x_1, x_2, ..., x_n$ and $max(z)$ that satisfy the presented set of equations. This equation can also be presented in a canonical form as

$$\begin{array}{lccll} \text{Maximise} & c^Tx & = & z & \\ \text{subject to} & Ax & \leq & b, & A : m \times n \\ & x & \geq & 0 & \end{array} \tag{2.2}$$

During the 1950's, it was recognised that these sorts of linear programming problems were appearing in a multitude of situations. The Simplex algorithm was published in 1951 [18], proving to be a very effective tool in practice for solving these sorts of problems. This publication of the work started in 1947 by George Dantzig described a finite set of steps to be performed for optimising a linear objective function that is subject to a finite set of linear constraints. Though we are able to represent linear programming problems in a mathematical format as seen in equations 2.1 and 2.2, real world problems might require some additional restrictions to the presented variables $x_1, ..., x_n$. This realisation was also made early in the 1950's, when a significantly increased applicability of the Simplex algorithm was recognised in models containing some amount of variables bound by integrality constraints. Even with a rising need for algorithms taking this integrality constraint into consideration, there was no known modelling method in order to solve these types of problems utilising linear programming. Thus, no general approach was known for solving these types of problems containing some sort of integral constraints applied to at least some variables, called mixed-integer linear programming (MILP). After less then a decade after Dantzig's paper on the Simplex algorithm, Ralph Gomory published his own work in 1958 [20] which described straightforward modifications that could

be adapted to Dantzig's work such that any MILP problem could be, similar to LP problems, solved optimally with a finite amount of steps. His work showed that the Simplex approach could be utilised in order to create additional inequalities that would be valid for all generated solutions that would satisfy the mentioned integrality constraints but which would be violated by the current program's optimal solution. These inequalities, nowadays referred to as Gomori cuts, were studied greatly due to theoretical reasons and for the promise they provided as a computational tool [27].

These additional integrality constraints can be formulated by adding the following restrictions to some or all the variables in the variable vector presented in the equation 2.2 as

$$x_n \in \mathbb{Z}, n \in [1, ..., n]. \tag{2.3}$$

In case this integrality constraint would be applied to all of the variables in a given problem, it would modify the problem formulation presented in 2.2 into the equation 2.4. Due to the variables being bound to non-negative values, they can only get positive integer values (including zero).

$$\begin{array}{rrcll} \text{Maximise} & c^T x & = & z & \\ \text{subject to} & Ax & \leq & b, & A : m \times n \\ & x & \in & \mathbb{Z}^+ & \end{array} \tag{2.4}$$

Since the creation of an algorithm capable of cracking problems containing variables bound by integrality over 60 years ago, the MILP theory and practice has progressed significantly due to intensive development. This tool utilised with computational resources has become a crucial part in business and engineering sectors. The main reasons for this popularity are the availability of linear programming based solvers on the market in addition to the modelling flexibility provided by MILP. Though there are various advanced techniques included in the most powerful solvers on the market, they all base the solving process around the branch-and-bound (B&B) algorithm [32].

Most real-world optimisation problems share two key properties. First, the problems are easy to state mathematically and simple to comprehend. Second, though their solution space is finite, it is often quite large due to a vast number of decision variables. The branch-and-bound technique is a commonly used approach for solving large, $\mathcal{NP}$-hard optimisation problems [51]. Even though the B&B approach is an algorithm paradigm and all specific problem types must be addressed separately, principles on the B&B design have emerged in past years. Optimisation problems containing integrality constraints applied to its variables requires a powerful algorithm, for which the B&B paradigm suits well. Usually these $\mathcal{NP}$-hard problems have a vast

solution space where explicit enumeration is not a viable approach due to an exponentially increasing set of possible solutions. Thus, utilising bounds for the target value of the problem in addition to the current best solution makes it possible to search some parts of the solution space only implicitly [8].

The branch-and-bound approach is a systematic method and a general technique for solving MILP problems. This way of solving optimisation problems can in the worst case lead to exponential time complexities, thus making the solving process significantly slower to normal LP problems. Though it has a worse performance, when applied correctly, it can solve problems reasonably fast, until a certain extent. The core idea behind this method is to search for the optimal solution node that satisfies the given properties using some form of searching tree data structure. Some of the most common ones used in the B&B method are the breadth first search (BFS), depth first search (DFS) or best first search. In these approaches not all nodes are generally explored (including their descendants). This is because the upper limit of the objective function for any sub-node is the objective function value of their parent node. Thus, in case two nodes would have solutions adhering to all of the constraints, the one with the lower objective function value can be ignored (with its sub-space), since its descendants do not contain solutions surpassing its value. In addition, in case a node returns a non-feasible solution, the entire branch can again be ignored since its descendants would also have a non-acceptable solution. Hence, based on the optimal values obtained on the explored nodes it can be determined which ones should be further expanded to reach the optimal solution fulfilling all of the problem constraints containing any form of integral values. [8]

Even though the branch-and-bound method provides a powerful approach for solving MILP problems, it does not fully support a sub-category of the problem, called a binary linear problem (BLP). A binary linear problem differs from the presented equation 2.4 with one simple, yet crucial aspect. In a BLP, some variables must be bound to binary values, but it is not required for all of them to conform to this regulation. Nevertheless, in the problem of this thesis all variables are bound by the following binary restriction, that can be expressed as

$$x_n \in \{0, 1\}, n \in [1, ..., n]. \tag{2.5}$$

The BLP model is an $\mathcal{NP}$-complete problem and until this day there has not been any polynomial algorithms invented for solving these types of models [36]. As will become apparent later in this work, increasing the amount of binary variables in a BLP/MILP will cause the algorithm to slow

down. Eventually it will reach a point where some clever modifications to the data and the algorithm have to be made in order to generate a solution to the largest presented problems, within a feasible amount of time.

2.2 The problem

In the retail business, getting any sort of edge over your competitors, no matter how small, can prove to be vital. There are many areas where a competitive edge can be acquired: maximising sales and profit margins with cutting-edge optimisation software, increasing availability while minimising costs related to inventory management, automatic ordering for replenishment purposes and many others. Though many more areas within a company can be optimised and automatised, most are not visible to a customer making decisions on where and what to shop. These company wide cost saving decisions on different sectors are made in the background and can save the company money solely based on the processes used. Hence, having certainty of the optimality of any given process ensures that it cannot be further improved and that it is as profitable as possible.

A research on customer behaviour based on external stimuli was presented in 2020 [50]. This paper brings up multiple factors affecting customer satisfaction and how a positive experience leading to repurchases intentions. There are many factors within a store influencing the atmosphere that attracts customers to make purchases, such as the layout, lightning, music, aroma, space management, etc. Though there are various external factors having influence on a customer's repurchase decision, the area of focus for this work is in space management. Space management is a process that enhances the sales by managing the store's space adequately for facilitating customers. A well designed and informed store layout enhances the customer purchase decision and they will spend more time within a store. Within space management, planograms are representing how and where specific retail merchandise should be placed. [50] As will be described in more detail in later paragraphs, in this thesis we are trying to build planograms that minimise the number of shelf breaches.

In figure 2.2 is represented a simple planogram, in which three shelves containing a multitude of products with a different amount of facings can be observed. Facing is a term that is used to represent how many consecutive representations of a single product are used on a given shelf. As an example, on the first shelf there are three facings for the first product and two for the second one. It should be stated that in the context of this thesis this term is used to specify how many consecutive representations are used horizontally

and does not include any information for the other dimensions, such as how many can be stacked on top of each other (see first products on shelves two and three, which are stacked).

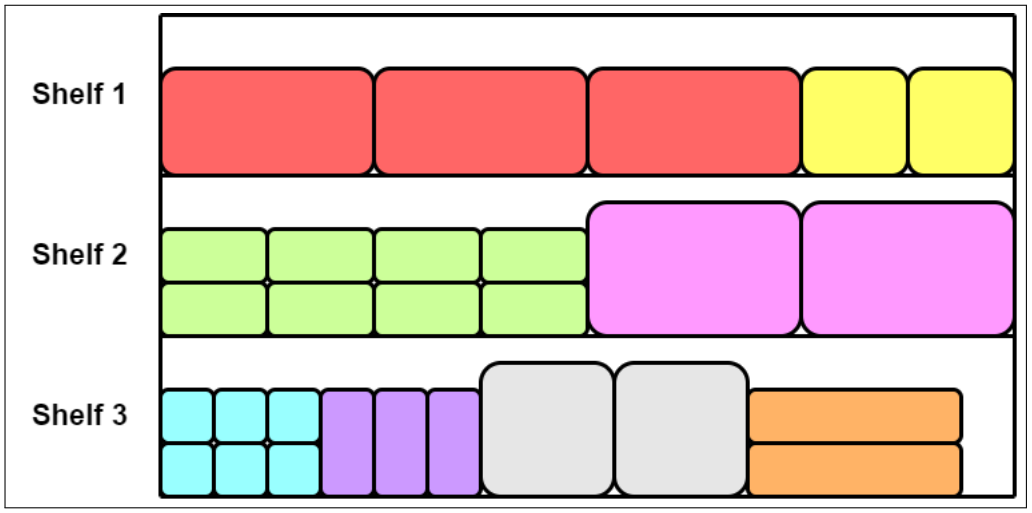

Figure 2.2: *Simple example of a planogram, containing shelves and products with various amount of facings*

Before looking deeper into the current approach utilised in order to find a local optimal value for any given planogram layout, presented in section 2.3, we can present the core of the problem as follows. Given a fixture F, which represents a physical element on which products can be placed (i.e. a set of shelves) and a finite set of products P, we can define a decreasing cost function c_p as

$$c_p : \mathbb{Z}^* \to \mathbb{R}^*, \ (p \in P) \tag{2.6}$$

and a minimum number of units of product p allowed on fixture F as

$$m_p \in \mathbb{Z}^+, \ (p \in P). \tag{2.7}$$

The cost function presented in equation 2.6 estimates the number of shelf breaches occurring during the implementation period for a given number of units placed. A shelf breach has taken place in case some delivered product has not fitted onto the shelf and needs to be moved into storage. This means the cost function tries to minimise the number of such occurrences, requiring additional manual labour to move the products into storage and back onto the shelf once some products have been sold. Thus, an optimal score out of this function is equivalent to minimising its resulting quantity. This function

will be used while defining the benefit values corresponding to the number of facings placed for a given product on a given shelf. The approach utilised for converting the cost function into a benefit is explained in more detail in section 2.6. In section 4.1 it will become apparent how this minimised cost quantity will be utilised in the linear programming approach in which we are trying to maximise an objective function value. With the properties defined in 2.6 and 2.7, we aim to minimise the sum of the costs c_p of all placed units (or facings) x_p as

$$\sum_{p \in P} c_p(x_p) \tag{2.8}$$

such that only whole units are to be placed (2.9) and the minimum constraints are complied with (2.10)

$$x_p \in \mathbb{Z}^+ \ (p \in P), \tag{2.9}$$

$$x_p \geq m_p \ (p \in P). \tag{2.10}$$

In addition to the restrictions affecting the cost function presented in equations 2.9 and 2.10, we must ensure that the products fit on a fixture. We first define the concept of a position to be a physical manifestation of a product on a shelf. We also make an assumption that each product can only be placed in one location on the fixture, thus resulting in each product having one position. This position includes a presentation, which is a specification for how a product is presented on a given component. Though the presentation for a product can include information in two dimensions (or three if depth is accounted for, though omitted in this work), such as how many facings of a product should be placed (horizontal) or if a product should be stacked (vertical), we can eliminate the vertical element from the optimisation process. Ignoring the vertical stacking of a product from the optimisation can be done due to it being determined entirely by the product configuration and shelf dimensions, thus having no impact in the result. Thus, the only decisions the optimisation process needs to make are the selection of shelf a position will be placed on as well as the amount of horizontal space this given position should occupy on the given shelf.

Since the width of a position is constrained by the width of the shelf it is placed on and possibly by a configured maximum number of facings, we can state that given a finite set of shelves S, there exists a finite and ordered set of possible presentations of product p on shelf s

$$P_{ps} \ (p \in P \ ; \ s \in S) \tag{2.11}$$

with the following properties persisting. The width of the i^{th} presentation in P_{ps}

$$w_{psi} \in \mathbb{Z}^+ \ (p \in P \ ; \ s \in S \ ; \ i \in P_{ps}), \tag{2.12}$$

the number of units or facings associated with the i^{th} presentation in P_{ps}

$$x_{psi} \in \mathbb{Z}^+ \ (p \in P \ ; \ s \in S \ ; \ i \in P_{ps}), \tag{2.13}$$

presentations are ordered by width

$$w_{psi} < w_{psj} \ (p \in P \ ; \ s \in S \ ; \ i < j), \tag{2.14}$$

and the wider the presentation of a single product on a particular shelf is, the more units are represented

$$x_{psi} < x_{psj} \ (p \in P \ ; \ s \in S \ ; \ i < j). \tag{2.15}$$

In addition to the properties 2.11 through 2.15 holding true, we have to impose the horizontal property of shelves in such a way that all products placed on a shelf must fit on it. Given the width of shelf s as

$$w_s \in \mathbb{Z}^+ \ (s \in S), \tag{2.16}$$

the set of products X_s is included in the union of the ordered sets of possible presentations (defined in 2.11) as

$$X_s \subset \bigcup_{p \in P} P_{ps} \ (s \in S) \tag{2.17}$$

and the i^{th} presentation in P_{ps} as

$$(p, s, i) \in P_{ps}, \tag{2.18}$$

we enforce that the products must fit on the shelf with the equation

$$\sum_{(p,s,i) \in X_s} w_{psi} \leq w_s \ (s \in S) \tag{2.19}$$

as well as the products must appear in order, meaning that product p must appear on that same or the previous shelf as product $p + 1$.

Now that we have defined the core of the problem, which is to minimise the sum of the costs of all placed units on a fixture satisfying certain properties, we can have a closer look on how a solution can be obtained to this problem. The solution is currently obtained with a local search metaheuristic approach called simulated annealing, which most likely results in a local optimum and in a best case scenario a global optimum.

2.3 Simulated annealing

Simulated annealing is a probabilistic method introduced by Kirkpatrick et al. in 1982 [31], making it a rather recent approach in comparison to LP solvers in the field of techniques utilised for optimisation problems. This proposed approach targets combinatorial optimisation problems such as finding an local optimum for a multivariable function [44].

This local search metaheuristic procedure is so named due to the process of physical annealing of solids, such as crystallines. In this physical procedure a crystalline solid is heated after which it cooled down gradually until achieving its minimum energy state in which its configuration is as regular as possible and where it is free of crystal defects. This thermodynamic behaviour is connected to the search of a global minimum of a discrete optimisation problem in the simulated annealing method, in which an algorithmic approach provides means to exploit such connection. [19]

Before introducing a pseudo-code presentation of this method, it is worth understanding how it works. This heuristic technique is an iterative approach often applied in discrete optimisation problems to return, instead of an optimal result, an acceptable one in a reasonable amount of time. During each iteration of this method, it compares the current solution to a newly selected one. While improved results are always accepted, the method accepts an inferior solution with some probability in order to escape a local minima in the search of the global one. The acceptance probability of the inferior solutions is defined by a temperature parameter, which is usually decreasing at each iteration of the process. Thus, the key feature of simulated annealing is the possibility to escape local optimum with hill-climbing moves that worsen the objective function value. While the number of iterations increases, the temperature variable decreases making the hill-climbing moves more infrequent. [44] Once the temperature variable reaches zero, the algorithm will return the best solution it came across during the process and not the one it ended at. Though this approach is heuristic and converges to a local optimum (that could also be a global one, in the best case), the benefit of it in practise is the possibility to control the amount of iterations, thus being able to control the runtime of the search process. One downside of this iterative method is the uncertainty of the optimality of the obtained result. Given enough time, it would be able to visit all local optima and certainty of the best objective function value would be obtained, but with a limitation on this property, we can only settle for a given outcome without concrete knowledge on the goodness of the result.

As mentioned prior, the approach currently utilised at a supply chain

company in order to find a suitable planogram is by utilising simulated annealing. In order to better understand how this approach works, we first define the solution space Ω that is the set containing all the possible solutions and the objective function defined on the solution space as $f : \Omega \to \mathbb{R}$. The goal is to obtain a global minimum $w^* \in \Omega$ such that

$$f(w^*) \leq f(w),\ w \in \Omega. \tag{2.20}$$

In addition, the objective function must be bounded to ensure that a global minimum exists. We also define $N(w)$, the neighbourhood function of $w \in \Omega$, such that these neighbouring solutions can be reached with a single iteration of a local search algorithm. [19]

The search process begins with an initial solution $w \in \Omega$. The next step is to generate a neighbouring solution $w' \in N(w)$ either randomly or with a pre-defined rule. Since the acceptance criterion can vary, we will only state that a candidate solution w' is accepted as the current solution based on the acceptance probability $p(w, w', t_k)$ as

$$P(\text{accept } w') = \begin{cases} p(w, w', t_k) & \text{if } \Delta_{w,w'} > 0 \\ 1 & \text{if } \Delta_{w,w'} \leq 0, \end{cases} \tag{2.21}$$

where

$$\Delta_{w,w'} = f(w') - f(w) \tag{2.22}$$

The last parameter to define is t_k, the temperature parameter at iteration k such that

$$t_k > 0,\ \forall k \tag{2.23}$$

and

$$\lim_{k \to \infty} t_k = 0. \tag{2.24}$$

This temperature parameter is given as an input to the acceptance probability function presented in equation 2.21. The more temperature there is left, thus the larger value t_k holds, the larger the acceptance probability is. This is because with the more time remaining in the simulation process, the larger number of possible states we want consider. Obviously, once the temperature lowers, we only want to explore the proximity of the current optimum to reach the best possible result within the given time.

With all these definitions in place, we propose the algorithm 1 below to represent the simulated annealing method. This given pseudo-code of

the simulated annealing approach results in $M_0 + M_1 + \ldots + M_k$ iterations in total, where k indicates the value of the cooling schedule t_k where the defined stopping criterion is met. Such stopping criterion can be a pre-defined number of iterations executed or a good enough solution being found.

Algorithm 1 Pseudo-code for a simulated annealing approach [19]

Set temperature change counter $k = 0$
Select initial solution $w \in \Omega$
Select temperature cooling schedule t_k
Select initial temperature $T = t_0 > 0$
Select repetition schedule M_k ▷ Iterations at each temperature t_k

```
repeat
    m ← 0                                          ▷ Repetition counter
    repeat
        Generate solution w' ∈ N(w)
        Calculate Δ_{w,w'}
        if Δ_{w,w'} ≤ 0 then
            w ← w'
        else if Δ_{w,w'} > 0 then
            w ← w' with probability p(w, w')
        end if
        m ← m + 1
    until m = M_k
    k ← k + 1
    Decrease T based on cooling schedule t_k
until stopping criterion
```

As said, we cannot know how good the obtained solution is compared to the global optimum unless we visit all local optima. Since the amount of computational resources is limited, we want to come up with a better approach for obtaining an optimal value for the objective function. This is where mixed integer linear programming might provide some assistance in generating a solution faster, resulting in a better objective function value or in the best case, fulfilling both of these criterion simultaneously.

2.4 Alternative approaches

Before implementing any algorithm capable of solving the given planogram problem, we performed a careful analysis of existing methods in order to find

the most suitable approach. As mentioned previously, the utilised approach should in the best case be able to find a better optimum than what is obtained by simulated annealing in a shorter amount of time. In case both criterion are not achievable, at least one of them should be met. It is worth noting that in case a better optimum is found, it has to be produced in a similar time frame as with the current approach, thus not being in a different order of magnitude.

The first step in selecting the desired approach is to understand what sort of restrictions have to be enforced in the model and how the problem could be formulated by utilising the selected method. The first distinction was made between linear (LP) and non-linear programming (NLP), of which the latter has one or more functions (objective and/or constraints) that are non-linear. Though both approaches seem like potential candidates for solving this problem, research on the topic seem to prefer the former in generating a solution faster [2].

One of the most important characteristics of a computer is the ability to perform repetitive operations efficiently. To be able to exploit this basic characteristic, most algorithms are built to solve problems in an iterative nature. Often programming problems are solved by selecting an initial solution vector x_0 and letting the algorithm produce an improved solution vector x_1. In this fashion, after each iteration of the algorithm we obtain an improved solution, yielding a sequence of ever-improving solutions $x_0, x_1, ...x_k, ..., x^*$. Though for linear programming problems the number of iterations is an unspecified number, it is repeated for a finite number of steps until an optimal result point x^* is obtained. The difference of the NLP approach compared to the LP one is that the sequence generally does not reach a solution point, but rather converges towards it. In practice this means that the search for an optimal point is terminated once a point is within a tolerable value of a solution point is reached. [34]

In addition of being different in the iterative process of generating the global optimum, the LP and a NLP methods differ also in their runtime complexities. A comparison between the two techniques in 3D conflict resolution of multiple aircraft revealed one key finding, though experimentally. Any given problem instance could be solved faster with MILP rather than with NLP. In addition to being faster, the gap between the runtimes increased as the problem grew in terms of variables and the oscillations were higher in runtimes for the NLP approach for larger problem instances. [6]

To support the experimental claims observed above, there exists literature and research papers backing up the longer runtimes for NLP. Though the complexity analysis for optimisation problems is difficult and distinctions between worst-case and average-case runtimes have to taken into consideration,

a linear programming algorithm utilising the simplex method usually takes polynomial time (though exponential in a worst case scenario) [35] whereas NLP optimisations are considered harder than linear ones [25] and often are not known to have any polynomial time algorithms [3].

Since research on runtime complexities would give the advantage for LP approaches instead of NLP ones, it was considered as the better alternative of the two to build the problem statement with. Complementing the research, two additional aspects further supported the selection of linear programming as the framework. The first was the ease of representing the problem as a (mixed integer) linear programming problem. As discussed more in detail in sections 2.6 and 4.1, we have variables representing the number of units of a specific product placed and a matrix containing corresponding benefit values to each positional variable, such that the objective function to be maximised and all the constraints are linear. Secondly, as will be described in chapter 3, there exists a wide amount of open source solvers supporting linear programming. Though there are nonlinear solvers of varying capabilities available, the linear solvers had better overall documentation, larger programming language variety and felt more robust in comparison to the nonlinear ones.

Based on all the various criterion mentioned in this section, choosing to utilise linear instead of nonlinear programming in order to solve the planogram problem was considered as the better of the two options available. Though a linear programming approach was selected, the next open question was what solver to choose. Since there exists a large variety of solvers using different algorithms, some thought had to be put in on this topic as well.

Many different algorithms exist for solving a linear programming problem. After some research on the topic, two different ones were considered mainly due their runtime complexities and their general availability in the existing solvers. The methods under consideration were Dantzig's simplex algorithm from the late 1940's (cited in [4]) and Karmarkar's algorithm from 1984 [1]. Since both methods generally run in polynomial time [21, 46] (more on simplex computational complexity in section 2.5), the decision came down to availability.

Even if the simplex algorithm has an exponential worst case computational complexity, it usually runs in polynomial time, which is why it is the standard algorithm for solving linear programming problems [7]. Since it is generally a fast approach for solving a system of linear equations, there are a vast amount of open source software available, in various programming languages. Even though Karmarkar's algorithm is a worthy alternative running in polynomial time, not as many solvers are available utilising it as are for Dantzig's simplex method.

Taking all this information into account, the decision to use a linear programming solver utilising the simplex algorithm was made, as will be discussed more in detail in chapter 3. Naturally, due to the nature of the problem statement, it was expected that the solver could handle binary and integer values in the problem variables and constraints, thus leading to a MILP problem.

2.5 Simplex

Dantzig's simplex method was selected to be the core linear programming algorithm used to solve the optimisation task of various planograms. The term simplex expresses some object in an n-dimensional space connecting $n + 1$ points. For example, in a one-dimensional space ($n = 1$), the simplex is a line segment connecting two points or in a three-dimensional space ($m = 3$), a simplex is a tetrahedron and its interior. The name of the Simplex method, which is related to linear inequality systems, can be described as the movement along the edges of convex polyhedral sets to achieve a global minimum of the objective function. This iterative search process ends in one of the following three outcomes: obtains a globally optimal solution (global minimum) for the objective function, creates a class of suitable solutions for which the objective function $z \to -\infty$ or concludes that the convex polyhedral set is infeasible and no solution exists. [14]

First, the distinguishing between the simplex method and the simplex algorithm has to be made. The simplex method starts with a linear program in a standard form and the simplex algorithm, which starts in a canonical form, forms the main subroutine of the simplex method by performing a sequence of pivot operations. The simplex method begins with the introduction of the problem statement in the standard form with the help of artificial variables in order to render the inequality equations into equality ones. The resulting auxiliary problem, which is in a canonical form, can be subjected to the simplex algorithm. In the first phase, a sequence of pivot operations are performed that produce a succession of different canonical forms. The objective of these operations is to generate a feasible solution, if such exists. In case the final canonical form yields such a solution, the algorithm is applied again in a second succession of pivot operations. The objective of these operations is to reach an optimal feasible solution such solution exists. [11]

In the following subsections the general concept of the Simplex algorithm will be presented, the computational complexity analysed and how the addition of integer variables modify the problem statement as well as the solving process.

2.5.1 Simplex algorithm

The Simplex algorithm is always initiated with a problem statement that has all of it's equations in a canonical form. Supposing such a problem consists of basic variables $x_{n+1}, x_{n+2}, ..., x_{n+m}$ and an objective function z, we want to find the values of

$$x_1 \geq 0, x_2 \geq 0, ..., x_{n+m} \geq 0 \text{ and } Min(z), \tag{2.25}$$

such that equations 2.26 and 2.27 are satisfied. [11]

$$\begin{array}{ccccccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & ... & + & a_{1n}x_n & + & x_{n+1} & & & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & ... & + & a_{2n}x_n & + & x_{n+2} & & & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & & \ddots & & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & ... & + & a_{mn}x_n & & & + & x_{n+m} & = & b_m \end{array} \tag{2.26}$$

$$(-z) \quad + \quad c_1x_1 \quad + \quad c_2x_2 \quad + \quad \ldots \quad + \quad c_nx_n \quad = \quad -z_0 \tag{2.27}$$

In these two equations a_{ij}, c_j, b_i and z_0 are constants and the canonical form of the basic solution is

$$z = z_0; x_{n+1} = b_1; x_{n+2} = b_2; \ldots; x_{n+m} = b_m; x_1 = x_2 = \ldots = x_n = 0. \tag{2.28}$$

Since the feasibility of the basic solution is assumed, the values of x_j in 2.28 are non-negative, thus resulting in

$$b_1 \geq 0, b_2 \geq 0, \ldots, b_m \geq 0. \tag{2.29}$$

Additionally, in case equation 2.29 holds, we can state that the linear program is presented in a feasible canonical form.

As shown, the canonical form of the problem statement can be used to provide an evaluation of the basic solution. In addition of determining whether a basic solution is feasible, the examination of the coefficients of equation 2.27 can be used to determine if the solution is minimal. The coefficients c_j in the objective function, also referred to as the relative cost factors due to their dependency of the basic variables, help us in this determination process. We can state that a basic feasible solution is a minimal feasible solution with an total cost of z_0 in case all of the relative cost factors are non-negative,

$$c_j \geq 0, \ j = 1, 2, ..., n \tag{2.30}$$

From the canonical form of the objective function, it is obvious that in case all the relative cost factors are non-negative, the smallest value of the sum

$$\sum c_j x_j \tag{2.31}$$

is zero for any choice of non-negative x_j. Furthermore, the minimum value of $z - z_0$ equals to zero and we can state that $Min(z) \geq z_0$. For the basic feasible solution, $z = z_0$ holds true, thus resulting in $Min(z) = z_0$. Based on these equalities it can be stated that the solution is optimal.

The canonical form of a problem can provide a way of testing the optimality of a basic feasible solution. In case this solution is not optimal, another solution is generated which has a reduced value of the objective function. This process of improving a non-optimal basic solution can be formalised as follows.

Given a non-optimal basic feasible solution, a new basic feasible solution can be constructed if at least one of the relative cost factors in the objective function 2.27 is negative under the assumption of non-degeneracy, $b_i > 0$ $(\forall i)$. This newly created solution will have a lower value of the objective function in case the value of a non-basic variable x_s is increased and the values of the basic variables are adjusted accordingly. For the selected non-basic variable x_s whose relative cost factor, denoted c_s, is negative, the index s can be selected in such a way that

$$c_s = Min(c_j) < 0. \tag{2.32}$$

The choice for s presented in equation 2.32 over any arbitrary j such that $c_j < 0$ is followed in most computational applications. This is due to it being the most convenient selection as well as being found leading in fewer iterations of the algorithm than with other selections of j.

In order to create a new feasible basic solution, we aim to eliminate the variable x_s from all other rows in equation 2.26 as well as the objective function of equation 2.27. This action (referred to as pivoting) consists of a number of elementary operations that replace a system with an equivalent system where a specified variable has a coefficient of unity in one equation and is zero elsewhere. The elimination of the variable x_s is the only necessary operation to reduce the system into a canonical form which is relative to the new set of variables. Since there is at most one equivalent canonical system with a fixed set of basic variable, the newly obtained solution is unique.

As was shown previously in this section, the Simplex algorithm produces a new feasible basic solution at each iteration. This newly obtained solution can be tested for optimality with the equation 2.32. In case the solution is

not optimal, we can select a new variable x_s with equation 2.32 to increase, thus producing one of the following two scenarios:

1) A class of solutions for which there is no lower bound existing for z in case all $a_{is} \leq 0$.

2) A new basic feasible solution for which the cost z is lower than for the basic feasible solution at the previous iteration. Only if the basic variables all have a strictly positive value can we be certain that the newly obtained value for z is smaller. Otherwise, it might be equal to the value of the previous iteration.

Thus, the Simplex algorithm repeats this cycle and terminates only when it has either constructed a class of feasible solutions for which z has no lower bound ($z \to -\infty$) or an optimal feasible solution where all $c_j \geq 0$.

We have to take into consideration the two possible cases for each iteration, degeneracy and non-degeneracy. Since it has been shown that both approaches terminate in a finite number of steps, we know that any method utilising the Simplex algorithm will terminate.

2.5.2 Simplex method

The aim of the Simplex method is to find values for $x_1, x_2, ..., x_N$ that simultaneously satisfy the system of equations

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & ... & + & a_{1N}x_N & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & ... & + & a_{2N}x_N & = & b_2 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ a_{M1}x_1 & + & a_{M2}x_2 & + & ... & + & a_{MN}x_N & = & b_M \end{array} \tag{2.33}$$

and minimise the objective function

$$c_1x_1 + c_2x_2 + ... + c_Nx_N = z, \tag{2.34}$$

where all the x_j $(j = 1, 2, ..., N)$ are restricted to be non-negative. The Simplex method utilises the Simplex algorithm presented in section 2.5.1 for solving the presented problem. [11]

Since no prior knowledge or assumptions can be made when starting to solve a linear programming problem, a general mathematical technique must be developed for solving any system of equations. Thus, the technique should be able to conclude that no feasible solution exists for problems where it might be the case without any prior knowledge. The Simplex method can be divided into two phases, of which the first one produces a starting feasible

canonical solution (if such exists) with the Simplex algorithm and the second one iterates towards a final optimal solution (or a class of solutions such that $z \to -\infty$).

Before presenting the outline of the outline of the procedure of the Simplex method, it is important to understand several key features of the first phase. Firstly, no assumptions can be made regarding the system. There is a possibility of it being unsolvable or solvable in non-negative numbers. Secondly, there is no need for eliminations in order to obtain an initial feasible solution in a canonical form. Lastly, the result obtained at the end of this first phase is a basic feasible solution (in case such solution exists) in a canonical form such that it can be utilised in the second phase of the approach.

The outline of the Simplex method procedure can be presented in the following simple steps.

1. Modify, where necessary, the original equations of the presented system in such a way that all the terms b_i are non-negative. This can be achieved by changing the signs on both sides of the equations.

2. Introduce a basic set of artificial variables $x_{N+1} \leq 0, x_{N+2} \leq 0, ..., x_{N+M} \leq 0$ so that the system can be presented as

$$\begin{array}{ccccccccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & ... & + & a_{1N}x_N & + & x_{N+1} & & & & & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & ... & + & a_{2N}x_N & + & x_{N+2} & & & & & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & & \ddots & & & \vdots & & \vdots \\ a_{M1}x_1 & + & a_{M2}x_2 & + & ... & + & a_{MN}x_N & & & + & x_{N+M} & & & = & b_M \\ c_1x_1 & + & c_2x_2 & + & ... & + & c_Nx_N & & & & + & (-z) & & = & 0 \end{array} \tag{2.35}$$

   where all b_i are non-negative and
   $x_j \geq 0$ $(j = 1, 2, ..., N, N+1, ..., N+M)$.

3. Utilising the Simplex algorithm presented in section 2.5.1, find a solution to the system of equations presented in 2.35 that minimises the sum of the artificial variables without restricting the sign of z as

$$x_{N+1} + x_{N+2} + ... + x_{N+M} = w \tag{2.36}$$

4. In case the value of $\text{Min}(w)$ is greater than zero, there is no feasible solution existing for the system and the process can be terminated. In case the value evaluates to zero, the second phase of the algorithm can be initiated.

5. Start the second phase of the Simplex algorithm in the case that $\text{Min}(w) = 0$. This phase applies the Simplex algorithm to the adjusted feasible canonical form in order to generate either a solution minimising z or a class of solutions where $z \to -\infty$.

As has been seen, the Simplex method utilising the Simplex algorithm is a finite process regardless if the problem has a valid solution of not. Throughout the presentation of the Simplex algorithm and method, we focused on systems that minimise the objective function whereas in the problem of this thesis, which we are trying to solve through linear programming, the quantity of the system is maximised. Since it is trivial to see that

$$\text{Minimise} \sum_{j=1}^{n} c_j x_j = \text{Maximise} \sum_{j=1}^{n} (-c_j) x_j, \tag{2.37}$$

any minimisation problem can be converted into an equivalent maximisation problem by simply changing the signs of all the relative cost factors in the objective function. [37] It is thus obvious that a similar transformation works when changing from a maximisation problem to a minimisation one.

2.5.3 Time complexity analysis

Now that we have seen that solving any linear programming system can be done with a finite number of steps, it is useful to understand how different aspects of a problem affect its runtime. Over the years, one of the standard ways to solve a linear programming problem

$$\begin{array}{cc} \text{maximise} & c^T x \\ \text{subject to} & Ax \leq 0 \\ \text{and} & x \geq 0 \end{array} \tag{2.38}$$

From a theoretical perspective, it has been customary to call an algorithm for a combinatorial problem good in case it can be solved within a polynomial number of steps regardless of the size of the instance. Categorising various algorithms with this rough definition helps differentiating between efficient algorithms and ones that require an exponentially increasing computational effort. Though the Simplex algorithm has been empirically observed to belong in this polynomial complexity category (and which is usually the case), the worst case of the number of iterations can formally be established as $\binom{n}{m} = \mathcal{O}(n^m)$. Thus, the bound for a worst case runtime would be exponential.

As was mentioned in the beginning of section 2.5, the Simplex method utilises a pivot rule that determines which variable is pivoted at each step. The selection of this pivot variable was presented in more detail in section 2.5.1 and in Dantzig's original formulation of the Simplex method, the non-basic variable with the most reduced cost as the pivot element is selected.

This pivoting rule has been shown to, in some cases, take an exponential computational time [17].

Based on the observations made above, it seems like the computational complexity of the Simplex method can be anywhere from polynomial to exponential and it depends on three different factors, which are the amount m of variables in the problem, the amount n of inequality constraints in the problem and the pivoting rule used to determine the pivot variable at each iteration. In the complexity analysis context, we can state that the input size of a problem is given by its dimensions n and m and the number of pivot operations indicate the performance of the problem [45].

As was discovered in section 2.5.2, the Simplex method is a finite process that can be used to solve a system of linear equations without cycling, thus not to visit any solution more than once. Since there is a finite number of bases the algorithm can visit without cycling, the upper limit for the number of iterations is bounded by an exponential function of the dimensions of the system. Thus far, it is not known if there exists a variation of the Simplex algorithm where the upper bound for the iterations would be bound by a polynomial function of the dimensions. Thus, having to call the Simplex algorithm as an exponential one does not make it justice. This is due to the lack of proof of it being exponential for every variant of the method, since the worst case analysis is not trivial in some cases. Most Simplex algorithms, including the one utilised in this work, can be described as the movement from one vertex of a polytope to an adjacent one, visiting each vertex at most one time. This method and its computational complexity has seen a lot of interest from people. The search for polynomial time algorithms grew in the field of linear programming algorithms around 1970s, trying to find dependencies between the number of pivoting steps and the dimensions n and m of a given problem. [35]

In order to tackle this open question on the computational complexity of the Simplex algorithm, we can introduce a new concept called smoothed analysis. This new hybrid approach of evaluating the worst and average case runtimes for an algorithm was introduced in 2004 in order to classify problems according to their underlying difficulty [5, 46]. The smoothed analysis approach measures the performance of a given algorithm with some minor random perturbations in the arbitrary inputs of a given problem. This approach mostly focuses on Gaussian perturbations of algorithms with real value inputs and measure the running times relative to the size of the input and the standard deviation of the perturbations.

Though the smoothed analysis does provide some conclusive results, the analysis still deserves further study. It was demonstrated that with a specific pivoting rule the computational complexity of the Simplex algorithm is poly-

nomial, though a large polynomial. For any other pivoting rules where the characterisation of the encountered vertices is determined by an iterative application of the pivot rule the probabilistic analysis is rendered difficult due to dependencies introduced by this characterisation process. Even though there is no conclusive evidence supporting that any pivoting rule would have a polynomial smoothed complexity, it would seem reasonable to think this would be the case knowing that it is polynomial for one specific rule. [46]

There are many tricks that can be tried when attempting to lower the upper bound of the Simplex method, but it seems unlikely that it could be lowered from an exponential complexity [45]. Though we do not have conclusive evidence whether the examples in our work run in exponential or polynomial time, it has been empirically shown that the algorithm is efficient, usually running in polynomial time, $\mathcal{O}((m+n)*m)$, where m represents the variables in the problem and n the number of inequality constraints in the problem [16].

Additionally, as will be seen in more detail in section 2.6, the problems we are trying to solve in this thesis all contain only binary variables. It has been shown that binary linear programming is an $\mathcal{NP}$-complete problem [29], thus requiring the simulation of all possible outcomes in order to be certain that a solution is optimal. Since the binary restriction require visiting all possible outcomes to find optimality, any polynomial or exponential complexity posed by a linear programming problem containing only real variables is insignificant.

2.5.4 Adding integer variables to the algorithm

The standard form of a linear programming problem was presented in equation 2.38, where the only restriction on the variables x was non-negativity. As seen in section 2.5.3, the complexity of a standard linear programming problem is most likely polynomial and in worst cases exponential. Since the solutions of the problems presented in this work must contain solely binary variables, we must impose some stricter restrictions to the problem variables than non-negativity, which might affect the complexity of the given problems. The two main restrictions that will be imposed to the problem variables during the solving process are the integrality constraint, where $x_i \in \mathbb{N}$ or the binary constraint, where $x_i \in \{0, 1\}$. A more specific description on the variable restrictions will be presented when discussing the results obtained with various different approaches in section 4.

Let us first consider a linear programming problem where some variables are restricted by integrality and the remaining ones are considered non-negative real value variables. The first instinct would be to solve it

with the Simplex method disregarding the integrality constraints. If it so happens that the obtained optimal solution would conform to the integrality constraints even if not explicitly imposed, it is just a stroke of luck. Most likely the optimal solution obtained will not conform the integrality restrictions prior to the relaxation of the variables and at least some of the variables obtained will be real values. [23] In case this happens, one can use the branch-and-bound algorithm to further process the resulting optimal solution.

The key concept of the branch-and-bound method is to decompose a problem that is difficult to solve directly into smaller partial problems in such a way that the any solution obtained by a partial solution is also a solution to the original problem [52]. In linear programming, this method keeps a list of nodes that all correspond to a single linear programming problem that is obtained by relaxing the integrality constraint on the variables bound by it in a parent node [30].

In general, the branch-and-bound algorithm works by applying the Simplex algorithm to the problem: relaxing all possible integality constraints to only non-negativity constraints. Once a solution is obtained, there are two possibilities. In the first, all variables are integers and the solving process is completed. Most likely this will not be the case and additional solving is required. The second possibility is that the solution contains at least one non-integer variable. If this is the case, one non-integer variable has to be chosen and two sub-problems have to be created and solved with the Simplex method. Assuming the selected variable x has a decimal value d, then the two sub-problems would have the variable constraints set as $x \geq \lceil d \rceil$ and $x \leq \lfloor d \rfloor$, respectively. After solving the two new sub-problems, we select the one resulting in a better optimal value and repeat the process until all variables are integers and thus the optimal result for the original problem is achieved. [33]

There exist various branching rules with which an initial problem can be decomposed into partial ones and similarly various bounding rules with which we can determine which nodes in the search tree need to be decomposed further. Additionally, various search strategies can be applied in the branch-and-bound approach in order to determine which node should be visited at the next iteration of the method. Such strategies include best-first and depth-first searches, which all lead to different search algorithms [52].

Similarly to the Simplex algorithm itself, the branch-and-bound has an exponential worst case complexity [41]. Though this method does not generally require visiting all the nodes to find an optimal solution, it still has a worst case computational complexity similar to a binary linear problem, which is an $\mathcal{NP}$-complete problem as mentioned in section 2.5.3.

The second sort of modification to a linear programming problem we have to consider is the addition of binary variables. In this case, the variables are bound to either zero or one in the final solution. These sort of problems have been shown to be $\mathcal{NP}$-complete [29], for which no polynomial complexity algorithm has been discovered in order for solving them. Problems in this complexity category are very difficult to solve and require brute forcing the solution space, since faster alternatives have not been found, but validating a solution is fast and can be done in polynomial time.

Since the problem formulation presented in more detail in section 2.6 restricts all variables to binary values, the larger number of variables clearly results in a larger complexity. In section 4, the results will be presented with the different approaches taken in order to solve problems of various sizes, based on the amount of variables and constraints. From the results it is clear that binary linear programming problems of a smaller size are solvable as presented, whereas ones with a larger size cannot be solved in reasonable time. In these cases, there are various different approaches that can be utilised for generating a result. Whereas the B&B method can solve a MILP, there are two tools the can help solving large BLPs. The first of these is trying to reduce the amount of binary variables in the problem by careful preprocessing in such a way that the resulting optimal result would still be a result to the original problem. The second is to use relaxation on some of the variables to ease the initial problem and solve it partially. The downside to this method is that it creates a partially valid solution where only some number of variables are binary bound. Once the optimum for the partially bound problem is found, it's binary variables are locked and the set variables are not modified in later iterations. Though a global optimum is found to the modified problem, it is not necessarily equal to the global optimum of the original problem due to the relaxation process at each iteration.

2.6 Presenting the problem as a LP problem

Now that we have seen how the Simplex method and algorithm work as well as how various variable restrictions in the problem affect the solving process, we can present the problem as a linear programming problem. For us to understand the canonical representation of it, we specify the following mathematical notations for different aspects of the problem.

First, let us define an ordered set of products P as

$$P \subset \mathbb{Z}^+ \tag{2.39}$$

and an ordered set of shelves S as

$$S \subset \mathbb{Z}^+. \tag{2.40}$$

Next, we define the maximum number of facings allowed if product q is placed on shelf t as

$$m_{qt} \in \mathbb{Z}^+ \ (q \in P \ ; \ t \in S), \tag{2.41}$$

the width of one facing of product q as

$$p_q^w \in \mathbb{Z}^+ \ (q \in P), \tag{2.42}$$

the width of shelf t as

$$s_t^w \in \mathbb{Z}^+ \ (t \in S) \tag{2.43}$$

as well as the binary variables determining if product q is placed on shelf t with f facings as

$$x_{qtf} \in \{0, 1\} \ (q \in P \ ; \ t \in S \ ; \ 0 < f \leq m_{qt}). \tag{2.44}$$

With the definitions 2.39 through 2.44 for formulating the problem, we still define the benefit of placing an f^{th} facing of product q on shelf t, used in the objective function of the MILP, as

$$b_{qtf} \in \mathbb{R}^* \ (q \in P \ ; \ t \in S \ ; \ 0 < f \leq m_{qt}). \tag{2.45}$$

Before expressing the problem in the standard form, it is useful to understand how the values b_{qtf} are generated. These are value that are calculated utilising a predefined cost function as follows. Given a decreasing function assigning a cost for placing a given number of units of product q as

$$c_q : \mathbb{Z}^* \to \mathbb{R}^* \ (q \in P) \tag{2.46}$$

and the amount of units that will be placed in case one facing of product q is placed on shelf t as

$$u_{qt} \in \mathbb{Z}^+ (q \in P \ ; \ t \in S), \tag{2.47}$$

the benefit for adding a facing can be formulated as

$$b^*_{qtf} = c_q((f-1)u_{qt}) - c_q(fu_{qt}), q \in P \ ; \ t \in S \ ; \ 0 < f \leq m_{qt}. \tag{2.48}$$

The equality represented in the equation 2.48 states that the benefit obtained by adding a facing equals to the total reduction in the cost function. Thus,

adding an additional facing for a given product $p \in P$ can only increase the cumulative benefit value b_{qtf}, since the cost function presented in equation 2.46 is decreasing. This cumulative value for b_{qtf} can be expressed as

$$b_{qtf} = \sum_{g=1}^{f} b^*_{qtg},\ q \in P\ ;\ t \in S\ ;\ 0 < f \leq m_{qt}\ ;\ g \in \mathbb{Z}^+. \tag{2.49}$$

All of the benefit values b_{qtf} have been precomputed with an existing cost function and used in this work as given values, separately for each problem. Hence, placing $f \leq m_{qt}$ facings for a given product q is at least as beneficial as placing $f - 1$ facings.

Utilizing the same data for the simulated annealing process as the mixed binary linear optimisation problem, yields comparable results in both complexity analysis as well as the obtained maximum for the sum of all selected benefits in the optimal solution. Now that we have an understanding on what the different variables and values quantify, we can formulate the problem in the standard linear programming form:

$$\begin{array}{lrcll}
\text{maximize} & \sum_{q\in P}\sum_{t\in S}\sum_{f=1}^{m_{qt}} b_{qtf}x_{qtf} & & & \\
\text{s.t.} & \sum_{t\in S}\sum_{f=1}^{m_{qt}} x_{qtf} & = & 1 & q \in P \\
& \sum_{t\in S}\sum_{f=1}^{m_{qt}} t(x_{ptf} - x_{qtf}) & \geq & 0 & (q,p) \in P, q = p-1 \\
& \sum_{f=1}^{m_{qt}}[x_{qtf} - (x_{ptf} + x_{prf})] & \leq & 0 & (q,p) \in P, (t,r) \in S, q = p-1, t = r-1 \\
& \sum_{q\in P}\sum_{f=1}^{m_{qt}} fp_q^w x_{qtf} & \leq & s_t^w & t \in S \\
& x_{qtf} & \in & \{0,1\} & q \in P, t \in S, 0 < f \leq m_q t
\end{array} \tag{2.50}$$

To get a better understanding of what the target function and the constraints used in 2.50 represent, we can take a closer look at them individually.

First, there is the objective function that we are going to maximise,

$$\sum_{q\in P}\sum_{t\in S}\sum_{f=1}^{m_{qt}} b_{qtf}x_{qtf}. \tag{2.51}$$

This quantity represents the sum of the benefits for all the placed facings. Thus, generating the optimal solution to this equation is achieved by finding the best combination of binary variables x_{qtf} such that when multiplied with their respective benefit value b_{qtf} and summed over q, t and f, yields the largest possible result. The obtained optimal solution must fulfil all of the following constraints subjected to the presented model.

Secondly, there are the constraints that are subjected to the model. These constraints are explained in more detail in equations 2.52 through 2.56. The first constraint formulated as

$$\sum_{t \in S} \sum_{f=1}^{m_{qt}} x_{qtf} = 1,\ q \in P, \tag{2.52}$$

ensures that a single product is placed on a single shelf. This constraint, together with the binary restriction of the variables x_{qtf}, ensures that a product can only be set on one shelf in the entire planogram. As was described in section 2.2, one product must reside on exactly one shelf in order for the planogram to be ordered, hence being pleasing to the eye and efficient for a customer to find their products.

The second constraint that can be expressed as

$$\sum_{t \in S} \sum_{f=1}^{m_{qt}} t(x_{ptf} - x_{qtf}) \geq 0,\ (q,p) \in P,\ q = p - 1, \tag{2.53}$$

ensures that the products placed must appear in order. This constraint makes sure that product $p = q + 1$ appears either on the same shelf or on later ones than product q. This holds true because of the previous constraint 2.52, stating that a product has to be placed on exactly one shelf. Though this constraint makes sure of the product order, it does not take into consideration of possible shelf gaps between two consecutive products. Hence, without constraint 2.54 in place, it would be possible for the algorithm to come up with a solution where empty shelves would appear between products q and p, which is something that cannot take place in the generated result. Thus, this constraint together with the one presented in 2.54 forces product order on successive shelves.

The third constraint formulated as

$$\textstyle\sum_{f=1}^{m_{qt}} [x_{qtf} - (x_{ptf} + x_{prf})] \leq 0,\ (q,p) \in P,\ (t,r) \in S,\ q = p - 1,\ t = r - 1, \tag{2.54}$$

states that consecutive products must appear either on the same shelf or on successive ones. Since we do not want empty shelves appear in the middle of the planogram, this constraint makes sure that in case product q is placed on shelf t, product p must be placed either on shelf t or r. As mentioned, this constraint together with the one presented in 2.53 ensure of product order on shelves in such a manner, that no empty shelves are between the first and last placed products. Though, it is acceptable and possible with this set of constraints that before the first or after the last placed products empty shelves could appear.

The fourth constraint applied to the problem is

$$\sum_{q \in P} \sum_{f=1}^{m_{qt}} f p_q^w x_{qtf} \leq s_t^w, \ t \in S. \tag{2.55}$$

This constraint implies that the sum of the widths of all the facings placed on a shelf cannot exceed the width of the shelf itself. The summed quantity is multiplied by the number of placed facings f, since the variable x_{qtf} represents a binary value expressing how many facings in total are placed.

The final constraint of the problem is

$$x_{qtf} \in \{0, 1\}, \ q \in P, \ t \in S, \ 0 < f \leq m_q t, \tag{2.56}$$

which makes sure that only whole facings are allowed to be placed on shelves. As mentioned earlier in this paragraph, this constraint joined with the first one ensures that only whole facings of products are placed and that each product can be placed only on one shelf.

Now that the baseline problem has been formulated and explained, we can solve it with available tools that support the Simplex method and additionally have the possibility of taking into account integer (and binary) variables in the process. In chapter 3 will be presented the tools and environments utilised in the task of solving the given problems and what were the criteria behind the selections.

Chapter 3

Environment

Being able to formulate a linear programming question mathematically is only half the battle. It is possible to solve these problems by hand, but having a large number of iterations would prove to be a lengthy and error prone procedure. Thus, we could benefit from utilising tools built for solving linear programming tasks by creating a program taking an input in a predefined format to generalise the solving process. In order to build such a generalised tool for tackling the problems presented in this thesis, many various aspects of the programming environment must be taken in consideration. Such aspects include the selection of the programming language, the programming platform, the mixed integer programming library and the solver utilised by the library. Since these four criterion are closely related, deciding one criterion heavily influenced the selection of the others, making it a straight forward process.

According to various sources, *Python* is one of the most used programming languages among developers, closely competing with *C* and *Java* [47, 48]. Since *Python* is one of the most used programming languages, it has a wide selection of standard libraries and toolkits available [15]. In addition of having linear programming tools available, it has a straight forward and simple syntax, which facilitates building any complicated software from scratch. Due to these reasons, it was selected as the programming language for the implementation. Using *Python* as the programming language with a Python IDE for building the application proved to be a powerful combination for an easy and straight forward implementation.

3.1 Linear programming modeler and solver

Since *Python* was selected as the programming language, a linear programming modeler had to be chosen from the selection available for that specific language. In short, a modeler is an application facilitating the implementation of an optimisation problem, which often is a ready built library that can be imported into an existing program. After some careful consideration, *PuLP* [43] was chosen as the modeler, since it is a *Python* library created for modeling linear and integer programs. This Python library is a project from The Computational Infrastructure for Operations Research (Coin-or) [10]. This non-profit educational foundation provides a vast amount of computational projects for cutting-edge computational research. Being developed by the community, this open source software is of high quality and performance due to the amount of testing and modification being done by a large user base. Since the algorithms are publicly available, it provides a platform for researchers to build their models on existing software. This gives them the ability to focus on research rather than reinventing the wheel and producing similar projects themselves, from the ground up.

In addition of being an open-source, cutting-edge software maintained and tested by the public, there were two additional reasons why *PuLP* was selected as the modeler for solving the given linear optimisation problems. The first reason was due to the ease of building and solving the problems. Once there was a procedure in place that could import the initial conditions of given problems into the modeler, the solving process did not require any additional installations nor troubleshooting. Since the modeler had a default solver available, there was no need to create a separate application that would do the solving part itself. Hence, any possible integration issues between multiple applications was averted by having one well built modeler capable of performing the entire process, from the problem statement building until the optimal solution generation.

The second additional reason for the selection of *PuLP* was the capability of calling various solvers in order to obtain an optimal result. A solver is an application including an algorithm capable of solving a given problem. By default, *PuLP* itself includes a solver, the Coin-or Branch and Cut mixed integer linear programming solver or CBC for short. This solver is designed to be used with the Coin-or Linear Programming solver (CLP), of which the main strength is the Simplex algorithm. [9] Though there are default solvers available built by the Coin-or organisation, applying a different solver for the modelled problem can be done trivially. The way this can be done is by first executing the built modeler that creates files containing the modelled

problem after which, a locally installed solver application can be executed to solve the modelled linear programming problem [42]. Since there are more powerful commercial solvers available, such as Gurobi [22], compared to the open source ones, once we are satisfied with the results obtained with the modelled problem we can switch to a commercial solver. By changing the solver used, we can potentially increase the performance of the solving process, thus obtain solutions in less time. In this thesis, we only use the default solvers provided by the Coin-or organisation to generate results and in case we see the need to further improve the runtimes of the final algorithms in the future, only then other solvers might be considered.

3.2 Result representation and testing

As seen earlier in this chapter, *Python* was selected to be the programming language and the contents of the *PuLP* library were used to model the problem statements and to solve them (modeler and solver). In addition to these, some additional *Python* libraries and functionalities were used for testing the optimal solutions and representing the results graphically.

First, some unit tests were created to validate obtained results. By using the physical limitations of the problem, such as product and shelf widths and the number of placed facings per product on specific shelves, any given solution was easily verifiable by assertion. Though in the case of some solutions the result was a local optimum due to relaxations made during the solving process, validating the solution was trivial and fast.

Second, two powerful open source libraries were used for data manipulation and scientific computing, *Pandas* and *NumPy* respectively. The *Pandas* library is a fast and powerful data analysis and manipulation tool [38]. It was utilised to build a generic, reliable and fast way of transferring the initial conditions of problems into the *PuLP* modeler. The *NumPy* library, which provides well-optimised and high performing scientific computing tools with *Python* [39], provided the framework for modifying and structuring the data in such a way that it can be easily applied correctly in various parts of the modelling process.

Finally, the results were presented graphically using *Jupyter Notebook* [28] and *Matplotlib* [49]. The *Jupyter Notebook* is a web application that provides a simple and streamlined way to create computational documents. The modular design enriches the functionality by providing the possibility to split the entire program into small executable subsections. By creating a subsection for each problem group, visualising the results could be done sequentially and separately. The results were visualised in this application

with the help of the *Matplotlib* library. This comprehensive visualisation library built in *Python* provided the necessary tools to create customised and clear representations of the results.

As seen, many open source libraries written in *Python* were used together to read the initial data into the program, model and solve it using the Simplex algorithm and create clear visualisation for results comparison. Having all of these tools built in the same programming language with a straight forward, high-level syntax, made the components worked well together and major issues related to the environment were not encountered.

3.3 Additional capabilities

Earlier in this chapter we have gone through the various components that were used during the solving procedure, from importing raw data until the graphical representation of the results. In addition to the tools used, there are various alternatives and optional modifications that could affect some problem variants. First, we must remember that in the final solution generated by any algorithm to any of the problems, the obtained variable values must be binary. Hence, in case we do not apply any relaxations at any point, none of the following additional tools will have a major impact in the time in which the results are generated. This is due to the binary linear programming problem being an $\mathcal{NP}$-complete problem [29], for which a polynomial complexity solving procedure has not been found.

One additional functionality that would have a positive impact on the algorithm runtime is parallelism. The CBC-solver has an in-built possibility of utilising a given number of threads in the solving process [40]. Having the ability of utilising N threads in the solving process could cut the execution time by this factor. This possibility was considered during the testing of the performance, but was not utilised since the idea was to find a fast way of solving the problems without relying to a large number of parallel processes. When trying to further improve the obtained results, this is one aspect to keep in mind.

Another additional aspect that could have a an impact on the solving time is the utilisation of a commercial solver, as presented in section 3.1. Again, the decision was made to use open source software during this work and in case promising results were obtained, further improvements could be achieved by upgrading the solver to a more powerful one.

In later chapters some preprocessing steps will be introduced, with which we were able to drop out redundant variables completely from the problems without affecting the optimal solution. For the more complex problems, some

preprocessing was done which can affect the generated result, since they remove variables providing less value to the problem than others. In addition to the preprocessing steps that were taken in this work, some further variable eliminations could be made in order to improve the solution generation process. Most likely they would have an effect on the obtained result, but in case they can provide a considerable time save, a small decrease in the obtained optimal solution could be tolerated. The only open question in this sort of variable elimination that can decrease the target value is that we do not have any reference to which the results could be compared to. Only in the case where a truly global optimum is obtained at some point of the solving procedure, would we be able to tell how much of a decrease takes place with some variable eliminations.

All of the functionalities and ideas presented in this section are some of the additional capabilities that could be examined closer in case further improvements to the results or runtimes would be required. As will be presented in later chapters, the obtained results are promising in both runtime and objective function values, compared to the current approach utilised in the result generation. Thus, being able to further improve in both of these categories has a direct effect in customer operations: a differently organised planogram can be obtained faster and can provide a larger value, which can on an aggregate level be a significant improvement.

Chapter 4

Experiments

Before implementing any process for solving the given problems, it was crucial to understand what information was provided for guiding this work. The given data can be divided into two different ensembles. The first of these is the raw data used with the simulated annealing approach for generating the benchmark results presented in more detail in table 4.1. The second ensemble contained the runtimes of the simulated annealing approach in addition to the heuristic optimal result obtained in the given runtime for a variety of different problems. As has been stated previously in section 2.4, the target of this work was ideally to generate global optimal results for all the presented problems in a shorter time than the given benchmark results. Though, as will become apparent in this chapter, modifications to some problem categories affecting the objective function value had to be made in order to generate any sort of results.

By fully understanding the data, we can build a sufficient knowledge in the solving process for us to recognise potential bottlenecks in it. This knowledge can provide useful insights to the solving process in various way. The applied techniques for generating results faster include such approaches as variable removal or relaxation, constraint modification or even the formulation of the solving algorithm. Thus, in this chapter we will first present the data in such a way that we understand what information is conveyed in it. Next, we will present the most basic formulation of the problem, i.e. the backbone for all the different approached tested. This will be followed by the presentation of various approaches tested during the span of this work and representing the outcomes obtained by the most promising ones compared to the benchmark results graphically. Finally, we will analyse different aspects that might have influenced the computational complexity of the solving procedure and sum our findings so far.

4.1 Data

The data provided for this thesis is real data from an existing customer project at a company specialising in retail and supply chain optimisation. As mentioned previously in equation 2.50, the objective function we are trying to maximize is of form

$$\sum_{q \in P} \sum_{t \in S} \sum_{f=1}^{m_{qt}} b_{qtf} x_{qtf}, \tag{4.1}$$

where q represents the products in the ordered set of products P, t represents the shelves in the ordered set of shelves S and f is the amount of facings placed, from 1 up to the maximum amount of m_{qt}. Thus, we are trying to maximise the objective function by multiplying the benefit values associated with the respective binary values representing the selection (1 = selected, 0 = not selected) of that specific number of facings f on shelf t for product q. The presented benefit values, which correspond to the benefit of adding f facings on a given shelf, are defined as the reduction in cost achieved by adding that facing's units. The cost function utilised in this process is described in more detail in section 2.2.

The original data set was given in a *CSV* (Comma Separated Value) format, containing roughly 30 different examples of varying sizes. Each of these required some processing for two purposes: getting the data into processable structures and being able to understand the content better once split up into smaller, structured collections. Essentially, the first processing step was getting the data into a format that could be easily modifiable and processable by the modeler. For each individual problem, which were all contained in separate files, the content of the data was split into $|P|$ matrices, where $|P|$ equals to the amount of products present in the given problem. Each of the $|P|$ matrices, one for each $q \in P$, contain the benefits b_{qtf} presented in equation 4.1. Hence, all of the matrices are of dimension $\mathbb{R}_{T \times m_{qt}}$, where $t \in S$, T is the number of shelves in the problem (size of S) and m_{qt} is the maximum amount of facings allowed. All of the entries in the matrices denote the benefit value for placing product q on shelf t with f facings. Now that the data is stored in a structured matter, any type of further modification and processing of the data can be done by accessing the correct matrix. This split into multiple matrices helps understanding all the values stored in these data structures, since any product-shelf-facing-combination can be accessed by knowing the values of p, t and f.

In addition to the raw data, the results utilising the adopted method of simulated annealing, were provided. This data contained the results obtained from applying simulated annealing to the different data sets as well as

the time required for achieving these results. The results achieved with this approach have some uncertainty associated with them, since without having visited all local optimum values, there is no guarantee a global optimum is attained. Each of the results provided had an associated value representing the runtime, which was used as a benchmark for the linear programming approach. For some of the largest problems in size, calculated as the product of the number of products and the number of shelves, two separate solutions with drastically different execution times were provided. This was to showcase that adequate results for large problems can be obtained relatively quickly: five to eight times more effort is required in order to obtain at most a 5 percentage improvement in the target function. All the results and runtimes obtained with simulated annealing are presented in detail in table 4.1, where the problems having two solutions with different runtimes are distinguished with suffix A (shorter time) and suffix B (longer time). These suffixes are only used for problems in difficulty categories *Difficult* and *Highly complex*, but not for all presented problems within the categories. This is due to the company providing the data not running two sets of simulations for all of the problems.

Each row within table 4.1 has four key values represented, conveying information about both the simulated annealing method and the linear programming approach used on one example set of data. The 28 problems presented have been categorised into smaller groups of varying difficulty levels based on the ease in which they could be solved with the linear programming approach. Hence, the *Problem description* is solely based on the amount of effort required for a given problem to generate an optimum result in a reasonable amount of time. The last column of the table, representing the number of variables in the initial problem statement, is roughly proportional to the difficulty category, but not fully (as seen for the two most challenging ones). This observation will be addressed in the following paragraphs. The remaining two columns in the middle of the table represent the original data: what is the maximum score obtained in a given amount of time utilising the simulated annealing method. These are the benchmark values which we are trying to overcome in this work: augmenting the maximum value and lowering the amount of time needed for finding this result.

Simulated annealing: maximum scores with respective runtimes			
Problem description (based on MILP solving difficulty)	Obtained result in given runtime	Runtime (seconds)	Total number of variables in corresponding, original MILP
Trivial 1	668.562	19	780
Trivial 2	668.73	22	780
Trivial 3	493.644	23	780
Trivial 4	669.356	11	780
Trivial 5	1539.734	28	1200
Trivial 6	1265.791	31	1200
Trivial 7	1439.743	33	1200
Trivial 8	1083.149	25	1200
Easy 1	1554.107	134	1800
Easy 2	1152.239	127	1800
Easy 3	1523.332	130	1800
Easy 4	1338.387	131	1800
Easy 5	965.393	126	1740
Easy 6	984.675	125	1740
Easy 7	786.829	29	1740
Easy 8	937.941	127	1740
Normal 1	1435.870	127	4730
Normal 2	1547.985	127	4730
Normal 3	1783.547	126	4730
Normal 4	1112.714	43	4730
Difficult 1A	11113.099	167	18070
Difficult 1B	11334.342	1183	18070
Difficult 2A	9845.303	171	18070
Difficult 2B	10087.450	916	18070
Difficult 3	11347.600	70	18070
Difficult 4A	6582.392	175	18070
Difficult 4B	6750.485	1291	18070
Highly complex 1	4254.521	199	10220
Highly complex 2A	4256.262	199	10220
Highly complex 2B	4396.745	1233	10220
Highly complex 3	4611.222	197	10220
Highly complex 4A	4178.736	195	10220
Highly complex 4B	4229.480	1179	10220

Table 4.1: *Problem descriptions, maximum scores obtained, runtimes for result generation and corresponding variable amounts in input matrices.*

As can be observed, the problems have been categorised into five separate difficulty groups. This categorisation was done based on the amount of effort required to solve the problems with linear programming in a reasonable amount of time, starting from to the most simple problem formulation presented in equation 4.2. As mentioned previously, the solving difficulty of the problems is roughly proportional on the number of variables present in the model. However, it is not exactly proportional due to other factors affecting the complexity of the problem. One such factor is the width of products p and shelves s in the problem. Lower product widths in addition to greater shelf widths imply a vaster solution space, since there is a large number of different combinations of product facings that can be fitted onto the given shelves. This is one aspect adding more complexity in the most challenging category, even if the number of variables is nearly half of the previous difficulty category, making it extremely demanding to find a solution within a reasonable time. As will be presented later in this chapter, different approaches were necessary in order to tackle problems from from the presented categories. Starting from the *Trivial* group, we had to introduce multiple steps that would speed up the solving process, even with a possible reduction in the global optimum value. Such steps include variable removal, data processing and a faster algorithm had to be used for being able to generate a result, and were applied to the problems once the solution generation process was taking seemingly too much time.

It is apparent from the categorisation naming convention that some problems were more challenging to solve in comparison to others. Thus, different approaches in their solving process was required for obtaining some optimal result with linear programming. As will be discussed in later chapters of this section, modifying the solving algorithm led in some cases to a suboptimal result to what the original problem definition would have yielded. Now that we have an understanding of the data, the benchmark results and the categorisation of various problems, can we start looking at what sort of approaches were taken in order to solve the different problems.

4.2 Backbone to the solution

As was presented in equation 2.50, the most simple format to present the the problems can be expressed with equation 4.2. All the various constraints and notations are presented in more detail in section 2.6.

$$
\begin{array}{lrcll}
\text{maximize} & \sum_{q\in P}\sum_{t\in S}\sum_{f=1}^{m_{qt}} b_{qtf}x_{qtf} & & & \\
\text{s.t.} & \sum_{t\in S}\sum_{f=1}^{m_{qt}} x_{qtf} & = & 1 & q \in P \\
 & \sum_{t\in S}\sum_{f=1}^{m_{qt}} t(x_{ptf} - x_{qtf}) & \geq & 0 & (q,p) \in P, q = p-1 \\
 & \sum_{f=1}^{m_{qt}}[x_{qtf} - (x_{ptf} + x_{prf})] & \leq & 0 & (q,p) \in P, (t,r) \in S, q = p-1, t = r-1 \\
 & \sum_{q\in P}\sum_{f=1}^{m_{qt}} fp_q^w x_{qtf} & \leq & s_t^w & t \in S \\
 & x_{qtf} & \in & \{0,1\} & q \in P, t \in S, 0 < f \leq m_q t
\end{array}
\tag{4.2}
$$

Essentially, this binary linear problem interpretation will provide the backbone for all of the other approaches or methods used later in this work. All the constraints presented in this model will have to hold true, for any solution obtained, even in the case where some modifications or additions will be introduced along the solving process. Even though some drastically different approaches will be tested that will introduce some variable relaxations to the original problem, the resulting optimum will have to conform to all the constraints presented in the optimisation formulation in equation 4.2.

After having built a program that could take the input data for any given problem, provide it to the modeler and then solving it with the open-source solver while taking into account the constraints presented, we could try to test its capabilities by trying to find an optimal result for the problems presented in the *Trivial* category. This category was chosen for testing out the solving process with the most basic problem formulation, since its problems contain the lowest amount of complexity. In case we can obtain an optimal solution to these eight problems faster than with simulating annealing, we can try to utilise the same formulation for the other difficulty categories as well to find their optimal solutions. In the figure 4.1 below are presented the results obtained to the problems in the *Trivial* category with both the simulated annealing approach as well as the linear programming method, based on the equations of 4.2. In addition to the results, the figure contains the time required for each problem in order to reach these results with both methods.

As can be observed in figure 4.1, the obtained results are quite promising. For all of the presented problems, the optimal result was achieved with the linear programming approach, which was larger than what was obtained with simulated annealing. Additionally, the time used for producing the optimal results were all considerably lower. On average 20 seconds less runtime provided a 15% score increase. Because optimal results to the original problems were yielded within this difficulty category in a rapid manner, we can be satisfied with the outcome and try to solve the problems in the next category, called *Easy*.

Next, all of the given problems in difficulty category *Easy* were formu-

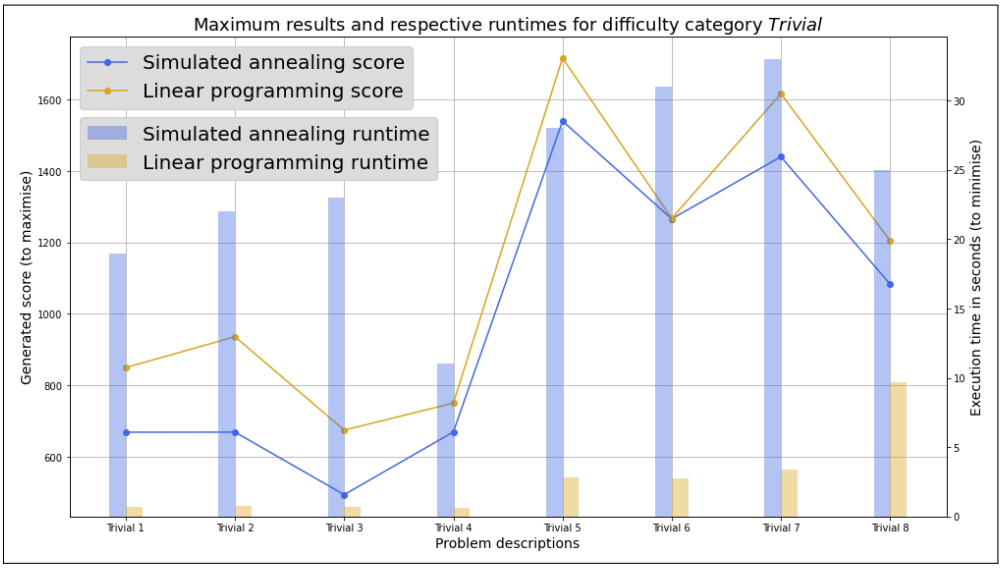

Figure 4.1: *Maximum scores and respective runtimes for the Trivial difficulty category*

lated with the linear optimisation equation 4.2. Since the number of binary variables in the model was greater than for the *Trivial* problems, it was expected that the solving process would take longer. The assumption about a longer runtime held true, since generating optimal results took anywhere between 25 seconds up until almost 10 minutes, which is visualised in figure 4.2 below.

Even if optimality was reached, it was clear that something needed to be done, since any category from this point on would have a larger amount of variables, implying even longer times for generating solutions. The basic problem formulation was even tested on the *Normal* problems, which could not generate any solutions within hours, thus proving this approach being inadequate for any other category than the *Trivial* and *Easy* ones.

4.3 Removing binary restrictions

Since there was a need to find some way that could generate results faster than the baseline model, different ideas had to be tested out. The first approach that was tested was modifying the equation 4.2 in such a way that we would replace the binary restriction subjected to the variables by an integer or real value restriction. This is to say that we would test out what sort of results could be obtained by replacing the binary restriction

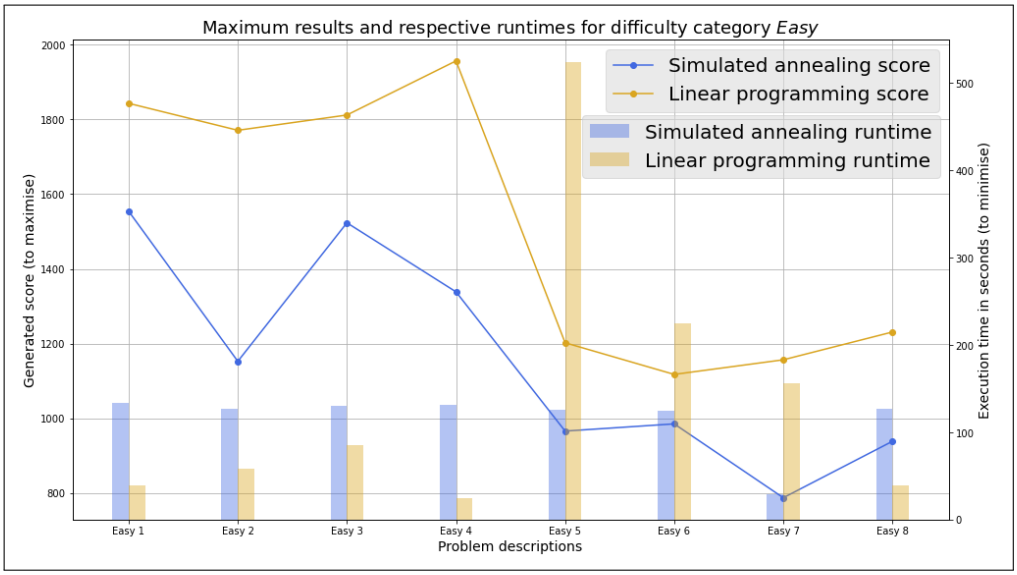

Figure 4.2: *Maximum scores and respective runtimes for the Easy difficulty category without data processing*

$$x_{qtf} \in \{0,1\},\ q \in P,\ t \in S,\ 0 < f \leq m_q t \tag{4.3}$$

with either an integer restriction

$$x_{qtf} \in \mathbb{Z}^+,\ q \in P,\ t \in S,\ 0 < f \leq m_q t \tag{4.4}$$

or a real value restriction

$$x_{qtf} \in \mathbb{R}^+,\ q \in P,\ t \in S,\ 0 < f \leq m_q t. \tag{4.5}$$

As was discussed in sections 2.5.3 and 2.5.4, solving a problem with the Simplex algorithm containing real value variables has smoothed polynomial complexity and utilising the branch-and-bound method with integer variables can we expect at most exponential complexity. In case we would be able to iteratively solve the problems with the modifications presented in equations 4.4 and 4.5, we would hopefully not need to visit every feasible solution for getting to the optimal solution, as is the case with a model with binary variables, which is an $\mathcal{NP}$-complete problem.

4.3.1 Integrality constraint

Let us make the adjustment presented in equation 4.4 into the simple problem representation from 4.2 in order to obtain the linear optimisation formulation

presented in 4.6.

$$
\begin{array}{lrcll}
\text{maximize} & \sum_{q\in P}\sum_{t\in S}\sum_{f=1}^{m_{qt}} b_{qtf}x_{qtf} & & & \\
\text{s.t.} & \sum_{t\in S}\sum_{f=1}^{m_{qt}} x_{qtf} & = & 1 & q \in P \\
& \sum_{t\in S}\sum_{f=1}^{m_{qt}} t(x_{ptf} - x_{qtf}) & \geq & 0 & (q,p) \in P, q = p-1 \\
& \sum_{f=1}^{m_{qt}}[x_{qtf} - (x_{ptf} + x_{prf})] & \leq & 0 & (q,p) \in P, (t,r) \in S, q = p-1, t = r-1 \\
& \sum_{q\in P}\sum_{f=1}^{m_{qt}} fp_q^w x_{qtf} & \leq & s_t^w & t \in S \\
& x_{qtf} & \in & \mathbb{Z}^+ & q \in P, t \in S, 0 < f \leq m_q t
\end{array}
\tag{4.6}
$$

The only change introduced in the problem formulation is the variable restrictions, which are bound to be positive integer values. Even though this modification would seems to shorten the solution generation time with the branch-and-bound technique, it actually does not. There are two possible approaches that can be taken with the integrality constraint. The first is to restrict the variables between zero and one, in which case the problem is relaxed and solved in such a way that all variables will be in the given range. After an initial result is obtained in which most likely all the variables are not either zero or one, but somewhere in between, the branch-and-bound method will start the solution process. Even if it sounds like a good approach, it will have to go through all the combinations, similar to the binary approach presented in section 4.2. Thus, this leaves us with the alternative method, where we do not bound the variables to anything else than being positive integers, not conforming them into a specific value range.

In this second method, the solving process is similar to the first one. First, the variables are relaxed to only positive real values without any upper limit. Once some initial solution is obtained, the branch-and-bound method will again start the iterative approach for generating the global optimum for the initial problem. Though it would seem as the problem could be somewhat different to the first approach presented, it actually is not. Since there is a constraint limiting the sum of all the variables for one product defined as

$$\sum_{t\in S}\sum_{f=1}^{m_{qt}} x_{qtf} = 1,\ q \in P \tag{4.7}$$

all of the resulting variables prior to starting the branch-and-bound will again be between zero and one, since none of them can be negative. Unfortunately, formulating the problem with integrality constraints did not provide any help in solving the more complex problem categories in a faster manner and therefore this approach was discarded.

4.3.2 Real value constraint

The next approach that was tested out for solving the given problems quicker, was by utilising the real value restriction from equation 4.5 instead of the binary restriction. Thus, the problem could be formulated with the notation presented in 4.8.

$$\begin{array}{lrcll} \text{maximize} & \sum_{q\in P}\sum_{t\in S}\sum_{f=1}^{m_{qt}} b_{qtf}x_{qtf} & & & \\ \text{s.t.} & \sum_{t\in S}\sum_{f=1}^{m_{qt}} x_{qtf} & = & 1 & q \in P \\ & \sum_{t\in S}\sum_{f=1}^{m_{qt}} t(x_{ptf} - x_{qtf}) & \geq & 0 & (q,p) \in P, q = p-1 \\ & \sum_{f=1}^{m_{qt}}[x_{qtf} - (x_{ptf} + x_{prf})] & \leq & 0 & (q,p) \in P, (t,r) \in S, q = p-1, t = r-1 \\ & \sum_{q\in P}\sum_{f=1}^{m_{qt}} f p_q^w x_{qtf} & \leq & s_t^w & t \in S \\ & x_{qtf} & \in & \mathbb{R}^+ & q \in P, t \in S, 0 < f \leq m_q t \end{array} \tag{4.8}$$

This was a more promising way for solving the problems, since a result could be generated in fractions of seconds, even for the most challenging categories. Despite the time used for generating the result was low, some issues related to the variables and their values was discovered. The core of the complication caused by the real value variables was linked to the constraints presented in equations 4.9 and 4.10.

$$\sum_{t\in S}\sum_{f=1}^{m_{qt}} t(x_{ptf} - x_{qtf}) \geq 0,\ (q,p) \in P,\ q = p-1. \tag{4.9}$$

$$\sum_{f=1}^{m_{qt}}[x_{qtf} - (x_{ptf} + x_{prf})] \leq 0,\ (q,p) \in P,\ (t,r) \in S,\ q = p-1,\ t = r-1. \tag{4.10}$$

The variables in the obtained solution adhere to these two constraints even with the real value relaxation, but fail to behave in the expected manner. The problem that is manifested by subjecting the variables only to a real value restriction is that multiple variables can have non-zero values spreading across a multitude of shelves. Even if the total sum of the variables for any given product is equal to one as specified in the problem statement, there is no way to enforce that they must be restricted to specific shelves without subjecting the model to some binary variables.

An approach was tested by introducing additional binary variables into the problem. First, we needed to restrict the sum of the variables for a given product q on any given shelf t to either zero or one. Alone, this constraint was not enough, since it could lead to a result where multiple variables for product q on shelf t would have non-zero values (summing up to one), which was not an acceptable solution since we must place only full facings onto the

planogram. Thus we needed to introduce additional binary variables in order to restrict the sum of variables for a given product q with a specific number of facings f also to zero or one. These two restrictions can be expressed with equations 4.11 and 4.12 respectively.

$$\sum_{f=1}^{m_{qt}} x_{qtf} = y_{qt},\ q \in P,\ t \in S,\ y_{qt} \in \{0,1\} \tag{4.11}$$

and

$$\sum_{t \in S} x_{qtf} = z_{qt},\ q \in P,\ 0 < f \leq m_q t,\ z_{qt} \in \{0,1\}. \tag{4.12}$$

With the help of these two equations we could be certain that only binary values were generated in an obtained solution. Also, we had certainty of only one of the variables for a given product q being set to one due to the constraint formulated as

$$\sum_{t \in S} \sum_{f=1}^{m_{qt}} x_{qtf} = 1,\ q \in P. \tag{4.13}$$

With the constraints presented in the equations 4.11 through 4.13, we could be certain that exactly one variable for each product would equal to one.

With these changes to the linear programming model, we could reduce the number of binary variables present in the problem from $q * t * f$ to $q * (t + f)$, where q equals the total number of products, t the total number of shelves and f the maximum number of facings possible. Unfortunately, reducing the number of binary variables even for the difficulty category *Normal* did not provide enough of a performance enhancement to the solving process, rendering the tested approach useless.

4.4 Preprocessing steps

Because neither the most simple model presented in 4.2 nor the variable restriction relaxations from sections 4.3.1 and 4.3.2 could perform adequately enough with a larger data set, our focus shifted to the data. In case we could remove some portion of the binary variables in the linear programming problems, the runtime would drop and results could be generated faster. Ideally, the variable removal process or any sort of data modification would not affect the global optimum result generated by the solving algorithm. After closer inspection of the data it became apparent that it is very primitive. This is to say that there were many unnecessary variables present from the solving

algorithm's perspective that if removed, would not have any affect on the result generated.

As was presented in more detail in section 4.1, the data consists of benefit values b_{qtf} that represent the benefit obtained by placing the respective variable, denoted as x_{qtf}, into the planogram. All of the benefit values presented in the data are cumulative sums for the number of facings a product can have on a specific shelf, which can be interpreted as

$$b_{qtf} \leq b_{qtg},\ q \in P,\ t \in S,\ 1 \leq f < g \leq m_{qt}. \tag{4.14}$$

Having two equal values consecutively, which can be represented as

$$b_{qtf} = b_{qtg},\ q \in P,\ t \in S,\ 1 \leq f < g \leq m_{qt}, \tag{4.15}$$

indicates that having g facings of product q on shelf t does not provide any additional benefit to having f facings of q on shelf t. Knowing that including an additional facing does not provide any more benefit to the optimal solution, these variables can be removed from the data completely, resulting in removing redundancy and complexity from it. The only noticeable difference to the model when selecting benefit b_{qtf} over the benefit b_{qtg}, which both have the same values, is that there is more space left on shelf t. Since $1 \leq f < g \leq m_{qt}$, the additional space provided can be more formally expressed as

$$p_q^w * (g - f),\ 1 \leq f < g \leq m_{qt}, \tag{4.16}$$

where p_q^w expresses the width of product q. Despite this change not having any effect on the result provided, it is worth while knowing that less space will be utilised, in case in further applications of this work there might be some applicability for this knowledge. Even if the result is not affected by this change, the runtime did benefit greatly from this data processing step together with the ones presented later in this section.

In addition to removing the non-beneficial variables from the model, we could further eliminate redundant ones by closer inspection of the data. The given data for the problems was elementary in the sense that it was not tailored for this specific linear optimisation approach. The first and most obvious way to remove redundant variables from the problem is to eliminate the ones that have a facing value f that is greater than the maximum amount allowed. More formally, we want to remove variables satisfying the condition that can be expressed as

$$f > m_{qt},\ q \in P,\ t \in S,\ 1 \leq f. \tag{4.17}$$

Though this can eliminate some variables, we can take this idea one step further. As said, the raw data was very elementary, not taking into account any illogicalities present. Hence, it was possible that the maximum amount of facings placed for a given product on a given shelf would breach one of the constraints presented in 4.2. For a given product, we can modify the constraint from the original problem formulation into the following format,

$$\sum_{f=1}^{m_{qt}} f p_q^w x_{qtf} \leq s_t^w,\ q \in P,\ t \in S, \tag{4.18}$$

in which the total width of the facings placed for a product q on shelf t must be smaller of equal to the width of the given shelf t. Knowing this, we could eliminate all the variables from the model satisfying the equation

$$p_q^w * f > s_t^w,\ q \in P,\ t \in S,\ 1 \leq f \leq m_{qt}. \tag{4.19}$$

Thus, all the variables for a given product q on a given shelf t, where the number of facings f multiplied by the product width p_q^w would exceed the shelf width s_t^w, could be removed. There was quite a lot of variation around the variable removal utilising this approach, since for some products no removal could be done whereas for some over half of the variables could be discarded. This elimination technique provided a large improvement to the problem complexities and utilised with the other rules presented in this section could lead us in the right direction for solving all problems optimally in a fast manner.

Before imposing the problems to the modifications presented prior, there is one more way of removing redundant variables without affecting the resulting optimum score. Since there is order in the product placement, denoted as

$$\sum_{t \in S} \sum_{f=1}^{m_{qt}} t(x_{ptf} - x_{qtf}) \geq 0,\ (q,p) \in P,\ q = p - 1, \tag{4.20}$$

we know that product p has to appear only after the product q has been placed. Evidently, only after all the facings for product q have been placed can we start placing any facings for product p. Utilising this information could we define the first possible appearances of all products in a given planogram. The way to find out the first possible occurrence of a given product q is by summing up the minimum number of facings $f = 1$ for each product, starting from shelf $t = 1$, until this sum is greater than the shelf width s_1^w. Assuming it takes n products to fill up the shelf $t = 1$ with exactly one facing of each product, the first possible appearance of product $n + 1$

must be on shelf $t + 1 = 2$, since all products must be placed on exactly one shelf in a defined order. The cut over for transitioning to the next shelf is when we cannot place product $n + 1$ to the current shelf without exceeding the shelf width s_t^w. This reasoning does not have any effect on the optimum generated and can be extended to the bottom part of the planogram, i.e. starting from the last shelf.

The way we can take this idea further is by making the assumption that the last product will have to be placed on the last shelf. With this approach we can do the reasoning process in reverse, starting from the last shelf instead of the first one. Similar to finding out the first possible shelf occurrence of a product, we find out the last possible shelf occurrence for product q by summing up the minimum number of facings $f = 1$ for each product, starting from the last possible shelf t. Assuming it takes n products to fill up the last shelf t with exactly one facing of each product, the first possible appearance of product $n - 1$ must be on shelf $t - 1$, since all products must be placed on exactly one shelf. Again, the cut over for transitioning to the previous shelf is when we cannot place product $n + 1$ to the current shelf without exceeding the shelf width s_t^w. It is worth noting that making the assumption of having the last product on the last shelf does not have any effect on the optimality, since it is based on physical constraints that must hold in all acceptable solutions.

It was mentioned in section 4.2 that the time utilised with the most elementary approach on the problems in the *Easy* category took anywhere from half a minute to roughly 10 minutes. Having a way of removing redundant variables from the model could have an impact on the time taken for generating the optimal solution. After applying all the various techniques presented in this section to the problems in the *Easy* difficulty category, we can observe the obtained results with respective runtimes in figure 4.3 below in comparison to the simulated annealing method.

It is clear that a significant time save has been obtained with the removal process of the redundant variables, since results were generated between one and two seconds for all eight problems in this difficulty category. Additionally, all the obtained results matched to the optimal ones obtained with the most basic approach, as was expected after the redundancy removal. Hence, optimal results were obtained in a very quick manner for these problems and we can shift our focus to the next category of problems and see whether the presented techniques would be powerful enough to solve them in adequate time.

Next, the problem category *Normal* had to be solved. First, by applying only the basic constraints without any data processing, we were able to solve all four problems. Though an optimal results was achieved, the time taken

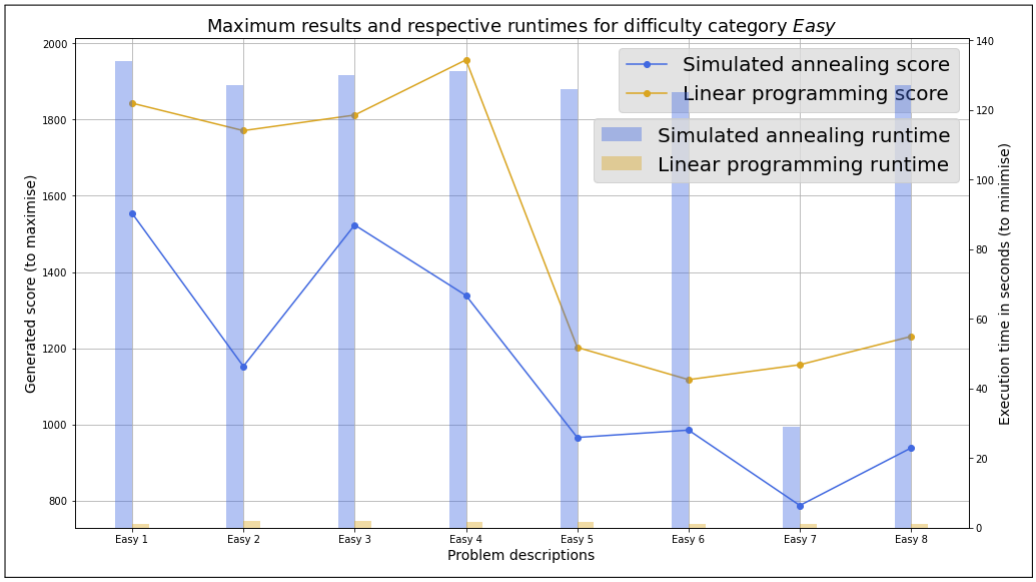

Figure 4.3: *Maximum scores and respective runtimes for the Easy difficulty category*

by the solver took anywhere from a couple of hours to tens of hours, which was not an adequate amount. Even if we could generate some solution for these problems, it was clear that something needed to be done to lower the runtimes. The next logical step in the process was to apply the data processing steps as was done to the *Easy* difficulty category in this section.

After applying the data processing steps presented in this section, a runtime of hours could be dropped down to values ranging between 5 and 37 seconds, which are presented in section 4.5 in figure 4.4. For all of the problems, the optimal solution was exactly the same as after the variable removal process, as was expected. Although this is a clear improvement to the solving process, nothing beyond the problem size within this difficulty category will be solvable in some reasonable amount of time. Thus, some clever way of finding a solution faster is required, even if it would come with a loss in the resulting score.

It is worth noting that the preprocessing steps are contained in the timing of the solving process. This means that the time used to generate the solutions for any of the problems, contain both the data preprocessing and the solving procedure. Because the size of the data sets are small, the time utilised for the data manipulation is only a fraction of what is required for the solver for generating a solution.

4.5 Iterative approach

Even if we could solve problems from the difficulty category *Normal* optimally and faster than what was done with simulated annealing, we wanted to find an even faster way for generating results. Some clever algorithm needed to be discovered in the hope of lowering the time taken by the solution generation process even if it would come at a cost, which was a lower objective function value. The pseudo-code for the algorithm designed for this purpose is presented in algorithm 2 below and will be described in more detail in the following paragraphs.

Algorithm 2 Pseudo-code for the iterative approach algorithm

Set Initial binary variable stepper $N_{step} > 0$
Set Current binary variable count $N = 0$
Set Current shelf to lock $S = 1$
  Create a new LP model M
  Add only real value variables to M
  Add initial constraints to M
  Generate optimal solution for M
  repeat
    if All variables set on S equal to one **then**
      Lock binary values of shelf S for later iterations
      $S \leftarrow S + 1$
      $N \leftarrow N_{step}$
    else
      $N \leftarrow N + N_{step}$
    end if
    Update variables in LP model:
      N binary ones in addition to the locked variables
    Update constraints in LP model M
    Solve model M
  until All variables in M are binary

The algorithm that was created for solving more complex problems than what were presented in the *Normal* difficulty category takes one parameter as an input. This parameter, defined as $N_{step}(\in \mathbb{Z}^+)$, indicates how many additional variables are considered as binary ones during the next iteration of the algorithm. In order to keep track of how many variables in total should be considered as binary ones, we can define the total binary variable count as $N(\in \mathbb{Z}^+)$. Thus, all the variables in the problem connected to these N products will be bounded to binary values, which can be formulated as

$$x_{qtf} \in \{0,1\},\ q \in [1,N],\ t \in S,\ 1 \leq f \leq m_{qt}, \tag{4.21}$$

and all other variables presented in the problem are bound to real values, which can be expressed as

$$x_{qtf} \in \mathbb{R}^{+},\ q \in [N+1,|P|],\ t \in S,\ 1 \leq f \leq m_{qt}. \tag{4.22}$$

With these variable restrictions in place, we could start to solve the problem. First, the preprocessing steps presented in section 4.4 were applied in order to remove possible redundancy. After the processing was done, N_{step} was arbitrarily selected. Additionally, the first shelf $t = 1$ was selected as the target shelf to be filled - the reasoning for this will become apparent in the next paragraphs.

Now, the iterative phase could begin by solving the binary linear programming problem containing a mixture of binary and real value variables. Since the count of binary variables is greatly reduced (depending on how large of a value is selected for N_{step}), the time utilised for solving this problem is considerably faster than the solving process of the original one. There are two scenarios that can happen once the iteration is completed. In case a solution is obtained where the selected target shelf does not solely contain binary variables, we increase the value of the binary variable counter N by the value of the input parameter N_{step} for the next iteration. The value for N during the iterative process can be expressed as

$$N_i = i * N_{step},\ i \in \mathbb{N}, \tag{4.23}$$

where i represents the number of completed iterations. After this operation, we proceed to increasing the number of binary variables in the problem according to equation 4.21 and lowering the amount of real ones based on equation 4.22, and iterate again. In the second scenario after i iterations are completed, a solution is obtained in which all the variables placed on the current target shelf are binary ones. If this is the case, we lock the binary variables for all the corresponding products on the target shelf, meaning that during the next iteration of the algorithm only one variable for each product on the target shelf equals to one and the remaining ones equal to zero. With this method, we can remove some complexity from the full problem at each step and iteratively find some result to the given problems, though not optimal.

Once the products set on the target shelf have been locked, we initialise the parameters for the next iteration by setting the number of binary variables back to N_{step}, increase the new target shelf by one and start a new

iteration of the model. Shelf by shelf, binary values for products are set until we reach a final solution in which all the values are binary ones, meaning that the final result has been achieved. Though a time save is obtained with this approach, it has a drawback of not necessarily yielding the true optimum for the original problem. In order to understand why it does not necessarily yield an optimal result, we must look closer at the iterative procedure in the algorithm. Since we relax some portion of the variables, the optimal solution obtained when locking the target shelf in place is optimal for that specific problem. This set of binary variables is not necessarily, but can be the same set that would be present in the optimal solution for the original problem statement. This means that when we lock a target shelf in place, though it is optimal for a specific iteration of the problem, it locks some variables in place which might not have been the values selected in the true optimum for the original problem. Thus, we have no certainty of how good the obtained solution is after the iterative approach in comparison to the true global optimum.

After testing out numerous values for the input parameter N_{step} for solving the problems in difficulty category *Normal*, the runtimes were almost identical, differing only by fractions of a second at most. It was found that the value $N_{step} = 5$ provided the best results on average for this difficulty category and was thus chosen as the one selected in the results comparison. In figure 4.4 are represented three different scores and their respective execution times: simulated annealing, only utilising data processing from section 4.4 and the iterative approach presented in this section with the data processing steps as well.

It is clear that an improvement in the runtime was achieved with the iterative approach. In comparison to the non-iterative approach, the iterative method was able to find a solution on average four times faster with only a 1.6% trade-off of the optimal result. With this new algorithm, the problems in the difficulty category *Normal* can be seen as solved, since in comparison to the simulated annealing approach, results (though not optimal) were generated on average roughly 18 times faster with a 50% increase in the end result.

4.6 Threshold variable removal

The complexity category *Difficult* is the first presented group of problems containing two different scores and respective runtimes for three out of the four problems, distinguished by suffixes A and B. The reason for this was to see how much the generated result could increase when given noticeably more

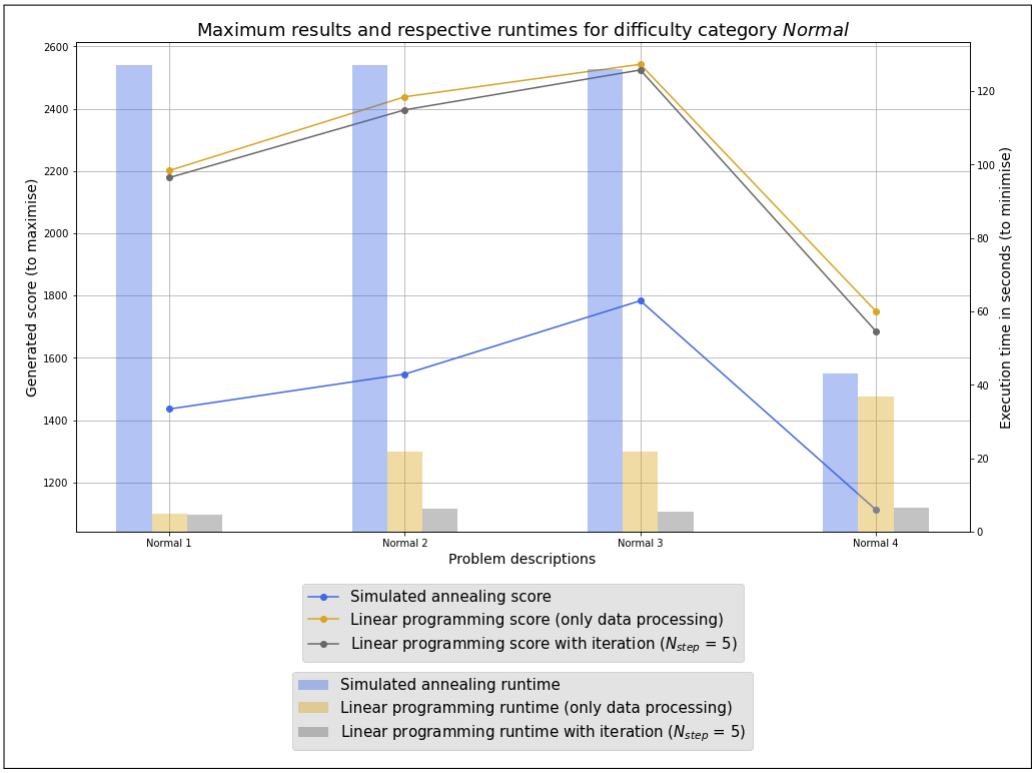

Figure 4.4: *Maximum scores and respective runtimes for the Normal difficulty category*

time during the simulated annealing process. From the linear programming perspective, the results are exactly the same regardless of the suffix, since the problem formulation for both are identical. This is why the results presented later in figure 4.5 are matching for the two linear programming methods and varying for the simulated annealing ones.

As was mentioned in the previous section, every problem in the difficulty category *Normal* could be solved with the three different approaches: with the original problem formulation presented in equation 4.2 with and without the data processing as well as the iterative method presented in the previous section. Since the problems in the difficulty category *Difficult* are much larger in size compared to the *Normal* ones, containing roughly four times more variables, no feasible solution could be generated even after the data processing steps from section 4.4. This means that we could not have any certainty about the optimal solutions for the problems, but could only try and come up with a better solution faster than what was obtained with simulated annealing. The only way for us to generate any sort of results was to utilise the iterative approach from section 4.5 after the removal process of

unnecessary variables presented in 4.4.

After some testing, it was found that selecting $N_{step} = 10$ yielded results as good as with other selected values, but in a quicker fashion. By applying the data processing with the iterative approach, the scores generated were on average roughly 7% greater that what was obtained with the shorter execution of the simulated annealing approach (suffix A) and 5% greater than the ones obtained with the lengthier (suffix B) approach. Additionally, the average time used to generate these results with linear programming was a couple of seconds faster. Even if this is very promising, one more variable removal process could be applied in the hope of trading a small portion of the obtained optimum for a faster result generation time.

In this variable removal process, we first have to come up with an arbitrary threshold percentage value denoted as $M > 0$. Once selected, we remove those variables from the entire set of variables that do not provide at least an additional M percent of benefit to the model. The way the additional benefit value is calculated is based on the previous selected value for a given product q on a given shelf t. For this given product on the given shelf, we select the benefit associated with the first possible variable x_{qt1}, which denotes placing only one facing into the planogram. After this, we compare the next possible benefit value for variable x_{qt2} to the previously selected benefit. In case it does provide at least an M percent benefit increase, it is not removed and this new value becomes the benefit that the remaining benefits will be compared to. In case it does not provide at least an M percent value increase compared to the previous selected benefit, the associated variable is removed from the final model. With this approach we can have some speed up in the solution generation process with at most a M percent trade-off in the resulting score.

In figure 4.5 below are presented three different results with respective runtimes: simulated annealing, linear programming with the iterative approach (with data processing) and linear programming with the threshold variable removal (with data processing). As mentioned, $N_{step} = 10$ was selected since it yielded the fastest times and a threshold value of 20% was selected after testing out numerous other values, since it provided the best objective function values in the least amount of time.

From the figure above we can conclude that a time save was indeed obtained between the two linear programming approaches, making the threshold approach roughly two times faster than the one without it. On average, these times are also faster compared to the simulated annealing approach. Though the problem *Difficult 3* was slower to solve with linear programming, it generated the most significant score increase of almost 9%. For the problems $1 - 3$ there was only a minimal score decrease between the iterative methods and one aspect that is surprising is that the threshold removal

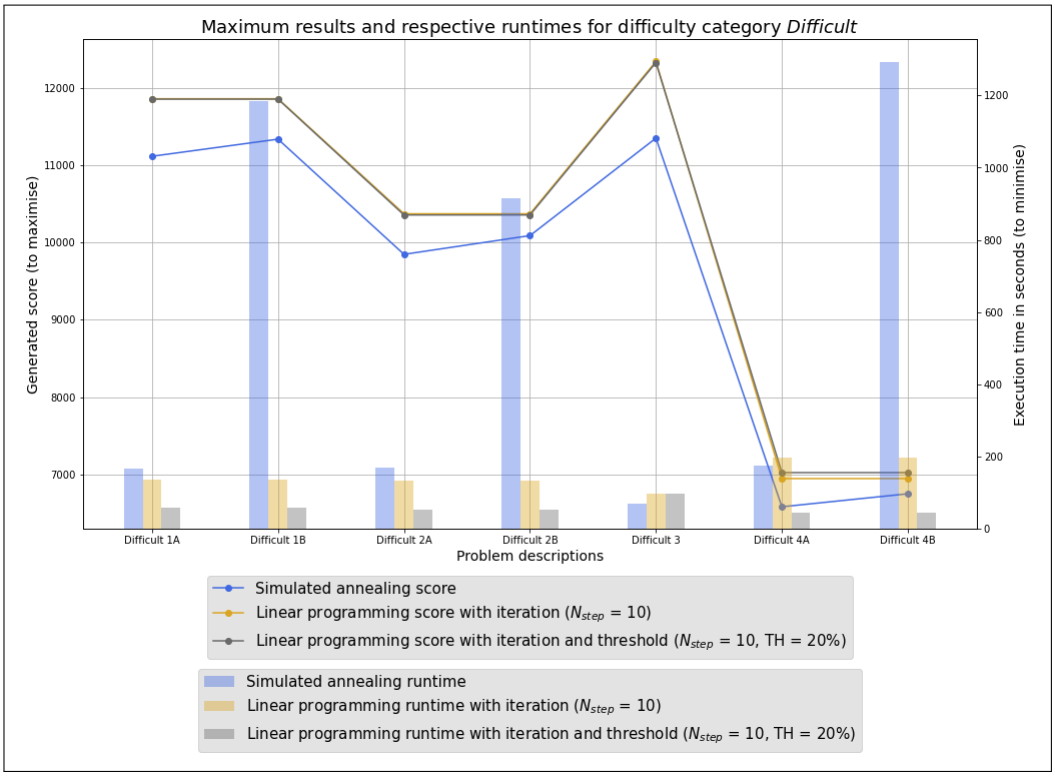

Figure 4.5: *Maximum scores and respective runtimes for the Difficult difficulty category. The scores obtained with the two LP approaches are nearly identical, thus they are overlapping in this figure.*

approach provided a small score improvement to the problem 4 even though we were expecting at most a 20% decrease. At first it seems quite odd, but there is a rational explanation to this event that can happen just as well as the decrease of the target function value.

Since we do not have certainty about what the true optimal value for the original problems are, we cannot make the assumption that any variable removal process would definitely decrease the target function value. At a first glance, it would seem reasonable to assume that removing variables that provide additional benefit into the model would lower the obtained value. After analysing this in more detail, we can indeed find a way to justify this increase. In the iterative algorithm, we process one shelf at a time by placing some amount facings for products (corresponding to the binary variables) onto it while relaxing the remaining part of the problem. When we have not removed any variables with respective benefit values from the process, the set of variables from which we select the optimal configuration from, shelf by shelf, differs from the one that remains after the threshold removal process.

When we think at the problem after the removal process, we have taken out a proportion of variables that would require placing multiple facings of some products in the final planogram. Thus, the optimal solution built can incorporate more products on the target shelves than the approach without the removal process, making the end result unpredictable. Even if we were expecting at most a 20% decrease, on average we obtained a minimal increase in the results.

4.7 Highly complex problems

The last difficulty category remaining is the *Highly complex* one containing four different problems. Similarly to the *Difficult* group, there are two different problems containing two different solutions distinguished by suffixes A and B. Even if the number of variables contained in these problems were approximately only half from the previous category (see table 4.1), the complexity of these problems comes from wider shelves. With the iterative approach we can lower the number of binary variables contained in the problem when we try to lock these products onto a target shelf. But since the shelves are very wide, they can contain a vast number of products, thus adding complexity. This means that while trying to find a solution where a shelf is set with some products, the amount of binary variables rises to a point, where it affects heavily the time to find a solution to the partial problem.

Naturally, the first instinct is to process the data and remove all redundancy followed by the iterative algorithm with the threshold variable removal. Again, after some testing the values $N_{step} = 10$ and $M = 20$ were selected for generating results. Though this method proved to be adequate, there was still one possible way of improving the time utilised in the process with a score trade-off. This was by noticing that in most results, the first product is placed on the first shelf and the last product on the last shelf. By applying this logic further, knowing that placed products have to placed in order and that they can be placed on exactly one shelf, we could do the following generalisation. In case product q is placed on shelf t, product $q+1$ has to be placed on either shelf t or $t+1$ and similarly product $q-1$ must be placed on either shelf t or $t-1$. By removing all redundant variables based on this rule, we could apply the iterative process to the smaller set of variables and obtain the results presented in figure 4.6.

It is clear that the results obtained with the iterative method are way better compared to the ones obtained with simulated annealing. The trade-off is also clearly visible, since applying the rule presented earlier in this section (locking the first product on the first shelf and the last product on

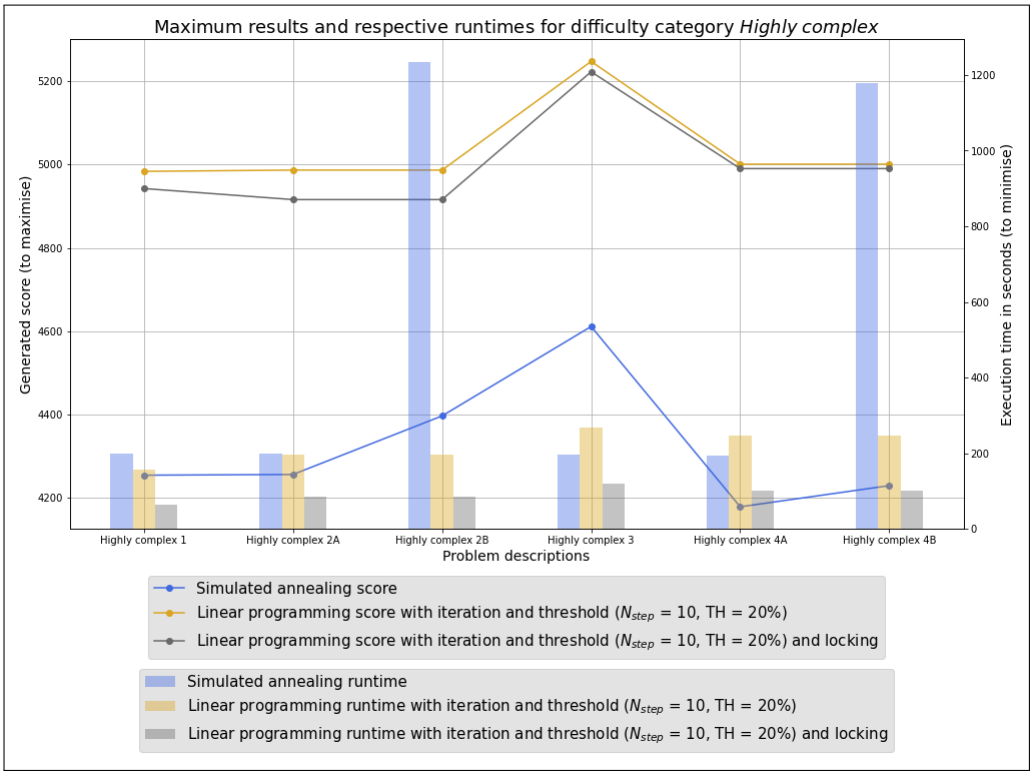

Figure 4.6: *Maximum scores and respective runtimes for the Highly Complex difficulty category*

the last shelf) yield slightly worse results but in a much faster manner. Compared to simulated annealing, applying the iterative process with parameters $N_{step} = 10$ and $M = 20$ as well as locking some products on specific shelves produced on average a 15% increase in the result in less than half the time. Even if these are most likely not the true optimal results for the original problems, the approach used for obtaining them provides a way of improving the end result in both the score and runtime.

Chapter 5

Discussion

As was presented in chapter 4, promising results were obtained for all of the difficulty categories presented. The split between these categories was made based on the effort required to solve the problems with linear programming, ranging from *Trivial* all the way to *Highly complex*. Even if we cannot have certainty about the optimality of the results generated for the last two groups of problems, we can be overall satisfied of the improvements achieved. In comparison to the current way of solving them utilising simulated annealing, the generated results were greater for all of the presented problems and the time used in the solving process was lower for all of the problems except for one of the *Difficult* problems. Though we can be satisfied that with linear programming we could obtain significant improvements, we cannot help but wonder if it would be possible to further enhance this method for even higher quality results.

One method for possibly achieving better results in a lower amount of time would be iteratively test different parameter combinations when solving some of the largest problems in terms of variables presented in them. Even if some combinations were tested out during this work, not every combination of N_{step} (number of binary variables added between iterations) and M (threshold benefit value percentage) were tried out, thus limiting us to the best combination of the ones tried out. There might exist some pair of values N_{step} and M for which greater results could be obtained faster or with which one of these would be improved. Even if with this method improvements could be achieved, we could not be certain that any obtained result would be optimal and could try to find alternative ideas that would be able to generate such results.

A second way to try and speed up the result generation process would be to find some clever way to remove additional variables from the problems. In an ideal situation, the removal procedure would not affect the optimal

result and we could obtain the maximal result for all the given problems. This would require, at least for the larger problems, the removal of a large portion of the binary variables, since the current number is so large that finding a solution could take any time from hours upwards. One aspect of the problems that could potentially be looked at in more detail is the fact that only one variable for each product equals to one and the rest equal to zero, which can be formulated as

$$\sum_{t \in S} \sum_{f \in m_{qt}} x_{qtf} = 1,\ q \in P,\ x_{qtf} \in \{0, 1\}. \tag{5.1}$$

Though no use for this piece of knowledge was found, it could provide some way of either removing a proportion of the variables or creating some additional constraints that could cut down the plausible combinations of product placements in the final planogram.

Thirdly, a possible approach that could be looked closer is the variables relaxation presented in section 4.3.2. It is clear that having a problem where values are only bound by non-negativity achieves a solution immensely faster than one where they are bound to binary values. Even though the problem *Highly complex 4* took the most time to solve even after the data processing, the iterative approach and threshold variable removal, it only took roughly a second to solve once all of the variables were relaxed to non-negative real values. The provided solution is in no way an acceptable one to the original problem, since even though most of the constraints can be met, we cannot formulate some of them in a way that would keep an order in the product placement (such that product q would be placed before product $q+1$). This is because the relaxation provides the capability of placing products on multiple shelves with multiple different facings, due to the lack of being able to restrict this behaviour in any way without adding binary values to the problem constraints. Hence, it is clear that in case clever constraints could be added to the problem formulation, we could achieve the optimal solution in a fraction of the time as was done with the iterative approach described in section 4.5.

One last way that could be exploited to come up to a better solution time or result would be to create a totally new algorithm, similar to the iterative one in section 4.5. Being able to relax the problem formulation in some way that the problem could be partially solved at each step could provide results faster. Unfortunately in our case it happened at the cost of not being able to find the true global optimum for the problems utilising this technique. Nevertheless, coming up with a totally new approach that could partially tackle the huge amount of complexity in the most difficult categories could

provide either a better score or a faster score generation depending on the core idea behind it. In case such approach cannot be discovered, it is possible to look into a completely different solving method for tackling the problems, such as Karmarkar's algorithm or even non-linear programming. Though in this work these methods were not looked into, they could provide a way to tackle the problem in a way that would perform in a superior way compared to linear programming.

In addition to the different methods that could be looked into presented in this chapter, there are other tools that could be utilised in future applications of this work in order to possibly find the true global optimum for all of the problems or get to a solution in a faster manner. It was mentioned in section 3.3 that the open-source solver used in this work provides the ability of utilising parallelism. Thus, depending on the number of threads available, the runtime of the solver could be dropped significantly. It is still good to note that in case a problem has a vast amount of binary variables, no amount of computational resources can lower the solution generation process enough, even with parallelism, due to the number of combinations possible in a given binary linear programming problem.

On top of utilising parallelism, we mentioned the use of commercial solvers in section 3.1. This is another aspect that could provide a faster way to obtain a solution, since one of the commercial solvers on the market named Gurobi [22] claims its superiority over its competitors in the solution generation time. Utilising a more powerful solver with parallelism would definitely enhance the time utilised by the program for reaching an optimal solution, thus providing a platform to tackle even more complex issues that were presented in this work. It is clear that adding parallelism and a more powerful solver could only enhance the time aspect of the problem solving process. In case these tools would be used with some cleverly designed algorithm capable of coming up with a better solution, it could be more valuable to a supply chain company. The reason for this is that the better the generated results are, the more optimal a given planogram can be, further influencing customer purchase behaviour. Obviously, the more the customer behaviour is influenced towards making a purchase decision, the more revenue it can possibly provide for the given place of business.

Chapter 6

Conclusions

A planogram is a representation of a real world fixture in which a set of products are placed on a given number of shelves. A product can be placed multiple times in succession, referred to as the number of facings of a product. This possibility of placing multiple facings for a product further increases the number of possible combinations in which the set of products can be placed onto the shelves. Some combinations are better than others, since the way they are presented can be linked to sales figures or can influence customer behaviour. Ideally the most optimal planogram could be built based on the best combination of predefined values, which represent the benefit of placing an arbitrary number of facings of a product on a given shelf.

In this work, we have built a linear programming approach in order to replace the current one utilising simulated annealing. Various different methods were tested and compared, from the most basic binary linear programming implementation to an iterative one where a proportion of the variables are relaxed and the problem is partially solved at each step. In addition to testing out different methods for solving the presented problems of varying sizes, the removal of both redundant variables and ones having only a small impact to the result was tested. By removing redundant variables, ones that could never be present in an optimal solution, we were able to achieve the optimal result faster. By dropping variables which contributed only a fraction to the end result compared to others came with a trade-off between the achieved score and the runtime of the algorithm. Thus, for most of the problems the runtime was faster but the value of the generated solution was lower in comparison to the linear programming approach in which variables with a small additional value were not removed.

Even if we cannot have certainty about the optimality of the results obtained for some of the problems with the iterative method presented in section 4.5, we can be satisfied of the improvement provided by linear programming

over simulated annealing. For the first three difficulty categories presented in this work, ranging from *Trivial* to *Normal*, the optimal results were achieved on average in a fraction of the time than what was utilised by simulated annealing. Over a 30% increase in the result was obtained 5 to 70 times faster on average, depending on the group of problems. The *Normal* difficulty category was the last which could be solved optimally in such a fast manner after which the iterative approach was utilised, due to the runtime increasing exponentially in relation to the binary variables present in the problems.

Utilising the iterative approach provided a way to solve the more complex problems faster on average, compared to simulated annealing, though it did not necessarily yield the true optimum. By removing some proportion of low impact variables from the *Normal* problems and applying the iterative method, could we trade-off 2% of the optimal score to achieve almost a 5 times faster runtime in comparison to the approach in which only the preprocessing presented in section 4.4 was applied. Applying these same techniques to the problems in the two most challenging difficulty categories, were we able to achieve on average 10% higher results in up to 10 times less effort, depending on which simulated annealing approach we compare the results to (shorter runtime with lower score or higher runtime with higher score).

By observing the results presented it is clear that linear programming performs better than simulated annealing, by producing better results (compared in objective function values) in a shorter period of time. Thus, linear programming could replace the current process used for the generation of the planograms. Even though there remains some uncertainty for some proportion of the presented problems, on how far from the true optimum we were, the results are outperforming the simulated annealing method. While we can be satisfied of the outcome obtained with linear programming, there is still room for further improvements. In case further enhancements would be necessary, some of the ideas presented in chapter 5 could be tested out.

Bibliography

[1] Adler, I., Resende, M. G., Veiga, G., and Karmarkar, N. An implementation of karmarkar's algorithm for linear programming. *Mathematical programming 44*, 1 (1989), 297–335.

[2] Avriel, M. *Nonlinear programming: analysis and methods*. Courier Corporation, 2003.

[3] Bellare, M., and Rogaway, P. The complexity of approximating a nonlinear program. In *Complexity in numerical optimization*. World Scientific, 1993, pp. 16–32.

[4] Bixby, R. E. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, 2012 (2012), 107–121.

[5] Bläser, M., and Manthey, B. Smoothed complexity theory. *ACM Transactions on Computation Theory (TOCT) 7*, 2 (2015), 1–21.

[6] Borrelli, F., Subramanian, D., Raghunathan, A. U., Biegler, L. T., and Samad, T. A comparison between mixed integer programming and nonlinear programming techniques for 3d conflict resolution of multiple aircraft. *Department of Aerospace Engineering and Mechanics, University of Minnesota, Minneapolis, Tech. Rep* (2003).

[7] Carnegie Mellon University. Lecture 18: LP algorithms and Seidel's algorithm, October 2017.

[8] Clausen, J. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen* (1999), 1–30.

[9] COIN-OR Foundation. COIN-OR Projects, 2016. `https://www.coin-or.org/projects/#ffs-tabbed-110`. Accessed 23.9.2021.

[10] COIN-OR Foundation. Computational infrastructure for operations research, 2016. `https://www.coin-or.org/`. Accessed 26.9.2021.

[11] Dantzig, G. B. *Linear Programming and Extensions*. Princeton University Press, 1963.

[12] Dantzig, G. B., Orden, A., and Wolfe, P. Notes on linear programming: Part 1. the generalized simplex method for minimizing a linear form under linear inequality restraints. Tech. rep., RAND CORP SANTA MONICA CA, 1954.

[13] Dantzig, G. B., and Thapa, M. N. *Linear programming 1: introduction*. Springer Science & Business Media, 2006.

[14] Dantzig, G. B., and Thapa, M. N. *Linear programming 2: theory and extensions*. Springer Science & Business Media, 2006.

[15] Darly Solutions. The most popular programming languages in 2021, 2022. `https://darly.solutions/the-most-popular-programming-languages-in-2021/`. Accessed 7.2.2022.

[16] Dobkin, D. P., and Reiss, S. P. The complexity of linear programming. *Theoretical Computer Science 11*, 1 (1980), 1–18.

[17] Fearnley, J., and Savani, R. The complexity of the simplex method. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing* (2015), pp. 201–208.

[18] for Numerical Analysis (US), I. *Problems for the Numerical Analysis of the Future*, vol. 15. US Government Printing Office, 1951.

[19] Gendreau, M., and Potvin, J.-Y. *Handbook of Metaheuristics*, vol. 146. Springer Science and Business Media, 2010.

[20] Gomory, R. E. Outline of an algorithm for integer solutions to linear programs and an algorithm for the mixed integer problem. In *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 77–103.

[21] Gonzaga, C. C. An algorithm for solving linear programming problems in $O(n^3L)$ operations. In *Progress in mathematical programming*. Springer, 1989, pp. 1–28.

[22] Gurobi Optimization. Gurobi, 2022. `https://https://www.gurobi.com/`. Accessed 8.2.2022.

[23] Gurobi Optimization, LLC. Mixed-Integer Programming (MIP) - A Primer on the Basics. `https://www.gurobi.com/resource/mip-basics/`. Accessed 24.1.2022.

[24] Hahn, C. K., Duplaga, E. A., and Hartley, J. L. Supply-chain synchronization: lessons from hyundai motor company. *Interfaces 30*, 4 (2000), 32–45.

[25] Hochbaum, D. S. Complexity and algorithms for nonlinear optimization problems. *Annals of Operations Research 153*, 1 (2007), 257–296.

[26] Huo, B. The impact of supply chain integration on company performance: an organizational capability perspective. *Supply Chain Management: An International Journal* (2012).

[27] Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G., and Wolsey, L. A. *50 Years of integer programming 1958-2008: From the early years to the state-of-the-art*. Springer Science & Business Media, 2009.

[28] Jupyter. Jupyter, 2022. `https://jupyter.org/`. Accessed 9.2.2022.

[29] Karp, R. M. On the computational complexity of combinatorial problems. *Netw. 5*, 1 (Jan 1975), 45–68.

[30] Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., and Dilkina, B. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2016), vol. 30.

[31] Kirkpatrick, S., Gelatt Jr, C. D., and Vecchi, M. P. Optimization by simulated annealing. *science 220*, 4598 (1983), 671–680.

[32] Land, A., and Doig, A. An automatic method of solving discrete programming problems. *Econometrica 28*, 3 (1960), 497–520.

[33] Lawler, E. L., and Wood, D. E. Branch-and-bound methods: A survey. *Operations research 14*, 4 (1966), 699–719.

[34] Luenberger, D., and Ye, Y. *Linear and Nonlinear Programming*. Springer Science & Business Media, 2008.

[35] Megiddo, N. On the complexity of linear programming. In *Advances in economic theory: Fifth world congress* (1987), Cambridge University Press London, pp. 225–268.

[36] Munapo, E. Solving the binary linear programming model in polynomial time. *American Journal of Operations Research 6*, 1 (2016), 1–7.

[37] National Institute of Technology Jamshedpur. Linear programming: The simplex method, n.d. `http://www.nitjsr.ac.in/course_assignment/CA02CA3103%20RMTLPP%20:%20Simplex%20Method.pdf`. Accessed 16.01.2022.

[38] numFOCUS. Pandas, 2022. `https://pandas.pydata.org/`. Accessed 9.2.2022.

[39] NumPy. NumPy, 2021. `https://numpy.org/`. Accessed 9.2.2022.

[40] Open source for the Operations Research Community. COIN-OR Branch-and-Cut solver, 2022. `https://github.com/coin-or/Cbc/`. Accessed 15.2.2022.

[41] Pataki, G., Tural, M., and Wong, E. B. Basis reduction and the complexity of branch-and-bound. In *Proceedings of the twenty-first annual ACM-SIAM symposium on discrete algorithms* (2010), SIAM, pp. 1254–1261.

[42] PuLP Documentation Team. How to configure a solver in PuLP, 2009. `https://coin-or.github.io/pulp/guides/how_to_configure_solvers.html`. Accessed 8.2.2022.

[43] Python Software Foundation. PuLP, 2021. `https://pypi.org/project/PuLP/`. Accessed 23.9.2021.

[44] Rutenbar, R. A. Simulated annealing algorithms: An overview. *IEEE Circuits and Devices magazine 5*, 1 (1989), 19–26.

[45] Shamir, R. The efficiency of the simplex method: A survey. *Management science 33*, 3 (1987), 301–334.

[46] Spielman, D., and Teng, S.-H. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Proceedings of the 33rd Annual ACMSymposium on Theory of Computing* (7 2001), 296–305.

[47] Statista. Most used programming languages among developers worldwide, as of 2021, 2022. `https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/`. Accessed 7.2.2022.

[48] StatisticsTimes. Top Computer Languages, 2021. `https://statisticstimes.com/tech/top-computer-languages.php`. Accessed 7.2.2022.

[49] The Matplotlib Development team. Matplotlib: Visualization with Python, 2021. `https://matplotlib.org/`. Accessed 9.2.2022.

[50] Thirumalazhagan, D., and Nithya, G. The influence of store atmosphere and planogram on customer repurchase intention. *International Journal of Advanced Research in Management and Social Sciences 9*, 2 (2020), 6–11.

[51] Van Leeuwen, J. *Handbook of theoretical computer science (vol. A) algorithms and complexity*. Mit Press, 1991.

[52] Zhang, W. Branch-and-bound search algorithms and their computational complexity. Tech. rep., University of Southern California Marina Del Rey Information Sciences inst, 1996.

[53] Zhao, L., Huo, B., Sun, L., and Zhao, X. The impact of supply chain risk on supply chain integration and company performance: a global investigation. *Supply Chain Management: An International Journal* (2013).