Optimal Configurations for Modular Systems

Zur Erlangung des akademischen Grades
einer Doktorin der Wirtschaftswissenschaften

Dr. rer. pol.

von der KIT-Fakultät für Wirtschaftswissenschaften
des Karlsruher Instituts für Technologie (KIT)

genehmigte **Dissertation**
von

Maren Beck, M.Sc.

Tag der mündlichen Prüfung:	11.02.2026
Hauptreferent:	Prof. Dr. Oliver Stein
Korreferent:	Prof. Dr. Stefan Nickel

Karlsruhe (2026)

Abstract

This work investigates how to determine optimal configurations for modular systems, a concept widely used in mechanical engineering to improve logistics and manufacturing processes. Although modular systems are common in industrial applications, systematically deriving an optimal configuration with methods of mathematical optimization remains an open challenge.

After introducing modular systems and reviewing relevant literature, we develop a mathematical optimization model that captures the structural relationships between components, variants, and the number of pieces in the modular system. Because the number of variants for the components is itself a decision variable, the initial formulation leads to a non implementable optimization model. To address this, we derive in a next step a well-defined mixed-integer optimization problem (MIP) that deals with the unknown number of variants. Although state-of-the-art MIP solvers such as Gurobi can solve MIPs with a large number of variables efficiently, numerical experiments reveal high computational effort for the modular system instances considered here, due to their strong combinatorial and logical structure.

To exploit this structure, we propose decomposition-based solution methods and analyze their numerical behavior. Motivated by these results, we introduce discrete functions as an alternative representation and examine discrete convexity concepts, including one newly developed in this work. Building on this, we adapt several derivative-free optimization techniques, such as Steepest Descent and Coordinate Search, and an adaption of the Nelder–Mead method to discrete functions.

Acknowledgements

My journey into optimization began almost ten years ago, in the summer of 2016, during a conversation with Nathan. That conversation eventually led me to Oliver's chair, and after attending my first lecture on parametric optimization in the winter term, I somehow never found my way out again. After several courses, seminar papers, and student assistant positions, I became involved in teaching while still working on my master's thesis. Looking back, it was a very rewarding, though often challenging, time.

My first and thanks go to you, Oliver. You showed me how the mathematical and logical thinking developed during my studies in mathematics can be applied in practice. Thank you for your excellent academic support and guidance in shaping the ideas behind this work. Thanks for your calmness and motivation that helped turn this project into a coherent piece of work and for encouraging my continued enthusiasm for optimization, even when things did not always go as planned.

My next thanks go to you, Nathan, for bringing me to Oliver's chair nearly ten years ago and for your practice-oriented perspective. Who would have thought that I would actually end up writing a dissertation? Your influence has shaped my path in ways I could not have anticipated. I look forward to continuing our collaboration at DHBW Mannheim and who knows what will come next.

I would also like to thank my current and former colleagues Stefan, Christoph, Paula, Christian, Robert, Felix, Marion, Martina, and Michaela for the great time at the chair: for the shared moments on the basketball court, for countless coffees and for our lunches on Gutenbergplatz.

Finally, my last thanks goes to my parents Claudia and Roland, to my brother Ricco, to Hanna, and to Flo. Thank you for catching me whenever I could no longer see the light at the end of the tunnel, for your unwavering support, and for continually motivating me throughout this journey.

I hope to stay in the optimization bubble for a long time to come. Whether in a job with an optimization background or in further teaching activities, so that I can show as many students as possible why everyone needs optimization in their lives.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The original idea for this work was developed as part of a collaboration with the Institute for Material Handling and Logistics (IFL) at the Karlsruhe Institute of Technology (KIT). As the title “Optimal Configurations for Modular Systems” suggests, this work deals with modular systems, which play an important role in mechanical engineering in industrial applications. Particularly in the field of design and logistics, the goal is to examine and improve existing structures [35]. Modular systems for the construction of products such as cars often offer an improvement, as they can provide great potential for improvement in the areas of transport and manufacturing [14, 62]. In order to take advantage of modular systems, the central question is what a good modular system should look like for desired products.

This is where optimization becomes relevant. “Optimization [or mathematical optimization] is a technology that can be used to devise effective decisions in a variety of contexts.” [8, 22] First, a mathematical model must be formulated that represents the desired situation as precisely as possible. We refer to such a model as an optimization model. In the next step, the optimization model should be solved using suitable numerical methods. If this is possible and successful, we end up with a best decision for the model. In the context of modular system problems, the best decision is an optimal configuration for the modular system. If the modular system has an optimal configuration, the costs incurred are as low as possible and all necessary constraints, such as resource limits, are met. In this work, we consider the determination of such an optimal configuration using methods from mathematical optimization.

In Chapter 2, we first provide an introduction to modular systems and present some examples. These are largely taken from the field of mechanical engineering. We present a modular system for crane bridges that was developed jointly by the IFL and Dietrich GmbH [6, 49]. This example will be revisited throughout many chapters of this work and is therefore examined in detail in this introductory chapter. After introducing the examples, we provide a literature review with a focus on specific optimization models that are intended to deliver optimal configurations of modular systems. It turns out that, although there are many applications that use modular systems, it remains unclear how to achieve an optimal configuration for the required modular system. This is therefore one of the goals of this work.

In Chapter 3, we set up an initial optimization model. The products to be built consist of a certain number of components, which may be available in different variants, e.g., sizes, in the modular system. As part of the optimization process, we look for the exact specifications of the variants, the number of variants per component, and the total number of pieces in the modular system. As described above, we want to use numerical methods to determine the optimal configuration of the modular system. However, such methods usually require a precise determination of how many variables the model is based on. This is not given here, since the number of variables for the specifications depends, for example, on the unknown number of variants.

In Chapter 4, we present a formulation of the optimization model that takes this dependency structure into account. The result is a mixed-integer optimization model for which the number of variables is clearly defined. For this reason, we refer to this as a mixed-integer optimization problem (MIP). In the optimization community MIPs and software for solving them (so-called solvers) are well known nowadays. Theoretical solution methods for MIPs were first mentioned in the 1950s [33, Part I], and the first solvers in the 1980s [34, Part III]. Today, there are many efficient solvers on the market that can be used depending on the problem formulation even if the optimization problems have many variables and complicating constraints. In this work, we choose the solver Gurobi, which has been available since 2008 and has been continuously developed since then [25]. In experiments with the running examples, it turns out that the solution time is very high, even though Gurobi is considered as one of the fastest solvers [26]. Since the derived optimization problem contains a large combinatorial and logical factor, we want to take advantage of this with the assumption that the problems can be solved faster this way.

Decomposition methods are known for their ability to handle optimization problems with complicating structures. The optimization problem derived in Chapter 4 has such a structure. Based on the decomposition idea, we derive an alternative solution method. In the following numerical tests, we notice that there is still potential for improvement in terms of runtime. Nevertheless, we find that the decomposition idea can be linked to so-called discrete functions, which is very promising. Instead of the numerical method derived in this chapter, we can also minimize a discrete function. In Chapter 5 we first provide an introduction to discrete functions. In the next step, we consider convexity concepts for discrete functions. In the context of optimization, convexity concepts are typically used in continuous optimization, i.e., for optimization problems without integer constraints. Since we want to minimize discrete functions, we must consider convexity concepts that are adapted to discrete functions. One of the concepts that we present is newly developed in the context of this work.

Since the runtime of numerical methods continues to be a challenge, we consider in Chapter 6 further approaches whose basic ideas lie in continuous derivative-free optimization. This idea came from the introduction of discrete functions which do not have derivatives in the classical sense. Two of the derivative-free methods can be transferred to discrete functions with little additional work (the Steepest Descent method and Coordinate Search). Another well-known and efficient method for continuous derivative-free optimization is the Nelder-Mead method [10]. As part of this work, we developed an adaptation of the method for discrete functions. Not all of the numerical methods considered here are guaranteed to terminate with an optimal point from which the optimal configuration of the modular system could be obtained. However, under convexity assumptions, we can at least conclude the desired result for the Steepest Descent method.

The main structure of this work is shown in Figure 1.1 and Figure 1.2. The important chapters are illustrated in blue boxes, the important sections in violet boxes. Motivation and explanations between chapters and sections are depicted in olive ones. The parts of the work that are based primarily on new ideas not found in the literature are marked with a double box border. Arrows for connections between chapters and sections are shown as solid lines. If there are further explanations between them, we use dashed arrows for presentation purposes.

Parts of this work have already been published in the article [3]. The article was joint work with Steffen Bolender (IFL) and Oliver Stein. In Chapter 2, this concerns Section 2.1, Subsection 2.2.3, and Section 2.3. In Chapter 3 this relates to the general optimization model in Section 3.1 and the application to the crane bridge example in Subsection 3.2.1. The adopted text content was written by me and the sketches were originally designed by Steffen Bolender and revised by me.

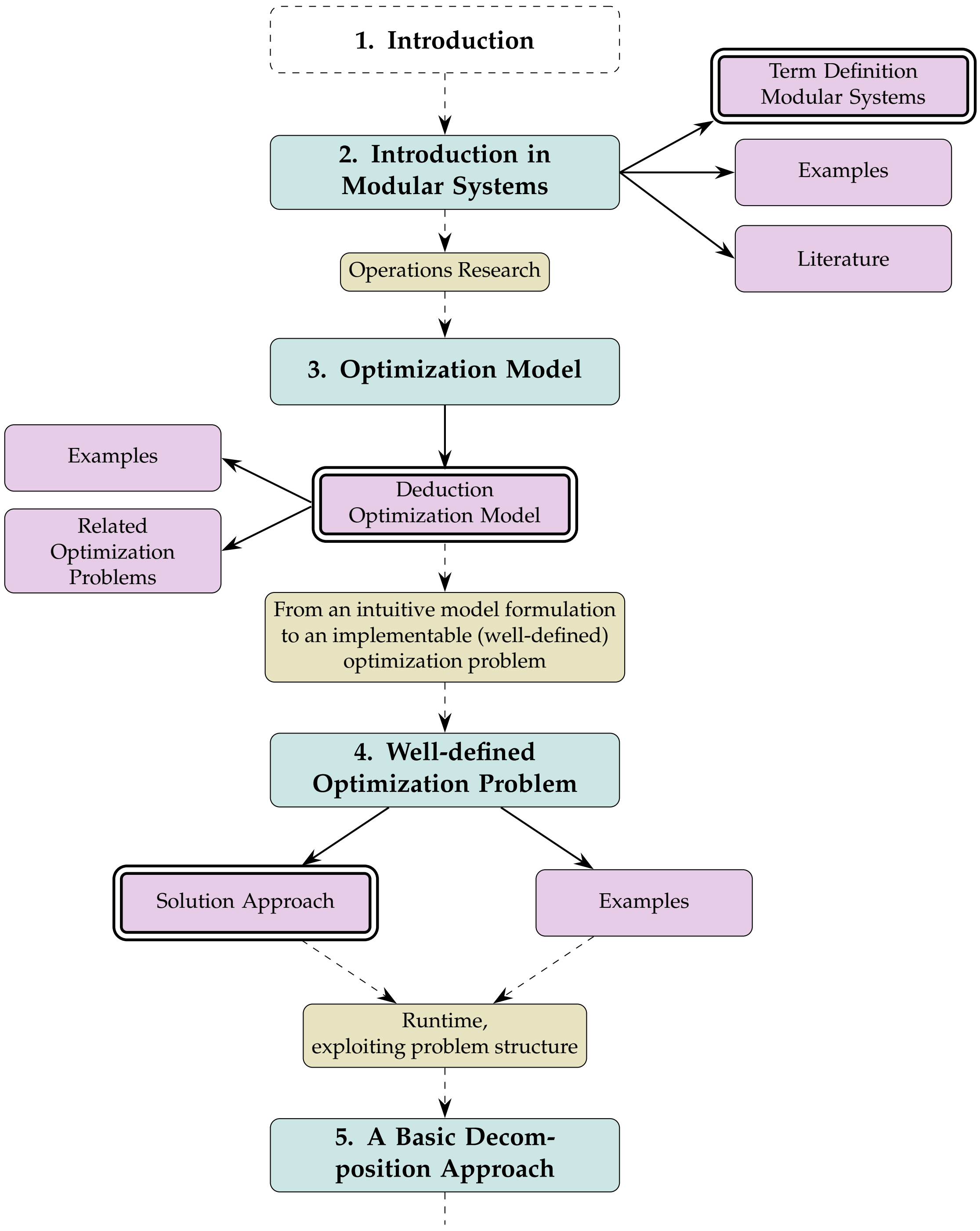
1. Introduction
Term Definition Modular Systems
2. Introduction in Modular Systems
Examples
Literature
Operations Research
3. Optimization Model
Examples
Deduction Optimization Model
Related Optimization Problems
From an intuitive model formulation to an implementable (well-defined) optimization problem
4. Well-defined Optimization Problem
Solution Approach
Examples
Runtime, exploiting problem structure
5. A Basic Decom- position Approach

FIGURE 1.1: Overview of the work, Part 1

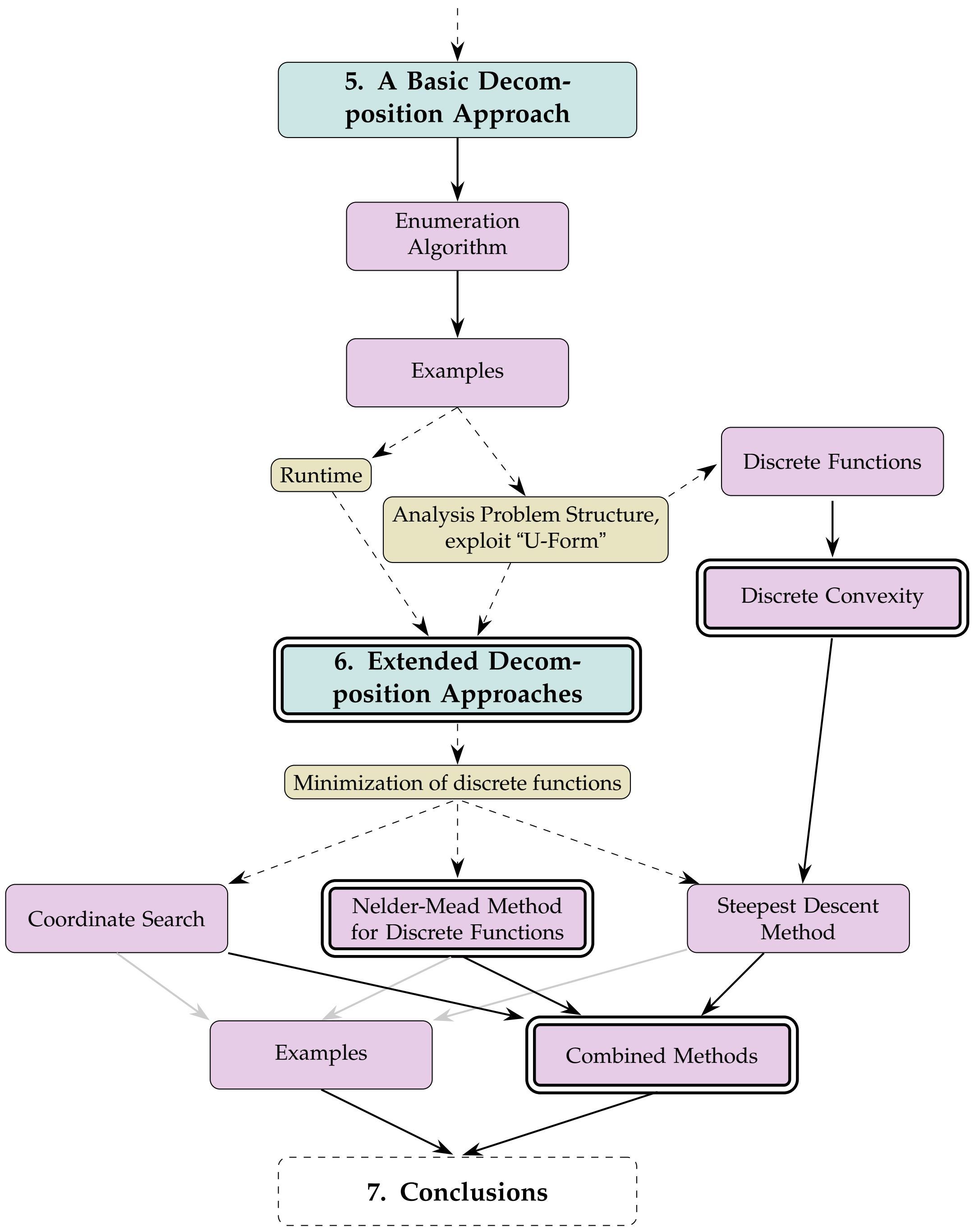
5. A Basic Decomposition Approach
Enumeration Algorithm
Examples
Runtime
Analysis Problem Structure, exploit "U-Form"
Discrete Functions
Discrete Convexity
6. Extended Decomposition Approaches
Minimization of discrete functions
Coordinate Search
Nelder-Mead Method for Discrete Functions
Steepest Descent Method
Examples
Combined Methods
7. Conclusions

FIGURE 1.2: Overview of the work, Part 2

Chapter 2

Introduction in Modular Systems

In this chapter, we give a general introduction to modular systems and discuss different examples and applications. In Section 2.1 we introduce modular systems in general. Then, we consider different examples in Section 2.2. The first two examples are modular systems that are frequently seen on the market. One is known from the automotive sector (Subsection 2.2.1), the other one from the construction industry (Subsection 2.2.2). The third and fourth examples are of high importance for the further work, as they will be revisited in all subsequent chapters. In Subsection 2.2.3, we introduce another example with technical background, namely the modular design and modular systems for crane bridges. Since we consider modular systems problems in this thesis from the perspective of mathematical optimization, we introduce in Subsection 2.2.4 an example that is well known in the context of mathematical optimization, namely a binpacking problem. We can adapt this example slightly to the context of modular systems. In the end, in Section 2.3, a detailed literature review on modular systems is given.

2.1 Modular Systems

Modular systems are proved to be very useful in practice for products that are assembled modularly from different components. The approach of modular systems, which can also be found in literature with the keyword modular design, is useful and suitable if a product can be subdivided into different parts. We call the parts components in this work. Each of the components can consist of different variants or sizes that can be combined with each other. Modular systems are a common concept in many fields of design and manufacturing, where the goal is to decompose a complex system into simpler modules in order to decrease complexity and increase cost-efficiency [59]. Through this concept, products can be highly individualized for specific users, and products have a well-structured design [51].

The different components for one product should be created independently [59]. An advantage of this independency is, for example, the reduction of dependency on suppliers when different components can be produced from various producers. Another advantage is that huge products, such as crane bridges, which are considered in the following, can be assembled at the final place of use. This reduces logistic and transportation costs. Through the combination of components, products with different required properties can be built, for example, products with different sizes or products with more stability.

One of the main challenges in the design of modular systems is to balance the system size and the product performance [59]. Increasing the number of different variants of individual components on the one hand decreases the cost of oversizing the assembled product, while on the other hand, the cost of maintaining the modular system increases. Maintenance costs of a modular system can, for example, be measured in the number of different variants for each component or in the size of the required warehouse area.

In general, it is important to mention that the terminology differs in the literature, especially what are variants, components, and modules. Through that we clearly fix them here and mention that in the literature, as, for example, a variant in [51] is not the same as a variant in this work. Some of these discrepancies might come from translations, since a lot of literature on the mechanical engineering side is written in German. We will discuss some of these points in the literature review in Section 2.3. But first, we consider different examples for modular systems to better understand the topic.

2.2 Examples for Modular Systems

In this subsection, we will look at three specific applications of modular systems that are used in practice and an example from the context of mathematical optimization. We start with an example that probably comes to mind very quickly when we think of modular systems, namely modular systems in the automotive industry.

2.2.1 Modular Systems in the Automotive Industry

Modular systems have been used in the automotive industry for many years. For car buyers, diversity means having a wide range of choices or being able to add a lot of individuality. From a small and agile car to a large family car. However, for manufacturers, this diversity means high costs, as each individual development is expensive and time-consuming. Many components have to be redesigned. It is therefore worthwhile for car manufacturers to use the technology as widely as possible [2].

The modular transverse modular system developed by the VW Group (german: Modularer Querbaukasten, MQB) is the most successful technology platform of VW [62]. This system celebrated its 10th anniversary in 2022. More than 32 million vehicles were produced within this technology. For example, the Polo, a small, agile car, and the Tiguan, a family car, are part of the MQB. These and many other models, for example, also models from Seat or Audi, that are brands that also belong to the VW Group, are based on the same architecture. In all these vehicles, the position of the engine is the same. This is installed transversely in the vehicle, hence the name of the modular system. For example, the VW Polo and the VW Tiguan can be fitted with the 1.5-liter TSI engine. The component engine is available in different variants in the MQB. For some models, for example, more powerful or less powerful engines are also compatible, but they are all installed in the same way and have identical installation positions [43]. The products VW Polo or VW Tiguan with the corresponding engine are then created by combining the individual components. Since components are installed in many models, development costs, manufacturing costs, and inventory costs are saved. However, this

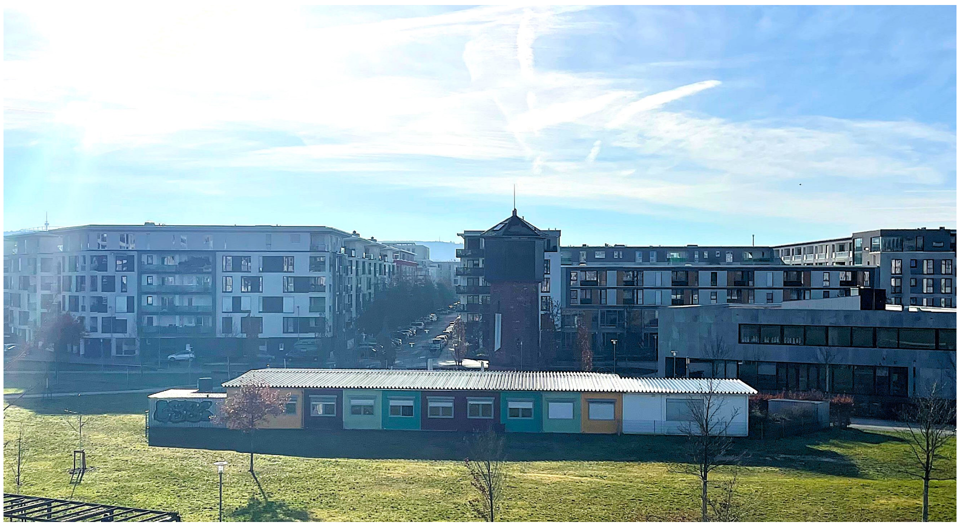

FIGURE 2.1: Container building childcare center Karlsruhe Citypark [9]

design naturally carries the risk that if a component installed millions of times has a fault, thousands of customers will suffer directly [2].

In addition to the automotive sector, there are also several modular system approaches in construction in the aircraft sector that work similarly, for details, see, for example, [52]. Next, we consider an example from the construction industry.

2.2.2 Modular Construction and Container Buildings

As a second example, we consider modular construction for buildings. "The modular design has been enjoying increasing popularity for some time." [41]

According to [41], modular construction is generally suitable for new buildings, renovations, or even extensions of buildings. Buildings whose units consist of containers are referred to as container buildings. Container buildings that are often found in many cities are usually intended as a short-term solution [41]. In Karlsruhe, several container buildings can be found in the cityscape of the Citypark. These are used as childcare centers or nurseries. Figure 2.1 shows a photo of a childcare center in the Citypark district of Karlsruhe.

Office buildings, refugee shelters, kindergartens, and schools are particularly well suited to modular construction, as they consist of many identical units. There are various suppliers on the market from whom suitable containers or modules can be purchased or rented ([39], [40]).

In the following, we take a closer look at a modular system for nurseries. This consists of individual components for the different rooms or units. There are containers for group or bedrooms, containers with sanitary units, containers for eating areas, or necessary intermediate elements such as corridors, stairs, etc. ([37]). Individual components are available in different variants. For example, a group room can be delivered with different furniture or a sanitary container with a different number of sinks. Individual containers are designed so that they can be connected and stacked on top of each other.

This construction method allows buildings to be built very quickly ([41]). If the units are built in standard containers, they can be easily transported to the location

FIGURE 2.2: Illustration for a modular construction of a building, [38]

by train or truck, and the building can be assembled quickly. Many suppliers also advertise that this construction method is very sustainable. If, for example, the container building is only an interim solution until another building is completed, the containers can be separated from each other again, as they are normally plugged or screwed together and not glued or welded([4]). They can then be transported away individually and reused after thorough cleaning or refurbishment. In addition, containers are usually made of steel, which is considered a very sustainable material, see [4] for more details.

There are also modular buildings that are not made of containers. These are considered an alternative to solid construction. According to [4], this construction method is an alternative with comparable quality to solid construction, and the buildings are usually completed up to 70 % faster. Figure 2.2 shows a modular building that is not made of containers.

Next, we will look at a not less well-known application, namely the modular design of crane bridges. This application will occur frequently in this work and therefore plays an important role. For this reason we consider the modular system for crane bridges in Subsection 2.2.3 in more detail than the above examples.

2.2.3 Modular Construction of Crane Bridges

Overhead cranes are mainly used to transport objects in production halls and warehouses. They consist of the crane bridge, the crab with hoist and trolley, and two end carriages at the ends of the crane bridge. The end carriages travel on the crane runway. Figure 2.3 shows a warehouse with an overhead crane and the crane bridge (in yellow).

End carriages, hoists, and trolleys are already offered by the respective manufacturers as modular systems. Crane bridges are usually built in one part in the shape of a box or I-profile girder, as in Figure 2.3. These are manufactured in series of various sizes. As warehouses are often very large and crane bridges are often built, as described above, in one part, the I-profile girders are very long. This is not suitable for transportation, so that special transports are needed. Such transports are often expensive and inconvenient.

In order to make the advantages of a modular system usable for crane bridges as well, a concept was developed in [6] and [49] on how crane bridges can also be constructed modularly. This segmented crane bridge is designed as a truss profile made of hollow profiles and diagonal connecting sheets. The components can be

FIGURE 2.3: Overhead crane with crane bridge (in yellow), [12]

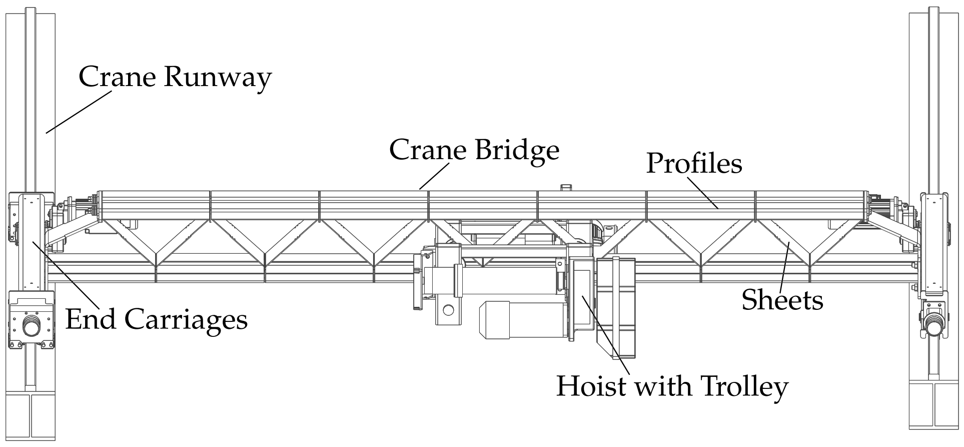

FIGURE 2.4: Crane concept with segmented crane bridge

mass-produced, easily transported and assembled at the crane's place of use. A detailed sketch of an overhead crane with a segmented crane bridge and the crane runway, end carriages, and a trolley is shown in Figure 2.4. A 3D-CAD illustration of the crane bridge is shown in Figure 2.5. Details about the construction and stability requirements and stability studies are explicitly explained in the papers mentioned above. The great advantage of transporting a segmented crane bridge is shown in Figure 2.6, where a disassembled crane bridge can be seen on only five Euro pallets. The assembled bridge in total has a length of 20 meters, but the pallets make the transportation process much less complicated.

The crane bridge modular system consists of the two components profile and sheet, each of which is designed as a series in different sizes, as well as matching end sheets, connecting elements and an individually manufactured compensating element. By combining a sheet size and a profile size as well as varying the number of these elements, various requirements and properties of the crane bridge can be covered. Profiles and sheets can differ in various geometry parameters, e.g. in the width or height. When we talk about a combination of these mentioned parameters, we are talking about one variant of the component profile or sheet. We

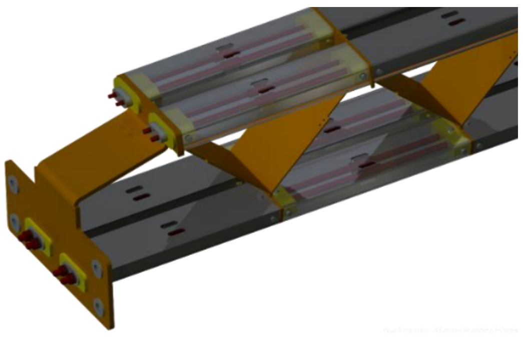
FIGURE 2.5: Cranebridge, detailled 3D-CAD illustration, [6, Figure 1, p. 4]

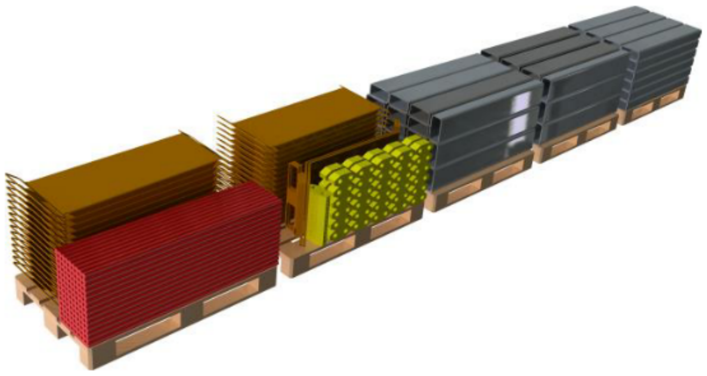
FIGURE 2.6: Assembled crane bridge on euro pallets, [6, Figure 1, p. 2]

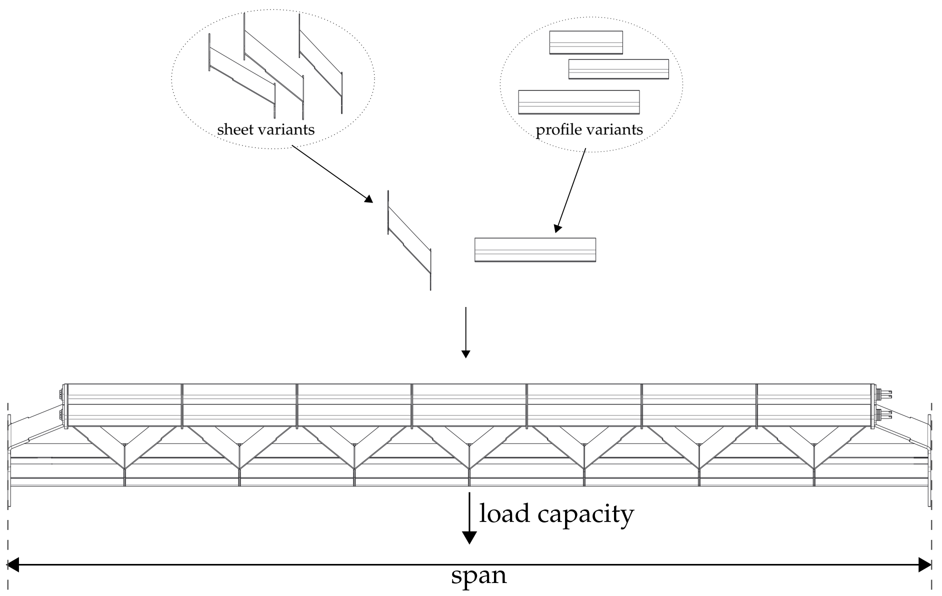

FIGURE 2.7: Sketch of the modular system for crane bridges

consider a demand for crane bridges to be produced, which have a specific span and which should carry a specific known load capacity. Figure 2.7 shows exemplary a sketch of the modular system for the crane bridges with some variants of the components (upper part of the figure) and a crane bridge built in this modular construction (lower part).

We now look at this example from the perspective of the crane bridge manufacturer. The manufacturer wants to create a modular system and knows its demand for specific crane bridges. As a manufacturer of crane bridges, we are interested in a modular system that is as cost-effective as possible and covers the required properties (span and load capacity) as well as possible. On the one hand, we want to avoid a large system size, measured in the number of different variants of the components, and on the other hand, we want to remain flexible in fulfilling the required properties of span and load capacity. In detail, this means that we want as few profile and sheet variants as possible but still want to cover span and load capacity as well as possible and thus want to avoid building oversized crane bridges. In our context, a crane bridge is oversized if the real load capacity is higher than the required one. However, a large number of variants is required for full flexibility.

As results, we want to know first how many different variants of profiles and sheets are necessary in this cost-effective modular system, second how these different variants look like, and third of how many parts the modular system consists of in total.

2.2.4 A Modular Approach for a Binpacking Problem

In this subsection, we look at an example with greater awareness in the field of mathematical optimization, a binpacking problem. This problem appears slightly differently in the literature, but can also be considered in the context of modular systems. First, we introduce the original problem and secondly we consider a binpacking example in relation to modular systems.

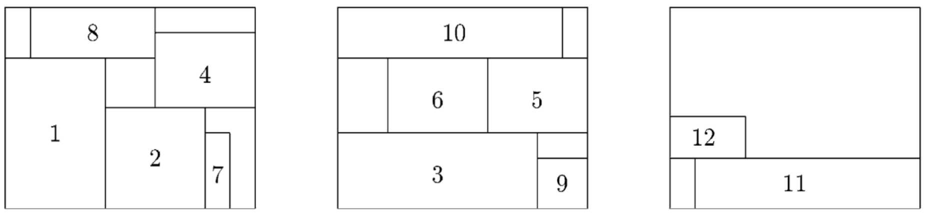

FIGURE 2.8: Two-dimensional binpacking problem, [34, Figure 5]

Binpacking problems occur in the problem class of combinatorial optimization problems. First, we look at the classical binpacking problems, where a lot of literature and algorithms exist. In the one-dimensional binpacking problem, we consider n objects of a given size a_i, $i = 1, \dots, n$, and an infinite number of bins or containers of a given size $A \in \mathbb{R}$, where $a_i \leq A$ for all $i = 1, \dots, n$. We want to assign the small objects in the bins and use as few bins as possible, [31, chapter 18]. In addition to the minimal number of bins, we also look for the allocation of the objects in the bins. Logically, the objects should not overlap and the maximum capacity, here in the form of size A, is not allowed to be exceed. An equivalent problem to the one-dimensional binpacking problem is the cutting stock problem [31]. The two-dimensional binpacking problem is very similar to the one-dimensional problem above. Now an infinite number of bins not only have a size A, but also have a width A and a height B and the objects are small rectangles with known width a_i and height b_i, [34]. We are again looking for a minimal number of bins and an allocation of the objects in the bins without overlapping and exceeding the total capacity. Figure 2.8 shows a feasible allocation of 12 objects. In this example, three bins are necessary. In this context, a feasible allocation means an allocation without overlapping of the objects and without exceeding the bins.

From now on, we consider a similar problem, an approach with a modular system for the binpacking problem. The main idea remains similar as before; we want to put objects in bins. Unlike the original problem, we have a fixed number of bins of different capacities. We measure the capacity of a bin only in its length, as in the one-dimensional case from above. Furthermore, we have objects of unknown length, which should be filled in the bins, again without overlapping and without exceeding the total length. The goal is to fill the bins as well as possible, which means that the sum of the remaining space in the bins should be minimized while using only a few different object lengths. In contrast to the original problem, we do not know the exact number and size of the objects, but we know the number of bins. In the end, the modular system contains the objects of different lengths with the corresponding number required. Besides the exact length of the objects, we also do not know how many variants of objects the modular system contains. As in the modular system for the crane bridges in Subsection 2.2.3 also the size of the modular system is significant and should be minimized.

We also want to introduce the term of components in this example. We consider different types of objects, for example different colors. The colors are easy to illustrate, and in applications, this could be, for example, different materials. This approach of colors or materials can easily be extended in a larger number of components in the modular system in this example. As we consider in this work from now on only the modular system point of view for the binpacking problem, we denote the corresponding problems as binpacking problems, and by that we mean the modular approach of the binpacking problem.

The number of components plays an important role in the subsequent work, so we give it a corresponding name. We define the *dimension* of a modular system as follows:

Definition 2.2.1 (Dimension Modular System)**.** *The **dimension** of a modular system is given by its number of components.*

Therefore, a modular system in the binpacking example with one color has dimension one. If we have a second color, we get dimension two. Through this approach, we can easily increase the dimension of the modular system without making the structure more complicated. The modular system of Subsection 2.2.3 contains through the concrete application two components, profiles and sheets and consequently has dimension two.

Through the binpacking and crane bridge example, we can get a first idea of how modular systems are an interesting topic from the optimization point of view. We now look at the entire situation from the perspective of the developer of the modular system. For example, we are a manufacturer of crane bridges and now want to know what the optimal modular system would look like for us, given a certain demand for crane bridges. We want to minimize costs and we want to know how different variants of the components of the products look like. In the case of the crane bridges, this means in particular, how long and thick the sheets and profiles are for example, and in the binpacking examples, how long the objects are. The last point, which we also consider is very important: we do not know how many different variants for a component are available in the modular system. This number is not known from the beginning.

In VW's MQB in Subsection 2.2.1 and the modular system for container buildings in Subsection 2.2.2, the different variants of the components, e.g., the engine or a sanitary unit, are already specified. So, in this work, we are still one step ahead of that and we are looking at it from the manufacturer's perspective rather than the customer's. As manufacturers, we first want to determine how the specific modular system is configured for a given demand. We want to consider this problem from the perspective of mathematical optimization.

There is only little existing literature for the combination of modular systems with mathematical optimization. The literature for the modular systems from Subsections 2.2.1, 2.2.2 and 2.2.3 is only from the applications point of view and not from the mathematical optimization. The binpacking example of the current subsection was developed specifically for this purpose, but is based on a well-known application from mathematical optimization. In the following section we give a general overview over literature for modular systems. After deriving a specific optimization problem in the following chapter, we will briefly revisit problems in the literature that are similar to what we want to achieve here. However, without specific optimization problems, this does not make sense at this point.

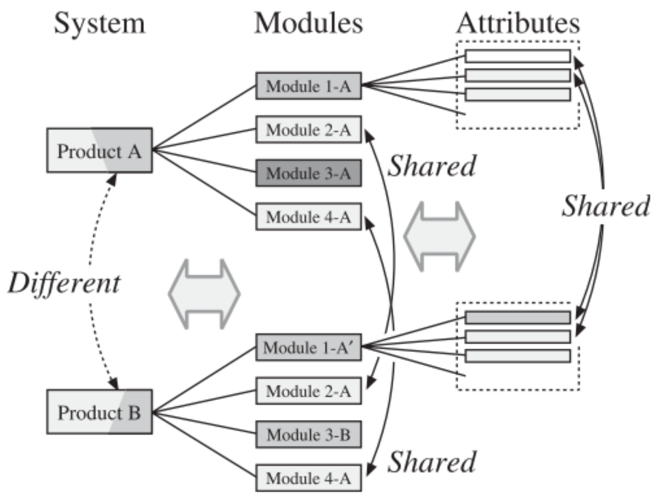

FIGURE 2.9: Product variety design, Fujita [19]

2.3 Existing Literature for Modular Systems

The aim of this thesis is to find the optimal configuration of a modular system using mathematical optimization. At first, it is not clear how many variants of the components exist and how the variants look like. In this regard, we examine the existing literature. The literature presented below comes from the mechanical engineering sector and not from the field of mathematical optimization. In the second area mentioned, no literature was found that refers to general modular systems. There are a few optimization problems that might go in a similar direction if we consider modifications of the original problems. However, these will only be considered in more detail after the model presentation in Section 3.3.

The literature by Fujita still fits quite well. In [19] an abstracted hierarchal representation of product variety design is introduced. Figure 2.9 shows the structure of the work of Fujita.

In our model, we consider different products that include components. The paper discusses modules and attributes. Modules are roughly equivalent to our components and the attributes to the design of the components, the specific variants in out context. In our context, we can provide the components independently of the product in different variants. This independency is not given in the structure of Fujita, which is clear by the exact definition of the variables in [19, Section 2.3]. At this point, it remains unclear whether the concept of variants of components is used. Obviously, there are, if we consider Figure 2.9 for example, different choices for module 1, A and A′, but also the attributes can be different.

In the papers of Fujita, [18, 17, 19] possible system boundaries and cost models for the optimization of modular systems are discussed. The optimization models are divided into three classes. Class I describes the optimization of the module attributes under a fixed combination of the modules. Class II describes the optimization of components combinations with explicitly given variants of the components. This leads to a combinatorial optimization problem, which is mostly binary. Class III describes the combination of Class I and II, the simultaneous optimization of the combination and of the design of the components. The problem we consider in this work fits best into the framework of Class III problems. Although the approaches are similar, there are some key differences to our approach: first, as mentioned above, the attribute variables in [19] depend on the products. Secondly, the fact that the same modules are used for different products is not determined by the optimization model but is defined in advance by constraints [Section 3.1][19]. The solution process for the optimization problem for Class III problems is subdivided into several subproblems which are solved by different optimization techniques, such as genetic algorithms and Branch-and-Bound techniques. In this work, we will present an optimization approach that does not require this subdivision.

A similar approach to the Class II approach of Fujita is given in the paper [64]. A model for the optimization of modular products in reconfigurable production lines is presented there using the example of a powertrain. The optimization problem is described as a subset selection problem. However, it is based on modules (here components) with predefined parameters (here variants), and there is no adjustment or optimization of the module parameters here.

Another keyword in the context of modular systems is the so-called *product family design*. In the product family of crane bridges, for example, we consider crane bridges with different properties, where a specific crane bridge is one product. Something similar happens at first glance when it comes to the modular construction of products. The idea here is to build products that consist of different components, where not every product contains exactly the same components. Again, the modules are the components of our modular system. The first difference in the concept is that it is permissible in one product to use a component that is not used in another product. This is not the case for our modular system concept, but it plays a subordinate role for now. The term product family design definitely includes many examples of how and why it makes sense to build products modularly, but the problem does not fit in our context. The main difference in these papers is the use of the term optimization. Optimization problems are mentioned, but neither the objective function nor the variables match those in our context. In [56], for example, optimization problems are specified and the values of design parameters are searched. However, the number of variants of the different components is not included in the optimization. Other papers mention variants and optimize several of them, but they talk about so-called “product variants”, which are something different. In the paper [61], for example, a cell phone example is given in which product variants are characterized by different components (with Wifi or Bluetooth instead) and not by different component parameters.

2.4 Conclusions

In this chapter, we presented in Section 2.1 what modular systems are in general, examined examples in detail in Section 2.2. We considered two examples from the field of mechanical engineering with automotive modular systems in Subsection 2.2.1 and the crane bridge modular system in Subsection 2.2.3. In addition to an example of a modular system in the field of construction with container buildings in Subsection 2.2.2, we also introduced a modular system approach for binpacking problems in Subsection 2.2.4. In the end, we reviewed the existing literature in Section 2.3.

In the literature research it becomes clear that there is quite a lot in this field but one must clearly differentiate how terms are used and what meaning they have. However, what also becomes clear is that very few can be found in combination with the field of mathematical optimization, which reinforces the topic of this thesis. In particular, no general optimization models are introduced, and, not to mention, no procedures are presented on how to solve such problems. Although there are many applications in the field of mechanical engineering, the focus is often different and solution methods from the perspective of mathematical optimization are not discussed. In many points, it remains completely unclear how to obtain optimal configurations for modular systems. This motivates the further course of the thesis, so, we proceed with optimization formulations for modular system problems.

In Chapter 3 we deduct a first general optimization model within the aspects discussed above. After that, we deduce concrete optimization models for the crane bridge and binpacking example which are introduced in this chapter.

Chapter 3

An Optimization Model for the Optimal Configuration of Modular Systems

In Section 3.1, we introduce a first optimization approach for the optimal configuration of modular systems. After the presentation of the general approach, we consider in Section 3.2 the crane bridge examples (Subsection 3.2.1) and various binpacking examples (Subsection 3.2.2). Lastly, we discuss the clustering problem in Section 3.3, which fits into the mathematical optimization context for modular system problems.

3.1 Optimization Model

In this section, we introduce a first approach with mathematical optimization. At some points, we briefly refer to the crane bridge problem to make the content easier to understand. The example will be presented in detail in the following section.

In some underlying market, we assume a demand of N products that can be characterized by the same s properties. Therefore, we have the products $p^\ell \in \mathbb{R}^s$, $\ell = 1, \ldots, N$. A product can be built of C different components. In the example of the crane bridges, we have a demand of N crane bridges built from $C = 2$ components, profiles and sheets. Crane bridges in the application are characterized by two properties, load capacity and span, so we have $s = 2$.

In the next step, we introduce the variable vectors of the optimization model. Each component has specific parameters, and through them the components in a modular system will differ. In our example, these are different geometry parameters such as length or width. For each component $c \in \{1, \ldots, C\}$, we assume $k_c \in \mathbb{N}_0$ different variants in the modular system. Depending on the application, a lower bound of zero or one is conceivable for k_c. The number of components C in the model is a fixed known number and the vector of the number of variants $\kappa = (k_1, \ldots, k_C)$ is variable.

As mentioned above, the second type of variables are the specifications of the different component variants. With $x_i^c \subseteq \mathbb{R}^{n_c}$ we denote the vector for variant $i \in \{1, \ldots, k_c\}$ of component $c \in \{1, \ldots, C\}$, where n_c is the number of geometry parameters of component c. If $k_c = 0$ for one component holds, the corresponding x_i^c clearly does not exist. Index sets with $i = 1, \ldots, k_c$ are in such cases empty by definition. For crane bridges, we have, for example, for profiles (component 1), three geometry parameters: length, thickness, and width; therefore, we have $n_1 = 3$. Additional constraints for x_i^c within the same component c may be present, such as box constraints. We collect them in the set X^c and require $x_i^c \in X^c \subseteq \mathbb{R}^{n_c}$. We summarize all x_i^c in the vector ξ with $\xi \in X(\kappa) \subseteq \mathbb{R}^{\sum_{c=1}^{C} k_c n_c}$, which is the second variable in our model. $X(\kappa)$ contains, in addition to the constraints from X^c, possible additionally coupling constraints between different variants. Using that, the set depends on the number of variants κ. Through this formulation, we obtain a first dependency structure of the length of the vector ξ on the variable κ.

In addition to the number of variants of the components and the specification of the components themselves, the number of pieces in the modular system is important and also unknown in the beginning. This is the third type of variable in the model. With $z_{i,\ell}^c \in \mathbb{N}_0$ we denote the number of pieces we choose from variant i of the component c for product ℓ with $i = 1, \dots, k_c$, $c = 1, \dots, C$, $\ell = 1, \dots, N$. We bundle the number variables for one product p^ℓ in the vector z^ℓ and all the number variables in the vector z. Thus, z is a vector of integer variables. To satisfy the required product properties, additional constraints on the variables z have to be expected, which again depend obviously on κ but can also depend on the vector ξ. We summarize all constraints in the set $Z(\kappa, \xi)$ and require $z \in Z(\kappa, \xi)$.

We remain quite general at this point and only mention the important dependencies. Examples about the possible structure of $Z(\kappa, \xi)$ are given in the following subsections.

As mentioned in the introductory examples, we want to minimize costs. Therefore, we consider two types of cost. Firstly, with given κ and ξ, for each product p^ℓ, we choose among all feasible configurations $z \in Z(\kappa, \xi)$ the cheapest one. Thus, we minimize for every product ℓ a cost function $c^\ell(\kappa, \xi, z^\ell)$, which typically models oversizing effects. In this part of the objective function we want to model how well the required properties are fulfilled and set

$$c^{prop}(\kappa, \xi, z^\ell) = \sum_{\ell=1}^{N} c^\ell(\kappa, \xi, z^\ell).$$

In addition, there are maintenance costs c^{size} that depend on the size of the modular system. The size of the system can be measured, for example, by the number of different variants, by the total weight of the system, or by the total volume of the system. Thus, c^{size} can depend on the three variables.

In many applications, oversizing costs and maintenance costs develop in opposite directions under changes in the configuration of the modular system. Together, we aim to minimize the sum of these two costs and obtain a first optimization model.

General Optimization Model

The general optimization model for the optimal configuration of a modular system is given by

$$
\begin{aligned}
P^{gen}: \quad \min_{\kappa,\xi,z} \ & c^{size}(\kappa,\xi,z) + c^{prop}(\kappa,\xi,z) \quad \text{s.t.} \\
& z \in Z(\kappa,\xi), \\
& \xi \in X(\kappa), \\
& \kappa \in \mathbb{Z}^{C}_{\geq 0},\ \xi \in \mathbb{R}^{\sum_c k_c n_c},\ z \in \mathbb{Z}^{N\sum_c k_c n_c}.
\end{aligned}
$$

Since we have continuous and integer variables, P^{gen} is in general mixed-integer. As mentioned above, the constraints yield a dependency structure between the variables. In fact, the length of the vector ξ is determined by the entries of the vector κ. This fact not only complicates the solution process but makes it impossible to use standard solvers for mixed-integer optimization problems. In implementations with standard optimization solvers for global minimization, it must be clear how many variables have to be defined. This number is not available at this time of consideration. This is also the reason why we denote P^{gen} as an optimization model and not an optimization problem. Later we also discuss a "well-defined" optimization problem, if the concrete number of variables is given. In this words, the model P^{gen} is a not well-defined optimization problem. This may not be a general procedure, but we use it in this work to clearly distinguish between an only modeled approach like P^{gen} and an optimization problem, which can be put into a solver. The concrete well-defined optimization problem is introduced in Chapter 4. Next, we deduct the optimization models for the examples of Chapter 2.

3.2 Optimization Models for the Examples

In Subsection 2.2.3 the idea of a modular system for crane bridges is introduced, and in Subsection 2.2.4 the idea of a modular system for binpacking problems. In the previous chapter, we did not consider optimization models. In Section 3.1 a general optimization model P^{gen} was derived for the cost-minimal configuration of the modular system. However, explicit forms of objective functions and constraints have not yet been discussed there. In the following two subsections, we consider concrete functions and constraints for the crane bridge and binpacking examples. We start with the optimization model for the crane bridge modular system.

3.2.1 Optimization Model for the Crane Bridge Modular System

A rough sketch of a crane bridge, which serves as an idea for the modular system, is shown in Figure 2.7. As mentioned above, we consider a demand for N crane bridges, the products p^ℓ, $\ell = 1, \ldots, N$, and we take into account the two properties span width L^ℓ, and load capacity M^ℓ. Thus, we have the product specifications $p^\ell = (L^\ell, M^\ell) \in \mathbb{R}^2$ for $\ell = 1, \ldots, N$ (with $s = 2$). The demand of the crane bridges is known in the beginning and for every crane bridge requested, it is clear how large the span is and what load capacity it must meet. The modular system then

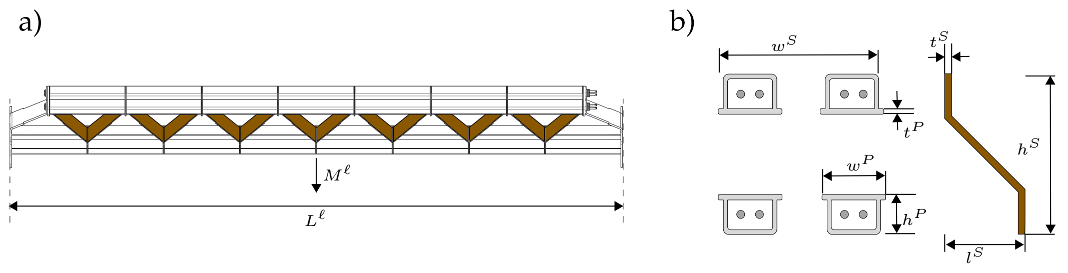

FIGURE 3.1: a) Segmented crane bridge and b) details for the geometric design of the profiles and sheets (brown)

contains the two components (thus $C = 2$) profile and sheet. For a crane bridge p^ℓ that is constructed from our modular system, we require that all profiles have the same geometric parameters. This should also hold for the sheets. We discuss this in more detail later. Firstly, we introduce the variables.

Variables

The modular system consists of $n := k_1 \in \mathbb{N}$ profile variants and $m := k_2 \in \mathbb{N}$ sheet variants, which are unknown but are assumed to be bounded from above. Thus, the first variable (vector) is $\kappa = (n, m) \in \mathbb{N}^2$ with $n \geq 1$ and $m \geq 1$.

The variable(vector) for the specification of the components is given by $\xi = (x_1^1, \ldots, x_n^1, x_1^2, \ldots, x_m^2)$. For simplification, we introduce some shorter and more understandable notation at this point. We denote profile variant i, $i = 1, \ldots, n$, with P^i, thus we have $P^i := x_i^1$. Profiles are characterized by three geometry parameters, height, thickness, and width, so we get the variables for the geometric design of the profiles with $P^i = \left(h_i^P, t_i^P, w_i^P\right) \in \mathbb{R}^3$, $i = 1 \ldots, n$. If we talk about a profile variant i, we mean a specific combination of the three ingredients of P^i. Of course, P^i is not negative and we also assume upper bounds for these variables and summarize them in the set X^1, so we have $P^i \in X^1 \subseteq \mathbb{R}^3$ for each profile variant i.

Sheets, on the other hand, are described by four geometry parameters: height, segment length, thickness, and width. For notational reasons, we use the index j for the different variants of the sheets and additionally set $S^j := x_j^2 \in \mathbb{R}^4$ with $j = 1, \ldots, m$. Thus, we obtain the variables for the geometric design of the sheets with $S^j = \left(h_j^S, l_j^S, t_j^S, w_j^S\right) \in \mathbb{R}^4$. We again consider some box constraints for S^j, which we collect in the set X^2, so we have $S^j \in X^2 \subseteq \mathbb{R}^4$ for each sheet variant j.

In total, we obtain $\xi = (P^1, \ldots, P^n, S^1, \ldots, S^m) \in \mathbb{R}^{3n+4m}$. For notational reasons, we write all the variables in the vector ξ one after the other. The notation with P^i and S^j is mainly used to make it easier to read which variables belong to the profiles and which to the sheets. It should be noted at this point that the length of the profiles is not explicitly included in the model but can be calculated from the other variables. This is characterized by the double segment length of the corresponding sheet.

In Figure 3.1 a) the segmented crane bridge with the profiles and sheets and the properties span and load capacity is illustrated once again. Part b) shows the geometric design of the profiles and sheets with the corresponding variables. The indices i and j for specific variants have been omitted to keep things simple. The sheets are colored brown in parts a) and b).

The variable vector for the number of pieces in the modular system is given by z. We consider the number variables regarding to profile or sheet and the variant and crane bridge. For the profiles $z^P_{i,\ell} := z^1_{i,\ell} \in \mathbb{N}_0$ is the number of sheets of variant i, $i = 1, \dots, n$, in crane bridge ℓ. For the sheets, we have, respectively, $z^S_{j,\ell} := z^2_{j,\ell} \in \mathbb{N}_0$, $j = 1, \dots, m$, $\ell = 1, \dots, N$.

After introducing the variables, we continue with the constraints of the model.

Constraints

In this example for simplicity, we do not require constraints that couple different geometry variables, that is, we only require the box-constraints for the geometry variables. We assume

$$P^i = \left(h^P_i, t^P_i, w^P_i\right) \in \underbrace{[40, 100] \times [6, 6] \times [100, 200]}_{=:X^1}, \quad i = 1, \dots, n, \tag{3.1}$$

$$S^j = \left(h^S_j, l^S_j, t^S_j, w^S_j\right) \in \underbrace{[400, 1000] \times [150, 600] \times [6, 6] \times [300, 400]}_{=:X^2}, \quad j = 1, \dots, m, \tag{3.2}$$

with the definitions of the sets X^1 and X^2. In the set notation of P^{gen}, this results in

$$\begin{aligned} X(n,m) = \big\{ (P^i)_{i=1,\dots,n} \in \mathbb{R}^{3n}, (S^j)_{j=1,\dots,m} \in \mathbb{R}^{4m} \big| \\ P^i \in X^1,\ i = 1, \dots, n, \\ S^j \in X^2,\ j = 1, \dots, m \big\} \end{aligned}$$

All numbers in (3.1) and (3.2) are measured in millimeters. As written in the equations above, for simplicity, we assume a fixed thickness of profiles and sheets (t^P_i and t^S_j) of 6 mm.

The remaining constraints refer only to the variables of z or of ξ and z. In P^{gen} these constraints are collected in the set $Z(\kappa, \xi)$. The most important condition is that we are allowed only to pick one profile and one sheet variant in the corresponding number for the crane bridge ℓ, see [6]. We model this with

$$\left|\{i \in \{1, \dots, n\} \,|\, z^P_{i,\ell} > 0\}\right| = 1, \quad \ell = 1, \dots, N, \tag{3.3}$$

$$\left|\{j \in \{1, \dots, m\} \,|\, z^S_{j,\ell} > 0\}\right| = 1, \quad \ell = 1, \dots, N. \tag{3.4}$$

Additionally, all number variables have to be nonnegative, thus we obtain

$$\begin{aligned} z^P_{i,\ell} &\geq 0, \quad i = 1, \dots, n,\ \ell = 1, \dots, N, \\ z^S_{j,\ell} &\geq 0, \quad j = 1, \dots, m,\ \ell = 1, \dots, N. \end{aligned} \tag{3.5}$$

Profiles of one variant and sheets of one variant must be taken from the modular system in such a way that the crane bridge results in the span L^ℓ (in meters) and that it carries at least a load of M^ℓ (in tons). For the chosen profile-sheet combination (i, j), $i \in \{1, \dots, n\}$, $j \in \{1, \dots, m\}$, we can calculate the total number $z^\ell_{1,i}$

of profiles and the total number of sheets $z^{\ell}_{2,j}$ explicitly through the construction of the bridge, see Figure 3.1 a). In this construction, the number of profiles depends on the geometry variables of the profiles and sheets. We only require a constraint for $z^{P}_{i,\ell}$ if the variant combination (i, j) is chosen. Thus, we get

$$\begin{gathered} z^{P}_{i,\ell},\ z^{S}_{j,\ell} > 0 \quad \Rightarrow \quad z^{P}_{i,\ell} = 4\left\lfloor \frac{L^{\ell}}{2l^{S}_{j}} \right\rfloor - 2, \\ i = 1,\dots,n,\ j = 1,\dots,m,\ \ell = 1,\dots,N. \end{gathered} \tag{3.6}$$

Together with the constraints (3.3) and (3.4), the variable $z^{P}_{i,\ell}$ is then only set to a value once. The arrow $\Rightarrow$ is only for the legibility of the condition. Such conditions are called "indicator constraints", and solvers such as Gurobi or CPLEX ([11], [30]) can handle them. For the number of sheets, we analogously obtain

$$\begin{gathered} z^{S}_{j,\ell} > 0 \quad \Rightarrow \quad z^{S}_{j,\ell} = 2\left\lfloor \frac{L^{\ell}}{2l^{S}_{j}} \right\rfloor - 2, \\ j = 1,\dots,m,\ \ell = 1,\dots,N. \end{gathered} \tag{3.7}$$

Through equations (3.6) and (3.7) the length L^{ℓ} of the crane bridges, which is one of the required properties, is fulfilled.

The second property is the load capacity of a crane bridge. Let M be the load capacity function for a combination of a profile P^{i} and a sheet S^{j}. Since the load capacity function is generally difficult to determine, we approximate it by the (rough) estimate

$$M(P^{i}, S^{j}) = \frac{c_1}{L^{\ell}}\left(c_2 h^{S}_{j} + c_3 h^{P}_{i} + c_4 w^{P}_{i} + c_5 w^{S}_{j} - c_6\left(\frac{h^{S}_{j} - 2h^{P}_{i}}{l^{S}_{j}} - \sqrt{3}\right)^{2}\right) \tag{3.8}$$

with parameters $c \in \mathbb{R}^{6}$. To motivate the shape of this estimate, note that the larger the span of the crane bridge, the lower the load capacity. Moreover, the height of the profiles and sheets is more important than their widths, so we choose c_2 and c_3 higher than c_4 and c_5, respectively. We chose $c = (50, 1, 3, \frac{2}{5}, \frac{1}{5}, 100)$. Parameter studies for a truss model have shown that the best distributions of forces in the truss are achieved at an angle of around 60° between the sheets and the profiles. In Figure 3.2, it can be seen that this holds for $\frac{h^{S}_{j} - 2h^{P}_{i}}{l^{S}_{j}} = \sqrt{3} = \tan(60°)$. Details can be found in [6]. The sketch of the bridge construction we use here corresponds to the bridges developed there. Various studies were carried out there on the static behavior of these bridges.

In summary, using the notation from above and the function M from (3.8), we obtain the following constraints:

$$\begin{gathered} z^{P}_{i,\ell},\ z^{S}_{j,\ell} > 0 \quad \Rightarrow \quad M(P^{i}, S^{j}) \geq M^{\ell}, \\ i = 1,\dots,n,\ j = 1,\dots,m,\ \ell = 1,\dots,N. \end{gathered} \tag{3.9}$$

As in the constraints before, the minimum load capacity M^{ℓ} must of course only be fulfilled for the used profile-sheet combination. In addition, the set of feasible configurations must take into account that not every profile-sheet combination is

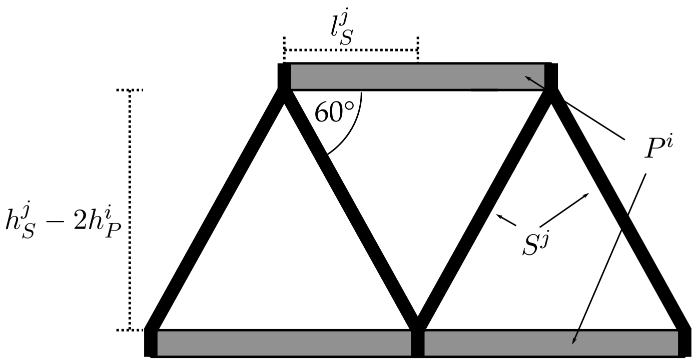

FIGURE 3.2: Composition and angles of profiles and sheets

possible due to possible instabilities or geometrical aspects. Indeed, the following additional constraints must also apply:

$$\begin{aligned} &z_{i,\ell}^P,\ z_{j,\ell}^S > 0 \quad \Rightarrow \\ &\qquad w_j^S \geq 2w_i^P + t_j^S,\ h_j^S \geq 3h_i^P,\ 2l_j^S \geq h_j^S,\ 2l_j^S \leq 3h_j^S, \\ &\quad i = 1,\ldots,n,\ j = 1,\ldots,m,\ \ell = 1,\ldots,N. \end{aligned} \tag{3.10}$$

With the first inequality, we achieve that there is a minimum distance between the profiles, see Figure 3.1b). The other three conditions achieve a boundary of the angle of the sheet, see Figures 3.1 and 3.2. We summarize these inequalities into a set F and shorten (3.10) to

$$\begin{aligned} &z_{i,\ell}^P,\ z_{j,\ell}^S > 0 \quad \Rightarrow \quad (P^i, S^j) \in F \\ &\qquad i = 1,\ldots,n,\ j = 1,\ldots,m,\ \ell = 1,\ldots,N. \end{aligned}$$

Altogether in the set notation, we obtain for crane bridge ℓ

$$\begin{aligned} Z_\ell(n,m,\xi) = \big\{(z_{i,\ell}^P)_{i=1,\ldots,n} \in \mathbb{Z}^n,\ (z_{j,\ell}^S)_{j=1,\ldots,m} \in Z^m \big| \\ & z_{i,\ell}^P \geq 0,\ z_{j,\ell}^S \geq 0, \\ & |\{i \in \{1,\ldots,n\}|\ z_{i,\ell}^P > 0\}| = 1, \\ & |\{j \in \{1,\ldots,m\}|\ z_{j,\ell}^S > 0\}| = 1, \\ & z_{i,\ell}^P,\ z_{j,\ell}^S > 0 \ \Rightarrow\ z_{i,\ell}^P = 4\left\lfloor \frac{L^\ell}{2l_j^S} \right\rfloor - 2, \\ & z_{j,\ell}^S > 0 \ \Rightarrow\ z_{j,\ell}^S = 2\left\lfloor \frac{L^\ell}{2l_j^S} \right\rfloor - 2, \\ & z_{i,\ell}^P,\ z_{j,\ell}^S > 0 \ \Rightarrow\ M(P^i, S^j) \geq M^\ell, \\ & z_{i,\ell}^P,\ z_{j,\ell}^S > 0 \ \Rightarrow\ (P^i, S^j) \in F, \\ & i = 1,\ldots,n,\ j = 1,\ldots,m\big\}. \end{aligned}$$

Thus, we set $Z(\kappa,\xi) = Z(n,m,\xi) = Z_1(n,m,\xi) \times \cdots \times Z_N(n,m,\xi)$. Finally, we consider the two-part cost function of the model.

Cost Function

On the one hand, we are interested in a small modular system and, on the other hand, in a modular system, so that the required spans and load capacities can be covered as well as possible. We measure the size of the modular system in the number of variants of the two components and sheets with

$$c^{size}(\kappa) = c^{size}(n, m) = c^P n + c^S m$$

with cost factors $c^P, c^S \in \mathbb{R}_{>0}$ for each profile and sheet variant. These costs obviously grow linearly with the number of variants. The second part of the objective function models oversizing costs that decrease for increasing numbers of variants. In fact, if profiles and sheets can be chosen from a large number of variants, we can expect that the desired load capacities will hardly be exceeded. However, if there is little choice, the crane bridges will be oversized and respective penalizing costs should occur. We model this with the cost function

$$c^{prop}(n, m, \xi, z) = c_d \sum_{\substack{i,j,\ell=1 \\ (z^P_{i,\ell}>0)\&(z^S_{j,\ell}>0)}}^{n,m,N} \left(M(P^i, S^j) - M^\ell \right), \tag{3.11}$$

with a cost coefficient $c_d \in \mathbb{R}_{>0}$ for each unit of deviation in the load capacity. These costs are obviously small, if the deviation of load capacity is small. The reason why the function above looks quite complicating is that the penalty costs must only be included in the objective function for the used profile-sheet combination of each crane bridge. We can model this later with additional binary variables. To save writing effort, we only write one "sum symbol", and mean that i goes from 1 to n, j from 1 to m and ℓ from 1 to N, respectively.

Depending on the use case, the weight of the crane bridges is also interesting, as lighter components are easier to transport, for example. If we use profile variant i and sheet variant j for a crane bridge ℓ, the corresponding weight is given by $w(P^i, S^j, z^P_{i,\ell}, z^S_{j,\ell})$. The weight of a crane bridge depends, in addition to the geometry variables, on the corresponding number of these variants. As in the part with the load capacity function, the weight enters the objective function only if the combination (i, j) is actually used. Thus, we obtain

$$c^w(n, m, \xi, z) = c_w \sum_{\substack{i,j,\ell=1 \\ (z^P_{i,\ell}>0)\&(z^S_{j,\ell}>0)}}^{n,m,N} w(P^i, S^j, z^P_{i,\ell}, z^S_{j,\ell}) \tag{3.12}$$

with a cost coefficient c_w per ton of weight.

Lastly, we obtain a first optimization model for the optimal configuration of a modular system for crane bridges:

Optimization model: modular system crane bridge

The optimization model is given by

$$
\begin{aligned}
P^{crane}: \quad & \min_{n,m,\xi,z} \; c^P n + c^S m + c^{prop}(n,m,\xi,z) + c^w(n,m,\xi,z) & \\
& \text{s.t.} \quad (3.3),(3.4),(3.5),\ (3.6),(3.7),(3.9),(3.10), & \# \; z \\
& \qquad\;\; (3.1),(3.2), & \# \; \xi \\
& \qquad\;\; n \geq 1,\ m \geq 1, & \# \; \kappa \\
& \qquad\;\; n,m \in \mathbb{Z},\ \xi \in \mathbb{R}^{3n+4m},\ z \in \mathbb{Z}^{N(3n+4m)} &
\end{aligned}
$$

with c^{prop} from (3.11) and c^w from (3.12).

If we do not want to consider the weight costs, we omit them from the objective function. The first line of the constraints are all constraints for the number variables z, which we denote with "# z", based on the Python syntax for comments. The second and third lines, respectively, to ξ and κ.

In the next subsection, we consider the second running example of this thesis, the modular system for the binpacking problem, and deduce a concrete optimization model.

3.2.2 Optimization Model for the Binpacking Modular System

In this subsection, we consider optimization models for the binpacking examples. In Subsection 2.2.4 the main idea for the binpacking application was introduced. We consider N bins of different lengths, which should be filled as much as possible. This is the only required property in this example. Thus, we have the product specification $p^\ell = L^\ell$, $\ell = 1,\dots,N$, where L^ℓ is the known length of bin ℓ. We divide this subsection into two parts. First we consider one-dimensional binpacking problems, then higher dimensional ones. We start with the one-dimensional example.

Optimization Model for the One-Dimensional Binpacking Problem

Firstly, we consider a one-dimensional binpacking example. With Definition 2.2.1, that means that we consider only one component ($C = 1$) in the modular system. The one component here is the color red, in applications, this can of course be a specific material or something similar. The objects can be available in n different variants in the modular system, in the general notation that means $n := k_1 = \kappa$ with $n \geq 1$ in general. For variables ξ and z, we first introduce the general notation and then shorten it for simplification. The objects can only differ in length (and not in height, for example), so we have $x_i^1 \in \mathbb{R}$, $i = 1,\dots n$, for the length of an object in variant i. For simplicity, we omit the 1 for the component in the variables, since we have only one component and write $\xi = \left(x_i^1\right)_{i=1,\dots,n} =: (x_1,\dots,x_n) \in \mathbb{R}^n$.

The corresponding number variable, that means, how often a variant i is present in a bin ℓ is given by $z_{i,\ell}^1 \in \mathbb{Z}$. We again shorten the notation and collect all variables in z, with $z = \left(z_{i,\ell}^1\right)_{\substack{i=1,\dots,n,\\ \ell=1,\dots,N}} =: (z_{1,1},\dots,z_{n,N})$.

Next, we look at the additional constraints for the variables. Firstly, we consider constraints that depend only on variables ξ. We have a lower bound $\underline{x} > 0$ and conditions that ensure that two variants differ sufficiently by a parameter $\tau > 0$. This condition is natural for many applications. We sort w.l.o.g. the entries of the vector ξ and require

$$x_i + \tau \leq x_{i+1}, \quad i = 1, \dots, n-1, \tag{3.13}$$

$$x_i \geq \underline{x}, \quad i = 1, \dots, n. \tag{3.14}$$

This gives us

$$X(n) = \big\{\xi \in \mathbb{R}^n \big|\, x_i \geq \underline{x},\ i = 1, \dots, n,\\ x_i + \tau \leq x_{i+1},\ i = 1, \dots, n-1\big\}.$$

Since we want to fill the bins with objects from the modular system without overlapping and exceeding the corresponding length, we need the constraints

$$\sum_{i=1}^{n} x_i z_{i,\ell} \leq L^\ell, \quad \ell = 1, \dots, N. \tag{3.15}$$

A natural condition for avoiding a high number of small objects, which we want to model, is a restriction of the total number of objects in each bin, which we model with

$$\sum_{i=1}^{n} z_{i,\ell} \leq \overline{o}_\ell, \quad \ell = 1, \dots, N \tag{3.16}$$

where $\overline{o}_\ell > 0$ is the upper bound of the number of objects in bin ℓ. With

$$z_{i,\ell} \geq 0, \quad i = 1, \dots, n,\ \ell = 1, \dots, N, \tag{3.17}$$

we summarize it to

$$Z_\ell(n, \xi) = \big\{(z_{i,\ell})_{i=1,\dots,n} \in \mathbb{Z}^n \big|\, z_{i,\ell} \geq 0,\ i = 1, \dots, n,\\ \sum_{i=1}^{n} x_i z_{i,\ell} \leq L^\ell,\\ \sum_{i=1}^{n} z_{i,\ell} \leq \overline{o}_\ell\big\}.$$

for bin ℓ. In this example, we consider the constraints for each product ℓ independently, since it is possible. In total, we have a cartesian product structure for Z from P^{gen}, given by $Z(n, \xi) = Z_1(n, \xi) \times \dots, Z_N(n, \xi)$.

On the one hand, we want to find a cheap size of the modular system, in our case a modular system with as few as possible variants, and on the other hand, we want to find a configuration of the modular system such that the objects fit as well as possible in the bins. The first part of the objective function is the function

$$c^{size}(n) = c_v n,$$

with a cost factor $c_v > 0$ for the variants. These costs obviously increase linearly with an increasing number of variants. In the second part of the cost function, we penalize remaining space in the bins, so we have for bin ℓ

$$c^\ell(\kappa, \xi, z) = c^\ell(n, \xi, z) = c_d \left(L^\ell - \sum_{i=1}^{n} x_i z_{i,\ell} \right).$$

with a cost factor $c_d > 0$ per each unit of deviation. Altogether we have

$$c^{prop} = c_d \sum_{\ell=1}^{N} \left(L^\ell - \sum_{i=1}^{n} x_i z_{i,\ell} \right).$$

For an increasing number of variants, the costs c^{prop} again decrease, since a large selection of variants results in few remaining space. As is often the case in modular system problems, the costs run in opposite directions.

In summary, we obtain the model

Optimization model: one-dimensional binpacking

The optimization model for the one-dimensional binpacking example is given by

$$\begin{aligned} P_{bin}^1 : \quad & \min_{n,\xi,z} \; c_v n + c_d \sum_{\ell=1}^{N} \left(L^\ell - \sum_{i=1}^{n} x_i z_{i,\ell} \right) & \\ & \text{s.t.} \quad (3.15), (3.16), (3.17), & \# \; z \\ & \qquad\;\; (3.14), (3.13), & \# \; \xi \\ & \qquad\;\; n \geq 1, & \# \; \kappa \\ & \qquad\;\; n \in \mathbb{Z}, \; \xi \in \mathbb{R}^n, \; z \in \mathbb{Z}^{Nn}. & \end{aligned}$$

We can easily extend P_{bin}^1 to any dimension p, which is done in the following.

Optimization Model for the p-Dimensional Binpacking Problem

We consider $p := C \in \mathbb{N}$ components, where p is fixed. Different components are again illustrated with different colors. The model complicates a bit, but the general structure remains the same, which is the main idea behind these examples. The notation of the variables is the same as in the problem P^{gen}. With $\kappa = (k_1, \ldots, k_p) \in \mathbb{Z}^p_{\geq 0}$ we have the number of variants for each component. $x_i^c \in \mathbb{R}$ is the variable for the length of variant i, $i = 1, \ldots, k_c$, of component c, $c = 1, \ldots, p$, and we have $\xi = (x_i^c)_{\substack{i=1,\ldots,k_c,\\ c=1,\ldots,C}} \in \mathbb{R}^{k_c p}$. The number of objects of variant i of component c in bin ℓ is as in P^{gen} denoted by $z_{i,\ell}^c$, so we have $z = (z_{i,\ell}^c)_{\substack{i=1,\ldots,n,\; \ell=1,\ldots,N,\\ c=1,\ldots,C}}$.

In this example, some k_c can be zero. The corresponding variables x_i^c and $z_{i,\ell}^c$ do not exist in such cases. Additionally, we require that at least one variant of any component is part of the modular system, so we require the following:

$$\sum_{c=1}^{C} k_c \geq 1. \tag{3.18}$$

For each component, different variants again have to differ by a parameter τ^c, now depending on the component. Together with a lower bound $\underline{x}^c$ we require

$$x_i^c \geq \underline{x}^c, \quad i = 1, \ldots, k_c,\ c = 1, \ldots, p, \tag{3.19}$$
$$x_i^c + \tau^c \leq x_{i+1}^c,\ i = 1, \ldots, k_c - 1,\ c = 1, \ldots, p \tag{3.20}$$

If we consider the set notation, we can subdivide according to the components and get for component c the set

$$X^c(k_c) = \big\{ (x_i^c)_{i=1,\ldots,n} \in \mathbb{R}^{k_c} \big|\ x_i^c \geq \underline{x}^c,\ i = 1, \ldots, k_c, \\ x_i^c + \tau^c \leq x_{i+1}^c,\ i = 1, \ldots, k_c - 1 \big\}.$$

Thus, we obtain the set $X(\kappa) = X^1(k_1) \times \cdots \times X^p(k_p)$.

The conditions for the variable z remain the same except that we also have to sum over the different components. In summary, we require

$$z_{i,\ell}^c \geq 0, \quad i = 1, \ldots, k_c,\ c = 1, \ldots, p,\ \ell = 1, \ldots, N, \tag{3.21}$$
$$\sum_{c=1}^{p} \sum_{i=1}^{k_c} x_i^c z_{i,\ell}^c \leq L^\ell, \quad \ell = 1, \ldots, N, \tag{3.22}$$
$$\sum_{c=1}^{p} \sum_{i=1}^{k_c} z_{i,\ell}^c \leq \overline{o}_\ell, \quad \ell = 1, \ldots, N, \tag{3.23}$$

where again L^ℓ is the known length of bin ℓ and $\overline{o}_\ell$ the upper bound for the total number of objects in bin ℓ.

If we use the set notation, we can subdivide here according to the bins and obtain for bin ℓ the set

$$Z_\ell(\kappa, \xi) = \big\{ \big(z_{i,\ell}^c\big)_{\substack{i=1,\ldots,n,\\ c=1,\ldots,C}} \in \mathbb{Z}^{\sum_c k_c} \big|\ z_{i,\ell}^c \geq 0,\ c = 1, \ldots, p, i = 1, \ldots, k_c, \\ \sum_{c=1}^{p} \sum_{i=1}^{k_c} x_i^c z_{i,\ell}^c \leq L^\ell, \\ \sum_{c=1}^{p} \sum_{i=1}^{k_c} z_{i,\ell}^c \leq \overline{o}_\ell \big\}.$$

Analogously to the one-dimensional case we get $Z(\kappa, \xi) = Z_1(\kappa, \xi) \times \ldots, Z_N(\kappa, \xi)$.

The cost functions remain similar to the one-dimensional case. The costs for the size of the modular system are given by $C(\kappa) = c_v^\top \kappa$, with $c_v \in \mathbb{R}_{\geq 0}^p$, where the entries of c_v^c are the cost factors for a variant of the component c. The deviation costs summed up for all bins are therefore given by

$$c^{prop}(\kappa, \xi, z) = c_d \sum_{\ell=1}^{N} \left(L^\ell - \sum_{c=1}^{p} \sum_{i=1}^{k_c} x_i^c z_{i,\ell}^c \right)$$

with the deviation cost factor $c_d > 0$. This leads to the following optimization model:

Optimization model: p-dimensional binpacking

The model for the p-dimensional case is given by

$$\begin{aligned} P^p_{bin}: \quad & \min_{\kappa,\xi,z} \; c_v^\top \kappa + c_d \sum_{\ell=1}^{N} \left(L^\ell - \sum_{c=1}^{p} \sum_{i=1}^{k_c} x_i^c z_{i,\ell}^c \right) \\ \text{s.t.} \quad & (3.21), (3.22), (3.23), && \# \; z \\ & (3.19), (3.20), && \# \; \xi \\ & \kappa \geq 0, \; (3.18) && \# \; \kappa \\ & \kappa \in \mathbb{Z}^p, \; \xi \in \mathbb{R}^{pk_c}, \; z \in \mathbb{Z}^{N \sum_c k_c}. \end{aligned}$$

After introducing the optimization models, we consider some visualizations for the modular system for the binpacking example.

Visualization Modular System Binpacking Example

The visualizations show a one- and a two-dimensional example with the allocation of the objects in the bins (Figures 3.3 and 3.5) and the modular system sorted by components and variants (Figures 3.4 and 3.6). The modular systems shown are not shown to be cost-optimal, they should only illustrate the present problem. The first component is associated with the color red, and the second with the color green. In the one-dimensional example, we consider six bins of different lengths. The one component (red) occurs in two different variants in the modular system. In the two-dimensional example, component one (red) occurs in two variants and component two (green) in three variants. The measure μ^ℓ indicates the percentage in which the bins are filled. Some of the bins in Figure 3.3 are completely filled (bin 3,6) and some not (bin 1,2,4,5). In these examples, we only allow two objects per bin. In the one-dimensional example, it can be assumed that a better modular system exists, since there is a lot of remaining space.

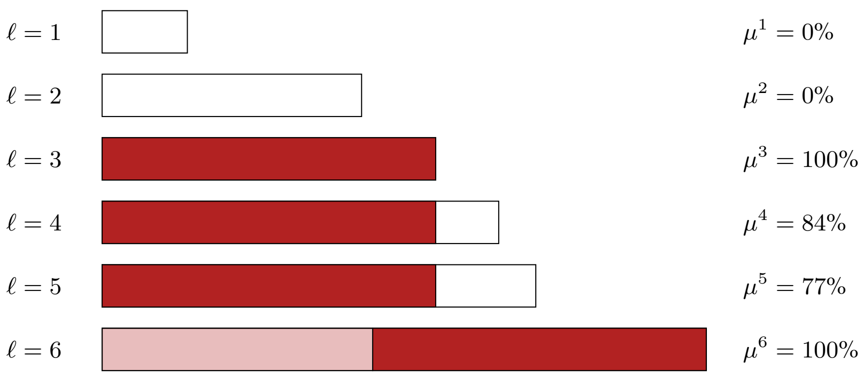

FIGURE 3.3: One-dimensional binpacking example (optimality not guaranteed); allocation of objects to bins

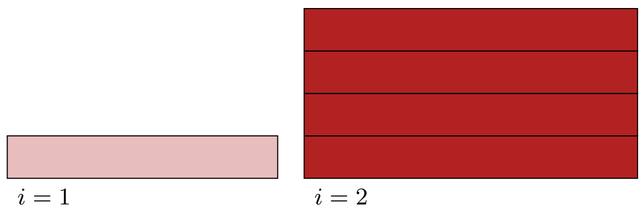

FIGURE 3.4: One-dimensional binpacking example (optimality not guaranteed); modular system

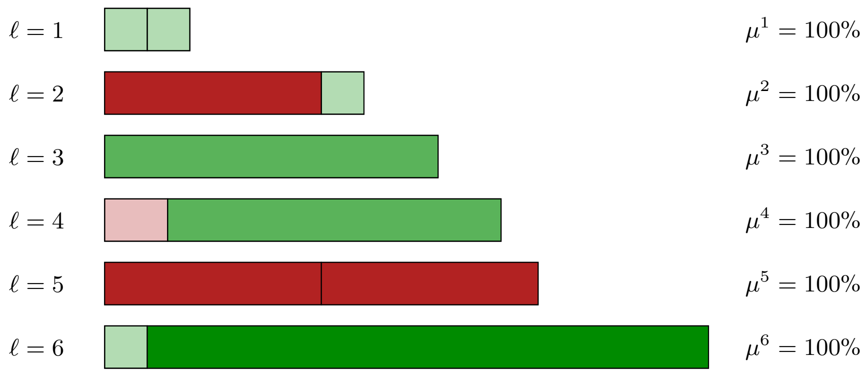

FIGURE 3.5: Two-dimensional binpacking example (optimality not guaranteed); allocation of objects to bins

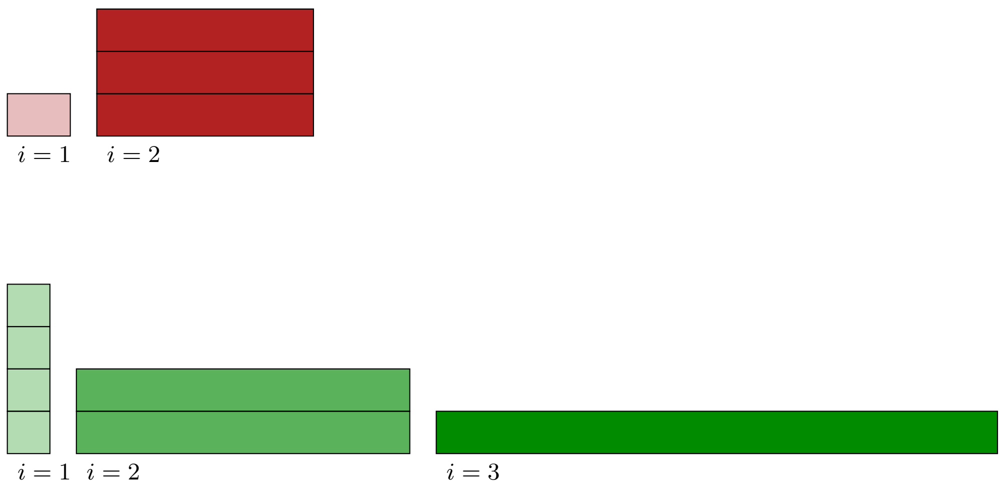

FIGURE 3.6: Two-dimensional binpacking example (optimality not guaranteed); modular system

After introducing the general optimization model and after introducing the optimization models for the running examples of this work, we have obtained an initial impression of the difficulties that arise with modular system problems. With this knowledge in mind, we briefly consider a well-known application of mathematical optimization that presents a very similar issue in Section 3.3. In the literature review in Section 2.3, it would have been too early to discuss this for the sake of clarity, so this example is first considered here. Specifically, it is about clustering problems and a special application thereof, facility location problems.

3.3 The Clustering and Facility Location Problem

As we already mentioned at the end of Section 3.1, the optimization approach presented is not well-defined, and we cannot determine exactly how many variables and associated constraints the optimization model consists of. However, since we have now considered a more concrete optimization approach compared to Chapter 2, we will return to the literature. In Section 2.3, a first literature overview was given. There, it was already explained that very little literature on modular systems and mathematical optimization exists. The literature presented there contains little to none optimization models and, above all, no concrete optimization problems that could be solved.

In this section, we look at an example that fits well into the setting in terms of structure, namely clustering problems. As part of this topic, a master thesis entitled “Numerical Methods in Optimization for Cluster Analysis in Facility Location Problems” was written by Fabian Habiger at our chair in the winter term 2022/2023, see [27]. Much literature was reviewed as part of this thesis, and we will now use it as a guide. According to [63], clustering is not uniformly defined. Basically, clustering deals with the partitioning of data into so-called clusters. The aim is for data within a cluster to be as homogeneous as possible, data from different clusters to be as heterogeneous as possible, and the measures used to examine the (dis)similarity of the data to have practical significance.

A classic use case are facility location problems. Cluster analysis can be used, for example, to determine where a manufacturer should build warehouses so that individual customers can be supplied quickly. In general, in many facility location problems, the sum of the distances between the customer and the assigned warehouse should be minimal. A common distance measure is the squared Euclidean norm.

The facility location problem fits into the category of facility location problems on the plane if warehouses can be built anywhere on the plane. Details on facility location problems can be found in [32]. Typical for location planning problems is that it is not clear from the beginning how many warehouses should be built. The variables in the location planning model described are precisely the coordinates of the locations to be opened. A resulting optimization problem is therefore not well-defined in our sense, and the connection to the optimization model for the modular system problems is thus immediately clear.

There is a large number of algorithms for clustering problems and thus also for facility location problems, see, for example, [63] for a broad overview. This paper shows that the most common methods are based heavily on problem-specific formulations of the clustering problem. It does not address general formulations

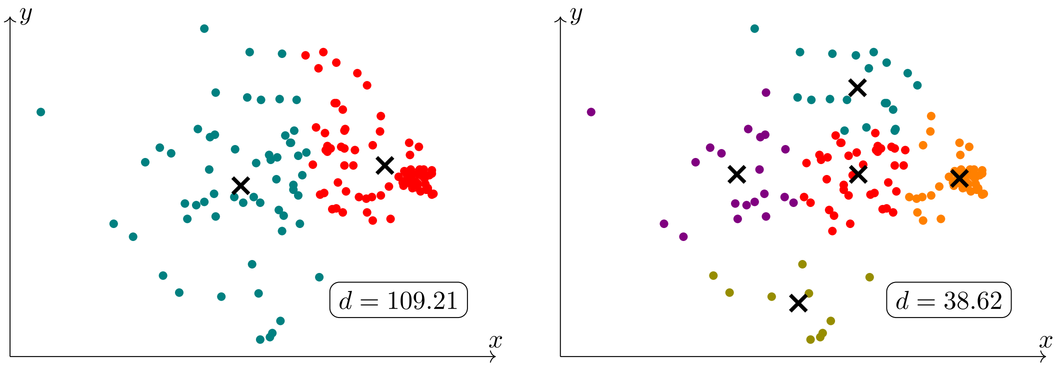

FIGURE 3.7: Data points with allocation in two (left) five (right) clusters and total squared Euclidean distance d to the allocated cluster

on how to deal with this unknown number of variables. This is precisely one goal of this work.

For the sake of completeness, we will briefly introduce one of the best-known solution methods for cluster problems, the k-means algorithm. We are doing this at this point to get a sense of how to deal with this particular type of problem. More detailed information on the method can be found, for example, in [29]. The input of the algorithm is a specific number of clusters k, which explains the k in the name of the method. At first, k cluster centers are randomly determined and then all data points are assigned to the nearest center. After this initialization step, the current cluster centers are replaced by the mean values of all data points contained in the corresponding cluster, and the data points are assigned to the new cluster centers. This is repeated until a termination criterion is satisfied.

Figure 3.7 shows data points and cluster centers determined by k-means for $k = 2$ and $k = 5$. The data correspond to randomly generated data and the results are obtained with python and the k-means method from the package *sklearn*. The colors indicate to which cluster a data point belongs. In the case of only two clusters, the total distance from the data points to the corresponding cluster is considerably larger (109.21) than when considering five clusters (38.62). We denote the total distance in the sketch with d and, as mentioned above, it is measured in the squared Euclidean norm.

As already mentioned above, there are no optimization approaches which handle the unknown number of clusters directly. In addition, no suitable literature has been found in the context of mathematical optimization for modular system problems. In this chapter, we have already been able to derive a general optimization model P^{gen} for modular system problems. This general approach can also be applied to clustering problems, as our student did in [27]. However, in this work, we continue to examine general modular system problems, and we do not consider cluster problems any further.

We now “only” have to deal with this unknown number of variables, which in the modular system problem is the unknown number of variants for the individual components. For this reason, in the following chapter, we first derive an optimization problem for modular systems, which can later also be passed on to a common solver.

3.4 Conclusions

In this chapter, in Section 3.1 we derived a general optimization model for modular system problems to obtain an optimal configuration of it. In general, we want to obtain products that consist of different components and that are part of the modular system in different variants.

Basically, the optimization models P^{gen} have a simple structure with three different types of variables. We have a variable vector for the number of different variants of the components, one for the number of parts in the modular system, and one for the corresponding (e.g. geometric) characteristics of the individual parts. The term "simple structure" refers here primarily to the description of the model, since many constraints can be collected appropriately and the objective function consists of two clearly defined parts that are summed accordingly. Modular system problems can therefore be modeled well on paper. So, we considered the running examples of this work, the crane bridge example in Subsection 3.2.1 and the binpacking example in Subsection 3.2.2.

However, the problem is that the variables for the number of parts and those for the (geometric) characteristics depend on the unknown number of variants. Such problems cannot be passed on to classical optimization solvers as the exact number of variables must be specified there. A similar problem occurs with clustering problems, which are presented in Section 3.3. There, we also presented a typical solution method, but it is completely tailored to clustering problems.

In the next chapter, we now consider a solution approach that deals with the unknown variable number and can be applied to modular system problems.

Chapter 4

A Well-Defined Optimization Problem

In Chapter 3, we introduced a first general optimization model P^{gen} for the optimal configuration for a modular system and considered the two running examples. The model is in our sense not "well-defined", because we cannot clearly determine the number of variables involved in the model. In P^{gen}, the length of the variables ξ and z depends on the number of variants. In particular, for a component c, which occurs in k_c different variants in the modular system, we have the length $C \cdot k_c$ for ξ and $N \cdot C \cdot k_c$, where C is the total (known) number of components and N the (known) number of required products, but k_c is part of the variable κ of the model and unknown at the beginning. Through this dependency structure, we denoted P^{gen} as an optimization model and not as an optimization problem.

In Section 4.1, we suggest a reformulation of P^{gen}, which allows us to formulate the optimization model as an optimization problem. The resulting problem can be solved with classical optimization software. As we use the solver Gurobi, we give in Section 4.2 a short introduction to Gurobi. Then, we present the concrete optimization problem for the crane bridge example in Subsection 4.3.1 and consider different numerical examples in Subsection 4.3.2. In the end, we introduce the simple optimization problem here for the configuration of the modular system for the binpacking example in Section 4.3.3 and again consider some numerical examples in Subsection 4.3.4.

We start with the deduction of the reformulation of P^{gen}, which is well-defined in the end.

4.1 Solution approach

A standard idea from the field of mixed-integer optimization is to introduce binary variables for "possible cases". In our case, that means we introduce for every possible variant i of a component c a binary variable. For this procedure, we need for the algorithmic handling that the number of these binary variables is finite, which means in this case that the number of variants is bounded. The boundedness from below by zero is directly given since the number of variants is nonnegative. However, boundedness from above is not a problem either. This assumption is completely natural in applications, since capacities, mainly spatial, are limited. As the optimization model P^{gen} is already mixed-integer, the additional binary variables and corresponding constraints do not change the problem structure, only the number of variables and constraints increases. Although the class of mixed-integer problems is NP-hard, in many real-world problems standard solvers can treat instances with several thousands of integer variables in reasonable time. In the case of linear mixed-integer optimization problems, [33, Figure 16.1, Table 16.1] shows, for example, how the solvers improved. The speed of the solvers was examined as well as how many problems could be solved.

More precisely, we set $k_c \in [0, \overline{k}_c]$ for each component c and $\overline{k} = (\overline{k}_1, \dots, \overline{k}_C)$, respectively. In some applications, such as the crane bridge example, the lower bound can be set to one since every component must be available in the modular system, but we can not say this in general. In the binpacking example we do not require this, for example, but the nonnegativity is obvious. We recall at this point that a variant i of a component is, for example, a specific combination of (different) geometry parameters.

The simple idea, as often used, is to introduce binary variables. These should indicate whether a theoretically possible variant for a component is available in the modular system or not. More precisely, we introduce variables $v_i^c \in \mathbb{B}$ for $i = 1, \dots, \overline{k}_c$ and $c = 1, \dots, C$. We introduce them for every theoretically possible variant, so the last index of i is obviously $\overline{k}_c$. Then the following relationship must apply:

$$v_i^c = \begin{cases} 1, & \text{if there exists at least one product } \ell \in \{1, \dots, N\} \text{ with } z_{i,\ell}^c > 0, \\ 0, & \text{if } z_{i,\ell}^c = 0, \text{ for all } \ell = 1, \dots, N. \end{cases} \tag{4.1}$$

We can write (4.1) with logical constraints. Thus, (4.1) is equivalent to

$$v_i^c = 0 \quad \Rightarrow \quad z_{i,\ell}^c = 0, \quad \ell = 1, \dots, N \tag{4.2}$$

for $i = 1, \dots, \overline{k}_c$ and $c = 1, \dots, C$. These constraints can be modeled directly with suitable optimization software or a big-M reformulation.

The number of variants of component c in the modular system is then given by at most $\sum_{i=1}^{\overline{k}_c} v_i^c$. We note at this point that the aforementioned sum becomes as small as possible in the optimization process, as it is included in the objective function to be minimized. Thus, the sum corresponds exactly to the number of variants per component at the end. With this sum or a similar sum, we can also model lower bounds for variants of components, as done in the following examples. In the next step, we replace all k_c through $\overline{k}_c$ in the formulations of the model P^{gen}, together with the constraints (4.2). With that, the number of variables of the optimization model is a fixed number (depending on the upper bounds). The resulting optimization model is “well-defined” in the context that it can be put in a general solver for optimization models.

After replacing the upper index k_c by $\overline{k}_c$ in every indexing of i, that is, in sums or sets, we only have to guaranty that parts of objective function and constraints only enter the optimization model in the case if a corresponding variant exists. Multiplying with the binary variables makes this easily realizable. In particular, with this modeling, $x_{\overline{i}}^c$ for an index $\overline{i} \leq \overline{k}_c$ can be positive, even though it does not occur in the modular system. In combination (by multiplying) with $v_{\overline{i}}^c = 0$ the value does not enter. In total, every function of the objective function and the constraints, which depends on a variant i of the component c must be supplemented respectively by the binary variable v_i^c.

We denote the new well-defined optimization problem by $\overline{P^{gen}}$ and replace every occurrence of κ with the new variables. We collect all binary variables v_i^c in the vector $v \in \mathbb{B}^{e^\top \overline{k}}$. As costs functions, we now consider $c^{size}(v, \xi, z)$ and $c^{prop}(v, \xi, z)$ instead of $c^{size}(\kappa, \xi, z)$ and $c^{prop}(\kappa, \xi, z)$ from P^{gen}. For the sets $X(\kappa)$ and $Z(\kappa, \xi)$ from P^{gen}, the replacement results in $X(v)$ or X, depending on the application, and in $Z(v, \xi)$. By allocating the restrictions given in Equation (4.2) to Z, we obtain the dependence of v in Z in the new formulation. We obtain a well-defined formulation of a general optimization problem for the optimal configuration of a modular system:

General Optimization Problem

The well-defined general optimization problem for the optimal configuration of a modular system is given by

$$\begin{aligned}\overline{P^{gen}}: \quad \min_{v,\xi,z} \; & c^{size}(v, \xi, z) + c^{prop}(v, \xi, z) \\ \text{s.t.} \quad & z \in Z(v, \xi), \\ & \xi \in X(v), \\ & z \in \mathbb{Z}^{N \cdot e^\top \overline{k}}, \xi \in \mathbb{R}^{e^\top \overline{k}}, v \in \mathbb{B}^{e^\top \overline{k}}\end{aligned}$$

with $\overline{k} = (\overline{k}_1, \ldots, \overline{k}_C)^\top$.

From an optimization point of view, it is interesting whether P^{gen} is solvable or not. In terms of application, we can assume that the costs for the size of the modular system c^{size} are nonnegative. As c^{prop} are costs for fulfilling the product properties, we model them so that they are also nonnegative. If the product properties are fulfilled perfectly, the costs are zero, otherwise positive. In total, we can assume as a natural condition given by the application that c^{size} and c^{prop} are nonnegative in general for feasible points, which we formulate in the following assumption.

Assumption 4.1.1. *The cost functions c^{size} and c^{prop} of $\overline{P^{gen}}$ are nonnegative for a feasible point of $\overline{P^{gen}}$, if such an point exists.*

With Assumption 4.1.1, the objective function of $\overline{P^{gen}}$ is nonnegative and, therefore, $\overline{P^{gen}}$ cannot be unbounded from below. As in applications the costs functions are often linear or quadratic, and in such cases always continuous, the following assumption is often not further restrictive:

Assumption 4.1.2. *The cost functions c^{size} and c^{prop} of $\overline{P^{gen}}$ are continuous over the feasible set.*

With Assumption 4.1.2 and if the feasible set is compact $\overline{P^{gen}}$ can only be solvable or inconsistent by Weierstrass. From the application, it is clear that the feasible set is compact. The number variables z and the variables ξ are bounded from below by zero and above by an upper bound, which is given through the application. We cannot rule out in general the possibility that the feasible set is empty.

Corollary 4.1.3. *The optimization problem $\overline{P^{gen}}$ is either solvable with minimal point $(v^\star, \xi^\star, z^\star)$ and optimal value obj or inconsistent.*

Before considering the examples and their numerical tests, we present the programming background of the work. There are many different types of optimization software and interfaces, from which we have chosen Python with the solver *Gurobi*. In the following section, we give a short introduction in the solver Gurobi to motivate the necessary reformulations in Section 4.3.

4.2 The Solver Gurobi

For modeling and solving the optimization models in this work, we use the *Gurobi Optimizer* or, shortly, *Gurobi*. Gurobi is a well-known choice for modeling and solving optimization problems to global optimality. The company's website advertises that it is the "World's fastest solver" [26]. Gurobi is available with different interfaces like *C, C++, Python, MATLAB*, and more. We use the Python interface with a current Python installation (version 3.12.7). The Gurobi Python interface can be easily used with its own package *gurobipy*. We used version 12.0.0 of gurobipy.

"Gurobi follows a model-and-solve paradigm"[24, Chapter 1]. As a user, we can simply translate the model formulation of our classical optimization problems with variables, constraints, and the objective function into "Gurobi syntax" and if the corresponding functions have a structure that is allowed for the respective problem, we can try to solve the problem.

The problem $\overline{P^{gen}}$ contains in general continuous variables (ξ), integer variables (z) and binary variables (z). For Gurobi, all of these variable types are allowed. Gurobi allows for several constraints types, in general. We are interested in quadratic constraints that are both convex and nonconvex, especially constraints of the form like

$$x^\top Qx + q^\top x \leq b$$

with $Q \in \mathbb{R}^{n\times n}$, $q \in \mathbb{R}^n$ and $b \in \mathbb{R}$. Since we do not require convexity, we have no special requirements for the matrix Q. Problems can certainly be solved more quickly if the constraint is convex, that is, if Q is positive semidefinite [24, 1.2.3]. Gurobi also includes "an additional set of higher-level constraints" [24, 1.2.4], which are denoted as *general constraints*. We are interested in such named "simple constraints", which allow one to model simple relationships between variables. We need "If-Else conditions", which are denoted as *Indicator constraints* in Gurobi. The idea is that "techniques for translating these conditions into lower-level modeling objects (typically using auxiliary binary variables [...]) are well known" and the reformulation will be part of the solution process and is not done by the user. This makes the model easy to read.

Gurobi allows, besides linear and piecewise linear, also quadratic objective functions, which is most interesting for us in addition to linear ones. In this case, also, the solver does not initially care whether the function is convex or not [24, 1.3].

"The Gurobi algorithms work on solving a model until they find a solution that is optimal [...] within the specified tolerances,"[24, 1.3]. Depending on the model (with or without integer variables), there are several algorithms and corresponding tolerances. The feasibility tolerances *FeasibilityTol* and *IntFeasTol* are interesting for us. These must both be taken into account if the model has both continuous and integer variables. As optimality tolerances, we take into account the tolerances *MIPGap* and *OptimalityTol*. If the runtime is too high, increasing the tolerances can be an approach, but we have to accept worse optimal points. The default values for the tolerances can be taken from the documentation [24, 5.1.2].

Even in nonconvex cases, if a solution exists and if there is enough time for the solution process, the "algorithms in Gurobi explore the entire search space [...] and [...] will find a globally optimal solution (subject to tolerances)" [24, 1.2.3]. The aspect of global optimality is very remarkable compared to other solvers, which often only provide a certificate of local optimality in the case of termination. However, depending on the problem, a long runtime can, of course, be a problem. However, in principle, the solver is a promising choice.

All experiments from the further work were run on an Intel i7 processor with 8 cores with 3.60 GHz and 32 GB of RAM. The Python code can be viewed in the GitLab repository [21]. The README-file located there explains how results can be reproduced.

Next, we consider the running examples and we deduce the concrete optimization problems.

4.3 Examples

In this section, we continue with the already known examples for the modular systems for crane bridges and the binpacking problem of Section 3.2. We apply the solution approach from Section 4.1 to them and formulate well-defined optimization problems. We start with the crane bridge application and deduce the optimization problem.

4.3.1 Optimization Problem for the Crane Bridge Modular System

The modular system, considered here, contains two components, profiles and sheets. We assume an upper bound of $\overline{n}$ for the profile variants and $\overline{m}$ for the sheet variants. This gives us, in relation to the solution approach presented, the variables $\xi = (P^1, \dots, P^{\overline{n}}, S^1, \dots, S^{\overline{m}})$. P^i contains the variables for the profile variant i with $P^i = \left(h_i^P, t_i^P, w_i^P\right) \in \mathbb{R}^3$ for $i = 1, \dots, \overline{n}$ and S^j the variables for the sheet variant j, more precisely $S^j = \left(h_j^S, l_j^S, t_j^S, w_j^S\right) \in \mathbb{R}^4$ with $j = 1, \dots, \overline{m}$. We obtain $\xi \in \mathbb{R}^{3\overline{n}+4\overline{m}}$.

The number variables for the number profiles in a crane bridge ℓ are given respectively by $z_{i,\ell}^P$ with $i = 1, \dots, \overline{n}$ and $\ell = 1, \dots, N$ and those for the sheets by $z_{i,\ell}^S$ with $j = 1, \dots, \overline{m}$. If we sum up all the number variables in a vector z, we obtain $z \in \mathbb{Z}^{N(\overline{n}+\overline{m})}$.

The variables for the existence of a specific variant are given by v_i^P for the profiles and, respectively, v_j^S for the sheets. Together we have $v \in \mathbb{B}^{\overline{n}+\overline{m}}$. We combine them in the sense that a number variable $z_{i,\ell}^P$, for example, has to be zero if the corresponding variant does not exist, that is, if $v_i^P = 0$. Thus, we get the following indicator constraints

$$v_i^P = 0 \Rightarrow z_{i,\ell}^P = 0, \quad i = 1, \dots, \overline{n},\ \ell = 1, \dots, N \tag{4.3}$$

$$v_i^S = 0 \Rightarrow z_{j,\ell}^S = 0, \quad j = 1, \dots, \overline{m},\ \ell = 1, \dots, N. \tag{4.4}$$

Such constraints can be modeled with Gurobi, see Section 4.2.

As in the model introduction in Subsection 3.2.1, we first consider the constraints that depend only on ξ and then the constraints that depend on ξ and z. For the first type of constraints, we only need box constraints for the corresponding geometry variables, which are already collected in the sets X^1 and X^2. Thus, we obtain

$$P^i = \left(h_i^P, t_i^P, w_i^P\right) \in X^1, \quad i = 1, \dots, \overline{n} \tag{4.5}$$

$$S^j = \left(h_j^S, l_j^S, t_j^S, w_j^S\right) \in X^2, \quad j = 1, \dots, \overline{m} \tag{4.6}$$

and the corresponding set

$$\begin{aligned} X = \big\{ &(P^i)_{i=1,\dots,\overline{n}} \in \mathbb{R}^{3\overline{n}},\ (S^j)_{j=1,\dots,\overline{m}} \in \mathbb{R}^{4\overline{m}} | \\ &P^i \in X^1,\ i = 1, \dots, \overline{n}, \\ &S^j \in X^2,\ j = 1, \dots, \overline{m} \big\}. \end{aligned}$$

Unlike in the problem $\overline{P^{gen}}$, here the set X does not depend on the variable v, since we do not need this dependency in this specific application.

The remaining constraints are given by

$$\begin{aligned} \left|\{i \in \{1, \dots, \overline{n}\} |\ z_{i,\ell}^P > 0\}\right| = 1, \quad \ell = 1, \dots, N, \\ \left|\{j \in \{1, \dots, \overline{m}\} |\ z_{j,\ell}^S > 0\}\right| = 1, \quad \ell = 1, \dots, N, \end{aligned} \tag{4.7}$$

$$\begin{aligned} z_{i,\ell}^P \geq 0, \quad i = 1, \dots, \overline{n},\ \ell = 1, \dots, N, \\ z_{j,\ell}^S \geq 0, \quad j = 1, \dots, \overline{m},\ \ell = 1, \dots, N, \end{aligned} \tag{4.8}$$

$$\begin{aligned} z_{i,\ell}^P,\ z_{j,\ell}^S > 0 \quad \Rightarrow \quad z_{i,\ell}^P = 4 \left\lfloor \frac{L^\ell}{2 l_j^S} \right\rfloor - 2, \\ i = 1, \dots, \overline{n},\ j = 1, \dots, \overline{m},\ \ell = 1, \dots, N, \end{aligned} \tag{4.9}$$

$$\begin{aligned} z_{j,\ell}^S > 0 \quad \Rightarrow \quad z_{j,\ell}^S = 2 \left\lfloor \frac{L^\ell}{2 l_j^S} \right\rfloor - 2, \\ j = 1, \dots, \overline{m},\ \ell = 1, \dots, N, \end{aligned} \tag{4.10}$$

$$\begin{aligned} z_{i,\ell}^P,\ z_{j,\ell}^S > 0 \quad \Rightarrow \quad M(P^i, S^j) \geq M^\ell, \\ i = 1, \dots, \overline{n},\ j = 1, \dots, \overline{m},\ \ell = 1, \dots, N, \end{aligned} \tag{4.11}$$

$$\begin{aligned} z_{i,\ell}^P,\ z_{j,\ell}^S > 0 \quad \Rightarrow \quad (P^i, S^j) \in F \\ i = 1, \dots, \overline{n},\ j = 1, \dots, \overline{m},\ \ell = 1, \dots, N. \end{aligned} \tag{4.12}$$

Details and interpretation of these constraints are given in Subsection 3.2.1. The corresponding set notation for crane bridge ℓ is then explicitly given by

$$
\begin{aligned}
Z_\ell(v,\xi) = \{(z^P_{i,\ell})_{i=1,\dots,\overline{n}} \in Z^{\overline{n}}\ (z^S_{j,\ell})_{j=1,\dots,\overline{m}} \in \mathbb{Z}^{\overline{m}} |\ & \\
& v^P_i = 0 \Rightarrow z^P_{i,\ell} = 0, \\
& v^P_i = 0 \Rightarrow z^S_{j,\ell} = 0, \\
& z^P_{i,\ell} \geq 0,\ z^S_{j,\ell} \geq 0, \\
& |\{i \in \{1,\dots,\overline{n}\}|\ z^P_{i,\ell} > 0\}| = 1, \\
& |\{j \in \{1,\dots,\overline{m}\}|\ z^S_{j,\ell} > 0\}| = 1, \\
& z^P_{i,\ell},\ z^S_{j,\ell} > 0 \ \Rightarrow\ z^P_{i,\ell} = 4\left\lfloor \frac{L^\ell}{2l^S_j} \right\rfloor - 2, \\
& z^S_{j,\ell} > 0 \ \Rightarrow\ z^S_{j,\ell} = 2\left\lfloor \frac{L^\ell}{2l^S_j} \right\rfloor - 2, \\
& z^P_{i,\ell},\ z^S_{j,\ell} > 0 \ \Rightarrow\ M(P^i,S^j) \geq M^\ell, \\
& z^P_{i,\ell},\ z^S_{j,\ell} > 0 \ \Rightarrow\ (P^i,S^j) \in F, \\
& i = 1,\dots,\overline{n},\ j = 1,\dots,\overline{m}, \}
\end{aligned}
$$

which clearly has the same structure as before, but all indices run to $\overline{n}$ and $\overline{m}$, respectively, and we add the constraints with the binary variables v. If we collect all sets, we obtain $Z(v,\xi) = Z_1(v,\xi) \times \cdots \times Z_N(v,\xi)$.

With equations (4.3), (4.4), and (4.7), it is not necessary to require that any profile variant and any sheet variant is available. This is implicitly given.

Altogether we obtain the well-defined optimization model for the crane bridge example

Modular System Crane Bridge

The well-defined optimization problem for the optimal configuration of a modular system for crane bridges is given by

$$
\begin{aligned}
\overline{P}^{crane,w}:\quad \min_{v,\xi,z}\quad & c^P \sum_{i=1}^{\overline{n}} v^P_i + c^S \sum_{j=1}^{\overline{m}} v^S_j \\
& + c_d \sum_{\substack{i,j,\ell=1 \\ (z^P_{i,\ell}>0)\&(z^S_{j,\ell}>0)}}^{\overline{n},\overline{m},N} \Big(M(P^i,S^j) - M^\ell \Big) \\
& + c_w \sum_{\substack{i,j,\ell=1 \\ (z^P_{i,\ell}>0)\&(z^S_{j,\ell}>0)}}^{\overline{n},\overline{m},N} w\left(P^i,S^j,z^P_{i,\ell},z^S_{j,\ell}\right) \\
\text{s.t.}\quad & (4.3),(4.4),(4.7),(4.8),(4.9),(4.10),(4.11),(4.12), \qquad \#\ z \\
& (4.5),(4.6) \qquad \#\ \xi \\
& z \in \mathbb{Z}^{N(\overline{n}+\overline{m})},\ \xi \in \mathbb{R}^{3\overline{n}+4\overline{m}},\ v \in \mathbb{B}^{\overline{n}+\overline{m}},
\end{aligned}
$$

where $\xi = (P^1,\dots,P^{\overline{n}},S^1,\dots,S^{\overline{m}})$.

The optimization problem $\overline{P}^{crane,w}$ is, in general, a mixed-integer problem. However, there are several difficulties. First, the logical conditions in the sum in the objective function. Since we do not want to introduce more variables here, we use the notation with a "logical and". In the implementation, we introduce at this point new binary variables that are one if and only if a profile-sheet-variant (i, j) is chosen (we denote them by $b^{PS}_{i,j,\ell}$). With a similar approach, the left-hand side of the indicator constraints in the set $Z(v, \xi)$ can be replaced, which fits in the notation of indicator constraints in Gurobi, see Section 4.2. The rounding conditions for the profile and sheet variants used for the different crane bridges can also not be modeled directly in Gurobi or other solvers, but can be reformulated easily, as well as the reformulation of the load capacity function M from (3.8) in a linear or quadratic function, which is necessary for our solver. As the reformulations make the model more difficult to read, we omitted the complete implementable model here and moved this to the appendix, Subsection A.1.1. The problems are denoted by $\overline{P}^{crane,w}_{complete}$ and $\overline{P}^{crane}_{complete}$ in the appendix.

In the following subsection, we consider different problem instances of the crane bridge example, where we optimize in particular, sometimes with and sometimes without total weight of the bridges, and compare the results. For simplicity we omit the $_{complete}$. We denote the optimization problem without weight minimization by $\overline{P}^{crane}$ and the other case as above by $\overline{P}^{crane,w}$. In $\overline{P}^{crane}$, the exact number of profiles and sheets of the modular system, which correspond to the variables z, does not enter the objective or another used function (such as the load capacity function M). It is only interesting which variant combination is used for a specific crane bridge. With the variables $b^{PS}_{i,j,\ell}$ above, this is easy to model and we obtain

$$c_d \sum_{i,j,\ell=1}^{\overline{n},\overline{m},\overline{N}} b^{PS}_{i,j,\ell} \left(M(P^i, S^j) - M^\ell \right)$$

as part of the objective function for the load capacity. There is no dependency on the variables z. For that reason, we could omit the variables z in the optimization problem as a variable, also the associated constraints, and calculate the exact number after the solving process through

$$z^P_{i,\ell} = 4 \left\lfloor \frac{L^\ell}{2l^S_j} \right\rfloor - 2, \quad i = 1, \ldots, \overline{n}$$

and

$$z^S_{j,\ell} = 2 \left\lfloor \frac{L^\ell}{2l^S_j} \right\rfloor - 2, \quad j = 1, \ldots, \overline{m}.$$

This should improve solution times, since we have fewer variables and fewer complicating constraints (indicator constraints and rounding conditions).

In problems with the minimization of the weight, the corresponding number of profiles and sheets, of course, is important and has to enter as variable in the optimization problem. The weight function w can be easily derived, as the profiles and sheets are constructed simply, as can be seen in Figure 3.1. The components are built of steel, see [6, 3.5] and the explicitly weight function and its derivation can

name	weight	description
CRANE_N5_1	0	small illustrative example
CRANE_N5_2	1	like CRANE_N5_1, but with weight opt.
CRANE_N5_3	0	like CRANE_N5_1, but larger $[\overline{n}, \overline{m}]$
CRANE_N5_4	0	like CRANE_N5_3, but larger $[\overline{n}, \overline{m}]$
CRANE_N20_1	0	higher demand of cranes
CRANE_N20_2	1	like CRANE_N20_1, but with weight opt.
CRANE_N20_3	0	like CRANE_N20_1, but smaller c^P
CRANE_N20_4	0	like CRANE_N20_1, but larger $\overline{m}$

TABLE 4.1: Overview problem instances crane bridge problem

be found in the Appendix in Equation (A.5). After the deduction of a well-defined optimization problem for the crane bridge example, in the following subsection we consider some numerical tests.

4.3.2 Numerical Tests for the Crane Bridge Modular System

We consider several input data for the crane bridge example and different objective functions, in particular with or without minimization of the total weight of the modular system. If the variable z does not explicitly enter in the objective function, which is the case, when we do not optimize the weight of the bridges, we omit z as a variable of the model and also omit the corresponding constraints. We calculate the number of profiles and sheets and the weight of the bridges in these cases after optimization. This allows us to reduce the number of variables, and we can omit some of the complicating constraints.

Table 4.1 gives an overview of the problem instances considered. We divide the tests into two parts. Firstly, we consider tests with a demand of five crane bridges (upper part of the table) and then examples with 20 crane bridges (lower part of the table). We start with five crane bridge examples.

Small Crane Bridge Examples

We distinguish between without and with weight optimization (CRANE_N5_1 vs. CRANE_N5_2), different maximum number of variants (CRANE_N5_1 vs. CRANE_N5_3, CRANE_N5_4). "N5_1" denotes in this case the first problem instance of an example with 5 crane bridges. We describe the input data and the results in more detail for the problems CRANE_N5_1 and CRANE_N5_2 in the following. For the other examples, we will focus only on the main aspects.

CRANE_N5_1	
$[\overline{n}, \overline{m}]$	[2, 2]
N	5
$[c^P, c^S]$	[10, 5]
c_d	10
runtime	0.33
total costs	36.1451
variant costs	30.00
deviation costs	6.15
weight	3.49
# cont. var.	42
# integer var.	24

ℓ	i	j	$M\,[t]$	$M^\ell\,[t]$	$L^\ell\,[m]$	$w\,[t]$
1	1	1	14.00	14.00	5.0	0.59
2	2	2	10.53	10.00	3.0	0.11
3	1	2	8.00	8.00	5.0	0.51
4	2	1	6.00	6.00	10.0	0.74
5	1	2	3.08	3.00	13.0	1.55

TABLE 4.2: CRANE_N5_1: input data, optimization results, mapping

i	w^P	h^P
1	200.00	98.03
2	100.00	40.00

j	w^S	h^S	l^S
1	400.00	966.47	600.00
2	400.00	489.61	550.41

TABLE 4.3: CRANE_N5_1: details profiles and sheets in millimeters

In problems CRANE_N5_1 and CRANE_N5_2 we consider a very small example with only two allowed variants of profiles and sheets. In the first example, we do not consider the weight of the bridges in the optimization problem, in the second, we do. Important input data and results for the first example are given in Tables 4.2 and 4.3 for CRANE_N5_1 and in Tables 4.4 and 4.5 for CRANE_N5_2. In the upper part of the first tables, respectively, we have on the left side, in addition to the maximum number of variants $(\overline{n}, \overline{m})$ and the number of crane bridges (N), the cost coefficients $(c^P,\, c^S,\, c_d)$. In the case of weight optimization, there is also the coefficient for the weight costs (c_w). In the lower part there are some optimization results and statistics, such as runtime, total costs, which means the optimal value of the problem, which is divided in the different parts. Also, the total weight of the bridges is given, which depending on the example is part of the optimization or not. If the weight of the bridges is part of the objective function, the total weight is obviously lower or equal than in the case without weight optimization (here 3.40 tons vs. 3.49 tons).

At the end of the left tables, there is the number of variables of the complete optimization model, which means the number of continuous variables and the number of integer variables, which includes the binary variables as well. If we compare CRANE_N5_1 and CRANE_N5_2, there is through the weight optimization in CRANE_N5_2 a higher number of variables since we need many auxiliary variables for modeling (66 vs. 144 in total).

The right part of the tables shows the explicit mapping of profile and sheet variant for each crane bridge, as well as the required span L^ℓ in meters of it, the weight w of a bridge in tons, and the actual and required load capacity (M and M^ℓ

CRANE_N5_2	
$[\overline{n}, \overline{m}]$	$[2, 2]$
N	5
$[c^P, c^S]$	$[10, 5]$
c_d	10
c_w	100
runtime	0.56
total costs	410.0638
variant costs	30.00
deviation costs	40.42
weight costs	339.64
weight	3.40
# cont. var.	90
# integer var.	54

ℓ	i	j	$M\,[t]$	$M^\ell\,[t]$	$L^\ell\,[m]$	$w\,[t]$
1	2	1	14.00	14.00	5.0	0.54
2	2	2	13.55	10.00	3.0	0.20
3	2	2	8.13	8.00	5.0	0.50
4	1	1	6.23	6.00	10.0	0.78
5	2	2	3.13	3.00	13.0	1.37

TABLE 4.4: CRANE_N5_2: input data, optimization results, mapping

i	w^P	h^P
1	100.00	55.00
2	172.41	100.00

j	w^S	h^S	l^S
1	400.00	970.00	593.86
2	344.81	518.79	597.73

TABLE 4.5: CRANE_N5_2: details profiles and sheets in millimeters

in tons). The span of the bridge must be exactly met, and the load capacity must not be exceeded. For this reason, the actual load capacity is also denoted in the table. In CRANE_N5_1, for example, crane bridges 2 and 5 have a higher load capacity as required, which is generally allowed in the optimization model. The explicit index of used variants does not have a specific meaning and is not starting at one in general. If we consider, for example, at most two variants of profiles ($\overline{n} = 2$) and if there is only one variant optimal, the index can be 1 or 2. Detailed results for the geometry variables are given in the corresponding Tables 4.3 and 4.5. In both examples, all possible variants of profiles and sheets are used in the solution, as we can see in the tables.

In the Appendix in Subsection A.1.2 additional input data are given, such as the tolerances for the solver, see Section 4.2. They become interesting in further examples and differ depending on the size of the problem. Some of the input data are quite difficult for the solver, and thus we choose the different tolerances (see Section 4.2) for the solvers in general with 10^{-3}, such that the results of the different examples are more comparable. The default values are significantly smaller. In this table, it is also denoted if the model was solved to global optimality or if the time limit was reached. If data from these tables are interesting and important, we will mention it.

Through numerical issues, the tolerances in example CRANE_N5_2 have to be increased compared to example CRANE_N5_1 and runtimes are not comparable in this case.

In the examples CRANE_N5_3 and CRANE_N5_4 we increase the maximum number of variants of profiles ($\overline{n}$) and sheets ($\overline{m}$) to 3 and 5 respectively. We optimize without the total weight, which means that we can compare the results with the results of example CRANE_N5_1. Input data and results are given in Tables 4.6 and 4.7 for example CRANE_N5_3 and in Tables 4.8 and 4.9 for CRANE_N5_4. In the example CRANE_N5_4 the tolerance for integer feasibility was increased because otherwise the solver did not finish within the time limit of 24 hours. From CRANE_N5_1 to CRANE_N5_3 we can see that we can reduce the total costs by allowing more variants. In the solution of CRANE_N5_3 three sheet variants are optimal and the costs are a little lower than before (35.76 vs. 36.14). It is clear that the total costs cannot increase as the previous solution remains feasible for the example with more allowed variants. A further increase of the maximum number of variants changes nothing and two profile and three sheet variants remain optimal. By increasing the maximum number of variants, the deviation costs can be reduced (from 6.15 to 0.7).

CRANE_N5_3	
$[\overline{n}, \overline{m}]$	[3, 3]
N	5
$[c^P, c^S]$	[10, 5]
c_d	10
runtime	44.02
total costs	35.7603
variant costs	35.00
deviation costs	0.76
weight	2.74
# cont. var.	84
# integer var.	36

ℓ	i	j	$M\,[t]$	$M^\ell\,[t]$	$L^\ell\,[m]$	$w\,[t]$
1	3	3	14.00	14.00	5.0	0.55
2	1	1	10.00	10.00	3.0	0.09
3	1	2	8.00	8.00	5.0	0.35
4	1	3	6.00	6.00	10.0	0.72
5	1	2	3.08	3.00	13.0	1.03

TABLE 4.6: CRANE_N5_3: input data, optimization results, mapping

i	w^P	h^P
1	100.00	40.00
3	184.29	99.95

j	w^S	h^S	l^S
1	300.00	471.58	505.24
2	330.55	573.89	289.38
3	374.43	970.71	595.60

TABLE 4.7: CRANE_N5_3: details profiles and sheets in millimeters

CRANE_N5_4	
$[\overline{n}, \overline{m}]$	[5, 5]
N	5
$[c^P, c^S]$	[10, 5]
c_d	10
runtime	10.25
total costs	35.7692
variant costs	35.00
deviation costs	0.78
weight	3.77
# cont. var.	210
# integer var.	60

ℓ	i	j	$M\,[t]$	$M^\ell\,[t]$	$L^\ell\,[m]$	$w\,[t]$
1	4	3	14.00	14.00	5.0	0.63
2	3	2	10.00	10.00	3.0	0.19
3	4	5	8.00	8.00	5.0	0.49
4	4	5	3.08	3.00	13.0	1.65
5	3	3	6.00	6.00	10.0	0.82

TABLE 4.8: CRANE_N5_4: input data, optimization results, mapping

i	w^P	h^P
3	100.00	45.86
4	200.00	99.96

j	w^S	h^S	l^S
2	300.00	404.59	288.87
3	400.00	942.83	473.92
5	400.00	483.12	528.13

TABLE 4.9: CRANE_N5_4: details profiles and sheets in millimeters

CRANE_N20_1	
$[\overline{n}, \overline{m}]$	[10, 5]
N	20
$[c^P, c^S]$	[20, 10]
c_d	1
runtime	266.87
total costs	90.4002
variant costs	40.00
deviation costs	50.40
weight	13.33
# cont. var.	1,150
# integer var.	315

ℓ	i	j	$M\,[t]$	$M^\ell\,[t]$	$L^\ell\,[m]$	$w\,[t]$
1	7	2	14.00	14.00	5.0	0.55
2	7	4	10.62	10.00	3.0	0.21
3	7	2	14.00	8.00	5.0	0.55
4	7	2	5.38	3.00	13.0	1.56
5	7	2	7.00	6.00	10.0	1.18
6	7	4	7.96	5.00	4.0	0.30
7	7	4	9.10	5.00	3.5	0.30
8	7	2	7.78	6.00	9.0	1.05
9	7	2	8.75	7.00	8.0	0.93
10	7	4	15.92	8.00	2.0	0.12
11	7	4	6.37	6.00	5.0	0.39
12	7	4	10.62	7.00	3.0	0.21
13	7	2	14.00	9.00	5.0	0.55
14	7	2	5.38	4.00	13.0	1.56
15	7	2	7.00	7.00	10.0	1.18
16	7	4	7.96	6.00	4.0	0.30
17	7	4	9.10	7.00	3.5	0.30
18	7	2	7.78	7.00	9.0	1.05
19	7	2	8.75	8.00	8.0	0.93
20	7	4	15.92	10.00	2.0	0.12

TABLE 4.10: CRANE_N20_1: input data, optimization results, mapping

i	w^P	h^P
7	145.52	87.49

j	w^S	h^S	l^S
2	400.00	1,000.00	500.00
4	300.00	400.00	421.92

TABLE 4.11: CRANE_N20_1: details profiles and sheets in millimeters

After considering four small examples, we continue with the examples with 20 crane bridges.

Larger Crane Bridge Examples

As five crane bridges are mostly not a realistic demand, in the following examples, we consider a demand of 20 crane bridges. This leads to a total number of variables that is many times higher. For the example without weight optimization, example CRANE_N20_1, a solution was found in an acceptable time and one profile and two sheet variants are optimal. In example CRANE_N20_2 we consider the same input data but with weight optimization. One special detail here was that we have to decrease the integer feasibility tolerance to 10^{-6}, since otherwise numerical issues occurred and integer variables variables are not recognized as integer ones. The other tolerances are set to 10^{-3} so that the solver terminates within the time limit of 24 h. The input data and results for CRANE_N20_1 are given in Tables 4.10 and 4.11 and for the examples with weight optimization in Tables 4.12 and 4.13.

The runtime and number of variables are significantly higher than in the examples with only five crane bridges, and also if we compare the examples CRANE_N20_1 and CRANE_N20_2 (without and with weight optimization). In

CRANE_N20_2	
$[\overline{n}, \overline{m}]$	[10, 5]
N	20
$[c^P, c^S]$	[20, 10]
c_d	1
c_w	10
runtime	289.37
total costs	293.7517
variant costs	30.00
deviation costs	162.49
weight costs	101.26
weight	10.13
# cont. var.	2,400
# integer var.	715

ℓ	i	j	$M\,[t]$	$M^\ell\,[t]$	$L^\ell\,[m]$	$w\,[t]$
1	10	5	14.00	14.00	5.0	0.42
2	10	5	23.33	10.00	3.0	0.17
3	10	5	14.00	8.00	5.0	0.42
4	10	5	5.38	3.00	13.0	1.17
5	10	5	7.00	6.00	10.0	0.92
6	10	5	17.50	5.00	4.0	0.29
7	10	5	20.00	5.00	3.5	0.17
8	10	5	7.78	6.00	9.0	0.80
9	10	5	8.75	7.00	8.0	0.67
10	10	5	35.00	8.00	2.0	0.04
11	10	5	14.00	6.00	5.0	0.42
12	10	5	23.33	7.00	3.0	0.17
13	10	5	14.00	9.00	5.0	0.42
14	10	5	5.38	4.00	13.0	1.17
15	10	5	7.00	7.00	10.0	0.92
16	10	5	17.50	6.00	4.0	0.29
17	10	5	20.00	7.00	3.5	0.17
18	10	5	7.78	7.00	9.0	0.80
19	10	5	8.75	8.00	8.0	0.67
20	10	5	35.00	10.00	2.0	0.04

TABLE 4.12: CRANE_N20_2: input data, optimization results, mapping

i	w^P	h^P
10	100.00	100.00

j	w^S	h^S	l^S
5	371.52	1,000.00	590.91

TABLE 4.13: CRANE_N20_2: details profiles and sheets in millimeters

CRANE_N20_3	
$[\overline{n}, \overline{m}]$	[10, 5]
N	20
$[c^P, c^S]$	[5, 10]
c_d	1
runtime	177.25
total costs	65.2619
variant costs	35.00
deviation costs	30.26
weight	10.42
# cont. var.	1,150
# integer var.	315

ℓ	i	j	$M\,[t]$	$M^\ell\,[t]$	$L^\ell\,[m]$	$w\,[t]$
1	4	2	14.00	14.00	5.0	0.55
2	4	5	10.62	10.00	3.0	0.22
3	7	2	12.39	8.00	5.0	0.39
4	7	2	4.76	3.00	13.0	1.11
5	7	2	6.19	6.00	10.0	0.84
6	2	5	6.73	5.00	4.0	0.20
7	2	5	7.70	5.00	3.5	0.20
8	7	2	6.88	6.00	9.0	0.75
9	7	2	7.74	7.00	8.0	0.66
10	2	5	13.47	8.00	2.0	0.08
11	4	5	6.37	6.00	5.0	0.40
12	2	5	8.98	7.00	3.0	0.14
13	7	2	12.39	9.00	5.0	0.39
14	7	2	4.76	4.00	13.0	1.11
15	4	2	7.00	7.00	10.0	1.19
16	7	5	6.56	6.00	4.0	0.20
17	7	5	7.49	7.00	3.5	0.20
18	2	2	7.00	7.00	9.0	0.77
19	4	2	8.75	8.00	8.0	0.94
20	2	5	13.47	10.00	2.0	0.08

TABLE 4.14: CRANE_N20_3: input data, optimization results, mapping

i	w^P	h^P
2	100.00	46.90
4	150.00	86.90
7	100.00	40.00

j	w^S	h^S	l^S
2	400.00	999.94	499.97
5	300.00	400.12	424.36

TABLE 4.15: CRANE_N20_3: details profiles and sheets in millimeters

the model with weight optimization, the total weight of the bridges is obviously smaller than in the other example, as expected (10.13 t vs. 13.33 t).

In the third experiment for the larger dataset, we consider cheaper profile variants and set $c^P = 1$ (instead of $c^P = 20$). We expect more profile variants, which is then the case since in the optimal modular system there are three profile variants optimal (instead of one). The input data and results are given in Tables 4.14 and 4.15.

In a last experiment, example CRANE_N20_4, we consider the same demand of crane bridges with the same costs as in CRANE_N20_1, but with more sheet variants (ten instead of five). The input data and results are given in Tables 4.16 and Table 4.17.

The runtime is non surprisingly higher (factor three) than in example CRANE_N20_1 as we have more variables and more constraints. In the optimal modular system, we have one profile and three sheet variants (see Table 4.17). The number of variants for profiles and sheets differs from example CRANE_N20_1, where we have only two sheet variants. In example CRANE_N20_4 the optimal value is slightly lower than in example CRANE_N20_1 (89.36 vs. 90.40). This fact is surprising, since the optimal point of example CRANE_N20_4 is also feasible for example CRANE_N20_1, but is not found by the solver in the previous exam-

CRANE_N20_4	
$[\overline{n}, \overline{m}]$	[10, 10]
N	20
$[c^P, c^S]$	[20, 10]
c_d	1
runtime	784.13
total costs	89.3689
variant costs	50.00
deviation costs	39.37
weight	13.21
# cont. var.	2,270
# integer var.	420

ℓ	i	j	$M\,[t]$	$M^\ell\,[t]$	$L^\ell\,[m]$	$w\,[t]$
1	7	9	14.00	14.00	5.0	0.55
2	7	10	10.72	10.00	3.0	0.21
3	7	3	9.00	8.00	5.0	0.49
4	7	3	3.46	3.00	13.0	1.49
5	7	9	7.00	6.00	10.0	1.18
6	7	10	8.04	5.00	4.0	0.30
7	7	10	9.19	5.00	3.5	0.30
8	7	9	7.78	6.00	9.0	1.05
9	7	9	8.75	7.00	8.0	0.93
10	7	10	16.08	8.00	2.0	0.12
11	7	10	6.43	6.00	5.0	0.47
12	7	10	10.72	7.00	3.0	0.21
13	7	3	9.00	9.00	5.0	0.49
14	7	9	5.38	4.00	13.0	1.56
15	7	9	7.00	7.00	10.0	1.18
16	7	10	8.04	6.00	4.0	0.30
17	7	10	9.19	7.00	3.5	0.30
18	7	9	7.78	7.00	9.0	1.05
19	7	9	8.75	8.00	8.0	0.93
20	7	10	16.08	10.00	2.0	0.12

TABLE 4.16: CRANE_N20_4: input data, optimization results, mapping

i	w^P	h^P
7	145.51	89.51

j	w^S	h^S	l^S
3	400.00	550.52	380.94
9	400.00	994.11	497.05
10	300.00	400.00	414.33

TABLE 4.17: CRANE_N20_4: details profiles and sheets in millimeters

ple. In other words, the optimal point of the problem in example CRANE_N20_1 is good enough within the tolerances of the optimization problem. We explain this by the inaccuracy of the solver. The difference in the objective function values is not significant, but the deviation in the number of variants at the minimal point is remarkable.

The data of the examples, which we consider here, are quite simple and small. If there are more crane bridges and a higher number of possible variants, the number of variables and constraints, especially the logical constraints (with the binary variables), becomes extremely large. Another point is the load capacity function, which occurs in the objective function and in the constraints, here even in an indicator constraint. In the current problem, we only considered a very rough estimation of the load capacity function, which we especially can reformulate as a linear function. If this function gets more complicated, there are many more variables and constraints, and the reformulation in a linear function will certainly be more difficult. It should also be mentioned that we increased the tolerances of the solver and the common default values are significantly smaller than our values. With the default values for the tolerances (mostly 10^{-6}) the solver does not even terminate in the example CRANE_N5_4, where only five crane bridges and a maximum number of five profile and sheet variants are considered within a time limit of 24 hours. This suggests that the solution approach for larger and more complicated problems is not practical, and other solution approaches may become interesting.

Before discussing other approaches in Chapters 5 and 6, we consider the modular system examples for the binpacking problems in the following subsections. In Subsection 4.3.3 we derive concrete optimization problems for different dimensions of the binpacking example, and in Subsection 4.3.4 we consider some numerical tests.

4.3.3 Optimization Problem for the Binpacking Modular System

As in Subsection 3.2.2, we divide this subsection into two parts. First, we consider the one-dimensional binpacking problem.

Optimization Problem for the One-Dimensional Binpacking Problem

In this example with the notation of Subsection 3.2.2 we allow a maximum number of $\overline{n}$ variants for the one component considered. So we have the variables $x_i \in \mathbb{R}$ with $i = 1, \dots, \overline{n}$ for the length of the objects, $z_{i,\ell} \in \mathbb{Z}$, $i = 1 \dots, \overline{n}$, $\ell = 1, \dots, N$ for the number variables and the new variables $v_i \in \mathbb{B}$, $i = 1, \dots, \overline{n}$. As for the first two variables, we omit in v_i the c, since we only consider one component in the one-dimensional case. Thus, we have $\xi = (x_i)_{i=1,\dots,\overline{n}} \in \mathbb{R}^{\overline{n}}$, $z = (z_{i,\ell})_{\substack{i=1,\dots,\overline{n},\\ \ell=1,\dots,N}} \in \mathbb{Z}^{\overline{n}\cdot N}$, and $v = (v_i)_{i=1,\dots,n} \in \mathbb{B}^{\overline{n}}$. In this section, the lengths of the variables are explicitly given, in contrast to Section 3.2.2. The condition that at least one variant has to be available in the modular system (the corresponding condition to $n \geq 1$), can easily be reformulated with

$$\sum_{i=1}^{\overline{n}} v_i \geq 1. \tag{4.13}$$

In this example, we require in addition to $z_{i,\ell} = 0$ also $x_i = 0$, if $v_i = 0$, compare to (4.2). In this simple model, we model this fact as classical in optimization problems with binary variables through

$$x_i \leq v_i \overline{x}, \quad i = 1, \dots, \overline{n}, \tag{4.14}$$

$$z_{i,\ell} \leq v_i \overline{z}^\ell, \quad i = 1, \dots, \overline{n},\ \ell = 1, \dots, N, \tag{4.15}$$

where $\overline{x}$ and $\overline{z}^\ell$ are suitable upper bounds for the corresponding variables x_i and $z_{i,\ell}$. Of course, Gurobi indicator constraints could also be used, but for the sake of simplicity, we will not use them here. It is clear from the constraints (4.14) and (4.15), that x_i and $z_{i,\ell}$ (for each bin ℓ) are zero if the corresponding v_i is zero. If $v_i = 1$, the variables can have a value. Next, the lengths of the objects must differ by a minimum value τ, and logically in this model only for existing variants. In total, we obtain, instead of (3.13), together with the lower bound $\underline{x}$, the following constraints:

$$x_i \geq v_i \underline{x}, \quad i = 1, \dots, \overline{n}, \tag{4.16}$$

$$v_i x_i + \tau v_i \leq v_{i+1} x_{i+1}, \quad i = 1, \dots, \overline{n} - 1. \tag{4.17}$$

Through the formulation in (4.17), the corresponding variables x_i for non-existing variants are sorted at the front and for the others, the variants differ at least by τ. Altogether, we obtain the set

$$\begin{aligned} X(v) = \{\xi \in \mathbb{R}^{\overline{n}} |\ & x_i \geq v_i \underline{x},\ i = 1, \dots, \overline{n}, \\ & x_i \leq v_i \overline{x},\ i = 1, \dots, \overline{n}, \\ & v_i x_i + \tau v_i \leq v_{i+1} x_{i+1},\ i = 1, \dots, \overline{n} - 1.\} \end{aligned}$$

Next, we consider the constraints with ξ and z. The constraints (3.15) and (3.16) become

$$\sum_{i=1}^{\overline{n}} x_i z_{i,\ell} \leq L^\ell, \quad \ell = 1, \dots, N, \tag{4.18}$$

$$\sum_{i=1}^{\overline{n}} z_{i,\ell} \leq \overline{o}_\ell, \quad \ell = 1, \dots, N \tag{4.19}$$

Together with

$$z_{i,\ell} \geq 0, \quad i = 1, \dots, \overline{n},\ \ell = 1, \dots, N \tag{4.20}$$

and equations (4.14), (4.15) we obtain for bin ℓ the set

$$\begin{aligned} Z_\ell(v, \xi) = \big\{(z_{i,\ell})_{i=1,\dots,\overline{n}} \in Z^{\overline{n}} |\ & z_{i,\ell} \geq 0,\ i = 1, \dots, \overline{n}, \\ & z_{i,\ell} \leq v_i \overline{z}_\ell,\ i = 1, \dots, \overline{n}, \\ & \sum_{i=1}^{\overline{n}} x_i z_{i,\ell} \leq L^\ell, \\ & \sum_{i=1}^{\overline{n}} z_{i,\ell} \leq \overline{o}_\ell \big\}. \end{aligned}$$

As in the previous chapter, we set $Z(v,\xi) = Z_1(v,\xi) \times \cdots \times Z_N(v,\xi)$. We obtain the following optimization problem for the one-dimensional case:

One-Dimensional Binpacking Problem

The well-defined optimization problem for the one-dimensional case is given by

$$\begin{aligned}
\overline{P}^1_{bin}: \quad \min_{v,\xi,z} \; & c_v \sum_{i=1}^{\overline{n}} v_i + c_d \sum_{\ell=1}^{N} \left(L^\ell - \sum_{i=1}^{\overline{n}} x_i z_{i,\ell} \right) \\
\text{s.t.} \quad & (4.15), (4.18), (4.19), (4.20), && \# \; z \\
& (4.14), (4.16), (4.17), && \# \; \xi \\
& (4.13), && \# \; v \\
& z \in \mathbb{Z}^{N\cdot\overline{n}}, \; \xi \in \mathbb{R}^{\overline{n}}, \; v \in \mathbb{B}^{\overline{n}}.
\end{aligned}$$

The problem $\overline{P}^1_{bin}$ is in general mixed-integer with a quadratic objective function and quadratic constraints, since in both there are products two variables of the form $x_i z_{i,\ell}$. In general, such functions are not convex, but we can still model them with Gurobi, see Section 4.2. There are also many linear conditions, but we always mention the most "severe", here quadratic, case.

Next, we briefly introduce the optimization problem for the general p-dimensional binpacking problem.

Optimization Problem for the p-Dimensional Binpacking Problem

Since the structure is quite similar to the one-dimensional case, we shorten it as much as possible. The main difference from the one-dimensional case is, as the title already says, that we consider p components. The upper bound for the number of variants per component c is given by $\overline{k}_c$ and we set $\overline{k} = (\overline{k}_1, \ldots, \overline{k}_p)^\top$. Then we have the variables for the lengths of the objects x_i^c, summarized to $\xi \in \mathbb{R}^{e^\top \overline{k}}$, the number variables $z_{i,\ell}^c$, summarized to $z \in \mathbb{Z}^{N\cdot e^\top \overline{k}}$, and finally the binary variables v_i^c, summarized to $v \in \mathbb{B}^{e^\top \overline{k}}$, with $i = 1, \ldots, \overline{k}_c$, $c = 1, \ldots, p$ and $\ell = 1, \ldots, N$ for the corresponding indices. The relation between the variables v and x or z remains the same, thus, we have

$$x_i^c \leq v_i^c \overline{x}, \quad i = 1, \ldots, \overline{k}_c, \; c = 1, \ldots, p, \tag{4.21}$$

$$z_{i,\ell}^c \leq v_i^c \overline{z}^\ell, \quad i = 1, \ldots, \overline{k}_c, \; c = 1, \ldots, p, \; \ell = 1, \ldots, N, \tag{4.22}$$

with suitable upper bounds $\overline{x}$ and $\overline{z}^\ell$. As in the first optimization model, we require that at least one object is available in the modular system, independently of the component. Thus, we have

$$\sum_{c=1}^{p} \sum_{i=1}^{\overline{k}_c} v_i^c \geq 1. \tag{4.23}$$

In addition, the constraints that belong to the set X remain the same in structure and are given by

$$x_i^c \geq \underline{x}^c, \quad i = 1, \dots, \overline{k}_c,\ c = 1, \dots, p, \tag{4.24}$$

$$v_i^c x_i^c + \tau v_i^c \leq v_{i+1}^c x_{i+1}^c, \quad i = 1, \dots, \overline{k}_c - 1,\ c = 1, \dots, p. \tag{4.25}$$

We obtain the set

$$\begin{aligned} X(v) = \{\xi \in \mathbb{R}^{e^\top \overline{k}} |\ & x_i^c \geq v_i^c \underline{x}^c,\ i = 1, \dots, \overline{k}_c,\ c = 1, \dots, p, \\ & x_i^c \leq v_i^c \overline{x},\ i = 1, \dots, \overline{k}_c,\ c = 1, \dots, p, \\ & v_i^c x_i^c + \tau v_i^c \leq v_{i+1}^c x_{i+1}^c,\ i = 1, \dots, \overline{k}_c - 1,\ c = 1, \dots, p\}. \end{aligned}$$

The remaining constraints (3.21)-(3.23) can be formulated in the binary variable approach as follows:

$$z_{i,\ell}^c \geq 0, \qquad i = 1, \dots, \overline{k}_c,\ c = 1, \dots, p,\ \ell = 1, \dots, N, \tag{4.26}$$

$$\sum_{c=1}^{p} \sum_{i=1}^{\overline{k}_c} x_i^c z_{i,\ell}^c \leq L^\ell, \quad \ell = 1, \dots, N, \tag{4.27}$$

$$\sum_{c=1}^{p} \sum_{i=1}^{\overline{k}_c} z_{i,\ell}^c \leq \overline{o}_\ell, \quad \ell = 1, \dots, N. \tag{4.28}$$

The corresponding set notation for bin ℓ is given by

$$\begin{aligned} Z_\ell(v, \xi) = \{ \left(z_{i,\ell}^c\right)_{\substack{i=1,\dots,\overline{k}_c, \\ c=1,\dots,C}} \in \mathbb{Z}^{e^\top \overline{k}} \Big|\ & z_{i,\ell}^c \geq 0,\ i = 1, \dots, \overline{k}_c,\ c = 1, \dots, p, \\ & z_{i,\ell}^c \leq v_i^c \overline{z}^\ell,\ i = 1, \dots, \overline{k}_c,\ c = 1, \dots, p, \\ & \sum_{c=1}^{p} \sum_{i=1}^{\overline{k}_c} x_i^c z_{i,\ell}^c \leq L^\ell, \\ & \sum_{c=1}^{p} \sum_{i=1}^{\overline{k}_c} z_{i,\ell}^c \leq \overline{o}_\ell\}. \end{aligned}$$

The objective function can be formulated analogously, only in higher dimensions as in $\overline{P}_{bin}^1$, giving us the following:

p-Dimensional Binpacking Problem

The well-defined optimization problem for the p-dimensional case is given by

$$\begin{aligned} \overline{P}_{bin}^p : \quad \min_{v,\xi,z} \ & \sum_{c=1}^{p} c_v^c \sum_{i=1}^{\overline{k}_c} v_i^c + c_d \sum_{\ell=1}^{N} \left(L^\ell - \sum_{c=1}^{p} \sum_{i=1}^{\overline{k}_c} x_i^c z_{i,\ell}^c \right) \\ \text{s.t.} \quad & (4.22), (4.26), (4.27), (4.28), && \#\ z \\ & (4.21), (4.24), (4.25), && \#\ \xi \\ & (4.23), && \#\ v \\ & z \in \mathbb{Z}^{N \cdot e^\top \overline{k}},\ \xi \in \mathbb{R}^{e^\top \overline{k}},\ v \in \mathbb{B}^{e^\top \overline{k}}. \end{aligned}$$

BP_DIM1_1	
N	10
c_v	[10]
c_d	0.20
$\overline{k}$	[10]
τ	[15]
$\overline{o}$	2
runtime	0.39
total costs	49.40
variant costs	40
deviation costs	9.40

BP_DIM1_1	x_i^1
1	16.50
2	53.50
3	146.50
4	250.00

TABLE 4.18: BP_DIM1_1: input data and optimization results

ℓ	1	2	3	4	5	6	7	8	9	10
L^ℓ	23	70	90	107	117	163	200	300	400	500

TABLE 4.19: BP_DIM1_1; lengths of bins

The optimization problem $\overline{P}_{bin}^p$, like $\overline{P}_{bin}^1$, is a mixed-integer optimization problem with quadratic objective function and quadratic constraints. It contains in total $(N+1)e^\top\overline{k}$ integer variables (including binary ones) and $e^\top\overline{k}$ continuous variables.

After introducing the optimization problems $\overline{P}_{bin}^1$ and $\overline{P}_{bin}^p$, we consider different numerical tests.

4.3.4 Numerical Tests for the Binpacking Modular System

Next, we consider various examples for the one- and multi-dimensional examples. The examples are denoted with “BP” for Binpacking, “DIMp” for the dimension p of the corresponding problem (number of components) and a counter in the form of a number at the end such that we can distinguish examples in the same dimension. We start with one-dimensional examples.

Basic One-Dimensional Example

First, we consider the one-dimensional example BP_DIM1_1 and we explain exemplarily the Tables 4.18 and 4.19 and Figures 4.1 and 4.2 in the next paragraph in detail. BP_DIM1_1 is an example with one component, in the following, red objects. The left table of Table 4.18 shows the input data in the upper part and some information after the solving process in the lower part. As important input data, which are denoted here, we first consider $N = 10$ different bins. The bins have different lengths, which can be taken from Table 4.19 or can be seen in Figure 4.1. $\overline{k}$ is again the vector of the maximum number of variants per component, i.e., in the one-dimensional exact $\overline{n}$ and here ten. Thus, theoretically, there are ten different variants possible. c_v and c_d are the cost factors that are part of the objective function, c_v for the different variants and c_d for the deviation costs. In higher dimensions, c_v is a vector that contains the cost factors separated by the different

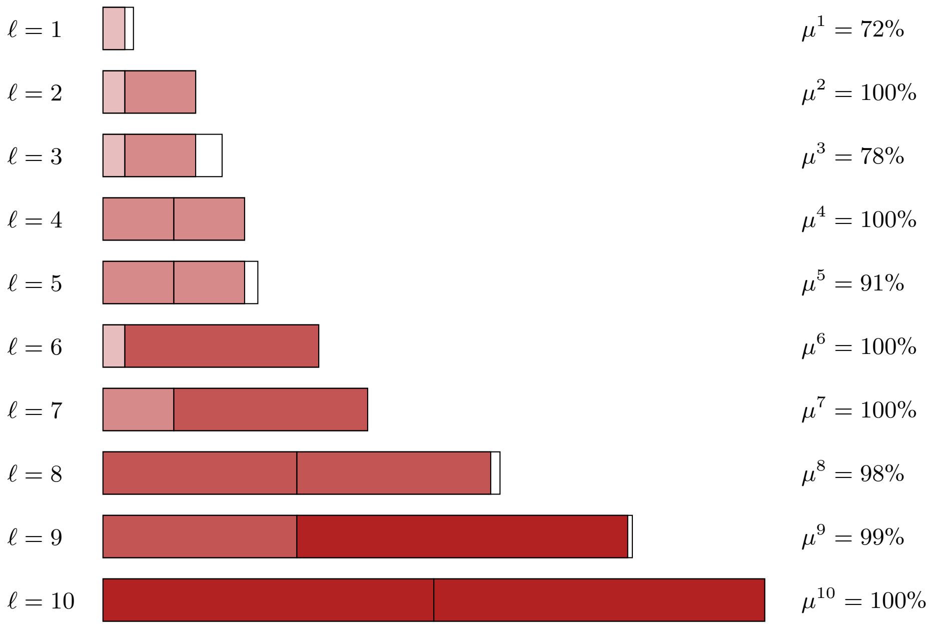

FIGURE 4.1: BP_DIM1_1: mapping

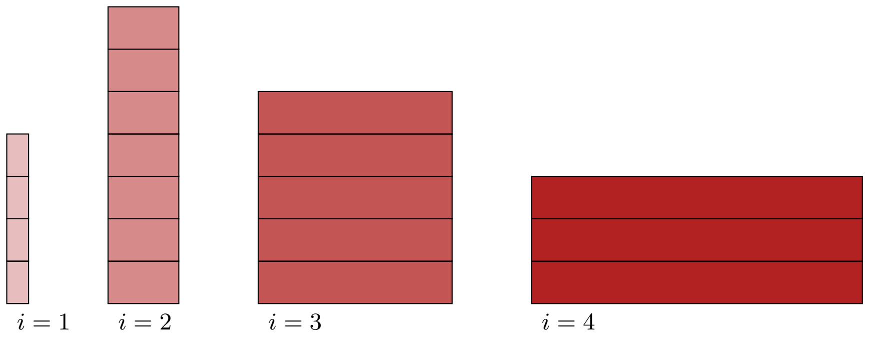

FIGURE 4.2: BP_DIM1_1: modular system

components. With $\tau = 15$ in this case, we ensure that the variants differ by at least 15 length units. In the following examples, we assume the same upper limit for the number of objects per bin in every bin, so we set $\overline{o}_\ell = \overline{o}$. In this example, only two objects per bin are allowed. Since this is a one-dimensional example, all vectors that depend on the dimension of the modular system, denoted by "[...]" in the table, have length one. In the one-dimensional examples, we could, of course, omit the brackets for better understanding. However, to keep the notation as uniform as possible, we will leave them.

The right part of the table shows the number of variants used, as well as the exact lengths of the variants (after the solving process). In this optimal configuration of the modular system, four (of the ten possible) variants with the lengths 16.50, 53.50, 146.50 and 250 are optimal. For better readability, we do not list the variants that are not used, i.e. those that are zero according to the model formulation of $\overline{P}^1_{bin}$. We start here counting at one.

The upper part of the left table of Table 4.18 shows, in addition to the runtime of the solver, which is here below one second, the total costs and the costs split in

name	parameter	description	total number variables (thereof integer)
BP_DIM1_2	$\overline{k}$	more variants are allowed	180 (165)
BP_DIM1_3	$\overline{o}$	more objects in a bin	120 (110)
BP_DIM1_4	τ	greater minimum distance	120 (110)
BP_DIM1_5	c_v	lower variant costs	120 (110)

TABLE 4.20: Overview for a case study in the one-dimensional binpacking example

the two parts, variant costs and deviation costs. The variant costs of 40 naturally fit to the results from the right-hand part of the table, since four variants are optimal and each variant has costs of $c_v = 10$. Figure 4.2 shows the modular system, sorted by the different variants. It shows on the one hand again how many different variants are available and how they look like (length) and on the other hand, the corresponding number. The mapping, which means which object is in which bin, is shown in Figure 4.1.

The figure again also shows the values μ^ℓ on the right side, which are the percentages of how filled the bins are. If there is no remaining space in bin ℓ, we have $\mu^\ell = 100\%$. In this example, the remaining space is also visible in the white parts at the end of a bin because it is big enough. In bin three for example, there is 22% space remaining since we have $\mu^3 = 78\%$.

Next, we make some modifications of the basic example.

Modifications of the Basic Example in the One-Dimensional Case

In the following paragraphs, we test different input data depending on the example BP_DIM1_1, which we denote in the following as "basic example". This means that we modify step by step in each of the following examples some of the input data of the basic example. Firstly, we consider only cases with one component. Table 4.20 gives an overview of the input data that are modified based on the input data of BP_DIM1_1. Since we do not change the lengths of the ten bins, we omit the corresponding tables here and move them to the appendix. The same applies to the illustration of the modular system, as we could theoretically count the corresponding objects in the figures with the mappings.

Firstly, in example BP_DIM1_2, we increase the number of possible variants and consider $\overline{k} = 15$ instead of $\overline{k} = 10$. This modification does not change anything major, since not all possible variants are exhausted in the basic example, but this serves to improve our understanding of the model. Of course, the number of variables in example BP_DIM1_2 is higher than in BP_DIM1_1, which, however, due to the good solver and the still small example, this does not have a major impact on the runtime. The input data and results, as well as the mapping, are shown in Table 4.21 and Figure A.1.

Secondly, in example BP_DIM1_3, we allow more objects in the bins compared to the basic example. Thus, we increase $\overline{o}$ to four (instead of two). For this example, it is clear that the total costs cannot be higher than the costs in the basic example, since the optimal configuration of the basic example is still feasible here. In the setting of BP_DIM1_3 it is possible to decrease the total costs from 49.40 to 32.30, since

BP_DIM1_2	
N	10
c_v	[10]
c_d	0.20
$\overline{k}$	[15]
τ	[15]
$\overline{o}$	2
runtime	0.45
total costs	49.40
variant costs	40
deviation costs	9.40

BP_DIM1_2	x_i^1
1	16.50
2	53.50
3	146.50
4	250.00

TABLE 4.21: BP_DIM1_2: input data and optimization results

BP_DIM1_3	
N	10
c_v	[10]
c_d	0.20
$\overline{k}$	[10]
τ	[15]
$\overline{o}$	4
runtime	0.39
total costs	32.30
variant costs	30
deviation costs	2.30

BP_DIM1_3	x_i^1
1	22.50
2	58.50
3	159.17

TABLE 4.22: BP_DIM1_3: input data and optimization results

more smaller objects are used and with that one variant can be saved. The input data and results are shown in Table 4.22, and the mapping is shown in Figure 4.3.

Next, in example BP_DIM1_4, we modify the parameter τ, which requires the minimum distance in the length of the different variants. We increase τ to $\tau = 80$, which ensures that the optimal configuration of the basic example is no longer feasible since, for example, $x_2^1 - x_1^1 = 37 \ngeq 80$. The enlargement of τ leads to the fact that only three variants are optimal and to higher total costs, especially higher deviation costs. The input data and results are shown in Table 4.23 and the mapping in Figure 4.4. In this example, it is also the first time that a bin (here bin 1) remains empty because the shortest object is longer than the length of bin 1 ($x_1^1 = 45 > 23 = L^1$). We remark at this point that this is allowed in the model formulation.

Lastly, in example BP_DIM1_5 we consider and modify the cost coefficients c_v and c_d, which become more important in Subsection 5.3.3. In detail, we decrease the costs for the variants in the modular system, which means that the second part of the objective function, the deviation costs, become more significant. In the optimal configuration, we obtain in this case more variants, five instead of four, since variants are cheaper, and lower deviation costs in total, here 3.00 instead of 9.40.

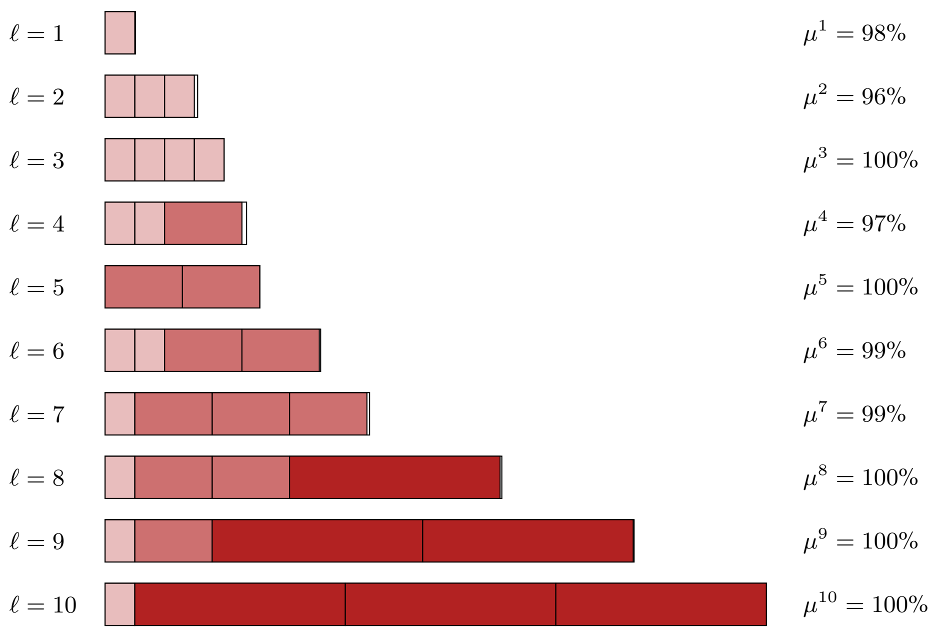

FIGURE 4.3: BP_DIM1_3: mapping

BP_DIM1_4	
N	10
c_v	[10]
c_d	0.20
$\overline{k}$	[10]
τ	[80]
$\overline{o}$	2
runtime	0.26
total costs	52.00
variant costs	30
deviation costs	22.00

BP_DIM1_4	x_i^1
1	45.00
2	150.00
3	250.00

TABLE 4.23: BP_DIM1_4: input data and optimization results

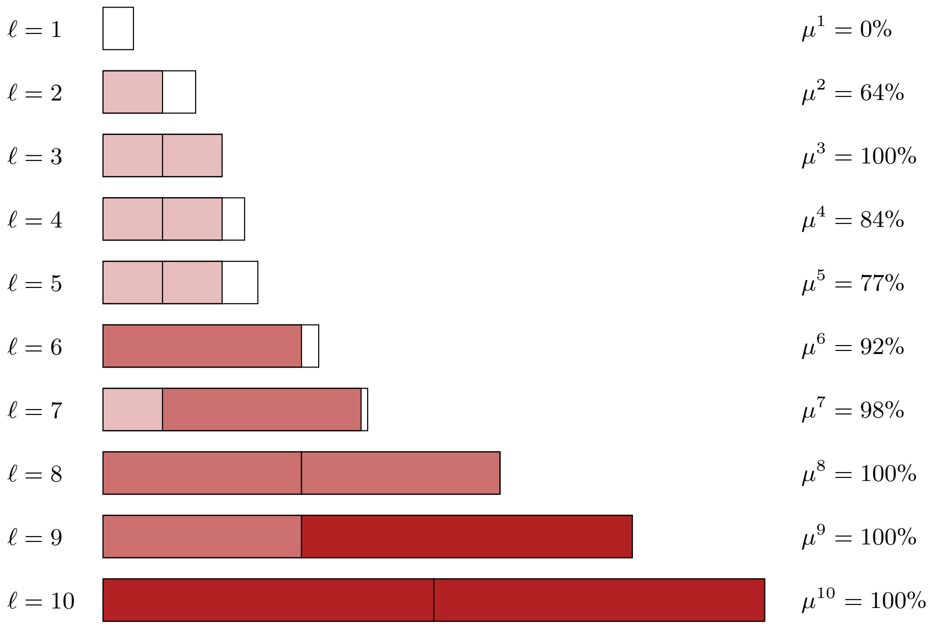

FIGURE 4.4: BP_DIM1_4: mapping

The input data and results are shown in Table 4.24 and Figure A.5.

BP_DIM1_5	
N	10
c_v	[5]
c_d	0.20
$\overline{k}$	[10]
τ	[15]
$\overline{o}$	2
runtime	0.79
total costs	28.00
variant costs	25
deviation costs	3.00

BP_DIM1_5	x_i^1
1	22.00
2	68.00
3	95.00
4	200.00
5	300.00

TABLE 4.24: BP_DIM1_5: input data and optimization results

BP_DIM2_1	
N	10
c_v	[10, 5]
c_d	5
$\overline{k}$	[10, 10]
τ	[15, 50]
$\overline{o}$	2
runtime	14.60
total costs	45.00
variant costs	45
deviation costs	0.00

BP_DIM2_1	x_i^1	x_i^2
1	31.50	11.50
2	58.50	75.50
3	-	131.50
4	-	200.00
5	-	300.00

TABLE 4.25: BP_DIM2_1: input data and optimization results

After considering many examples of the one-dimensional binpacking problem, we now proceed with the numerical examples for the higher-dimensional examples. In particular, this means that we are considering more components in the modular system, which corresponds to multiple colors here.

Higher dimensions

In this part, we again consider as base the basic example BP_DIM1_1 and increase step by step the dimension of the modular system. We recall that the dimension of the modular system was exactly the number of components which are the colors of the objects in this example. We consider step-by-step up to four components with corresponding color sequence red, green, blue and yellow. We introduce a new component c always with lower variant costs c_v^c but a higher τ^c compared to the previously introduced component. This motivates on the one hand the usage of the new component through the lower variant costs. On the other hand, the larger τ^c hopefully prevents that only the new component is used.

In the two-dimensional case, example BP_DIM2_1, both components are used (which was the idea behind the choice of the corresponding parameters) and each bin is fully filled. Details can be found in Table 4.25 and Figure 4.5. We can also see that there are more variants of the second component, which makes sense, since they are cheaper. Compared to the basic example, the total costs are lower (45.00 vs. 49.50), which is also a logical point, since the optimal configuration of the one-dimensional example remains feasible, since we do not force the usage of all components.

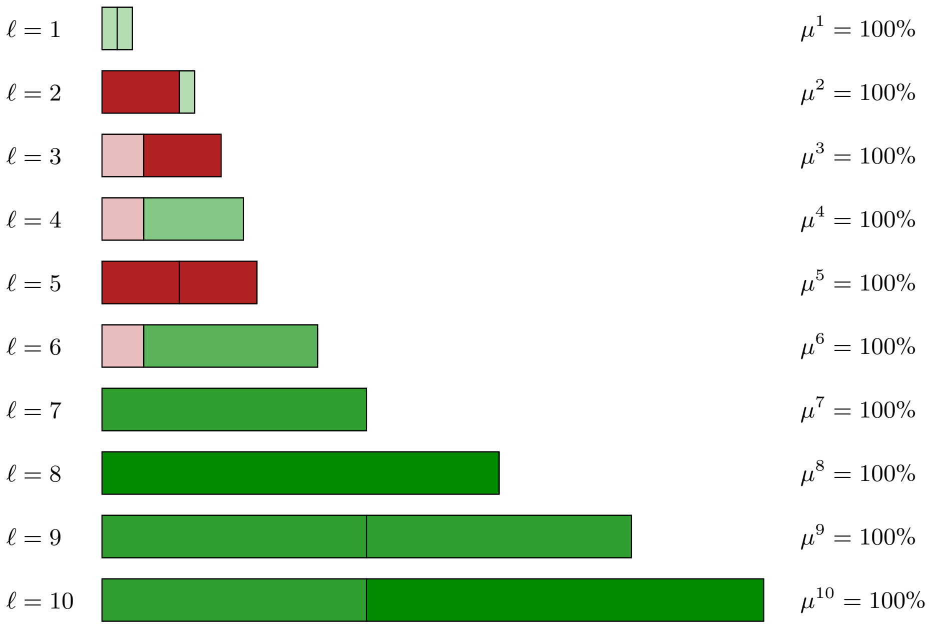

FIGURE 4.5: BP_DIM2_1: mapping

In the three-dimensional case, example BP_DIM3_1, the results are quite similar, which means that all components are used (red, green and blue) and the last component (here the blue objects) have the most variants. The total costs continue to decrease to 24.00. Details can be found in Table 4.26 and Figure 4.6.

BP_DIM3_1	
N	10
c_v	[10, 5, 1]
c_d	5
$\overline{k}$	[10, 10, 10]
τ	[15, 50, 75]
$\overline{o}$	2
runtime	110.66
total costs	24.00
variant costs	24
deviation costs	0.00

BP_DIM3_1	x_i^1	x_i^2	x_i^3
1	58.50	45.00	11.50
2	-	107.00	118.00
3	-	-	200.00
4	-	-	300.00

TABLE 4.26: BP_DIM3_1: input data and optimization results

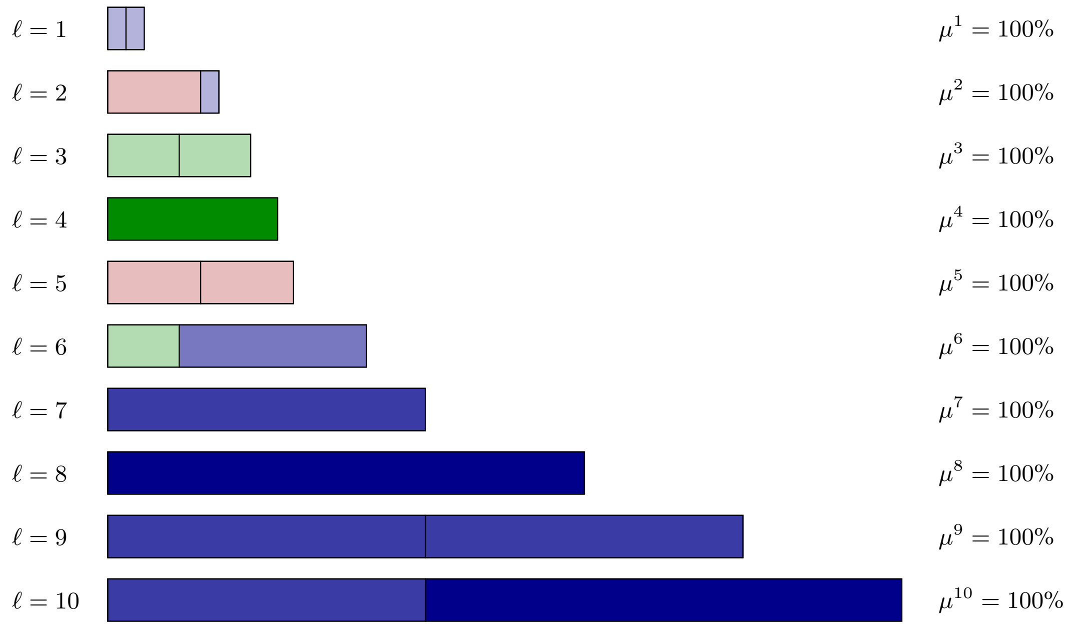

FIGURE 4.6: BP_DIM3_1: mapping

In the last experiment, the modular system of dimension four, BP_DIM4_1, it is interesting that the most expensive component (here component one) is no longer used. We have no red objects, but green, blue and yellow ones. The other results behave structurally as before and as expected. Details are given in Table 4.27 and Figure 4.7.

With this procedure, it is very simple to bring the modular system in higher dimensions without changing the main structure of the problem. The problems are still solved in little time, also in dimension four, with around two minutes. The results can also be illustrated in a good way in higher dimensions, and it is easy to understand what is happening. In applications we often have more than two components; also the crane bridge model is a simplified model. This example still has a long solution time and has “only” dimension two. With that in mind, we proceed with the conclusions of this chapter.

BP_DIM4_1	
N	10
c_v	[10, 5, 1, 0.5]
c_d	5
$\overline{k}$	[10, 10, 10, 10]
τ	[15, 50, 75, 80]
$\overline{o}$	2
runtime	29.20
total costs	9.00
variant costs	9.00
deviation costs	0.00

BP_DIM4_1	x_i^1	x_i^2	x_i^3	x_i^4
1	-	58.50	2.50	11.50
2	-	-	78.50	104.50
3	-	-	-	200.00
4	-	-	-	300.00

TABLE 4.27: BP_DIM4_1: input data and optimization results

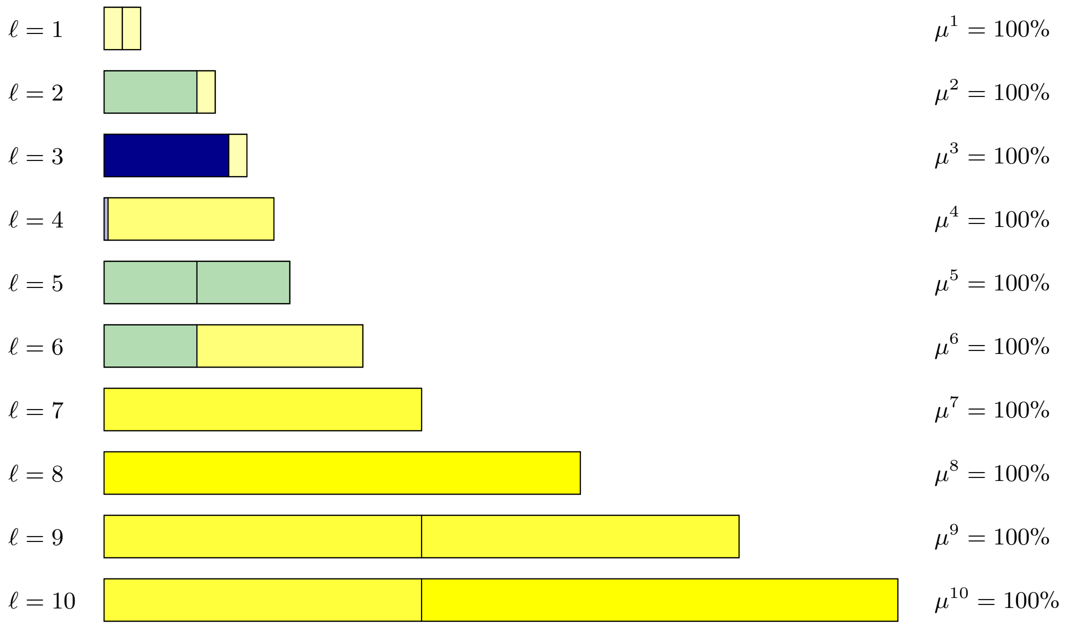

FIGURE 4.7: BP_DIM4_1: mapping

4.4 Conclusions

In this chapter, we derived a well-defined optimization problem for the modular system problems. Through the introduction of suitable binary variables, we could find in Section 4.1 an approach to formulate the optimization model P^{gen} from Chapter 3 into a completely well-defined optimization problem. The resulting problem is a mixed-integer optimization problem. Such problems were not found in the literature, although the approach with binary variables is frequently used in mixed-integer optimization. Thus, this work formulated for the first time a complete optimization problem that can be theoretically transferred to a solver to determine an optimal configuration for modular system problems.

This solution approach worked and behaved as desired, as we can see from the investigations in Section 4.3. There we deduced the corresponding mixed-integer optimization problems for the crane bridge and binpacking problems. For the binpacking application we showed how we can consider modular systems with an arbitrary number of components, that means a modular system with an arbitrary dimension. The resulting problems are, in all cases, mixed-integer optimization problems with quadratic constraints and quadratic objective functions. Such problems can be solved by the solver Gurobi, which we briefly introduced in Section 4.2.

In the numerical tests, we noticed that the problems can generally be solved. It was positive to note that all problems in the binpacking application could be solved very quickly, even though we increased the dimension of the modular system to four. This behavior explains the general introduction of this example. Especially in the larger crane bridge examples, numerical difficulties have occurred. This is probably due to the complicated functions involved, such as the load-capacity function, and also to the many binary variables used to model the corresponding combinations and logical conditions. In even larger applications, the solution times will become longer, which motivates different solution methods.

In the following two chapters, we consider such solution methods. They are mainly based on a decomposition approach, which is considered in the next chapter.

Chapter 5

A Basic Decomposition Approach

In Chapter 4, we solved the optimization problems $\overline{P^{gen}}$ to obtain an optimal configuration for the modular system. It is particularly remarkable in applications for the crane bridge modular system that the runtime of the solver becomes considerably long. In this example, some simplifications have already been made through assumptions, and it is still only a modular system consisting of two components. This example suggests that, for more complicated modular system problems with a larger number of components, the problems $\overline{P^{gen}}$ cannot be solved in a reasonable amount of time.

The optimization problem $\overline{P^{gen}}$ generally contains three types of variable vectors, the binary variable (vector) v, which specifies whether a variant of a component exists in the modular system, the continuous variable ξ, which characterizes the geometry parameter of the different variants, and the integer variable z, which is the corresponding number of each variant in the modular system. The binary variables from v lead to a large number of combinatorial and coupling constraints in the optimization problem. Although the constraints can be easily modeled as linear, quadratic constraints or with Big-M reformulations, they are generally not easy for solvers to handle. A classic approach, if complicating variables can be identified in an optimization problem, is a decomposition approach.

Decomposition approaches are used when it can be assumed that a problem can be solved more easily by fixing the complicating variable and solving the resulting problem. Ideally, the large problem is divided into many smaller problems that can be solved in parallel. Well-known decomposition approaches include the generalized Benders decomposition, which goes back to Jacques Benders and Arthur Geoffrion in 1972 ([5], [20]), and the Lagrangian decomposition ([23]). In these decomposition approaches the optimization problems need to fulfill special structures in the objective function and constraints. In our case, the problems are complicated, but we do not need such a complicated decomposition approach, since we consider a fairly simple variable, the binary variable v, as the complicating variable. The decomposition approach and the resulting algorithm are presented in Section 5.1. In the following section, Section 5.2, we apply the approach of Section 5.1 to the modular systems for the crane bridge and the binpacking examples.

By carefully examining the examples to which we apply the decomposition approach and the resulting algorithm, we notice that the problem can also be viewed in a different way. At the end of Section 5.2, we conclude that determining an optimal configuration for the modular system is equivalent to determining a minimal point of a discrete function. In Section 5.3, we first introduce discrete functions and discuss convexity concepts for them later.

5.1 First Decomposition and Enumeration Approach

In this section, we consider a first decomposition approach to avoid solving the optimization problem $\overline{P^{gen}}$. To avoid duplication, we will not reintroduce the variables of the general modular system problem here. Only necessary new constraints are discussed here. In principle, we try to keep the problem structure as similar as possible to the previous one. For exact variables and indices, we refer to Section 3.1.

As described above, we fix the binary variables v, where v_i^c is one if variant i of component c is part of the modular system. In general, there is no sorting or other idea in the order of v, so instead of looking at all possible $0/1$ combinations of v_i^c for the components, we can fix the total allowed number of variants per component to $\widehat{k}_c$ for $c = 1, \dots, C$. The number of variants divided into the components is now given explicitly through $\widehat{k} = (\widehat{k}_1, \dots, \widehat{k}_C) \in \mathbb{N}^C$. Clearly, we have $\widehat{k}_c \leq \overline{k}_c$ with the maximum allowed number of variants $\overline{k}_c$ for component c. In general, we require $\widehat{k}_c \geq \underline{k}_c$, where $\underline{k}_c$ is the lower bound for the number of variants for component c. Depending on the application, $\underline{k}_c$ is mostly zero or one. In the crane bridge example, both components, for example, are necessary, in the binpacking problem not necessarily.

In the following, we deduce an optimization problem $\overline{P^{gen}}(\widehat{k})$, which is similar to $\overline{P^{gen}}$ but with a fixed known number of variants for each component. Instead of solving $\overline{P^{gen}}$, we can solve $\overline{P^{gen}}(\widehat{k})$ for every $\widehat{k}$ and compare the optimal values. The number of problems that need to be solved is finite, since we have $\widehat{k}_c \in \{\underline{k}_c, \dots, \overline{k}_c\}$. For that, we continue with the new optimization problem.

To enforce that there are exactly $\widehat{k}_c$ variants of component c, we add the constraints

$$\sum_{i=1}^{\overline{k}_c} v_i^c = \widehat{k}_c, \quad c = 1, \dots, C, \tag{5.1}$$

if we want to enforce exactly $\widehat{k}_c$ variants for component c. Thus, $\widehat{k}_c$ enters as a parameter in (5.1). Additionally, in general, we need to require two more things besides concrete fixing of the binary variables. Firstly, we must ensure that the variants we request are also used in the modular system for at least one product. This is not necessarily given if we require only Equation (5.1) because $z_{i,\ell}^c = 0$ for all products ℓ for a component c and a corresponding variant i with $i \leq \widehat{k}_c$ may occur. Thus, we require

$$\sum_{\ell=1}^{N} z_{i,\ell}^c \geq v_i^c, \quad i = 1, \dots, \overline{k}_c,\ c = 1, \dots, C. \tag{5.2}$$

This ensures that this variant is used at least in one product, if $v_i^c = 1$.

Secondly, we need to require that two different variants of a component are not similar or not too similar. In the model $\overline{P^{gen}}$, such cases obviously do not occur, since every additional variant generates costs, and therefore identical variants are not part of the solution, since omitting an identical variant reduces total costs in all cases. In the binpacking problem, we require different variants from the beginning. In the general optimization problem $\overline{P^{gen}}$ or in the optimization problem for the crane bridge example $\overline{P}^{crane}$ this was not explicitly considered.

Since the constraints in (5.1) are the most important aspect here, we explicitly mention it in the optimization problem, while we write the other conditions (for variant differences and (5.2)) in the sets $\widehat{X}(v)$ and $\widehat{Z}(v,\xi)$, which are the sets $X(v)$ and $Z(v,\xi)$ from $\overline{P^{gen}}$ extended by the new constraints.

Together, we obtain the optimization problem $\overline{P^{gen}}(\widehat{k})$ with a given fixed number of variants $\widehat{k}$:

Optimization Problem with a Fixed Number of Variants

The general optimization problem for the optimal configuration of a modular system for a fixed variant number $\widehat{k}$ is given by

$$\begin{aligned}\overline{P^{gen}}(\widehat{k}):\quad \min_{v,\xi,z}\ & c^{size}(v,\xi,z)+c^{prop}(v,\xi,z)\\ \text{s.t.}\quad & z\in\widehat{Z}(v,\xi),\\ & \xi\in\widehat{X}(v),\\ & (5.1),\\ & z\in\mathbb{Z}^{N\cdot e^\top\overline{k}},\xi\in\mathbb{R}^{e^\top\overline{k}},v\in\mathbb{B}^{e^\top\overline{k}}\end{aligned}$$

for $\underline{k}\le\widehat{k}\le\overline{k}$ with $\underline{k}=(\underline{k}_1,\dots,\underline{k}_C)^\top$ and $\overline{k}=(\overline{k}_1,\dots,\overline{k}_C)^\top$.

It should be mentioned that instead of fixing the binary variables v_i^c to value one, which is done implicitly in Equation (5.1), we can set up a completely new model by only initializing the variables x_i^c for $i=1,\dots,\widehat{k}_c$, for example. In $\overline{P^{gen}}(\widehat{k})$ we have x_i^c for $i=1,\dots,\overline{k}_c$. With this new formulation, the number of variables is much smaller if $\widehat{k}_c$ is significantly smaller than $\overline{k}_c$. However, since in presolving processes fixed binary variables are removed, we refrain from setting up a new model in order to keep the formulation of $\overline{P^{gen}}(\widehat{k})$ as similar as possible to the one of $\overline{P^{gen}}$. Secondly, we remark that the inequalities $\underline{k}\le\widehat{k}$ and $\widehat{k}\le\overline{k}$ are meant component-wise.

In general, we cannot say much more about the solvability of any problem $\overline{P^{gen}}(\widehat{k})$ for a given $\widehat{k}$ than in the previous chapter. Thus, $\overline{P^{gen}}(\widehat{k})$ is either solvable or inconsistent. One other thing we can say is that at least one problem $\overline{P^{gen}}(\widehat{k})$ is solvable if $\overline{P^{gen}}$ is solvable. If we require too many variants of a component, for example, and variants must differ significantly, problems can be unsolvable through inconsistency. However, if the variant number is too low or lower than in the optimal case, it is also not guaranteed that there is a feasible configuration which can fulfill all product properties at the same time.

The decomposition approach, instead of solving $\overline{P^{gen}}$, is now to solve problem $\overline{P^{gen}}(\widehat{k})$ for all possible $\widehat{k}$. We note that $\widehat{k}$ is a vector, and with "for all $\widehat{k}$" we mean all possible combinations for $\widehat{k}=(\widehat{k}_1,\dots,\widehat{k}_C)$ with $\widehat{k}_c\in[\underline{k}_c,\overline{k}_c]$. The resulting solution approach is similar to a classical enumeration approach in the hope that the problems to be solved are easy. We collect all possible combinations in a list $\mathcal{L}$. The calculation of $\mathcal{L}$ is done by a function `combinations`, which we introduce here for the following algorithm. Thus, we have $\mathcal{L}=$ `combinations(`$\underline{k},\overline{k}$`)`, where $\underline{k}$ is the vector of all lower bounds for $\widehat{k}$. $\mathcal{L}$ is a list of tuples, where each tuple has length C. The cardinality of $\mathcal{L}$ is given by

$$|\mathcal{L}| = \prod_{c=1}^{C} (\overline{k}_c - \underline{k}_c + 1),$$

which can be very high in general. After determining all possible combinations, each problem $\overline{P^{gen}}(\widehat{k})$ is passed to a suitable solver.

As we do not know whether a problem is solvable or inconsistent in general, we have to consider different cases. In the case that $\overline{P^{gen}}(\widehat{k})$ is solvable for a variant combination $\widehat{k}$, we store the corresponding results: the costs $\widehat{obj}$, the geometry variables $\widehat{\xi}$ and the corresponding number variables $\widehat{z}$. More precisely, we add $\left(\widehat{k} : \left(\widehat{obj}, \widehat{\xi}, \widehat{z}\right)\right)$ to a set $\mathcal{S}$. The notation is based on a Python dictionary object with a key $\widehat{k}$ and the corresponding data, here the tuple $\left(\widehat{obj}, \widehat{\xi}, \widehat{z}\right)$. Although the cardinality of $\mathcal{S}$ can be very high in general, we store all data for further considerations in Subsection 5.3.3. If no further considerations are made, of course, only the element with the currently lowest optimal value can be stored.

In the end, we select the tuple from $\mathcal{S}$ with the lowest costs $\widehat{obj}$. The hope is that many of the problems with a fixed number of variants can be solved quickly, but it is clear that we have to solve a lot of problems. Altogether we get Algorithm 1.

Algorithm 1: First Decomposition Approach

```
   Input: Bounds $\underline{k}, \overline{k}$ for $\widehat{k}$, necessary input data for problems $\overline{P^{gen}}(\widehat{k})$
   Output: $(k^\star : (obj, \xi^\star, z^\star))$ or flag=inconsistent
 1 Determine the list $\mathcal{L}$ = combinations($\underline{k}, \overline{k}$).
 2 Set $\mathcal{S} = \emptyset$.
 3 for $\widehat{k} \in \mathcal{L}$ do
 4     Try to solve $\overline{P^{gen}}(\widehat{k})$.
 5     if $\overline{P^{gen}}(\widehat{k})$ solvable with optimal point $(\widehat{v}, \widehat{\xi}, \widehat{z})$ and optimal value $\widehat{obj}$ then
 6         $\mathcal{S} = \mathcal{S} \cup \left\{\left(\widehat{k} : \left(\widehat{obj}, \widehat{\xi}, \widehat{z}\right)\right)\right\}$
 7     end
 8 end
 9 if $\mathcal{S} \neq \emptyset$ then
10     Choose $\left(\widehat{k} : \left(\widehat{obj}, \widehat{\xi}, \widehat{z}\right)\right) \in \mathcal{S}$ with smallest $\widehat{obj}$ and set $k^\star = \widehat{k}$, $\xi^\star = \widehat{\xi}$,
         $z^\star = \widehat{z}$ and $obj = \widehat{obj}$.
11 else
12     Set flag = inconsistent.
13 end
```

Algorithm 1 returns either the best variant combination $k^\star$ with the corresponding variables $\xi^\star$ and $z^\star$ and the objective function value obj or the information that all problems are *inconsistent*.

After introducing the decomposition idea for the optimization approach for modular systems, we consider the running examples.

5.2 Decomposition Approach for the Modular System Examples

In this section, we again consider the crane bridge and the binpacking examples. In Subsection 5.2.1 we consider the crane bridge example. After introducing the new optimization problem with fixed variants, we consider some selected data sets. In Subsection 5.2.2 we do the same with the Binpacking example.

5.2.1 Decomposition Approach Crane Bridge Modular System

We start with the deduction of the concrete optimization problem.

Optimization Problem

In Subsection 4.3.1 the well-defined optimization problem for the optimal configuration for a modular system is introduced. The complete implementable optimization problem is given in the appendix in Section A.1. We distinguish between optimization problems where the weight of the bridges is part of the objective function or not.

As we have many constraints in these optimization problems, we only discuss the additional constraints here. We have a maximum number of profiles of $\overline{n}$ and sheets $\overline{m}$. We fix the number of variants to $\widehat{k} = (\widehat{k}_P, \widehat{k}_S)$, where $\widehat{k}_P \leq \overline{n}$ is the fixed number of profile variants, and $\widehat{k}_S \leq \overline{m}$ the number of the sheets. Similarly to (5.1), we thus require

$$\begin{aligned} v_i^P &= 1, \quad i = 1, \ldots, \widehat{k}_P, \\ v_i^P &= 0, \quad i = \widehat{k}_P + 1, \ldots, \overline{n}, \end{aligned} \tag{5.3}$$

$$\begin{aligned} v_j^S &= 1, \quad j = 1, \ldots, \widehat{k}_S, \\ v_j^P &= 0, \quad j = \widehat{k}_S + 1, \ldots, \overline{m}. \end{aligned} \tag{5.4}$$

According to the general constraints in (5.2), we require for the two components of the modular system

$$\begin{aligned} \sum_{\ell=1}^{N} z_{i,\ell}^P &\geq v_i^P, \quad i = 1, \ldots, \overline{n}, \\ \sum_{\ell=1}^{N} z_{j,\ell}^S &\geq v_j^S, \quad j = 1, \ldots, \overline{m}. \end{aligned} \tag{5.5}$$

We remarked in the previous chapter that the number variables z do not necessarily have to be implemented, in particular in the cases where the weight of the bridges is not part of the objective functions. In these cases, we can replace the variables $z_{i,\ell}^P$ and $z_{j,\ell}^P$ in (5.5) by the corresponding binary variables $b_{i,\ell}^P$ and $b_{j,\ell}^S$, which are accordingly one if variant i of profiles and variant j of sheets are used in the crane bridge ℓ.

The remaining constraints remain unchanged and are listed here again for the sake of completeness. The corresponding MILP-reformulations can be done similarly here and are therefore omitted here. We also consider here only the problems with the number variables z. Thus, we have

$$z^P_{i,\ell} \le v^P_i, \quad i = 1,\dots,\overline{n},\ \ell = 1,\dots,N, \tag{5.6}$$
$$z^S_{j,\ell} \le v^S_i, \quad j = 1,\dots,\overline{m},\ \ell = 1,\dots,N, \tag{5.7}$$
$$P^i = \left(h^P_i, t^P_i, w^P_i\right) \in X^1, \quad i = 1,\dots,\overline{n}, \tag{5.8}$$
$$S^j = \left(h^S_j, l^S_j, t^S_j, w^S_j\right) \in X^2, \quad j = 1,\dots,\overline{m}, \tag{5.9}$$

$$\begin{aligned} \left|\{i \in \{1,\dots,\overline{n}\} \mid z^P_{i,\ell} > 0\}\right| = 1, \quad \ell = 1,\dots,N, \\ \left|\{j \in \{1,\dots,\overline{m}\} \mid z^S_{j,\ell} > 0\}\right| = 1, \quad \ell = 1,\dots,N, \end{aligned} \tag{5.10}$$

$$\begin{aligned} z^P_{i,\ell} \ge 0, \quad i = 1,\dots,\overline{n},\ \ell = 1,\dots,N, \\ z^S_{j,\ell} \ge 0, \quad j = 1,\dots,\overline{m},\ \ell = 1,\dots,N, \end{aligned} \tag{5.11}$$

$$\begin{aligned} z^P_{i,\ell},\ z^S_{j,\ell} > 0 \quad \Rightarrow \quad z^P_{i,\ell} = 4\left\lfloor \frac{L^\ell}{2l^S_j} \right\rfloor - 2, \\ i = 1,\dots,\overline{n},\ j = 1,\dots,\overline{m},\ \ell = 1,\dots,N, \end{aligned} \tag{5.12}$$

$$\begin{aligned} z^S_{j,\ell} > 0 \quad \Rightarrow \quad z^S_{j,\ell} = 2\left\lfloor \frac{L^\ell}{2l^S_j} \right\rfloor - 2, \\ j = 1,\dots,\overline{m},\ \ell = 1,\dots,N, \end{aligned} \tag{5.13}$$

$$\begin{aligned} z^P_{i,\ell},\ z^S_{j,\ell} > 0 \quad \Rightarrow \quad M(P^i, S^j) \ge M^\ell, \\ i = 1,\dots,\overline{n},\ j = 1,\dots,\overline{m},\ \ell = 1,\dots,N, \end{aligned} \tag{5.14}$$

$$\begin{aligned} z^P_{i,\ell},\ z^S_{j,\ell} > 0 \quad \Rightarrow \quad (P^i, S^j) \in F \\ i = 1,\dots,\overline{n},\ j = 1,\dots,\overline{m},\ \ell = 1,\dots,N. \end{aligned} \tag{5.15}$$

The constraints that depend only on P^i, S^j and v^P_i and v^S_j belong to the set $\widehat{X}(v)$, all others to the set $\widehat{Z}(v,\xi)$, if we refer to the syntax from the general problem $\overline{P^{gen}}(\widehat{k})$. To avoid duplication, the sets are not explicitly written down at this point.

The idea behind Algorithm 1 is only formally correct if we additionally require that the enforced variants are sufficiently different. We must require that the profile variants $P^r = (h^P_r, t^P_r, w^P_r)$ and $P^s = (h^P_r, t^P_r, w^P_r)$ for $r, s = 1,\dots,\widehat{k}_P$ with $r \neq s$ are not too similar. For that, we require

$$\max\left\{|h^P_r - h^P_s|, |t^P_r - t^P_s|, |w^P_r - w^P_s|\right\} \ge \tau^P, \quad r, s = 1,\dots,\widehat{k}_P,\ r \neq s \tag{5.16}$$

with a parameter $\tau^P > 0$. That means that some of the geometry parameters can be very similar or identical, but at least in one of the parameters, a minimum difference of τ^P is necessary. The maximum function and the absolute values in Equation (5.16) can be reformulated with new binary variables and big-M reformulations into constraints which Gurobi can handle.

For the sheets we analogously require

$$\max\left\{|h_r^S - h_s^S|, |l_r^S - l_s^S|, |t_r^S - t_s^S|, |w_r^S - w_s^S|\right\} \geq \tau^S, \quad r, s = 1, \ldots, \widehat{k}_S,\ r \neq s. \tag{5.17}$$

In total, we obtain the optimization model for the optimal configuration for the crane bridges including minimizing the total weight of the bridges:

Modular System Crane Bridge with Fixed Number of Variants

The optimization problem for the optimal configuration of a modular system for crane bridges with exactly $\widehat{k}_P$ profile variants and $\widehat{k}_S$ sheet variants is given by

$$\begin{aligned}
\overline{P}^{crane,w}(\widehat{k}) : \quad \min_{v,\xi,z} \quad & c^P \widehat{k}_P + c^S \widehat{k}_S + c_d \sum_{\substack{i,j,\ell=1 \\ (z^P_{i,\ell}>0)\&(z^S_{j,\ell}>0)}}^{\overline{n},\overline{m},N} \left(M(P^i, S^j) - M^\ell\right) \\
& + c_w \sum_{\substack{i,j,\ell=1 \\ (z^P_{i,\ell}>0)\&(z^S_{j,\ell}>0)}}^{\overline{n},\overline{m},N} w\left(P^i, S^j, z^P_{i,\ell}, z^S_{j,\ell}\right) \\
\text{s.t.} \quad & (5.5), (5.6), (5.7), (5.10), (5.11), (5.12), (5.13), (5.14), (5.15), \qquad \# \ z \\
& (5.8), (5.9), (5.16), (5.17), \qquad \# \ \xi \\
& (5.3), (5.4), \qquad \# \ v \\
& z \in \mathbb{Z}^{N(\overline{n}+\overline{m})},\ \xi \in \mathbb{R}^{3\overline{n}+4\overline{m}},\ v \in \mathbb{B}^{\overline{n}+\overline{m}},
\end{aligned}$$

with $\xi = (P^1, \ldots, P^{\overline{n}}, S^1, \ldots, S^{\overline{m}})$, $\widehat{k} = (\widehat{k}_P, \widehat{k}_S)$ and $\widehat{k}_P \leq \overline{n}$ and $\widehat{k}_S \leq \overline{m}$.

We remark at this point that the problems $\overline{P}^{crane,w}(\widehat{k})$ and $\overline{P}^{crane,w}$ from the previous chapter do not fit exactly together, since there are explicit differences in the variants required in this chapter. We could, of course, also insert the additional conditions (5.16) and (5.17) into the optimization problem $\overline{P}^{crane,w}$ from the previous chapter, repeat the computations, and then compare them exactly. However, it is noticeable that the results of the computations do not differ in the optimal costs, i.e., the differences between the variants are already taken into account or another optimal point exists that fulfills the conditions.

After introducing the optimization problem with fixed variants, we consider Algorithm 1 with some of the problem instances from Subsection 4.3.2.

Results Algorithm 1

For the crane bridge example, we consider three problem instances. The first example is example CRANE_N5_4 from Subsection 4.3.2. In the second example, we look at example CRANE_N20_1, which has a higher demand of crane bridges (20). The third example is a new small example. In all examples, we choose a minimum difference for profile variants of $\tau^P = 15\,(\text{mm})$ and for sheet variants of $\tau^S = 15\,(\text{mm})$.

We start with example CRANE_N5_4 with the input data from Table 4.8 (upper part of the left table) with the demand of five crane bridges and $\overline{n} = \overline{m} = 5$ as the maximum variant number. From the application, it is clear that we have to consider a minimum number of one for the variants, as both components have to be part of the crane bridge. Thus, we have $\underline{k}_1 = \underline{k}_2 = 1$. In this case, during Algorithm 1 we have to solve $\overline{n} \cdot \overline{m} = 25$ optimization problems. The results for all combinations are shown in Table 5.1. All but one problem can be solved in a few seconds. The smallest optimal value is given for $k^\star = (2, 3)$. This configuration is the same as in the optimization problem in Chapter 4, see Table 4.9, despite the fact that there are no constraints for variant differences in Chapter 4. The optimization problem with $\widehat{k} = (3, 2)$ needs more than one minute, but remains an exception with this runtime.

CRANE_N5_4			
$\widehat{k}$	$\widehat{obj}$	deviation costs	runtime $[s]$
(1, 1)	242.18	227.18	0.0
(1, 2)	64.1	44.1	0.07
(1, 3)	41.95	16.95	1.09
(1, 4)	36.92	6.92	1.37
(1, 5)	41.15	6.15	0.89
(2, 1)	181.15	156.15	0.04
(2, 2)	36.15	6.15	1.38
(2, 3)	35.77	0.77	2.47
(2, 4)	40.0	-0.0	2.15
(2, 5)	45.0	-0.0	1.01
(3, 1)	185.5	150.5	0.14
(3, 2)	45.15	5.15	89.45
(3, 3)	45.01	0.01	1.16
(3, 4)	50.35	0.35	3.17
(3, 5)	55.0	0.0	3.79
(4, 1)	195.73	150.73	0.17
(4, 2)	54.83	4.83	4.4
(4, 3)	73.24	18.24	0.86
(4, 4)	60.0	0.0	6.58
(4, 5)	65.0	-0.0	3.2
(5, 1)	206.68	151.68	1.5
(5, 2)	64.88	4.88	5.34
(5, 3)	65.0	-0.0	4.56
(5, 4)	69.99	-0.01	6.72
(5, 5)	75.0	-0.0	5.41

TABLE 5.1: CRANE_N5_4: Results fixed variants

In the second experiment, we consider again example CRANE_N20_1 with 20 crane bridges and at most ten profile variants and five sheet variants. In the previous chapter, it is remarkable in this example that the solver does not find the optimal configuration of one profile variant and three sheet variants. We knew that there is a better feasible point from the example CRANE_N20_4. The solver in the previous chapter terminates with one profile variant and two sheet variants and costs of 90.40, where a modular system with one profile variant and three sheet variants should be cheaper. With this prior knowledge, we will now look at the results of Algorithm 1. The results are given in Table 5.2.

Several things are noticeable at this point. Not all optimization problems are solved within the set time limit (here two hours). In these cases, $\widehat{obj}$ is marked with "- (1)", and the runtime of 7200 s is specified. The solver does not find any unsolvability issues due to inconsistency within the time limit in this example. The best variant configuration found is $k^\star = (1, 3)$ with the already known costs of 88.4. Here, it is once again apparent that the modular system with $\widehat{k} = (1, 2)$ is more expensive than the configuration just discussed, since it has costs of 90.39. What is also noticeable here, however, is that some problems have a very long runtime, if they terminate at all. Other problems, on the other hand, such as those where $\widehat{k}$ is close to the optimal $k^\star$ in particular, are solved in seconds. Why the solver has such difficulties with some problems remains unclear at this point. It is at least noticeable that for all problems that are not solved within the time limit, $\widehat{k}_S = 2$ (and $\widehat{k}_P \geq 5$) applies. However, in the previous chapter, the solver required "only" 266 seconds. The time required for Algorithm 1, where all problems have to be solved, is therefore much higher for this specific example than in the solution approach in Chapter 4. However, at this point, it is noticeable that problems where $\widehat{k}$ is close to $k^\star = (1, 3)$, i.e., problems with $||k^\star - \widehat{k}||_\infty \leq \omega$, where $\omega \in \mathbb{N}_0$ is a small parameter, can be solved quickly. Furthermore, the optimal values in this neighborhood are quite small. For example, for $\widehat{k} = (1, 4)$ we have costs of 94.09, for $\widehat{k} = (2, 3)$ costs of 90.54, and for $\widehat{k} = (2, 4)$ costs of 97.27. If we only search for the best configuration in this "area", we would be done quickly. This idea is further explored in Chapter 6.

Another thing that stands out in this example is that for very low and very high numbers of variants, the costs for the modular system become very high and are among the highest. This makes intuitive sense and was expected. The highest costs are for $k^\star = (10, 4)$ at 366.21. What is unexpected at this point is that for $\widehat{k} = (10, 5)$, the costs are slightly lower again at 286. We assume that this is due to the fairly large tolerance limits set for continuous variables. This is 10^{-3} in the results from Table 5.2. If we reduce the tolerance to 10^{-4} and try to solve this selected problem, the solver does not find a solution within the timelimit of two hours. We mark optimal values that are unexpected with a * or $^+$, depending on whether they can be solved with the new tolerance (then with *) or not (with $^+$). At first glance, the results for $\widehat{k} = (2, 2)$ and $\widehat{k} = (4, 3)$ are also noticeable. Resolving the problem $\overline{P}^{crane}((2, 2))$ leads to an optimal value of 89.76, which fits into the expected range, as it lies between the optimal values of the problems with $\widehat{k} = (2, 1)$ and $\widehat{k} = (2, 3)$. For the problem $\overline{P}^{crane}((4, 3))$, we obtain costs of 126.8, which fits better with expectations. Some other optimal values are marked in the table. We discuss the behavior of the costs and why and whether we can expect a certain behavior at all in Subsection 5.3.3. Here, too, it becomes clear that the optimization problems for crane bridge problems are difficult to handle numerically, in particular if we choose tight tolerances.

We omit at this point the results for example CRANE_N20_4, which has the

CRANE_N20_1			
$\widehat{k}$	$\widehat{obj}$	deviation costs	runtime $[s]$
(1, 1)	192.49	162.49	0.02
(1, 2)	90.39	50.39	0.48
(1, 3)	88.47	38.47	11.23
(1, 4)	94.09	34.09	10.4
(1, 5)	102.07	32.07	43.81
(2, 1)	171.79	121.79	0.13
(2, 2)	223.26*	163.26	0.22
(2, 3)	90.54	20.54	26.37
(2, 4)	97.27	17.27	130.02
(2, 5)	105.32	15.32	453.33
(3, 1)	189.75	119.75	3.03
(3, 2)	106.76	26.76	22.56
(3, 3)	107.72	17.72	2745.3
(3, 4)	118.84	18.84	139.46
(3, 5)	125.01	15.01	345.32
(4, 1)	209.41	119.41	8.86
(4, 2)	126.29	26.29	78.72
(4, 3)	223.62*	113.62	4.26
(4, 4)	152.95	32.95	29.69
(4, 5)	155.6	25.6	67.41
(5, 1)	229.24	119.24	10.68
(5, 2)	- (1) -	-	7200.01
(5, 3)	151.92	21.92	1206.22
(5, 4)	160.14	20.14	81.44
(5, 5)	172.13	22.13	102.92
(6, 1)	249.25	119.25	21.22
(6, 2)	166.23	26.23	970.76
(6, 3)	168.1	18.1	3549.17
(6, 4)	185.17	25.17	55.85
(6, 5)	301.2	131.2	14.1
(7, 1)	269.3	119.3	131.88
(7, 2)	- (1) -	-	7200.01
(7, 3)	293.45^{+}	123.45	8.73
(7, 4)	212.84	32.84	36.12
(7, 5)	218.8	28.8	310.01
(8, 1)	289.42	119.42	134.53
(8, 2)	- (1) -	-	7200.01
(8, 3)	210.04	20.04	179.83
(8, 4)	221.66	21.66	72.05
(8, 5)	232.63	22.63	119.82
(9, 1)	309.55	119.55	233.42
(9, 2)	- (1) -	-	7200.02
(9, 3)	340.43^{+}	130.43	11.07
(9, 4)	258.07	38.07	87.96
(9, 5)	377.58	147.58	24.37
(10, 1)	329.96	119.96	156.04
(10, 2)	- (1) -	-	7200.01
(10, 3)	356.45	126.45	18.62
(10, 4)	366.21	126.21	24.51
(10, 5)	286.17^{+}	36.17	155.03

TABLE 5.2: CRANE_N20_1: Results fixed variants

CRANE_N2_1			
$\widehat{k}$	$\widehat{obj}$	deviation costs	runtime $[s]$
$(1,1)$	- (3) -	-	0.0
$(1,2)$	349.82	223.46	1.18
$(2,1)$	- (3) -	-	0.0
$(2,2)$	330.65	201.7	0.44

TABLE 5.3: CRANE_N2_1: Results fixed variants

same input data as CRANE_N20_1, but with ten instead of five sheet variants. Due to the large number of variables, the solver had difficulties, and the tolerances for the solver had to be adjusted. As a result, the results no longer match exactly. Furthermore, no new insights can be gained since the best configuration is anyway at $k^\star = (1,3)$, and at this point, $k_S^\star < 5$ applies.

The last experiment, example CRANE_N2_1 in this subsection, is a new data set that we have not considered yet. This example is very small, with only two crane bridges and $\overline{n} = \overline{m} = 2$. The special aspect is that the two crane bridges requested are very different. We consider two crane bridges that have to carry the same load capacity of 5 t, but have very different spans, here one meter and 13 meters. Additionally, we require here that the number of profiles in a crane bridge is bounded (here by 41). This condition can be natural in the application, but is not considered yet to keep the optimization model as simple as possible. Thus, we have the new constraints

$$z_i^P \leq \overline{z}^P, \quad i = 1, 2,$$

with $\overline{z}^P = 41$.

If we only allow a bounded profile number in a crane bridge, one sheet variant may not be enough, for example, for the different spans. This can be seen in Table 5.3, as the corresponding optimization problems are infeasible if $\widehat{k}_S = 1$. They are marked with "- (3) -", which is the flag for infeasibility.

For the sake of completeness, the necessary input data for example CRANE_N2_1 and the results corresponding to the optimization problem $\overline{P}^{crane,w}$ can be found in the appendix in Tables A.5, A.6 and A.7.

Next, we look at the Binpacking examples.

5.2.2 Decomposition Approach Binpacking Modular System

We start with binpacking problems with only one component. Then we consider the general p-dimensional binpacking examples followed by the numerical results.

Optimization Problem - One-Dimensional Binpacking

In Subsection 4.3.3, we introduced the well-defined optimization problem for the one-dimensional binpacking example, which is denoted by $\overline{P}_{bin}^1$. We now consider the decomposition solution approach and fix the number of variants. In this example with only one component (color), we again omit the index c in the variables.

Since the optimization problems are very similar to those in the previous chapter, we will only discuss the new constraints in detail. The unchanged constraints are listed below for clarity.

For notational reasons, we interpret $\widehat{k}$ as a number (not a vector of length one). Firstly , we fix exactly $\widehat{k}$ variables v_i to one. We again do this by

$$\begin{aligned} v_i &= 1, \quad i = 1, \dots, \widehat{k}, \\ v_i &= 0, \quad i = \widehat{k}+1, \dots, \overline{n}. \end{aligned} \tag{5.18}$$

Secondly, we have to ensure that the fixed variants are used. We do this corresponding to Equation 5.2 by

$$\sum_{\ell=1}^{N} z_{i,\ell} \geq v_i. \tag{5.19}$$

The remaining constraints that depend on the set $\widehat{Z}(v, \xi)$, which can be exactly transferred from the previous chapter, are given by

$$z_{i,\ell} \geq 0, \quad i = 1, \dots, \overline{n}, \ \ell = 1, \dots, N \tag{5.20}$$

$$z_{i,\ell} \leq v_i \overline{z}^{\ell}, \quad i = 1, \dots, \overline{n}, \ \ell = 1, \dots, N, \tag{5.21}$$

$$\sum_{i=1}^{\overline{n}} x_i z_{i,\ell} \leq L^{\ell}, \quad \ell = 1, \dots, N, \tag{5.22}$$

$$\sum_{i=1}^{\overline{n}} z_{i,\ell} \leq \overline{o}_{\ell}, \quad \ell = 1, \dots, N. \tag{5.23}$$

The constraints depending on the set $\widehat{X}(v)$ are given by

$$x_i \geq v_i \underline{x}, \quad i = 1, \dots, \overline{n}, \tag{5.24}$$

$$x_i \leq v_i \overline{x}, \quad i = 1, \dots, \overline{n}, \tag{5.25}$$

$$v_i x_i + \tau v_i \leq v_{i+1} x_{i+1}, \quad i = 1, \dots, \overline{n} - 1. \tag{5.26}$$

They are also the same as in the previous chapter. The constraint that at least one variant must be used is replaced by (5.19), respectively. In Section 5.1 it is mentioned that it is still important at this point to require that the variants differ. In the one-dimensional binpacking problem, this is already given through the constraints in Equation (5.26).

The complete optimization problem, with a fixed number of variants $\widehat{k}$ for the objects is then given by

One-Dimensional Binpacking Problem with Fixed Number of Variants

The optimization problem for the one-dimensional binpacking problem with a fixed number of variants $\widehat{k}$ is given by

$$\begin{aligned}
\overline{P}^1_{bin}(\widehat{k}): \quad & \min_{v,\xi,z} \; c_v\widehat{k}+c_d\sum_{\ell=1}^{N}\left(L^\ell-\sum_{i=1}^{\overline{n}} x_i z_{i,\ell}\right) & \\
\text{s.t.} \quad & (5.19), (5.20), (5.21), (5.22), (5.23), & \# \; z \\
& (5.24), (5.25), (5.26), & \# \; \xi \\
& (5.18), & \# \; v \\
& z \in \mathbb{Z}^{N\cdot\overline{n}}, \; \xi \in \mathbb{R}^{\overline{n}}, \; v \in \mathbb{B}^{\overline{n}}, &
\end{aligned}$$

with given $\widehat{k} \in \mathbb{Z}$ and $\widehat{k} \in [1, \overline{n}]$.

The problem $\overline{P}^1_{bin}(\widehat{k})$ can be inconsistent, especially if the conditions (5.26) are too restrictive for a high $\widehat{k}$. If $\overline{P}^1_{bin}(\widehat{k})$ is inconsistent for a $\tilde{k}$, it is clear by the construction of the example that all k with $k > \tilde{k}$ do not need to be considered further and can be omitted from the set $\mathcal{L}$, since they cannot be solvable. On the other hand, this knowledge makes it easy to generate problem data and choose $\overline{n}$ so that such an unsolvability does not occur.

Before we continue with the numerical results, we consider the higher-dimensional problem.

Optimization Problem - p-Dimensional Binpacking

As in the one-dimensional case, we do not have to change much. As new constraints in the optimization problem for the p-dimensional binpacking problem with fixed variants $\widehat{k} = (\widehat{k}_1, \dots, \widehat{k}_p)$ and $\widehat{k}_c \leq \overline{k}_c$, we get

$$\begin{aligned}
v_i^c &= 1, \quad i = 1, \dots, \widehat{k}_c, \; c = 1, \dots, p, \\
v_i^c &= 0, \quad i = \widehat{k}_c + 1, \dots, \overline{k}_c, \; c = 1, \dots, p,
\end{aligned} \tag{5.27}$$

$$\sum_{\ell=1}^{N} z_{i,\ell}^c \geq v_i^c, \quad 1, \dots, \overline{k}_c, \; c = 1, \dots, p. \tag{5.28}$$

The constraints that depend on z or ξ and z are given by

$$z_{i,\ell}^c \leq v_i^c \overline{z}_\ell^{ub}, \quad i = 1, \dots, \overline{k}_c, \; c = 1, \dots, p, \; \ell = 1, \dots, N, \tag{5.29}$$

$$z_{i,\ell}^c \geq 0, \quad i = 1, \dots, \overline{k}_c, \; c = 1, \dots, p, \; \ell = 1, \dots, N, \tag{5.30}$$

$$\sum_{c=1}^{p}\sum_{i=1}^{\overline{k}_c} x_i^c z_{i,\ell}^c \leq L^\ell, \quad \ell = 1, \dots, N, \tag{5.31}$$

$$\sum_{c=1}^{p}\sum_{i=1}^{\overline{k}_c} z_{i,\ell}^c \leq \overline{o}_\ell, \quad \ell = 1, \dots, N. \tag{5.32}$$

The constraints that depend only on ξ and v are given by

$$x_i^c \geq 0, \quad i = 1, \dots, \overline{k}_c,\ c = 1, \dots, p, \tag{5.33}$$

$$x_i^c \leq v_i^c \overline{x}^{ub}, \quad i = 1, \dots, \overline{k}_c,\ c = 1, \dots, p,, \tag{5.34}$$

$$v_i^c x_i^c + \tau v_i^c \leq v_{i+1}^c x_{i+1}^c, \quad i = 1, \dots, \overline{k}_c - 1,\ c = 1, \dots, p. \tag{5.35}$$

The complete optimization problem for the p-dimensional binpacking problem with fixed variants is given by

p-Dimensional Binpacking Problem with Fixed Number of Variants

The optimization problem for the p-dimensional binpacking problem with fixed number of variants $\widehat{k} = (\widehat{k}_1, \dots, \widehat{k}_p)$ is given by

$$\begin{aligned}
\overline{P}^p_{bin}(\widehat{k}) : \quad \min_{v,\xi,z} \ & \sum_{c=1}^{p} c_v^c \widehat{k}_c + c_d \sum_{\ell=1}^{N} \left(L^\ell - \sum_{c=1}^{p} \sum_{i=1}^{\overline{k}_c} x_i^c z_{i,\ell}^c \right) \\
\text{s.t.} \quad & (5.28), (5.29),\ (5.30), (5.31),\ (5.32), && \#\ z \\
& (5.33), (5.34), (5.35), && \#\ \xi \\
& (5.27), && \#\ v \\
& z \in \mathbb{Z}^{N \cdot e^\top \overline{k}},\ \xi \in \mathbb{R}^{e^\top \overline{k}},\ v \in \mathbb{B}^{e^\top \overline{k}}
\end{aligned}$$

with given $\widehat{k}_c \in ([0, \overline{k}_c] \cap \mathbb{Z})$.

Similarly to the one-dimensional case, a problem $\overline{P}^p_{bin}(\widehat{k})$ can be inconsistent if the parameters τ^c become too large and the variants cannot be accommodated in the bins. It is also clear that if $\overline{P}^p_{bin}(\tilde{k})$ with the parameter $\tilde{k}$ is inconsistent then the problem $\overline{P}^p_{bin}(\widehat{k})$, where $\widehat{k}_c > \tilde{k}_c$ for any $c = 1, \dots, p$ holds, is also inconsistent. However, inconsistency can effectively be ruled out by selecting suitable bin lengths and corresponding τ^c values, and so all problems in Algorithm 1 have an optimal point.

After introducing the optimization problems for the binpacking application, we continue with numerical experiments.

Results Algorithm 1 - One-Dimensional Binpacking

We start with the already known basic one-dimensional example BP_DIM1_1 that was introduced in Subsection 4.3.4. Concrete input data, such as length of the bins, allowed number of objects in a bin, the minimum difference for the length of the objects are the same as in the previous chapter. Details can be found in Table 4.18 (upper left part) and Table 4.19. Based on Algorithm 1, we fix the variants from one to ten. All optimization problems are solvable in only a few seconds. In Table 5.4, we show the optimal values of the problems $\overline{P}^1_{bin}(\widehat{k})$ for all considered $\widehat{k}$, as well as the costs for the deviation from the total length and the runtime for solving the optimization problem.

In the table for each $\widehat{k}$ the optimal value $\widehat{obj}$, the total deviation (sum over all bins), and the runtime are given. The smallest optimal value is given for $\widehat{k} = 4$

BP_DIM1_1			
$\widehat{k}$	$\widehat{obj}$	deviation costs	runtime $[s]$
(1)	164.0	154.0	0.01
(2)	76.0	56.0	0.03
(3)	52.0	22.0	0.1
(4)	49.4	9.4	0.18
(5)	53.0	3.0	0.32
(6)	60.4	0.4	0.46
(7)	70.0	0.0	0.38
(8)	80.0	0.0	0.26
(9)	90.0	0.0	0.16
(10)	100.0	0.0	0.13

TABLE 5.4: BP_DIM1_1: Results fixed variants

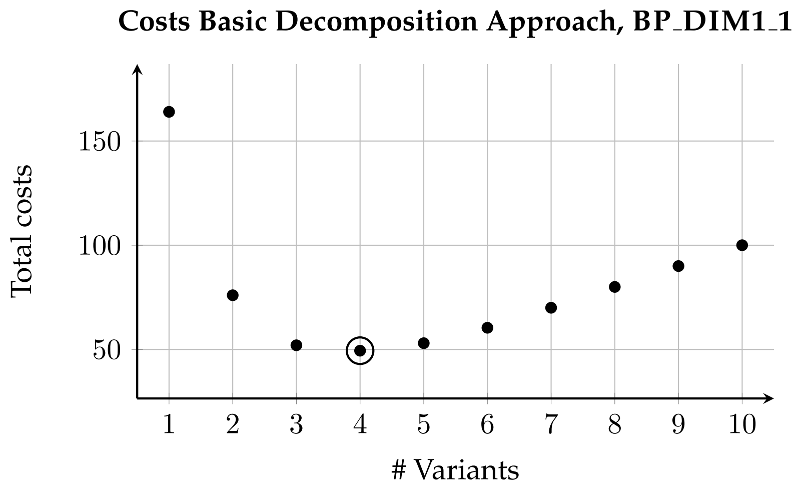

FIGURE 5.1: BP_DIM1_1: Plot optimal values fixed variants

with $\widehat{obj} = 49.40$, which is clearly the same as the result of the previous section, see Table 4.18. In this example, it is also noticeable that there is no free space left in the bins from $\widehat{k} = 7$ onwards. For this reason, the total costs grow linearly from this point onwards by exactly the cost factor $c_v = 10$. For further considerations in Subsection 5.3.3, we plot the total costs for each $\widehat{k} \in \{1, \dots, 10\}$.

The data can be taken from Figure 5.1 , which shows that the total costs are highest if the number of variants is either small or large. This is not particularly surprising and is to be expected from the application. Somewhere in the middle, here at $\widehat{k} = 4$ variants (marked with an additional circle at the data point), we find the optimal number of variants for which the modular system is most cost-efficient. The fact that the total costs are growing linearly is also visible in the plot.

Next, we consider two new datasets in the examples BP_DIM1_6 and BP_DIM1_7 for the one-dimensional binpacking example. The example BP_DIM1_7 in particular will be interesting in Section 5.3.3 with some more modifications. Table 5.5 gives a brief overview of the upcoming examples here with a short explanation.

name	description
BP_DIM1_6	deviation costs increasing again
BP_DIM1_7	more than one global minimal point

TABLE 5.5: Overview for a case study in the one-dimensional binpacking example (Part 1)

In example BP_DIM1_1 the sum of deviations is zero at one point ($\widehat{k} = 7$) and remains zero as $\widehat{k}$ increases. This is not universally valid, as the constraints for larger $\widehat{k}$ can lead (due to the minimum differences between variants) to free space remaining in the bins. In example BP_DIM1_6, we consider this case. As we did not consider this example before, we add the input data in the appendix, Table A.20 for the lengths of the bins, and Table A.21 some other input data. Table 5.6 shows the corresponding facts and Figure 5.2 shows the costs in the plot for the decomposition approach. For $\widehat{k} = 2$, the total costs of the modular system are minimal and there is no remaining space in the bins. From $\widehat{k} = 4$ onwards, free space is available again in the bins.

After the first two examples (BP_DIM1_1 and BP_DIM1_6) it seems that the lowest total cost is always assumed for a unique number of variants. However, this is generally not valid for the very simple binpacking problem. We consider BP_DIM1_7, which is an example with new input data that becomes more important in Subsection 5.3.3. We consider eight bins. The input data are again given in the appendix in Table A.22 and Table A.23. In Table 5.7 and Figure 5.3 the results corresponding to Algorithm 1 are shown.

It is immediately apparent that for three and five variants, the modular system has a total cost of 75, which is the minimum total cost. In this example, it is therefore not unique for how many variants the cheapest modular system is available. We should mention here that we are only interested in the uniqueness of the number of variants at this point and not in the uniqueness of the length of the objects. At the same time, it could, of course, still be possible that the lengths of the objects are also not unique. However, we will not examine this further at this point. In Subsection 5.3.3 we consider the example BP_DIM1_7 in more detail and with some

BP_DIM1_6			
$\widehat{k}$	$\widehat{obj}$	deviation costs	runtime $[s]$
(1)	50.0	40.0	0.02
(2)	20.0	0.0	0.01
(3)	30.0	0.0	0.01
(4)	42.0	2.0	0.02
(5)	54.0	4.0	0.0

TABLE 5.6: BP_DIM1_6: Results fixed variants

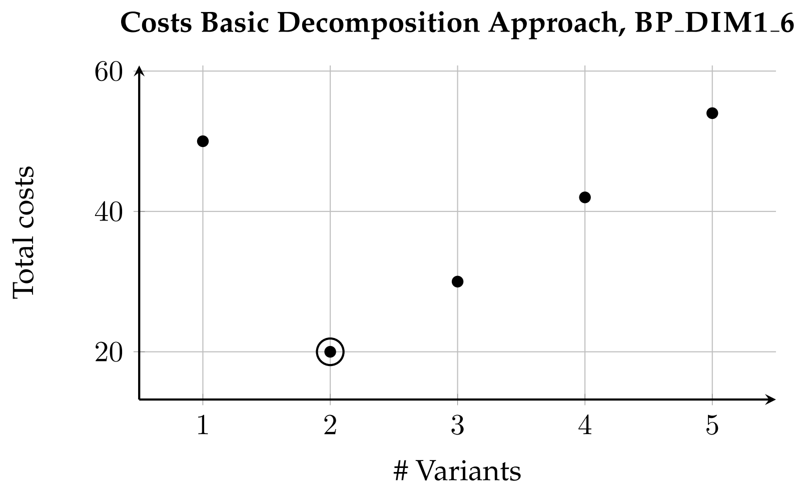

FIGURE 5.2: BP_DIM1_6: Plot optimal values fixed variants

BP_DIM1_7			
$\widehat{k}$	$\widehat{obj}$	deviation costs	runtime $[s]$
(1)	295.0	280.0	0.02
(2)	110.0	80.0	0.02
(3)	75.0	30.0	0.04
(4)	80.0	20.0	0.12
(5)	75.0	0.0	0.06
(6)	90.0	0.0	0.04
(7)	105.0	0.0	0.03
(8)	120.0	0.0	0.03

TABLE 5.7: BP_DIM1_7: Results fixed variants

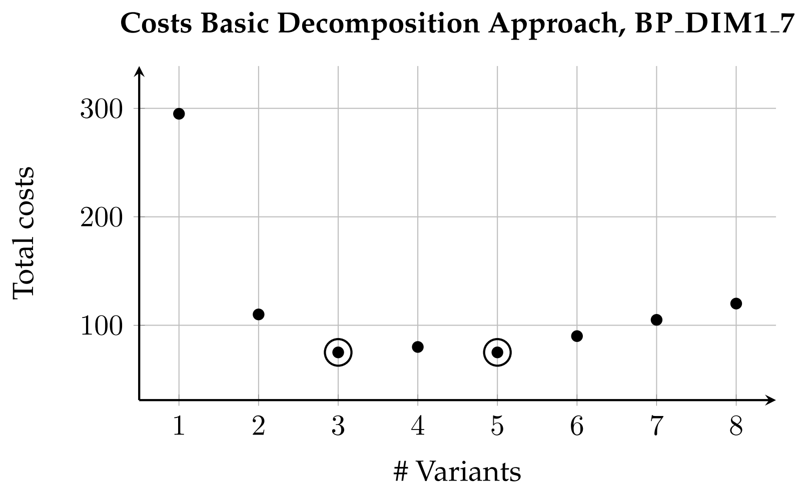

FIGURE 5.3: BP_DIM1_7: Plot optimal values fixed variants

Costs Basic Decomposition Approach, BP_DIM2_1

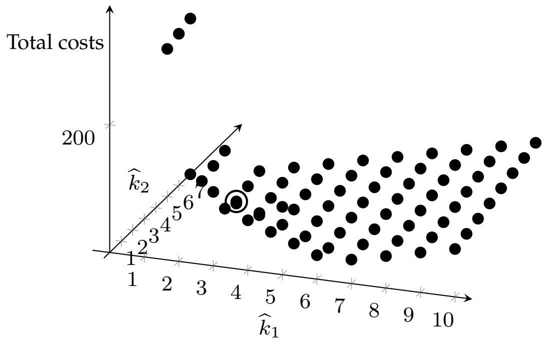

FIGURE 5.4: BP_DIM2_1: Plot optimal values fixed variants

modifications. Before we can continue with that, we need some basic mathematical concepts, which will be introduced in the next section. However, before doing this, we discuss the results for the higher-dimensional binpacking problems.

Results Algorithm 1 - p-Dimensional Binpacking

In the last part of Subsection 4.3.4 we extended the basic one-dimensional example BP_DIM1_1 to higher dimensions, in particular up to dimension four. The lengths of the bins remain the same as in the one-dimensional case. We only allow more components (colors).

In the example BP_DIM2_1, we consider two components and allow ten variants for each component, thus we have $\overline{k} = (10, 10)$. As we allow $\widehat{k}_c = 0$ for one of the two components, in Algorithm 1 we have to consider 120 combinations of variants, thus 120 optimization problems need to be solved. The input data are given in Table 4.25 (upper left part). We obtain the cost minimal modular system for $\widehat{k} = (2, 5)$ with $\widehat{obj} = 45$. In this example, not all problems $\overline{P}^2_{bin}(\widehat{k})$ have an optimal point, as can be seen in Table 5.8. Inconsistent problems are marked with "-". All problems $\overline{P}^2_{bin}(\widehat{k})$ with $\widehat{k}_2 \geq 8$ are inconsistent. Even in the case if $\widehat{k}_1 = 0$, there is no optimal point, since there is no feasible point as the bins are too short for the objects with a difference of at least $\tau^2 = 50$. In the case of very few variants, the total costs $\widehat{obj}$ are very high, since we only allow two objects per bin, and thus the deviation costs are very high.

In the case of two components, we can plot the calculations of Algorithm 1 as well as in the case of only one dimension. We obtain a three-dimensional plot, with $\widehat{k}_1$ on the x-axis, $\widehat{k}_2$ on the y-axis, and the object values of the corresponding problem $\overline{P}^2_{bin}(\widehat{k})$ on the z axis. In Figure 5.4 the optimal values over $\widehat{k}$ are visualized for object values below 250. For a clear visualization, we only plot a selection of the data points. In the upper right part of the figure, the points lie along a hyperplane. This is due to the fact that the total deviation for these fixed variants is zero and thus the values of the objective function only increase by factor c^c_v. The best variant number is again marked with an additional circle.

Next, we consider the extension to three components. In the algorithm a lot of problems, in particular $11^3 - 1 = 1330$ problems, need to be solved if we choose

$\overline{k} = (10, 10, 10)$ as in the already considered example BP_DIM3_1. As we increase the parameter τ^c for the new cheaper component, a lot of problems are also inconsistent. For that reason, we reduce $\overline{k}$ to $\overline{k} = (10, 7, 6)$, such that most problems have an optimal point, and denote the example with BP_DIM3_2. Now we have to consider $11 \cdot 8 \cdot 7 - 1 = 615$ combinations. The optimal values of all the problems are given in Tables A.24 and A.25. Most problems have an optimal point. The lowest optimal value is 24.00 and is only reached for $\widehat{k} = (1, 2, 4)$. For the sake of completeness, we put the results from the model without fixed variants in the Appendix, Table A.18.

Lastly, we consider the example with four components. Instead of $\overline{k} = (10, 10, 10, 10)$ of example BP_DIM4_1, we consider $\overline{k} = (5, 5, 5, 5)$ such that most of the 1295 problems are not inconsistent. We denote the problem by BP_DIM4_2. The tables can be found in the appendix in Tables A.26-A.28. The configuration $\widehat{k} = (0, 1, 2, 4)$ leads to the cheapest modular system with costs of 9.0. This variant combination is unique in this example. For the sake of completeness, we put the results from the model without fixed variants in the Appendix, Table A.10.

After considering the examples, in particular CRANE_N20_1 or BP_DIM3_2 and BP_DIM4_2, it is clear that solving the problems with all possible $\widehat{k}$ is not an efficient way to find an optimal configuration for a modular system. When we presented the results for Algorithm 1 for example CRANE_N20_1, we already remarked that there are a few "good fixations" around the best $k^\star$. In the one- and two-dimensional examples for the binpacking problem, we already plotted the costs for the modular system over the corresponding $\widehat{k}$ and thus gave an impression of what the represented, so-called discrete function might look like. In example BP_DIM1_7 this function had in particular more than one minimal point, which is interesting.

We now discuss the behavior of the costs by introducing discrete functions. We look at them both in theory and by considering the known examples.

BP_DIM2_1

$\widehat{k}$	$\widehat{obj}$	deviation	runtime $[s]$
(0,1)	3855.0	3850.0	0.01
(0,2)	1410.0	1400.0	0.03
(0,3)	565.0	550.0	0.04
(0,4)	257.5	237.5	0.04
(0,5)	225.0	200.0	0.05
(0,6)	230.0	200.0	0.07
(0,7)	235.0	200.0	0.06
(0,8)	-	-	-
(0,9)	-	-	-
(0,10)	-	-	-
(1,0)	3860.0	3850.0	0.01
(1,1)	1415.0	1400.0	0.03
(1,2)	570.0	550.0	0.12
(1,3)	260.0	235.0	0.32
(1,4)	87.5	57.5	0.38
(1,5)	65.0	30.0	0.31
(1,6)	70.0	30.0	0.15
(1,7)	75.0	30.0	0.33
(1,8)	-	-	-
(1,9)	-	-	-
(1,10)	-	-	-
(2,0)	1420.0	1400.0	0.04
(2,1)	575.0	550.0	0.1
(2,2)	265.0	235.0	0.59
(2,3)	85.0	50.0	0.64
(2,4)	50.0	10.0	0.59
(2,5)	45.0	0.0	0.54
(2,6)	50.0	0.0	0.31
(2,7)	55.0	0.0	0.26
(2,8)	-	-	-
(2,9)	-	-	-
(2,10)	-	-	-
(3,0)	580.0	550.0	0.06
(3,1)	270.0	235.0	0.33
(3,2)	90.0	50.0	0.55
(3,3)	55.0	10.0	1.16
(3,4)	50.0	0.0	0.3
(3,5)	55.0	0.0	0.21
(3,6)	60.0	0.0	0.08
(3,7)	65.0	0.0	0.07
(3,8)	-	-	-
(3,9)	-	-	-
(3,10)	-	-	-
(4,0)	275.0	235.0	0.14
(4,1)	95.0	50.0	0.63
(4,2)	60.0	10.0	1.32
(4,3)	55.0	0.0	0.71
(4,4)	60.0	0.0	0.17
(4,5)	65.0	0.0	0.11
(4,6)	70.0	0.0	0.06
(4,7)	75.0	0.0	0.08
(4,8)	-	-	-
(4,9)	-	-	-
(4,10)	-	-	-
(5,0)	125.0	75.0	0.22
(5,1)	65.0	10.0	1.17
(5,2)	60.0	0.0	0.84
(5,3)	65.0	0.0	0.14
(5,4)	70.0	0.0	0.1
(5,5)	75.0	0.0	0.08
(5,6)	80.0	0.0	0.07
(5,7)	85.0	0.0	0.07
(5,8)	-	-	-
(5,9)	-	-	-
(5,10)	-	-	-
(6,0)	70.0	10.0	0.62
(6,1)	65.0	0.0	0.25
(6,2)	70.0	0.0	0.25
(6,3)	75.0	0.0	0.12
(6,4)	80.0	0.0	0.1
(6,5)	85.0	0.0	0.11
(6,6)	90.0	0.0	0.11
(6,7)	95.0	0.0	0.06
(6,8)	-	-	-
(6,9)	-	-	-
(6,10)	-	-	-
(7,0)	70.0	0.0	0.32
(7,1)	75.0	0.0	0.17
(7,2)	80.0	0.0	0.12
(7,3)	85.0	0.0	0.11
(7,4)	90.0	0.0	0.1
(7,5)	95.0	0.0	0.12
(7,6)	100.0	0.0	0.04
(7,7)	105.0	0.0	0.07
(7,8)	-	-	-
(7,9)	-	-	-
(7,10)	-	-	-
(8,0)	80.0	0.0	0.12
(8,1)	85.0	0.0	0.23
(8,2)	90.0	0.0	0.11
(8,3)	95.0	0.0	0.11
(8,4)	100.0	0.0	0.11
(8,5)	105.0	0.0	0.1
(8,6)	110.0	0.0	0.06
(8,7)	115.0	0.0	0.11
(8,8)	-	-	-
(8,9)	-	-	-
(8,10)	-	-	-
(9,0)	90.0	0.0	0.13
(9,1)	95.0	0.0	0.13
(9,2)	100.0	0.0	0.05
(9,3)	105.0	0.0	0.1
(9,4)	110.0	0.0	0.12
(9,5)	115.0	0.0	0.15
(9,6)	120.0	0.0	0.09
(9,7)	125.0	0.0	0.12
(9,8)	-	-	-
(9,9)	-	-	-
(9,10)	-	-	-
(10,0)	100.0	0.0	0.16
(10,1)	105.0	0.0	0.13
(10,2)	110.0	0.0	0.1
(10,3)	115.0	0.0	0.13
(10,4)	120.0	0.0	0.08
(10,5)	125.0	0.0	0.11
(10,6)	130.0	0.0	0.15
(10,7)	135.0	0.0	0.21
(10,8)	-	-	-
(10,9)	-	-	-
(10,10)	-	-	-

TABLE 5.8: BP_DIM2_1: Results fixed variants

5.3 Discrete Functions and Their Properties

It is clear that applying Algorithm 1 is identical to determining a global minimal point of a discrete function. Graphs of these discrete functions can be seen, for example, in Figure 5.1 or 5.4. We obtain the plots by solving the problem $\overline{P^{gen}}(\widehat{k})$ for all possible combinations of variants $\widehat{k}$ and plotting the optimal values over $\widehat{k}$. Since discrete functions are not widely used, we consider them in more detail in this section. In Subsection 5.3.1, we first introduce what we mean by a discrete function and present concepts for the optimal points of them. As in the continuous case, convexity plays an important role in minimizing a discrete function. In Subsection 5.3.2, convexity concepts for discrete functions are introduced based on the existing literature. The literature is also presented there. In Subsection 5.3.3, the convexity concepts are applied to discrete functions from the modular system problems. In the last subsection, Subsection 5.3.4, further discrete functions are introduced for testing purposes, which are also briefly examined for convexity. These will play a greater role in the numerical experiments in Chapter 6.

5.3.1 Introduction and Definition of Discrete Functions

To further examine the problem structure of modular systems, we need the concept of discrete functions. Discrete functions differ fundamentally from the classic continuous functions in their domain, which now contains only vectors of integers $x \in \mathbb{Z}^n$ instead of $\mathbb{R}^n$. The image space can still be real. In this section, we will mainly refer to the literature of Kazuo Murota [45, 47, 46, 44]. In his work, the domain of the discrete function is the set $\mathbb{Z}^n$ in general. In our application, we only need to consider the sets $X = [\underline{x}, \overline{x}] \cap \mathbb{Z}^n$ as a domain, where the points in X correspond to the possible number of variants for each component.

In the p-dimensional binpacking problem, for example, we have $\widehat{k} \in [0, \overline{k}] \cap \mathbb{Z}^n$ with $\overline{k} = (\overline{k}_1, \dots, \overline{k}_p)$ and where $\overline{k}_c$ is the upper bound for the number of variants of component c (here color c). The corresponding discrete function for this modular system problem therefore has exactly the domain $[0, \overline{k}] \cap \mathbb{Z}^n$.

The intervals from the paragraph above are n-dimensional intervals that are defined as in [57, Subsection 3.3.2]:

Definition 5.3.1. *(n-Dimensional Interval, Box) For $\underline{x}, \overline{x} \in \mathbb{R}^n$ with $\underline{x} \leq \overline{x}$ the set*

$$[\underline{x}, \overline{x}] = \{x \in \mathbb{R}^n \mid \underline{x} \leq x \leq \overline{x}\}$$

is called an n-dimensional interval or a box.

For notational reasons, we also define in the following a discrete analogue, the term of a “discrete box”. In addition, we need its continuous relaxation:

Definition 5.3.2 (Discrete Box and its Continuous Relaxation)**.** *Let the set $[\underline{x}, \overline{x}]$ with $\underline{x} \leq \overline{x}$ and $\underline{x}, \overline{x} \in \mathbb{Z}^n$ be a box with integer bounds. Then $X = [\underline{x}, \overline{x}] \cap \mathbb{Z}^n$ is called a discrete box and the set (box) $\widehat{X} = [\underline{x}, \overline{x}]$ is the continuous relaxation of X.*

Thus, we define discrete functions, similar as in [45], as follows:

Definition 5.3.3 (Discrete Function)**.** *Let $X \subseteq \mathbb{Z}^n$ be a discrete box. Then a function $f : X \to \mathbb{R}$ is called a discrete function.*

We now consider the optimization problems $\overline{P^{gen}}(\widehat{k})$ of Section 5.1 and their optimal values $\widehat{obj}$. In order to indicate the explicit dependence on $\widehat{k}$ in the optimal value of the problem, we now denote this by $v(\widehat{k})$. In the numerical results for Algorithm 1, we have already plotted $v(\widehat{k})$ for all $\widehat{k} \in \big([1, \overline{k}] \cap \mathbb{Z}^n\big)$, for example, in Figure 5.1. The function values of the shown discrete function are the optimal values of the optimization problem $\overline{P^{gen}}(\widehat{k})$ in which $\widehat{k}$ enters as a parameter. The domain of the function is in our case a grid of integer numbers of one or more dimensions. The illustration of a two-dimensional discrete function is shown in Figure 5.4.

As we are interested in the variant vector $\widehat{k}$ with the lowest total costs, we want to solve an optimization problem

$$P: \quad \min_{x \in \mathbb{R}^n} \; f(x) \quad \text{s.t.} \quad x \in M$$

with a discrete feasible set $M \subseteq \mathbb{Z}^n$ and a discrete function $f : X \to \mathbb{R}$ with $M \subseteq X \subseteq \mathbb{Z}^n$. In the following, we usually use $M = X$ with a discrete box X. When the current optimization literature refers to discrete optimization problems, this only means that the feasible set is integer. In addition, in discrete optimization f is usually assumed to be defined on the continuous relaxation of the feasible set (or a superset of it), see [13, Chapter 1]. In our case, the optimal value function is a discrete function that is initially defined only for discrete points from X. Very little literature can be found for this situation. The optimal modular system, i.e., the most cost-effective modular system for us, is available for $k^\star \in [\underline{k}, \overline{k}]$, for which v has the lowest value. In the example BP_DIM1_1, to which Figure 5.1 belongs, this is $k^\star = 4$ or, in the two-dimensional example BP_DIM2_1 with Figure 5.4, $k^\star = (2, 5)$. In the context of this section, we now denote the point $k^\star$ as the global minimal point of the discrete function v. The global minimal points of discrete functions are defined similarly as in the continuous case:

Definition 5.3.4 (Global Minimal Point Discrete Function)**.** *The point $x^\star \in M$ is a global minimal point of P if and only if $f(x^\star) \leq f(x)$ holds for all $x \in M$.*

The function v of the examples BP_DIM1_1 and BP_DIM2_1 have unique global minimal points, but example BP_DIM1_7 has two global minimal points, as can be seen in Figure 5.3.

Altogether, running Algorithm 1 is equivalent to searching for a global minimal point $k^\star$ of the function v on the discrete box $M = X = [\underline{k}, \overline{k}] \cap \mathbb{Z}^C$ by evaluating the function v at all points in M. Evaluating the function v is equivalent to solving an optimization problem at this point.

In addition to global optimal points, continuous optimization also includes the concept of local optimal points. In continuous optimization, a point $\tilde{x}$ is a local minimal point, if its function value is at least as low as the function values of the points in a small neighborhood U. An important question now is how to define the neighborhood U if we transfer this concept to discrete optimization. In the continuous case we usually choose boxes or balls around the point $\tilde{x}$ for the set U, so we define accordingly $U = U(\tilde{x}) = \{x \in \mathbb{R}^n | \; ||x - \tilde{x}|| \leq \varepsilon\}$, with radius $\varepsilon > 0$. The Euclidean norm or the maximum norm is often chosen as norm. No matter which norm we choose, when checking for local minimality, all possible directions around the point $\tilde{x}$ are examined.

A natural definition of a discrete neighborhood $U \subseteq \mathbb{Z}^n$ around a point $\widehat{x}$ is given, for example, by

$$N^B(\widehat{x}) = \{x \in \mathbb{Z}^n | \, ||x - \widehat{x}||_\infty \leq 1\}. \tag{5.36}$$

All points in $N^B(\widehat{x})$ lie in a box around $\widehat{x}$, while the distance to $\widehat{x}$ is at least one. We say the points in $N^B(\widehat{x})$ are the points in the "box neighborhood" of $\widehat{x}$. From now on we set $M = X$ in general. Then local minimal points in the box-sense are called box-local minimal points and are defined as follows:

Definition 5.3.5 (Box-Local Minimal Point). *Let X be a discrete box, and $f : X \to \mathbb{R}$ be a discrete function. Then the point $\tilde{x} \in X$ is called a box-local minimal point of f if*

$$f(\tilde{x}) \leq f(x) \; \forall \, x \in \left(N^B(\tilde{x}) \cap X\right)$$

holds.

The notation of local minimal points is based on [57, Definition 1.1.3] and can also be found in the literature of Murota, [45, Section 2.1]. In the papers of Murota, several other local minimality concepts are introduced, depending on the type of convexity he introduces. The box neighborhood concept is introduced in the context of the L$^\#$-convexity, which we also introduce in Subsection 5.3.2, so this concept is also useful for us. It can be found for example in [47, Theorem 7.14]. An advantage of concept in Definition 5.3.5 is that we do not need to check too many points to verify if a point is box-local minimal point. However, it is precisely this aspect that we are only considering a few points that will prove problematic.

However, it is unnatural that we only consider a very small number of "directions" around a point $\tilde{x}$. This fact sounds very abstract here, but it will become very important later when we want to derive relationships between local and global minimal points under convexity assumptions, similar to the ones known from continuous functions. For this reason, we introduce another neighborhood concept, the "direction neighborhood" N^D. We say that a point $x \in \mathbb{Z}^n$ lies in the direction neighborhood of $\tilde{x} \in \mathbb{Z}^n$ if there is no integer point on the line-segment strictly between x and $\tilde{x}$ (denoted by $\overline{x\tilde{x}}$). Thus, we define

$$N^D(\tilde{x}) = \{x \in \mathbb{Z}^n | \;\; \nexists \widehat{x} \in \mathbb{Z}^n : \widehat{x} \in \overline{x\tilde{x}}\}. \tag{5.37}$$

The line-segment between $\widehat{x}$ and x does not contain any integer point if the greatest common divisor of the entries of the vector $d = \widehat{x} - x = (d_1, \ldots, d_n)$ is 1. Graphically speaking, this neighborhood concept is somewhat closer to the continuous idea. All points that can be reached by moving from $\tilde{x}$ to another integer point without visiting another integer point (or in other words that are visible from $\tilde{x}$) are in the neighborhood $N^D(\tilde{x})$. We denote the corresponding concept of a local minimal point with visibility-local and define it through as follows:

Definition 5.3.6 (Visibility-Local Minimal Point). *Let X be a discrete box and $f : X \to \mathbb{R}$ a discrete function. Then the point $\tilde{x} \in X$ is called a visibility-local minimal point of f, if*

$$f(\tilde{x}) \leq f(x) \; \forall \, x \in \left(N^D(\tilde{x}) \cap X\right)$$

holds.

x_1	x_2	$f(x)$
1	1	308.00
1	2	84.00
2	0	304.33
2	1	66.00
2	2	100.00
3	0	91.00
3	1	82.00
3	2	116.00
4	0	64.00
4	1	98.00
4	2	132.00
5	0	80.00
5	1	114.00
5	2	148.00

TABLE 5.9: Example 5.3.8: Data

In principle, however, significantly more points must be considered in order to check whether a point is locally minimal in this sense. In fact, almost all points must be taken into account, thus the verification process is rather more complex. If you take it accurately, the term "local" is no longer a good choice, since we do not stay in a neighborhood close to the point $\tilde{x}$ when we determine $N^D(\tilde{x})$. However, we will see later that this type of local optimality from Definition 5.3.6 is essential with regard to convexity. In the case of $n = 1$ we have $N^B(\tilde{x}) = N^D(\tilde{x})$. For a point $\tilde{x}$, it is obvious that $N^B(\tilde{x}) \subseteq N^D(\tilde{x})$ holds. This implies

Corollary 5.3.7. *Let X be a discrete box and $f : X \to \mathbb{R}$ be a discrete function. Let $\tilde{x}$ be a visibility-local minimal point. Then it is also a box-local minimal point. The converse of this statement is not valid in general.*

To show, that the converse statement of Corollary 5.3.7 does not hold in general, we consider the following example.

Example 5.3.8. We consider a small example with the data from Table 5.9. These data are not chosen randomly, they fit to a binpacking modular system example from Subsection 5.3.3. For the moment, for presentation purposes, we consider $X = \{(x_1, x_2) \in \mathbb{Z}^2 |\, x_1 \in [1, 5],\ x_2 \in [0, 2]\} \setminus \{(1, 0)\}$. However, in general, nothing changes in this example if we later consider a larger set X. In Figure 5.5 the data are illustrated and a part of the convex hull of the points in the space is marked. The faces should indicate how the points are distributed in the space. In addition, the two points $x^\star = (4, 0)$ and $\widehat{x} = (2, 1)$ are highlighted. The point $x^\star$ is the global minimal point of the data considered here, as it has the lowest function value with $f(x^\star) = 64$.

Besides the global minimal point, we are interested in local minimal points here. First, we consider the box neighborhood. In Figure 5.6, the domain of f is shown with the points $x^\star$, $\widehat{x}$ and its box neighborhoods $N^B(x^\star)$ and $N^B(\widehat{x})$. In addition, all directions (or connections) between $x^\star$ ($\widehat{x}$) and the points of $N^B(x^\star)$ ($N^B(\widehat{x})$) are illustrated with dashed lines. If we consider the corresponding function values it

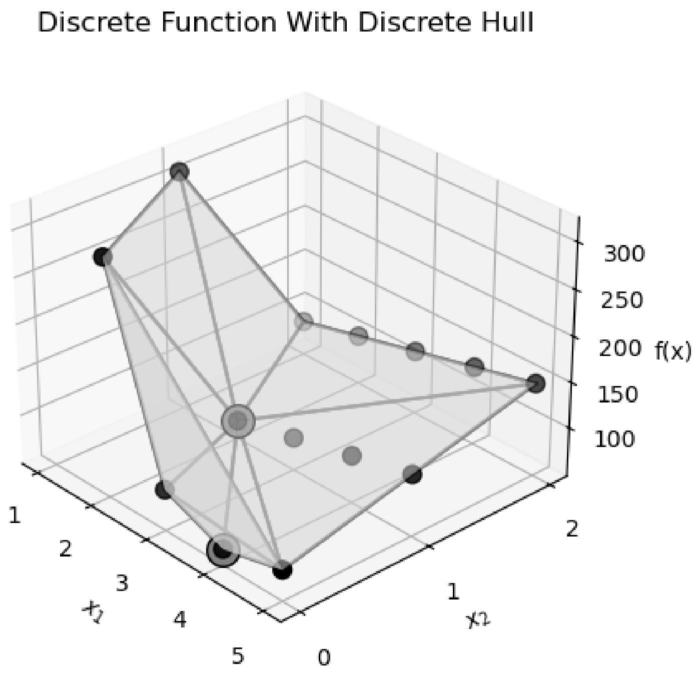

FIGURE 5.5: Example 5.3.8: Plot of the data and parts of the convex hull

is easy to verify that $\widehat{x}$ is a box-local minimal point. Since $\widehat{x}$ is box-local minimal point, we know that the function values must increase along the violet dashed lines. However, it is important to note that we are not considering the direction leading to point $x^\star$. In this direction, the function values decrease, as can be also be seen in Figure 5.5.

If we now consider the direction neighborhood instead of the box neighborhood, it is clear that $\widehat{x}$ is not a visibility-local minimal point, since the comparison of the function values now involves comparing $f(\widehat{x})$ and $f(x^\star)$. The corresponding illustration on the grid can be found in Figure 5.7. The consideration of the direction neighborhood thus also fits "better" with Figure 5.5. This will become interesting later on.

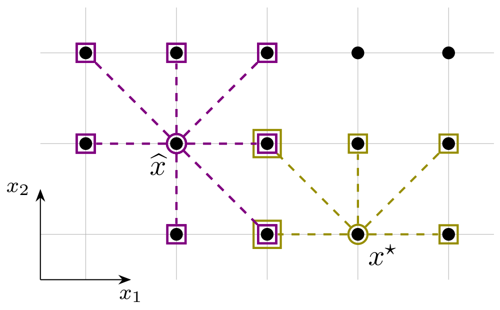

FIGURE 5.6: Example 5.3.8: Grid with box neighborhood

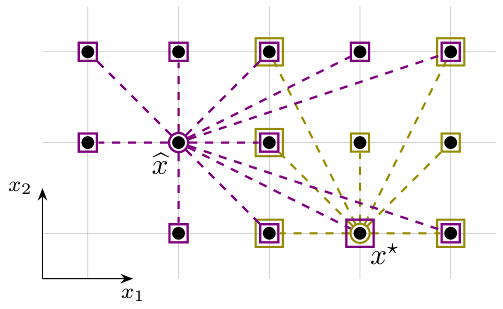

FIGURE 5.7: Example 5.3.8: Grid with direction neighborhood

Now that we have introduced discrete functions and local and global minimal points in this context, we move on to an important property of functions that arises in this context for continuous functions. This property is convexity.

5.3.2 Discrete Convexity

Convexity in general is a useful function property in continuous optimization. Convexity says something about the curvature behavior of a function. If a continuous function is convex local and global minimal points coincide. For that reason, if convexity of the function is known, numerical solvers, which only terminate with guaranteed local minimal points, can be used. For discrete functions such concepts are of course also interesting, in particular, under which conditions local minimal points are also global ones. Here we must clearly distinguish which concept of neighborhood we are using and which is used in the literature. We begin with the most important convexity definitions and connections for continuous functions.

Convexity of Continuous Functions

We consider a function $\overline{f} : \widehat{X} \to \mathbb{R}$ with a convex set $\widehat{X} \subseteq \mathbb{R}^n$. A set $\widehat{X}$ is convex if for every $x, y \in \widehat{X}$ and $\lambda \in [0,1]$ the convex combination $(1-\lambda)x + \lambda y$ also lies in $\widehat{X}$. We again mark sets that are without integer constraints with a hat symbol. In general, the convexity concepts for continuous functions are defined for arbitrary sets $\widehat{X}$, later we use boxes. For the moment, $\widehat{X}$ is an convex set without integer constraints. To distinguish between discrete and continuous functions, we denote continuous functions with a bar from now on.

In one dimension a function $\overline{f}$ is said to be convex, if all line segments between two arbitrary points $(x, \overline{f}(x))$ and $(y, \overline{f}(y))$ lie above the graph of $\overline{f}$ for $x, y \in \widehat{X}$ and $\widehat{X}$ convex, or in general as in [7, Section 3.1]:

Definition 5.3.9 (Convexity Continuous Function). *Let $\widehat{X}$ be a convex set. A function $\overline{f} : \widehat{X} \to \mathbb{R}$ is convex on $\widehat{X}$ if*

$$\overline{f}((1-\lambda)x + \lambda y) \leq (1-\lambda)\overline{f}(x) + \lambda \overline{f}(y)$$

holds for all $x, y \in \widehat{X}$ and $\lambda \in [0,1]$.

For sufficient smooth functions, convexity can be mostly easily verified through characterizations (C^1- or C^2). The C^1-characterizations of convexity is given through the following theorem, based on [7, Section 3.1.2]:

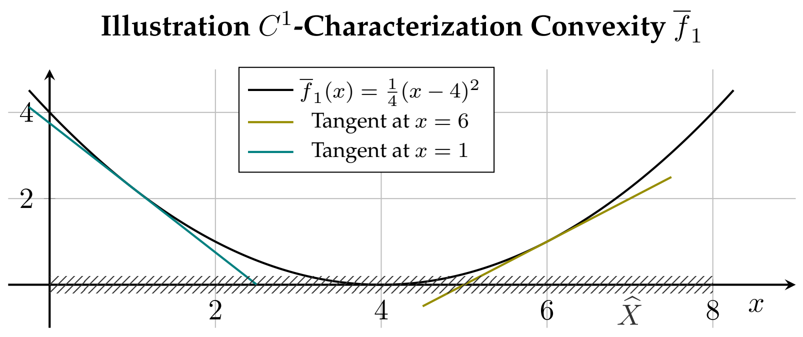

FIGURE 5.8: Illustration C^1-characterization convexity for $\overline{f}_1(x) = \frac{1}{4}(x-4)^2$ with $\widehat{X} = [0, 8]$ and two tangents

Theorem 5.3.10 (C^1-Characterization Convexity)**.** *Let $\widehat{X}$ be a convex set. A function $\overline{f} \in C^1(\widehat{X}, \mathbb{R})$ is convex on $\widehat{X}$ if and only if*

$$\overline{f}(y) \geq \overline{f}(x) + \nabla \overline{f}(x)^\top (y - x)$$

holds for all $x, y \in \widehat{X}$.

Theorem 5.3.10 says "that a function is convex [. . .] if and only if its graph lies above each of its tangent spaces"([57, Subsection 2.2.2]). If $n = 1$ the tangent spaces are simple tangents.

Next, we consider an example.

Example 5.3.11. We consider two functions $\overline{f}_1$ and $\overline{f}_2$ on a set X. Firstly, we have $\overline{f}_1(x) = \frac{1}{4}(x-4)^2$ and $\widehat{X} = [0, 8]$. An illustration with two tangents is shown in Figure 5.8. It is easy to see that for every point in $\widehat{X} = [0, 8]$, a tangent can be found that lies below the graph of $\overline{f}_1$. So $\overline{f}_1$ is a convex function on $\widehat{X}$ according to Theorem 5.3.10.

Secondly, we consider the function $\overline{f}_2(x) = 4\cos(\frac{\pi}{4}x) + 4$, which is illustrated in Figure 5.9. $\overline{f}_2$ is obviously nonconvex on $\widehat{X} = [0, 8]$, since, for example, we cannot find a tangent at the point $x = 6$ that lies completely below the graph of $\overline{f}_2$.

For the transition to discrete convexity, we again consider two functions f_1 and f_2, that are now discrete and which fulfill $\overline{f}_1(x) = f_1(x)$ and $\overline{f}_2(x) = f_2(x)$ for $x \in X$ where $X = \{1, \ldots, 8\}$ is a discrete box and $\widehat{X} = [0, 8]$ is its continuous relaxation. The resulting discrete functions are illustrated in Figures 5.10 and 5.11.

In our understanding, the function f_1 should now be discretely convex, and the function f_2 should not. Similarly to the continuous case, for f_2, the bulges or kinks are now at $x = 2$ and $x = 7$, points at which the function "does not behave convexly".

If the feasible set M of the optimization problem is a convex set and the (continuous) function $\overline{f}$ is convex on M, then every local minimal point is always a global minimal point, see [7, Section 4.2.2]. In this sense, convexity is a very helpful concept for continuous optimization problems, which we now want to consider also in the discrete case. If we can apply the concept of convexity to the discrete case, for example, in Algorithm 1, we can stop when a point has been identified as a local minimal point. In such a case, a high number of function evaluations of v, which

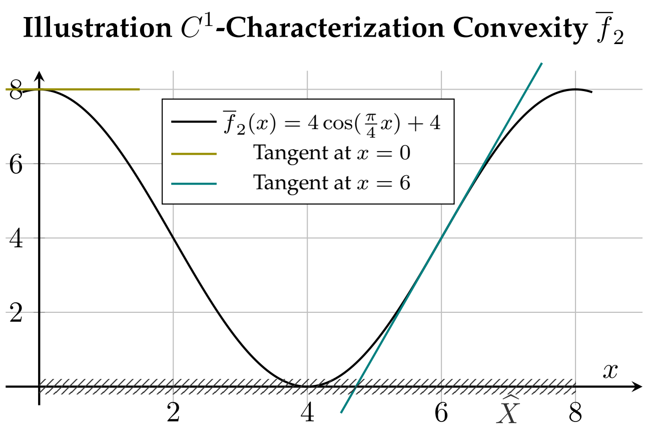

FIGURE 5.9: Illustration C^1-characterization convexity for $\overline{f}_2(x) = 4\cos(\frac{\pi}{4}x) + 4$ with $\widehat{X} = [0, 8]$ and two tangents

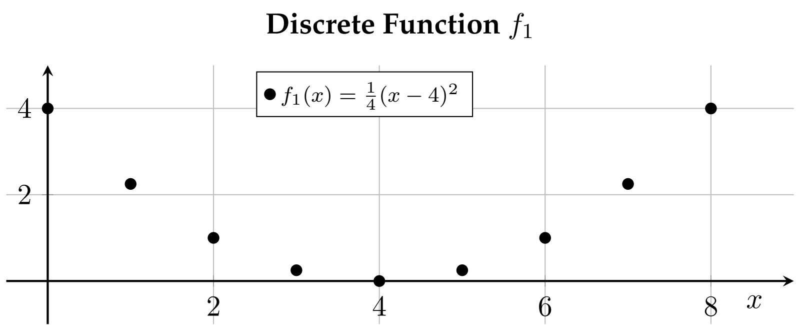

FIGURE 5.10: Discrete function f_1 on $X = [0, 8] \cap \mathbb{Z}$

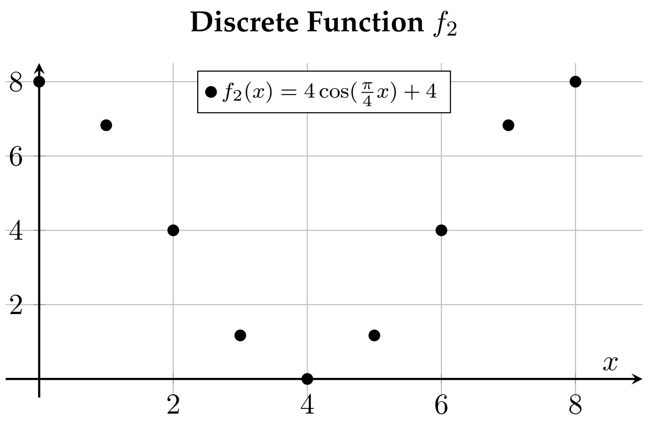

FIGURE 5.11: Discrete function f_2 on $X = [0, 8] \cap \mathbb{Z}$

in our case corresponds to solving an optimization problem, can be prevented. So we need a convexity concept for discrete functions that can be verified easily and under which local minimal points are also global minimal points. At the same time, we must specify what we mean by a local minimal point. As mentioned above, the neighborhoods can be defined differently.

There are relatively few authors with articles in the literature under the keyword discrete convexity. One mathematician who repeatedly stands out is the Japanese Kazuo Murota, whose discrete convexity concepts are presented in the following paragraph. We have already referred to his literature for the definition of convex functions and local minimality (in the N^B case).

Literature on Discrete Convexity and Definitions for Discrete Convexity

Murota introduced different types of convexity, so-called L-, L-natural, M-and M-natural convexity, in 1998 in [46]. These concepts are revisited in many of his papers ([45], [47], [60]). We consider the for us most intuitive convexity definitions of Murota, the L-natural and M-natural convexity, which we shorten into L$^{\#}$- and M$^{\#}$-convexity.

We take our information from two recent papers, the overview paper [45] and the more detailed paper [47]. In addition to Murota, we also quickly come across the physicist Bruce Miller, who also introduced discrete convexity concepts in the early 1970s in his paper [36]. However, the article [60] by Murota et al. from 2013 argues that the convexity concept introduced by Miller corresponds to Murota's L$^{\#}$-convexity. For this reason, we now limit ourselves to Murota's convexity concepts and present them below. There are many other papers by Murota besides those mentioned here, which go more into the mathematical framework and also cover applications. For example, the paper [28] introduces concepts that are supposed to correspond to the C^2-characterization of convexity. Since we do not need them in the further work, we refer to the aforementioned paper.

We remark again at this point that the concepts for convexity of Murota are introduced in his papers for functions $f : \mathbb{Z}^n \to \mathbb{R}$, or $\overline{f} : \mathbb{R}^n \to \mathbb{R}$, respectively, which means that the whole $\mathbb{Z}^n$ or $\mathbb{R}^n$ is chosen as domain. As we do not need the properties on the entire space, we again consider boxes $X \subseteq \mathbb{Z}^n$ and its continuous relaxations $\widehat{X} \subseteq \mathbb{R}^n$ in the formulations below. According to [45], a function f that is called discrete convex needs to satisfy two properties, which we discuss in more detail here. First, a local minimal point must also be a global minimal point. This is a point that we also mention above that it would be "natural" if a discrete convex function fulfills that. In addition, a discrete convex function f should be extendable to a continuous convex function $\overline{f}$ on $\widehat{X}$. This is, based on Murota, defined as follows:

Definition 5.3.12 (Convex Extensibility). *Let $X \subseteq \mathbb{Z}^n$ be a discrete box and $\widehat{X} \subseteq \mathbb{R}^n$ its continuous relaxation. Let $f : X \to \mathbb{R}$ be a discrete function. Then f is called convex extensible if there exists a convex function $\overline{f} : \widehat{X} \to \mathbb{R}$, with $\overline{f}(x) = f(x)$ for all $x \in X$. $\overline{f}$ is then called a convex extension of f.*

According to Murota, convex extensibility alone is not a "fruitful theoretical framework". Why this is the case, remains open from the literature. Nevertheless, this motivates the introduction and definition of L$^{\#}$- and

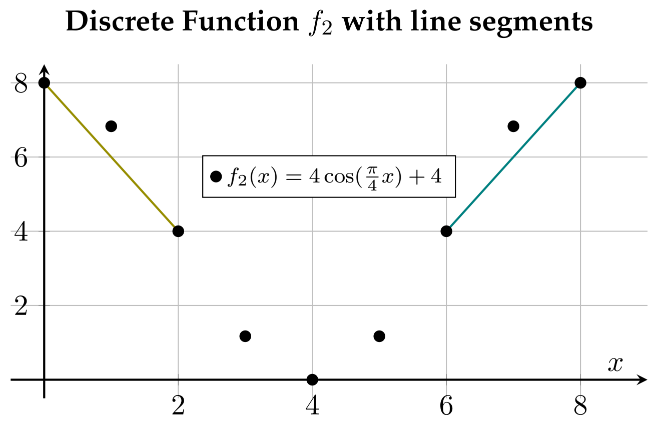

FIGURE 5.12: Discrete function f_2 on $X = [0, 8] \cap \mathbb{Z}$ with line segments

M$^{\#}$-convexity, which can be proven algorithmically. In the papers of Murota, L- and M-convexity (without the natural symbols) are also introduced. However, the "natural" variants are sufficient for us. An additionally important fact is that under L$^{\#}$- and M$^{\#}$-convexity local and global minimal points coincide, according to Murota. We discuss this later with the specification of which term of local minimal point is suitable and with the corresponding literature.

Before introducing M$^{\#}$-convexity and L$^{\#}$-convexity in general, we consider the case of univariate functions, which means $X \subseteq \mathbb{Z}$. Here, following Murota ([45, Section 2.2.1]), we define discrete convexity as follows:

Definition 5.3.13 (Discrete Convexity Univariate Function). *Let $X = [\underline{x}, \overline{x}] \subseteq \mathbb{Z}$ be a discrete box. Then a function $f : X \to \mathbb{R}$ is discrete convex if*

$$f(x) \leq \tfrac{1}{2} f(x-1) + \tfrac{1}{2} f(x+1), \quad \forall x \in [\underline{x}+1, \overline{x}-1]$$

holds.

This means that the graph of f lies under the line segment between $(x-1, f(x-1))$ and $(x+1, f(x+1))$ for every $x \in X$. According to Murota, a discrete univariate convex function f satisfies the property that local minimal points are also global minimal points ([45, Theorem 2]). Murota uses in this case the box-local definition from Definition 5.3.5. Since we have $n = 1$, all visibility-local minimal points are also global minimal points.

The functions f_1 and f_2 of Example 5.3.11 are univariate, so we can check the convexity with Definition 5.3.13.

Example 5.3.14. *(Sequel of Example 5.3.11)* With Figures 5.10 and 5.12 it can be easily verified that the function f_1 is discrete convex and f_2 not. The function f_2 and the line segments that violate the inequality in the definition are illustrated in Figure 5.12.

We remark at this point that f_2 has only one box- (and visibility) local minimal point that is also global in the discrete case, although f_2 is not discrete convex.

As we do not only obtain functions with a one-dimensional domain in modular system problems, we are also interested in discrete convexity concepts for functions with $X \subseteq \mathbb{Z}^n$ with $n > 1$. We continue to refer to [45]. There, another special case is considered, the discrete convexity for separable functions.

Let $X \subseteq \mathbb{Z}^n$ be a discrete box, $X_i \subseteq \mathbb{Z}$ and $X = X_1 \times \cdots \times X_n$. We consider functions $f : X \to \mathbb{R}$ with $f(x) = \sum_{i=1}^{n} f_i(x_i)$. Moreover, we have univariate functions $f_i : X_i \to \mathbb{R}$. If all f_i are discrete convex on X_i according to Definition 5.3.13, box-local minimal points are also global minimal points, see [45, Section 2.2.1].

So far, we have introduced discrete convexity concepts for special class of functions. Now we consider non-univariate and non-separable functions, and the aforementioned L$^\#$- and M$^\#$-convexity are introduced. Both convexity concepts depend on "discretization[s] of two different characterizations of convexity"([45]). We start with L$^\#$-convexity and the so called "midpoint convexity". A convex function $\overline{f} : \widehat{X} \to \mathbb{R}$, according to Definition 5.3.9, satisfies

$$\overline{f}\left(\frac{1}{2}x + \frac{1}{2}y\right) \leq \frac{1}{2}\overline{f}(x) + \frac{1}{2}\overline{f}(y). \tag{5.38}$$

with $\lambda = \frac{1}{2}$ and $x, y \in \widehat{X}$. Equation (5.38) is "called midpoint convexity [and] is known to be equivalent to convexity if f is a continuous function"([45, Subsection 2.2.2]). Equation (5.38) is equivalent to

$$\overline{f}\left(\frac{x+y}{2}\right) + \overline{f}\left(\frac{x+y}{2}\right) \leq \overline{f}(x) + \overline{f}(y). \tag{5.39}$$

If we now consider a discrete function, the inequality in (5.39) does not make sense for arbitrary $x, y \in \mathbb{Z}^n$, as $\frac{x+y}{2}$ is not integer in general. For that reason, we define the midpoint or, so called, L$^\#$-convexity for discrete functions as in [45, Section 2.2.2] as follows:

Definition 5.3.15 (L$^\#$-Convexity). *Let $X \subseteq \mathbb{Z}^n$ be a discrete box. A function $f : X \to \mathbb{R}$ is L$^\#$-convex on X, if*

$$f\left(\left\lfloor \frac{x+y}{2} \right\rfloor\right) + f\left(\left\lceil \frac{x+y}{2} \right\rceil\right) \leq f(x) + f(y)$$

holds for all $x, y \in X$.

The $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ statements in Definition 5.3.15 are meant component-wise. According to [47, Theorem 7.14], a point $x \in X$ is a global minimal point for the L$^\#$-convex function f, if and only if x is a local minimal point in the sense of the box neighborhood (Equation (5.36)). Additionally, L$^\#$-convex functions are convex extensible, see [45, Theorem 3].

In our specific application, we are only interested in testing discrete convexity as simple as possible. Murota mentions in his papers that checking whether functions are L$^\#$-convex (on $\mathbb{Z}^n$) can be done in polynomial time. How this should work remains unclear if we choose $\mathbb{Z}^n$, as Murota does in his work, instead of the discrete box X. As we have to consider many combinations of points x and y, proving L$^\#$-convexity on a discrete box X is nevertheless very time consuming, in particular in higher dimensions. To avoid this, we hope that the introduction of a weaker concept of convexity will bring about an improvement. But before doing that, we briefly consider the other main convexity concept of Murota, the M$^\#$-convexity.

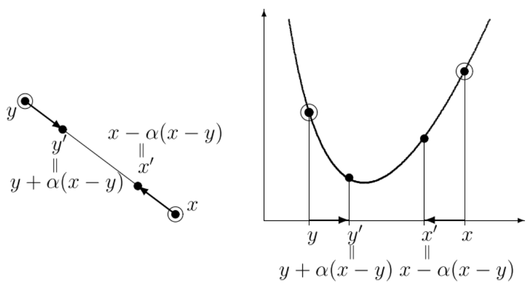

FIGURE 5.13: Equidistance convexity according to Murota, [45, Figure 2]

The deduction of M$^{\#}$-convexity is based on a discretization approach similar to that of the deduction of L$^{\#}$-convexity. A continuous convex function $\overline{f}$ satisfies in general

$$\overline{f}(x-\alpha(x-y))+\overline{f}(y+\alpha(x-y)) \leq \overline{f}(x)+\overline{f}(y) \tag{5.40}$$

for every $\alpha \in [0,1]$. This corresponds to applying Definition 5.3.9 twice with $\lambda = \alpha$ and the suitable choice of the points x and y in the inequality. The representation in (5.40) is interpreted as “equidistance convexity”, which is illustrated for the one-dimensional case in Figure 5.13.

The inequality in (5.40) means that “the sum of the function evaluated at two points, x and y, does not increase, if the two points approach each other by the same distance on the line segment connecting them”([45, Section 2.2.3]). As in L$^{\#}$-convexity, the points $x' = x-\alpha(x-y)$ and $y' = y+\alpha(x-y)$ are not guaranteed to be integer. For a discrete function, Murota simulates equidistance convexity “by moving a pair of points (x,y) to another pair (x',y') along the coordinate axes rather than on the connecting line segment”. In mathematical notations in the paper this yields

$$(x',y') = (x-\chi_i+\chi_j, y+\chi_i-\chi_j)$$

with $i \in \text{supp}^+(x-y)$ and $j \in \text{supp}^-(x-y) \cup \{0\}$. χ_i is the i-th unit vector, χ_0 the zero vector, and the special index sets are defined through

$$\text{supp}^+(z) = \{i |\, z_i > 0\}, \quad \text{supp}^- = \{i |\, z_i < 0\}.$$

This construction means that we have to go one unit in coordinate i and at most one unit in another coordinate j to determine (x',y') from (x,y).

Since the definitions are very complex, we consider a small example.

Example 5.3.16. We consider a two-dimensional domain and the points $x = \begin{pmatrix}5\\1\end{pmatrix}$ and $y = \begin{pmatrix}1\\4\end{pmatrix}$. We have $\text{supp}^+(x-y) = \{1\}$ and $\text{supp}^-(x-y) = \{2\}$. With this, two choices for (x',y') are possible. First, $(x',y') = \left(\begin{pmatrix}4\\1\end{pmatrix}, \begin{pmatrix}2\\4\end{pmatrix}\right)$, if we choose $i = 1$ and $j = 0$ (violet in the illustration) or second $(x'',y'') = \left(\begin{pmatrix}4\\2\end{pmatrix}, \begin{pmatrix}2\\3\end{pmatrix}\right)$ (olive in the illustration) for $i = 1$ and $j = 2$. The points are shown in Figure 5.14. It becomes clear that if the dimension of the domain is higher, there are a lot of more possibilities.

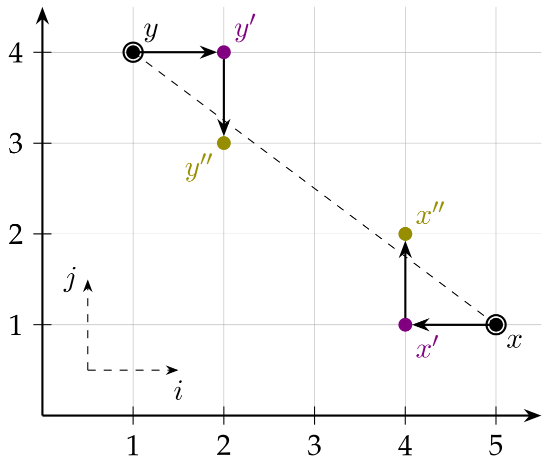

FIGURE 5.14: Illustration domain M$^{\#}$-convexity, based on [45, Figure 3]

With the formulations and definitions above, the analogue of equidistance convexity can be formulated for discrete functions. A discrete function that fulfills the equidistance convexity is called M$^{\#}$-convex from now on.

Definition 5.3.17 (M$^{\#}$-Convexity). *Let $X \subseteq \mathbb{Z}^n$ be a discrete box. Then $f : X \to \mathbb{R}$ is M$^{\#}$-convex on X, if there exists a $j \in supp^-(x-y) \cup \{0\}$ with*

$$f(x - \chi_i + \chi_j) + f(y + \chi_i - \chi_j) \leq f(x) + f(y)$$

for any $x, y \in X$ and any $i \in supp^+(x-y)$.

This definition and further considerations can be found in [47]. If we want to check M$^{\#}$-convexity for the example of Figure 5.14, we have to compare function values for many combinations of points and indices. It seems obvious that this cannot be done efficiently. Again, the question arises of how this should be done if we have $\mathbb{Z}^n$ instead of the discrete box X. How and how quickly we can verify M$^{\#}$-convexity is not clear from the papers by Murota that we considered. At this point, it is important to note that the definition of convexity in the one-dimensional domain case of Definition 5.3.13 is identical to Definition 5.3.17. This is obvious since for any $i \in \text{supp}^+(x-y)$ we have $j = 0$. With that we have $\chi_i = 1$ and $\chi_j = 0$ and the inequality from Definition 5.3.17 reduces to the inequality in Definition 5.3.13. It should also be mentioned that L$^{\#}$- and M$^{\#}$-convexity are the same concepts, for separable functions, see [45, Figure 5]. The most important property of M$^{\#}$-convex functions is that a local minimal point is always a global minimal point [45, Theorem 7]. A point is characterized as a local minimal point in the paper if

$$f(x) \leq f(x - \chi_i + \chi_j)$$

holds for all $i, j \in \{0, \ldots, n\}$. This characterization looks like the definition of a box-local minimal point (Definition 5.3.5) except that the diagonal element from x is not explicitly considered. More precisely $-\chi_i + \chi_j$ cannot be **1** or **-1**, where **1** is a vector of ones. This may possibly lead to the number of local minimal points differing from those defined by us. Incidentally, this problem does not arise with L$^{\#}$-convex functions, since local minimal points are defined differently in the paper. This aspect is particularly noticeable in Murota's papers. Various convexity concepts are introduced, and different definitions of local minimality are given in relation to them. Since L$^\#$-convexity and the box neighborhood considered in this context seem logical and natural to us, we now only consider L$^\#$-convexity.

The question remains at this point as to how we can verify or disprove L$^\#$-convexity in a fast and efficient way. In the papers, no ideas are presented in addition to the definition itself. In the continuous case, we usually work with C^1- and C^2-characterizations. In the process of our work, we asked ourselves whether such a concept, in particular, an equivalent to the C^1-characterization also exists for discrete convexity concepts for discrete functions. In the literature itself, we did not find such a simple concept that fits the C^1-characterization, which should, however, be easy to transfer geometrically. In the next paragraph, we consider this idea in more detail and introduce a new convexity concept, the so-called subgradient-convexity.

Subgradient-Convexity

We use the concept of supporting hyperplanes for convex sets. For the deduction of the concept, we start again with continuous functions. The convexity characterization with the epigraph of the function will turn out to be useful. A function $\overline{f} : \widehat{X} \to \mathbb{R}$ is convex if and only if its epigraph epi$(\overline{f}, \widehat{X}) = \{(x,t) |\, \overline{f}(x) \leq t,\ x \in \widehat{X}\}$ is a convex set, see [7, Section 3.1.7]. For this result, we do not need $\overline{f}$ to be smooth. According to [7, Section 2.5.2], convex sets have at least one supporting hyperplane at each boundary point of the set. By construction, the boundary of epi$(\overline{f}, X)$ is exactly the graph of $\overline{f}$. We now assume $\overline{f}$ to be convex. Then, according to [54, Section 23], the set

$$\mathcal{H} = \{z \in \mathbb{R}^n |\, H(z) = 0\} \tag{5.41}$$

with $H(z) := \overline{f}(x) + g^\top(z-x)$ is a non-vertical supporting hyperplane to the convex set epi$(\overline{f}, \widehat{X})$ at the point $(x, f(x))$, where $g \in \mathbb{R}^n$ is a subgradient of the function $\overline{f}$. The hyperplane $\mathcal{H}$ is called a supporting hyperplane to epi$(\overline{f}, \widehat{X})$ at the boundary point $(x, f(x))$, since $H(x) = \overline{f}(x)$ and $H(z) \leq \overline{f}(z)$ for all $z \in \widehat{X}$ since g is a subgradient of $\overline{f}$ which is defined as follows: A vector g is a subgradient of the convex function $\overline{f}$ at a point x, if

$$\overline{f}(z) \geq \overline{f}(x) + g^\top(z-x) \quad \forall z \in \widehat{X} \tag{5.42}$$

holds. With (5.42) it is clear that $\mathcal{H}$ from (5.41) is a supporting hyperplane. If $\overline{f}$ is differentiable at x, we can choose $g = \nabla\overline{f}(x)$ and we get exactly the inequality of the C^1-characterization of convexity. If $\overline{f}$ is not smooth at x, the choice of g is not unique, but possible, as for convex functions, these hyperplanes exist. In Figure 5.15 the set epi$(\overline{f}_1, \widehat{X})$ is shown with a supporting hyperplane at $x = 6$. Figure 5.16, on the other hand, shows the set epi$(\overline{f}_2, \widehat{X})$, where, for example, for $x = 2$, no supporting hyperplane can be found.

We now return to discrete functions and apply the idea with supporting hyperplanes. According to Murota, a discrete L$^\#$-convex function is convex extensible to a convex function $\overline{f}$. The set epi$(\overline{f}, \widehat{X})$ is thus convex as well, and for every $x \in \widehat{X}$ it has at least one supporting hyperplane at the point $(x, \overline{f}(x))$. This also holds in particular for all discrete points, since we have $X \subseteq \widehat{X}$ if we choose $\widehat{X}$ as continuous

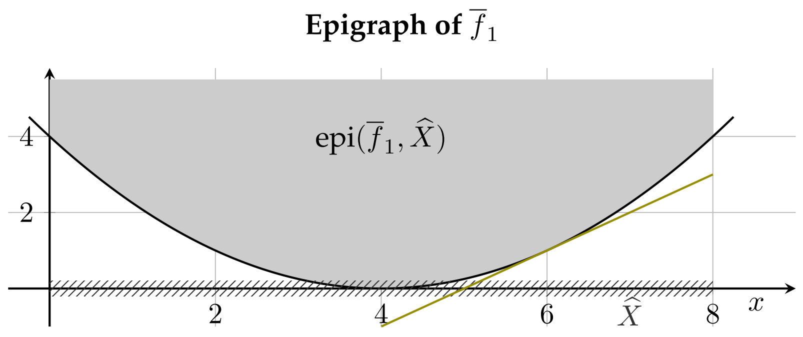

FIGURE 5.15: Epigraph of the continuous function $\overline{f}_1$ and a supporting hyperplane at $\overline{x} = 6$

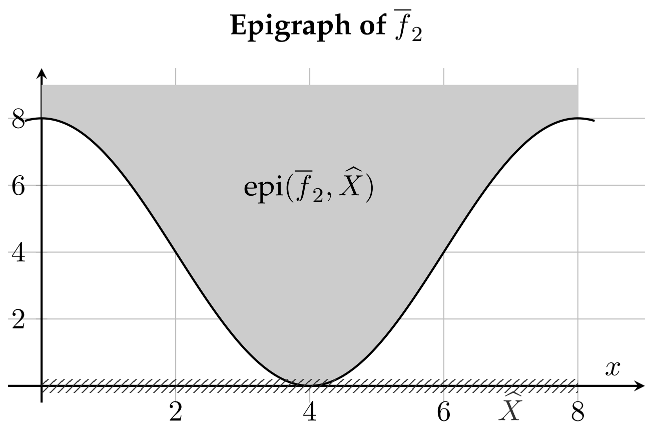

FIGURE 5.16: Epigraph of the continuous function $\overline{f}_2$

relaxation of X. So, there exists a vector $g \in \mathbb{R}^n$ such that

$$f(x) + g^\top(z - x) \leq f(z), \quad \forall z \in X, \tag{5.43}$$

is fulfilled for a point $x \in X$ since $\overline{f}(x) = f(x)$ holds for integer points (compare to Definition 5.3.12). We can apply this, of course, to all points $x \in X$ and obtain the following theorem:

Theorem 5.3.18. *Let $X \subseteq \mathbb{Z}^n$ be a discrete box and $f : X \to \mathbb{R}$ be an L$^\#$-convex function on X. Then for every $x \in X$ there exists a supporting hyperplane at the point $(x, f(x))$ that fulfills* (5.43). *The converse of the statement does not hold in general.*

Proof. "$\Rightarrow$" is done in the previous paragraph. A counterexample for the violation of the converse statement is given now. We consider a function $f : X \to \mathbb{R}$ with

$$f(x) = \max\{-2x_1 - 3x_2 - 2, 3 - 2x_2, -2x_1 + 3x_2 - 1\}$$

and $X = [-1, 1]^2 \cap \mathbb{Z}^2$. For the convex extension we choose the same function with domain $\widehat{X} = [-1, 1]$ respectively. According to [7, Section 3.2.3] $\overline{f}$ is a convex function on $\widehat{X}$, so for every $x \in \widehat{X}$ there exists a supporting hyperplane to the set epi$(\overline{f}, \widehat{X})$ in $(x, \overline{f}(x))$. With the corresponding subgradients g, (5.43) is fulfilled at the discrete points $x \in X$. With $x = \begin{pmatrix}-1\\0\end{pmatrix}$ and $y = \begin{pmatrix}1\\1\end{pmatrix}$ we have $\left\lfloor \frac{x+y}{2} \right\rfloor = \begin{pmatrix}0\\0\end{pmatrix}$ and $\left\lceil \frac{x+y}{2} \right\rceil = \begin{pmatrix}0\\1\end{pmatrix}$ and

$$\begin{aligned}
f(x) &= \max\{0, 3, 1\} = 3,\\
f(y) &= \max\{-7, 1, 0\} = 1,\\
f\left(\left\lfloor \frac{x+y}{2} \right\rfloor\right) &= \max\{-2, 3, -1\} = 3,\\
f\left(\left\lceil \frac{x+y}{2} \right\rceil\right) &= \max\{-5, 1, 2\} = 2.
\end{aligned}$$

This implies

$$f(x) + f(y) = 4 < 5 = 3 + 2 = f\left(\left\lfloor \frac{x+y}{2} \right\rfloor\right) + f\left(\left\lceil \frac{x+y}{2} \right\rceil\right),$$

which disproves that f is L$^\#$-convex. □

For discrete one-dimensional functions, the converse of the statement of Theorem 5.3.18 holds.

Theorem 5.3.19. *Let $X \subseteq \mathbb{Z}$ be a discrete box and $f : X \to \mathbb{R}$ be a discrete function on X. Then f is L$^\#$-convex if and only if there exists for every $x \in X$ a supporting hyperplane at the point $(x, f(x))$ that fulfills* (5.43).

Proof. We only have to show "$\Leftarrow$". Let f be a discrete function such that for all $x \in X$ there is a corresponding supporting hyperplane. Let $x \in X$ be now arbitrarily chosen with $x \notin \{\underline{x}, \overline{x}\}$. Then there exists at least one $g \in \mathbb{R}$, such that

$$\mathcal{H} = \{z \in \mathbb{R}^n | \ H(z) = 0\}$$

with $H(z) = f(x) + g(z - x)$ is a supporting hyperplane at the point $(x, f(x))$. That means in particular that we have $H(z) \leq f(z)$ for all $z \in X$. We also have $H(x-1) \leq f(x-1)$ and $H(x+1) \leq f(x+1)$. This is equivalent to

$$\begin{aligned} f(x) + g(x - 1 - x) &\leq f(x-1), \\ f(x) + g(x + 1 - x) &\leq f(x+1) \end{aligned}$$

and thus to

$$\begin{aligned} f(x) - g &\leq f(x-1), \\ f(x) + g &\leq f(x+1). \end{aligned}$$

The last two equations directly imply $2f(x) \leq f(x-1) + f(x+1)$ which is exactly the inequality of Definition 5.3.13. Since x is arbitrarily chosen, f is discrete convex according to Definition 5.3.13. This definition is equivalent to the definition of L$^{\#}$-convexity (Definition 5.3.15) in the one-dimensional case (see [45]). $\square$

In the one-dimensional case, we have now at least found an equivalent to the C^1-characterization, see Theorem 5.3.19. For higher dimensions, the equivalent does not hold, see Theorem 5.3.18.

In the following paragraph, we present an approach for easily checking whether there is a supporting hyperplane that satisfies (5.43). In the context of optimization problems, this can be done by a linear optimization problem. The question we ask ourselves because of Theorem 5.3.18 is whether, for every point $x \in X$, there exists a vector $g \in \mathbb{R}^n$ such that

$$f(x) + g^\top (z - x) \leq f(z)$$

holds for every $z \in X$. Since we assume that X is a discrete box, it is bounded, so we can set $X = \{x^1, \ldots, x^d\}$ and we can reformulate the above inequality. For a fixed $x \in X$ we have the following question:

$$\exists g \in \mathbb{R}^n \quad \text{with} \quad \max_{i=1,\ldots,d} g^\top (x^i - x) + f(x) - f(x^i) \leq 0?$$

Equivalently we can check if the optimal value of the problem

$$\min_{g \in \mathbb{R}^n} \max_{i=1,\ldots,d} g^\top (x^i - x) + f(x) - f(x^i)$$

is less than or equal to zero. With the epigraph reformulation ([57, Section 1.1]), we can instead solve the problem

$$P^{hyp}(x): \quad \min_{(g,t) \in \mathbb{R}^n \times \mathbb{R}} t \quad \text{s.t.} \quad g^\top (x^i - x) + f(x) - f(x^i) \leq t, \quad i = 1, \ldots, d.$$

and check whether the optimal value is less than or equal to zero. The optimization problem $P^{hyp}(x)$ is a standard linear (continuous) optimization problem, which can be solved efficiently with LP-solvers, for example Gurobi. If the optimal values of $P^{hyp}(x)$ are less than or equal to zero for all $x \in X$, all corresponding supporting hyperplanes exist. For this case and to shorten the notation in Theorem 5.3.18, we introduce the so-called subgradient convexity in the following definition:

Definition 5.3.20 (Subgradient Convexity)**.** *Let $X \subseteq \mathbb{Z}^n$ be a discrete box and $f : X \to \mathbb{R}^n$ a discrete function. Then f is called subgradient-convex if the problems $P^{hyp}(x)$ have an optimal value less than or equal to zero for all $x \in X$.*

For the sake of completeness, we have the following:

Corollary 5.3.21. *Let $X \subseteq \mathbb{Z}^n$ be bounded and $f : X \to \mathbb{R}$ a discrete function. If f is $L^{\#}$-convex, then f is also subgradient-convex. The converse of the statement is not valid in general. In the case of $n = 1$ the converse of the statement can be shown.*

Before heading to local and global optimality of L$^{\#}$- and subgradient-convex functions, we consider again Example 5.3.8.

Example 5.3.22 (Sequel of Example 5.3.8)**.** f is subgradient-convex, as we can see in Figure 5.5. For every x we can find a hyperplane which intersects the illustrated convex hull at more than one point.

f is not L$^{\#}$-convex. Let us consider $x = x^\star = (4, 0)$ and $y = \widehat{x} = (2, 1)$. Then we have $\lfloor \frac{x+y}{2} \rfloor = (3, 0)$ and $\lceil \frac{x+y}{2} \rceil = (3, 1)$. So we have

$$\left\lfloor \frac{x+y}{2} \right\rfloor + \left\lceil \frac{x+y}{2} \right\rceil = 91 + 82 = 173 \not\leq 124 = 64 + 66 = f(x) + f(y).$$

This implies that f is not L$^{\#}$-convex.

From a numerical point of view, we are primarily interested in convexity in terms of whether local and global minimal points coincide. This statement holds in the continuous case, and according to Murota, it also holds for discrete functions, at least if they are L$^{\#}$-convex. This statement includes Definition 5.3.5 (box-local minimal point). Since subgradient-convexity and L$^{\#}$-convexity are equivalent in the one-dimensional case, we also know that for one-dimensional subgradient-convex functions, box-local and global minimal points coincide. Graphical considerations quickly lead to the assumption that local and global minimal points also coincide for subgradient-convex functions. The question then, of course, is under which definition of neighborhood this applies. Examples 5.3.8 and 5.3.22 show that subgradient-convexity does not imply that box-local and global minimal points coincide. We now look at exactly what happens here and why we can conclude that there is a connection for the direction neighborhood.

Local and Global Minimality for Subgradient-Convex Functions

We need to return to convex extensibility (see Definition 5.3.12). According to Murota, a L$^{\#}$-convex (discrete) function is convex extensible on $\widehat{X}$. However, how to determine functions $\overline{f}$ remains open in the overview papers [45, 47]. In another more detailed work by Murota in [44], details in a highly mathematical framework with set functions are given. Fortunately, we do not need the exact function formula. Furthermore, it is unclear whether $\overline{f}$ is smooth. It seems reasonable to assume that this is not the case. However, smoothness is not a prerequisite for our considerations, so it is not important whether this applies. The subgradient ideas from the previous paragraph do not require smoothness assumptions for $\overline{f}$.

We briefly look at one running example of this section and consider possible extensions of the function $f_1(x) = \frac{1}{4}(x-4)^2$ with $X = [0, 8] \cap \mathbb{Z}$. A possible choice

Discrete Function f_1 with Convex Hull Extension

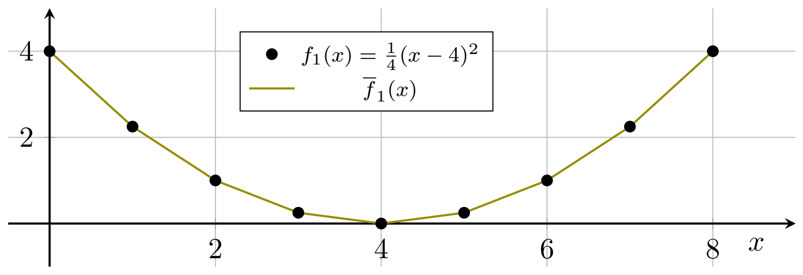

FIGURE 5.17: Discrete function f_1 and its convex hull extension $\overline{f}_1$

for $\overline{f}_1$ is obviously $f_1(x) = \frac{1}{4}(x-4)^2$ on $\widehat{X} = [0,8]$. This is through our construction of the discrete function from a continuous one which is clear and obvious. Another option is, of course, the piecewise linear function, which connects all image points linearly. This function is also convex, but not smooth. The piecewise linear function is a special convex extension, which we denote, similarly to [57], as a *convex hull extension* of f:

Definition 5.3.23. *Let $X \subseteq \mathbb{Z}^n$ be a discrete box with continuous relaxation $\widehat{X} \subseteq \mathbb{R}^n$. Let $f : X \to \mathbb{R}$ be a discrete function, and there exists a convex extension $\overline{f} : \widehat{X} \to \mathbb{R}$ of f. Then $\overline{f}$ is called a convex hull extension of f, if*

$$\overline{f}(x) \geq \tilde{f}(x), \quad x \in \widehat{X}$$

for all convex extensions $\tilde{f}$ of f holds.

Figure 5.17 shows the discrete function f_1 with its convex hull extension $\overline{f}_1$.

If a discrete function f is convex extensible in general, it is obvious that a convex hull extension also exists. For L$^\#$-convex functions, convex extensions exist, according to Murota. For subgradient-convex functions, we also have:

Theorem 5.3.24. *Let $X \subseteq \mathbb{Z}^n$ be a discrete box with continuous relaxation $\widehat{X} \subseteq \mathbb{R}^n$. Let $f : X \to \mathbb{R}$ be a subgradient-convex discrete function. Then there exists a convex hull extension $\overline{f} : \widehat{X} \to \mathbb{R}$.*

We give a short sketch of the proof with graphical arguments that are valid for an arbitrary dimension.

Proof. Let f be subgradient-convex. Next, we create the convex hull of all points $(x, f(x))$ with $x \in X$. This procedure is always possible, and the resulting set is a polytope. The only question is whether there are points $(\tilde{x}, f(\tilde{x}))$ that do not lie on a face of the convex hull. A point cannot lie outside of the polytope, since the polytope is the convex hull of the image points. If a point lies inside the polytope, f cannot be subgradient-convex. So, all $(x, f(x))$ lie on faces of the considered convex hull. By choosing suitable faces from the polytope, the convex hull extension can be built. □

However, determining a concrete functional description is not easy, but fortunately is not necessary for now. The existence of a convex hull extension implies that the global minimal points of f and the integer global minimal points of $\overline{f}$ coincide. Graphically in our examples, this statement is obvious. In general, we have:

Theorem 5.3.25. *Let $X \subseteq \mathbb{Z}^n$ be a discrete box with $|X| \geq 1$ and let $\widehat{X} \subseteq \mathbb{R}^n$ be its continuous relaxation. Let f be a discrete function with $f : X \to \mathbb{R}$ that is convex extensible with the convex hull extension $\overline{f} : \widehat{X} \to \mathbb{R}$. Then we have:*

$$\operatorname*{arg\,min}_{x\in X} f(x) = \left(\operatorname*{arg\,min}_{x\in \widehat{X}} \overline{f}(x)\right) \cap \mathbb{Z}^n.$$

Proof. Both $\operatorname*{arg\,min}_{x\in X} f(x)$ and $\left(\operatorname*{arg\,min}_{x\in \widehat{X}} \overline{f}(x)\right) \cap \mathbb{Z}^n$ are non-empty if we assume that X contains at least one integer point and if we assume that $\widehat{X}$ is the continuous relaxation of X.

We start with "$\supseteq$". Let $x^\star$ be a global minimal point of $\overline{f}$ that is integer, thus $x^\star \in \left(\left(\operatorname*{arg\,min}_{x\in \widehat{X}} \overline{f}(x)\right) \cap \mathbb{Z}^n\right)$. Then we have $x^\star \in X$ and through the construction of $\overline{f}$ (as the convex hull extension) directly $x^\star \in \operatorname*{arg\,min}_{x\in X} f(x)$.

For the other direction "$\subseteq$" we consider a point $x^\star \in \operatorname*{arg\,min}_{x\in X} f(x)$. Let $y \in \widehat{X}$ be an arbitrary point. We consider the graph of $\overline{f}$, which is defined by

$$\text{graph}\overline{f} = \left\{(x, \overline{f}(x)) \mid x \in \widehat{X}\right\}.$$

Through the construction of $\widehat{X}$ and the convex hull extension $\overline{f}$, it is clear that every point $(y, \overline{f}(y)) \in \text{graph}\overline{f}$ can be written as a convex combination of a finite number of points $(x^i, \overline{f}(x^i))$, $i = 1, \ldots, m$, with $x^i \in X$ (with $m \in \mathbb{N}$ and $m \leq |X|$). Since $\overline{f}$ is piecewise linear and X a finite set, each $(y, \overline{f}(y))$ lies on a face of graph$\overline{f}$ and the vertices of the face are exactly points $(x^i, \overline{f}(x^i))$ with $x^i \in X$. In particular, we have

$$(y, \overline{f}(y)) = \left(\sum_{i=1}^{m} \lambda_i x^i, \sum_{i=1}^{m} \lambda_i \overline{f}(x^i)\right)$$

with $\lambda \in \Lambda = \{\lambda \in \mathbb{R}^m \mid \lambda_i \geq 0,\ e^\top \lambda = 1\}$. An example for the faces and how the convex hull extension for a two-dimensional function can look like is shown in Figure 5.5. With that construction we have

$$\overline{f}(y) = \sum_{i=1}^{m} \lambda_i \overline{f}(x^i) \overset{(\star)}{=} \sum_{i=1}^{m} \lambda_i f(x^i) \overset{(\star\star)}{\geq} \sum_{i=1}^{m} \lambda_i f(x^\star) \overset{\lambda\in\Lambda}{=} f(x^\star) \overset{(\star)}{=} \overline{f}(x^\star),$$

where we use in $(\star)$ that $f(x) = \overline{f}(x)$ at the integer points holds and in $(\star\star)$ that $x^\star$ is the global minimal point of f on X. Since $y \in \widehat{X}$ is arbitrary chosen, we have $x^\star \in \left(\operatorname*{arg\,min}_{x\in \widehat{X}} \overline{f}(x)\right) \cap \mathbb{Z}^n$.

□

The global minimal points of $\overline{f}$ that are integer therefore coincide with the global minimal points of f. The question is now whether the statement of Theorem 5.3.25 also holds if we do not consider the whole sets X or $\widehat{X}$ but only a subset of them. The proof can be written down exactly as it appears above if a convex hull extension $\tilde{f}$ of f on a relaxation of a neighborhood is exactly the same as for the convex hull extension $\overline{f}$ of f on $\widehat{X}$. This is only the case for the direction neighborhood term. If we choose the box neighborhood this is not the case in general. To show this, we consider again the two-dimensional example.

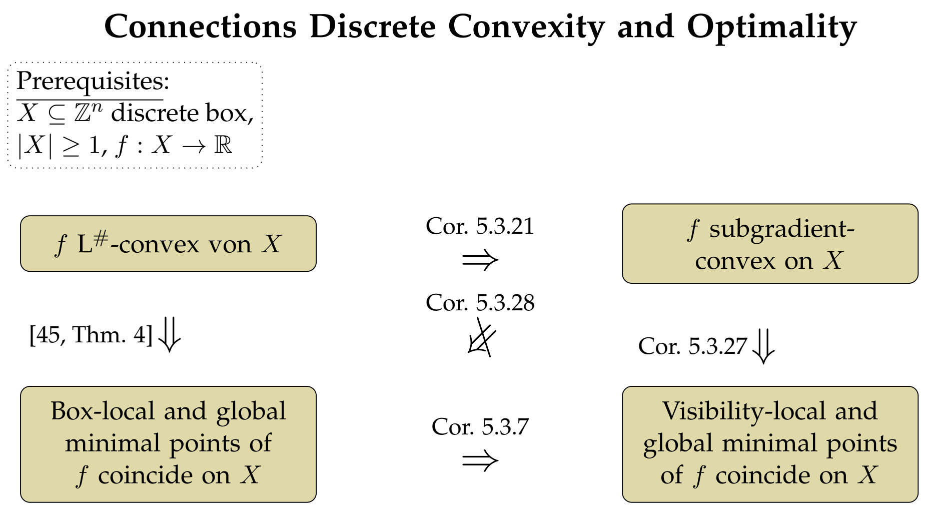
Connections Discrete Convexity and Optimality
Prerequisites:
$X \subseteq \mathbb{Z}^n$ discrete box,
$|X| \geq 1$, $f : X \to \mathbb{R}$
f L$^{\#}$-convex von X
Cor. 5.3.21 $\Rightarrow$
f subgradient-convex on X
[45, Thm. 4] $\Downarrow$
Cor. 5.3.28 $\not\swarrow$
Cor. 5.3.27 $\Downarrow$
Box-local and global minimal points of f coincide on X
Cor. 5.3.7 $\Rightarrow$
Visibility-local and global minimal points of f coincide on X

FIGURE 5.18: Connections for optimality and convexity for discrete functions

Example 5.3.26 (Sequel of Example 5.3.22). Let us consider the point $\widehat{x} = (2,1)$ with the box neighborhood $N^B(\widehat{x})$ and its relaxation $\widehat{N^B(\widehat{x})} = [1,3] \times [0,2]$. Let $\overline{f}^B$ be the convex hull extension on $\widehat{N^B(\widehat{x})}$. Let $\overline{f}$ be the convex hull extension on $\widehat{X}$, which is shown in Figure 5.5. Then we have in particular at $\tilde{x} = (3, 0.5)$ the function value $\overline{f}^B(\tilde{x}) = \frac{1}{2}(f((3,0)) + f((3,1))) = 86.5$. From Figure 5.5, we get $\overline{f}(\tilde{x}) = \frac{1}{2}(f(\widehat{x}) + f(x^\star)) = 65$ and so he have $\overline{f}(\tilde{x}) \neq \overline{f}^B(x)$. Through the definition of the direction neighborhood N^D, such a case cannot occur.

Thus, we have the following two corollaries:

Corollary 5.3.27. *Let f be subgradient-convex. Then all visibility-local minimal points are global minimal points of f.*

Corollary 5.3.28. *Let f be subgradient-convex. Then there can exist box-local minimal points that are not global minimal points.*

To conclude this Section, Figures 5.18 and 5.19 show the important results and connections between the convexity approaches.

Geometrically speaking, we have found and defined a concept of discrete convexity that fits the C^1-characterization of convexity. Under subgradient-convexity local and global minimal points coincide if we consider the "suitable" neighborhood definition. The subgradient-convexity is weaker than the L$^{\#}$-convexity of Murota, but is, we hope, easier to prove.

Next, we consider the Modular System Examples and check the convexity behavior of the corresponding discrete functions.

5.3.3 Convexity Considerations for the Modular System Examples

The function values of the discrete functions belong to the results of the decomposition approach (Algorithm 1 of Section 5.1). We remark at this point that the functions values of the discrete functions are obtained by solving a mixed-integer optimization model for each point in the domain of the function. The domain of

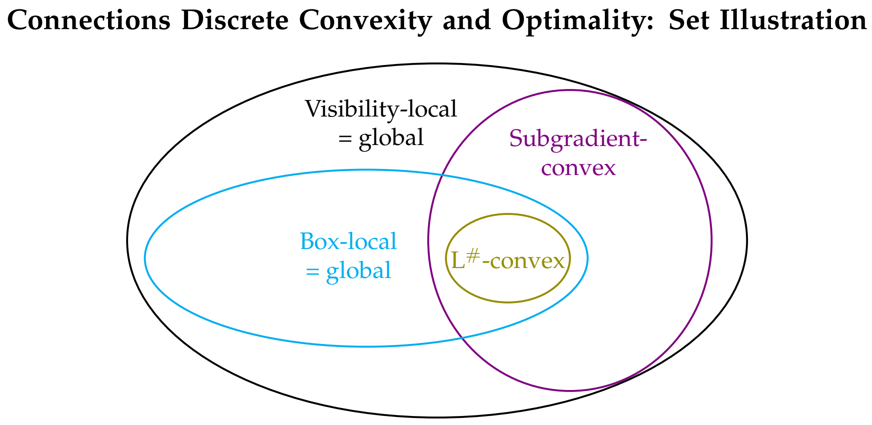
Connections Discrete Convexity and Optimality: Set Illustration
Visibility-local = global
Subgradient-convex
Box-local = global
L$^{\#}$-convex

FIGURE 5.19: Connections for optimality and convexity for discrete functions with sets

the discrete function corresponds to the possible combinations of fixed variants for each component of the modular system. The solving process for several problems could be very time consuming, especially for the large crane bridge problems.

In the examples of the higher-dimensional binpacking modular system, we found that these could not always be solved for a large number of fixed variants. Due to the constraint that only a maximum number of objects is allowed per bin, it is not always possible to find a feasible point for a correspondingly large number of required variants. In the example BP_DIM4_2 from Table A.28, for example, for the fixation on 5 variants for each component, i.e., $\widehat{k} = (5,5,5,5)$, the optimization problem $\overline{P}_{bin}^{\widehat{k}}$ is inconsistent. Strictly speaking, the related discrete function is therefore not defined on a discrete box but only on a subset of it. However, the terms and procedure for checking L$^{\#}$- or subgradient-convexity do not change, since the concrete structure of X is actually irrelevant. The following remark summarizes the last paragraph:

Remark 5.3.29. *The definition and results of discrete convexity in Subsection 5.3.2 can also be applied to discrete functions $f : X \to \mathbb{R}$ if X is only a subset of a discrete box $Y \in \mathbb{Z}^n$ and $n \geq 2$.*

The crucial point is that X has a finite number of elements, and this is still the case if the domain of the discrete function is only a subset of a discrete box. The reason we exclude $n = 1$ in Remark 5.3.29 is because Definition 5.3.13 is not exactly transferable. However, we do not have any issues with the one-dimensional examples anyway.

In Table 5.10 the results of the convexity analysis for the problems of the binpacking modular systems that we already considered are shown, in Table 5.11 for the crane bridge modular system, respectively. All but one problem are not L$^{\#}$-convex. Only the problem BP_DIM1_1 is L$^{\#}$-convex, which is already known since the problem is one-dimensional and therefore the discrete convexity from Definition 5.3.13 is relatively easy to prove with the graph of the function itself from Figure 5.1. For example BP_DIM1_7 it is easy to see with Figure 5.3 that the discrete function is neither L$^{\#}$- nor subgradient-convex. With a domain dimension greater than two, it is no longer possible to check discrete convexity simply by looking at the graph of the function, and an investigation using for example Python is necessary. We check L$^{\#}$-convexity by considering all possible point combinations from the constrained set X. Depending on the dimension, there are, of course, a lot of these. We check subgradient-convexity by solving the problems $P^{hyp}(x)$ for all $x \in X$. The number of problems to be solved is obviously given by the number of elements in X. In general, with higher dimension of the domain, the number of linear optimization problems to be solved grows slower than the number of combinations that we need to consider for the L$^{\#}$-convexity. Nevertheless, linear optimization problems must be solved in order to verify subgradient-convexity. According to Tables 5.10 and 5.11, checking L$^{\#}$-convexity is slightly faster up to dimension three, but similar to the subgradient-convexity from dimension four onwards. It seems reasonable to assume that it is easier in terms of time to check subgradient-convexity for higher dimensions or larger sets X. For these selected examples, box-local, visibility-local, and global minimal points are identical. This is surprising since almost all functions are not L$^{\#}$-convex. For the crane bridge problems, neither type of convexity is given, but also box-local, visibility-local and global minimal points are identical.

The tables also show runtimes, for which it is important to note that they do not include the time required to determine the optimal values. We have taken the results of Algorithm 1, i.e., in particular, the optimal values of the individual optimization problems $\overline{P^{gen}}(\widehat{k})$ for the corresponding fixation $\widehat{k}$ as a fixed input.

Altogether, it is certainly interesting that relatively simple binpacking problems lead to at least subgradient-convex discrete functions in almost all cases. In all binpacking examples we considered in the above part, visibility-local and global minimal points coincide, although not all problems are subgradient-convex. For the discrete functions from the crane bridge examples, no convexity concept can be shown. It can still be shown that there exist no local minimal points (in both concepts) which are not global. From a numerical point of view, this is at least a good result. However, the coincidence of visibility-local and global minimal points does not generally apply to modular system problems, as further analysis and modifications of example BP_DIM1_7 show. As can be seen in Table 5.10, the corresponding discrete function is neither L$^{\#}$-convex nor subgradient-convex. Due to the lack of subgradient-convexity and the fact that the example has a very simple structure and is also one-dimensional, we can use the example well to further explore structural issues.

Name	$L^{\#}$-convex	Runtime $L^{\#}$-convex	Subgradient--convex	Runtime Subgradient-convex	Box-local =Global	Visibility-local =Global
BP_DIM1_1	**True**	0.000	**True**	0.004	**True**	**True**
BP_DIM1_7	False	0.000	False	0.003	**True**	**True**
BP_DIM2_1	False	0.037	**True**	0.194	**True**	**True**
BP_DIM3_2	False	4.122	**True**	7.569	**True**	**True**
BP_DIM4_2	False	40.228	**True**	39.688	**True**	**True**

TABLE 5.10: Convexity analysis for the discrete functions of the modular system for the binpacking problems

Name	$L^{\#}$-convex	Runtime $L^{\#}$-convex	Subgradient--convex	Runtime Subgradient-convex	Box-local =Global	Visbility-local =Global
CRANE_N5_4	False	0.002	False	0.043	**True**	**True**
CRANE_N20_1	False	0.011	False	0.068	**True**	**True**

TABLE 5.11: Convexity analysis for the discrete functions of the modular system for the crane bridge problems

Further Analysis of Example BP_DIM1_7

The corresponding discrete function for example BP_DIM1_7, that we denoted by v, has two global minimal points, see Figure 5.3. As already mentioned, v is not subgradient-convex, since we cannot find a supporting hyperplane in the point $(4, v(4))$.

This example clearly illustrates the role that the cost factors of the objective function play and how they can affect the solution to the modular system problem. In some applications, cost factors cannot be adjusted, as they are determined externally, for example, by market prices. At this point, we assume that we can determine and adjust the cost factors ourselves. By choosing suitable manufacturers for certain materials, we may be able to influence cost factors.

As example BP_DIM1_7 is one-dimensional, v is a discrete function with $v : X \to \mathbb{R}$ with the discrete box $X = [1, \overline{n}] \cap \mathbb{Z}$ and the function values are the optimal values of the optimization problem $\overline{P}^1_{bin}(\widehat{k})$ (from Subsection 5.2.2) for all $\widehat{k} \in X$. Since the objective function of the problem $\overline{P}^1_{bin}(\widehat{k})$ is considered here again, we specify it once more. It is given by

$$c_v\widehat{k} + c_d \sum_{\ell=1}^{N} \left(L^\ell - \sum_{i=1}^{\overline{n}} x_i z_{i,\ell} \right),$$

where the x_i and $z_{i,\ell}$ enter as variables.

In all of our modular systems and the associated optimization problems from the decomposition approach (Algorithm 1), the costs for the variants, in particular $c_v\widehat{k}$, are included as a constant, since we fix $\widehat{k}$ in each iteration. This part of the objective function in $\overline{P}^1_{bin}(\widehat{k})$, or in the general problem $\overline{P^{gen}}(\widehat{k})$, does not need to be considered in the optimization process. It is sufficient if these costs are added in the end.

If we consider the second part of the objective function, it is clear that the optimal point of the optimization problem is independent of the choice of c_d, since c_d only enters as a positive factor. For the following tests, we define the ratio r for the cost factors as follows:

$$r = \frac{c_d}{c_v}.$$

If we change the ratio r, the optimal points of the problems remain the same and therefore the deviations in the bins remain identical. However, the total costs do change, of course, as they depend on the cost factors.

In the following tests we slowly increase r from 0.4 to 1.25, by reducing c_v from 25 to 8 while keeping c_a constant at $c_a = 10$. We use example BP_DIM1_7 as a reference for now with the cost ratio $r = \frac{2}{3}$. In the examples BP_DIM1_8, BP_DIM1_9 and BP_DIM1_10 we increase the variant costs, such that r becomes smaller than $\frac{2}{3}$. In examples BP_DIM1_11, BP_DIM1_12 and BP_DIM1_13, we reduce the variant costs based on example BP_DIM1_7 such that r becomes larger than $\frac{2}{3}$. This leads to an increasing cost ratio r. Variant costs, the different ratios and the corresponding color are given in Table 5.12 and the discrete functions are illustrated with the same colors in Figure 5.20.

Increasing c_v leads to higher variant costs, and this leads to a rotation of the graph of v counterclockwise, which we can see later in a visualization. For one

Name	Variant Costs c_v	Cost Ratio r	Color
BP_DIM1_8	17	≈ 0.6	▨
BP_DIM1_9	20	0.5	▨
BP_DIM1_10	25	0.4	▨
BP_DIM1_11	14	≈ 0.7	▨
BP_DIM1_12	10	1	▨
BP_DIM1_13	8	1.25	▨

TABLE 5.12: Overview variant costs and cost ratio for examples BP_DIM1_7 - BP_DIM1_13

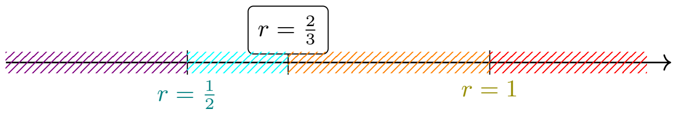

FIGURE 5.20: Cost ratios for examples BP_DIM1_7-BP_DIM1_13

variant, the total cost increases by the cost factor c_v for two variants by $2 \cdot c_v$ and so on. We remark at this point again that the deviation costs do not change if we only modify c_v. When we reduce the variant costs, the function rotates clockwise accordingly. The discrete functions of the examples mentioned here, are shown in Figure 5.21.

The colors of the data points match those of Figure 5.20. The black points “in the middle” correspond to example BP_DIM1_7. It is the same plot as in Figure 5.3. If we now reduce r, the discrete functions rotate through the higher variant costs counterclockwise, and we get the violet points (example BP_DIM1_10). If we increase r we end up with example BP_DIM1_13 and the red points. This is a rotation from the black function clockwise.

It is interesting to note that the geometric structure of the function remains unchanged. It is only rotated and shifted. The kink structure around the global minimal points is retained but rotated. The global minimal points of the discrete functions are again denoted by an additional circle. We recognize that for the examples with $r < \frac{2}{3}$ the global minimal point of the discrete function is unique and at $k^\star = 3$. With $r = \frac{2}{3}$ (BP_DIM1_7) we have the two global minimal points $k^\star = 3$ and $k^\star = 5$. For $r > \frac{2}{3}$ the global minimal point is again unique and at $k^\star = 5$.

It is also noticeable that none of the discrete functions is subgradient-convex since the kink structure remains even when rotated. For all functions, no supporting hyperplane can be found at $(4, v(4))$. According to Figure 5.18 and Theorem 5.3.18, the functions are therefore also not L$^\#$-convex. A change in the cost factors therefore does not affect the structure of the discrete function, which is an interesting fact at this point.

This example shows us not only that changing cost factors can change global minimal points but also that global minimal points can become local minimal points and vise versa. Since we have a one-dimensional example, box-local and visibility-local minimal points are the same. So we only speak of local minimal points in this part of the work. Let us return to the example BP_DIM1_7 in black. Here, there are two global minimal points $k = 3$ and $k = 5$. If r becomes smaller, the global minimal point $k = 5$ first becomes a local minimal point (BP_DIM1_8 and BP_DIM1_9,

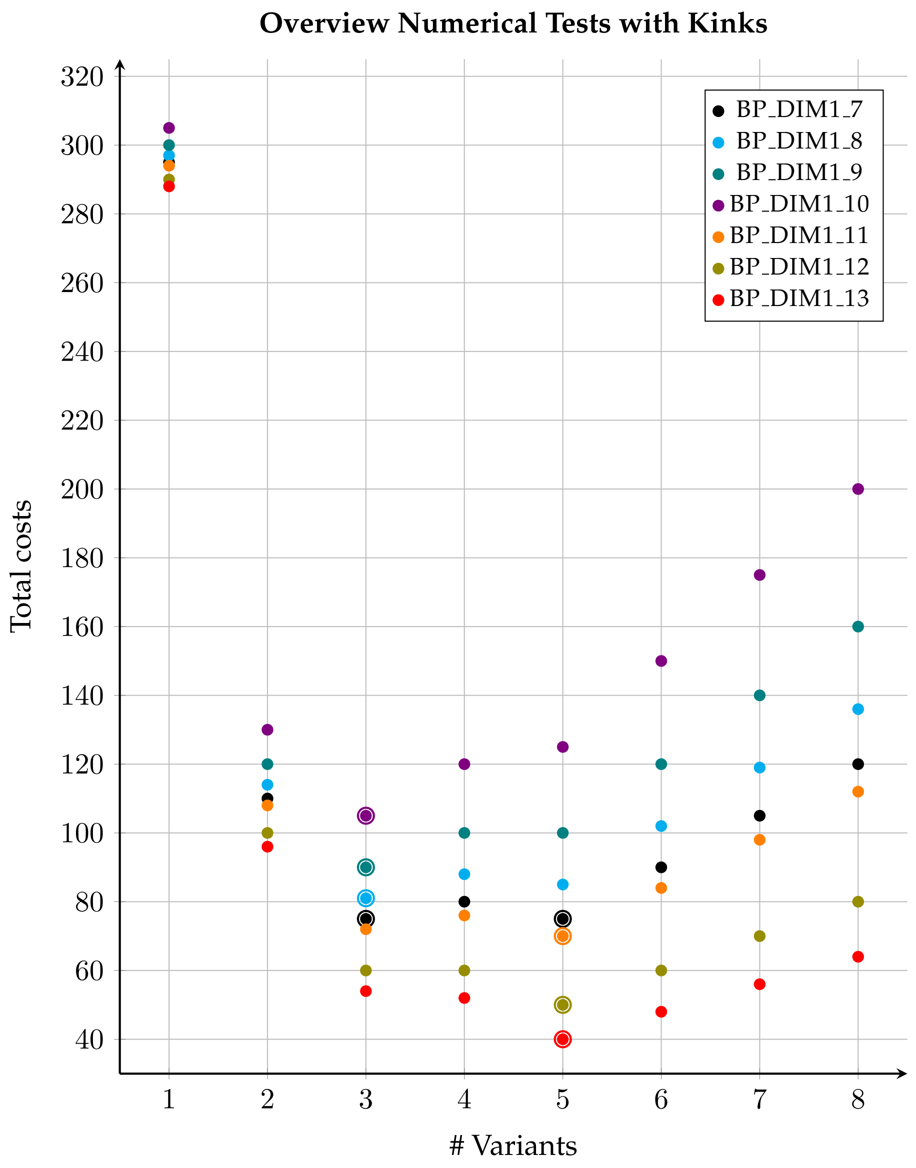

FIGURE 5.21: Total costs for examples BP_DIM1_7-BP_DIM1_13

blue and teal). For example BP_DIM1_9, although v has the same function value for $k = 4$ and $k = 5$, $k = 5$ is still local minimal because $v(5) \leq v(6)$ and $v(5) = v(4)$. For $r < 0.5$ (e.g., example BP_DIM1_10), $k = 5$ is no longer a local minimal point, and v only has one global minimal point at $k = 3$. A similar phenomenon occurs when we increase r based on example BP_DIM1_7. First, the global minimal point $k = 3$ becomes a local one (BP_DIM1_11 and BP_DIM1_12, orange and olive), then from $r > 1$ onwards, only the global minimal point $k = 5$ remains (BP_DIM1_13, red). The color marking in Figure 5.20, for example, red in the range $r > 1$, is intended to show that all problems with a cost ratio $r > 1$ behave like example BP_DIM1_13, i.e., that the only global minimal point is the point $k = 5$.

At the beginning of our analysis of the convexity of binpacking problems, we stumbled upon the input data from example BP_DIM1_7 by chance. The subsequent analytical consideration and minor modification of the input data is based on geometric and logical ideas derived from the structure of the problem. However, it is noticeable that this example is the only one that is not subgradient-convex. Since the number of bin packing problems we have considered so far is small, we randomly generate additional input data for binpacking problems of dimensions one to four.

More Binpacking Problems

We created 15 examples with different numbers of bins (up to 15) and different number elements per bin and different variant differences. The concrete input data can be found in [21]. As in the examples BP_DIM3_2 and BP_DIM3_2, inconsistent problems arise when the number of variants is too large, but the domain of the discrete function remains finite.

In the following, we execute Algorithm 1 for the new input data and check the discrete function for $L^{\#}$- and subgradient-convexity and compare the box-local (visibility-local) and global minimal points. In Table 5.13 the results of the analysis are given for the two-dimensional random binpacking examples. There are no $L^{\#}$-convex discrete functions, but all except one problem are subgradient-convex. The runtimes are very low for both verifications. It is noticeable that visibility-local and global minimal points coincide for all problems, even for the non subgradient-convex function. The data of Example 5.3.8 that we consider in Subsection 5.3.1 correspond to example BP_RD_DIM2_4. In Example 5.3.8 a slightly smaller domain X is considered. The results corresponding to the local and global minimal points do not change significantly if we now consider higher dimensions.

In the three-dimensional examples, see Table 5.14, most of the examples are also subgradient-convex. None of the discrete functions are $L^{\#}$-convex. And again, visibility-local and global minimal points are identical for all problems. For the four-dimensional examples, basically nothing changes, see Table 5.15. The only thing that is noticeable is that, for some problems, the time required to check $L^{\#}$-convexity is very high. In these cases, checking subgradient-convexity is significantly less time-consuming. We expected this phenomenon because the number of point-by-point comparisons becomes very large for high dimensions and large maximum numbers of variants. The number of linear optimization problems (LPs) to be solved for verifying subgradient-convexity does not grow as rapidly, and LPs can still be solved efficiently and quickly.

Name	$L^{\#}$-convex	Runtime $L^{\#}$-convex	Subgradient- -convex	Runtime Subgradient-convex	Box-local =Global	Visibility-local =Global
BP_RD_DIM2_1	False	0.003	**True**	0.046	**True**	**True**
BP_RD_DIM2_2	False	0.002	**True**	0.019	**True**	**True**
BP_RD_DIM2_3	False	0.052	**True**	0.246	**True**	**True**
BP_RD_DIM2_4	False	0.009	**True**	0.057	False	**True**
BP_RD_DIM2_5	False	0.020	**True**	0.111	**True**	**True**
BP_RD_DIM2_6	False	0.012	**True**	0.073	**True**	**True**
BP_RD_DIM2_7	False	0.051	False	0.240	False	**True**
BP_RD_DIM2_8	False	0.020	**True**	0.110	False	**True**
BP_RD_DIM2_9	False	0.034	**True**	0.170	**True**	**True**
BP_RD_DIM2_10	False	0.010	**True**	0.061	**True**	**True**
BP_RD_DIM2_11	False	0.003	**True**	0.026	**True**	**True**
BP_RD_DIM2_12	False	0.034	**True**	0.169	**True**	**True**
BP_RD_DIM2_13	False	0.028	**True**	0.146	False	**True**
BP_RD_DIM2_14	False	0.002	**True**	0.014	**True**	**True**
BP_RD_DIM2_15	False	0.003	**True**	0.023	**True**	**True**

TABLE 5.13: Convexity analysis for the discrete functions of the modular system for randomly generated two-dimensional binpacking problems

Name	$L^{\#}$-convex	Runtime $L^{\#}$-convex	Subgradient- -convex	Runtime Subgradient-convex	Box-local =Global	Visibility-local =Global
BP_RD_DIM3_1	False	6.440	**True**	10.189	**True**	**True**
BP_RD_DIM3_2	False	4.889	**True**	8.200	**True**	**True**
BP_RD_DIM3_3	False	0.118	**True**	0.454	**True**	**True**
BP_RD_DIM3_4	False	0.222	**True**	0.766	**True**	**True**
BP_RD_DIM3_5	False	7.396	False	11.184	False	**True**
BP_RD_DIM3_6	False	0.512	False	1.455	**True**	**True**
BP_RD_DIM3_7	False	7.836	**True**	11.410	**True**	**True**
BP_RD_DIM3_8	False	1.553	**True**	3.552	**True**	**True**
BP_RD_DIM3_9	False	0.015	**True**	0.083	**True**	**True**
BP_RD_DIM3_10	False	0.018	**True**	0.103	**True**	**True**
BP_RD_DIM3_11	False	0.334	**True**	1.061	False	**True**
BP_RD_DIM3_12	False	2.699	**True**	5.242	False	**True**
BP_RD_DIM3_13	False	3.672	**True**	6.735	**True**	**True**
BP_RD_DIM3_14	False	0.156	**True**	0.603	**True**	**True**
BP_RD_DIM3_15	False	7.758	**True**	11.435	**True**	**True**

TABLE 5.14: Convexity analysis for the discrete functions of the modular system for randomly generated three-dimensional binpacking problems

Name	$L^{\#}$-convex	Runtime $L^{\#}$-convex	Subgradient- -convex	Runtime Subgradient-convex	Box-local =Global	Visibility-local =Global
BP_RD_DIM4_1	False	41.295	**True**	40.714	**True**	**True**
BP_RD_DIM4_2	False	4.630	**True**	8.185	**True**	**True**
BP_RD_DIM4_3	False	1.817	**True**	4.146	**True**	**True**
BP_RD_DIM4_4	False	1828.971	False	557.695	False	**True**
BP_RD_DIM4_5	False	7.729	**True**	12.182	**True**	**True**
BP_RD_DIM4_6	False	8.600	**True**	12.386	False	**True**
BP_RD_DIM4_7	False	1300.190	**True**	446.213	False	**True**
BP_RD_DIM4_8	False	20.937	**True**	24.914	False	**True**
BP_RD_DIM4_9	False	1383.346	False	453.949	**True**	**True**
BP_RD_DIM4_10	False	245.058	**True**	144.738	**True**	**True**
BP_RD_DIM4_11	False	694.275	False	278.335	**True**	**True**
BP_RD_DIM4_12	False	426.489	**True**	210.769	**True**	**True**
BP_RD_DIM4_13	False	7.881	**True**	11.820	**True**	**True**
BP_RD_DIM4_14	False	8.657	**True**	12.817	**True**	**True**
BP_RD_DIM4_15	False	23.073	**True**	25.204	**True**	**True**

TABLE 5.15: Convexity analysis for the discrete functions of the modular system for randomly generated four-dimensional binpacking problems

After a detailed consideration of the discrete functions that belong to the modular system problems, we discuss the specific applicability of the convexity concepts for the modular system problems in more detail.

Critical View of the Convexity Concept for the Modular System Problems

As already mentioned above, we obtained the function values of the discrete functions considered in the examples in this subsection after applying Algorithm 1. We only checked the $\mathrm{L}^{\#}$- and subgradient-convexity once all the function values were known. This is, of course, very time-consuming and, above all, once all data are available, the minimal point can be determined directly. In principle, however, we do not need all function values if we can prove in another way that the discrete function is at least subgradient-convex. In this case, we can stop during an algorithm as soon as a local minimal point (in a suitable neighborhood) is reached. This local minimal point is then always also a global minimal point. If we cannot check convexity beforehand, there is still a high chance that a local minimal point is a global minimal point. As long as we choose at least subgradient-convexity and the direction neighborhood, the chance is very high, as we can see from the examples discussed above.

The function values that we determine during Algorithm 1 are the corresponding optimal values of an optimization problem, i.e., the discrete function corresponds to an optimal value function of a parametric optimization problem. However, it is completely unclear at this point how and whether this optimal value function can be specified concretely with an exact formulation of the function. For continuous parametric optimization problems, it is sometimes possible to find a functional description for the optimal value function for specific feasible sets, see [58]. In general, the problems that occur in [58] have a continuous parameter range and continuous variables that do not fit in our context. In the literature, mixed integer parametric optimization problems are mentioned in some places, for example, in the preprint [50]. Here, the parameter of the problem is included as the right-hand side of a MILP, which does not fit into our context either. In our examples, the variables that we fix (v) also occur in the objective functions, and the problems are not necessarily linear. In the binpacking example, we have, for example, products of ξ and z, even if we fix v.

We did not find any literature specifically for our problem during the process of our work. The question of whether the optimal value function can be determined and ideally further investigated remains open in this work. However, since the examples are promising and local minima are usually also global, in the next chapter we look at methods for determining local minima of discrete functions. However, the crucial difference at this point is that we do not determine all function values in advance but only when we need them.

In the next Subsection, we consider more discrete functions that become interesting in Chapter 6. These are independent of the modular system problems, but will later be used to test numerical methods. We introduce the other discrete functions at this point and examine them for $\mathrm{L}^{\#}$- and subgradient-convexity.

5.3.4 Discrete Test Functions

In the context of operations research, there are several test libraries with optimization problems, classified by their type, linear or nonlinear, or with or without integer variables. In most cases, also the objective function with gradient and/or hessian as explicit function is given. Normally, the test problems from the libraries are used to test solvers developed by the user, as the local and/or global minimal points are already known for many problems.

Here and in the next chapter, we need discrete functions for which we have not found a specially created library. Since we want to minimize the functions and are interested in local and global minimal points, we choose a library as described above. The objective functions of the test problems are usually continuous here, but we can easily build a discrete function on top of them by evaluating the continuous function only at the discrete points. We determine the domains of the selected functions by ourselves. In the following, we consider the test library "CUTEst" and its Python package "PyCUTEst".

CUTEst and PyCUTEst

A large and well-known function library is CUTEst, which is presented in [16]. The CUTEst library from 2015 is a library with more than 1,500 test problems, based on the original library CUTE from 1995 and an expansion of it, CUTEr, from 2003 (see [53]). Many problems can be parameterized to allow suitable different dimensions of the same problem. The Python package PyCUTEst gives the user access to the CUTEst collection of test problems. A detailed documentation of the package can be found on the website [53].

In general, the PyCUTEst package does not provide specific function specifications for the functions contained in the library. For this reason, these are not specified here in detail. However, some functions, such as the Rosenbrock function, later labeled with "ROSENBR",

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,$$

which is first mentioned in the paper [55], are generally known. However, with the help of the PyCUTEst package, we can generate the function values for the points in the domain. In Table 5.16 the considered test problems from the PyCUTEst test library are listed.

We consider functions of dimensions two, three, and four. We again have as the domain of the discrete function a discrete box $X \subseteq \mathbb{Z}^n$ with its continuous relaxation $\widehat{X}$. The sets $\widehat{X}$ are also shown in the table. In the last column, all box-local minimal points are given, and the ones that are global are additionally printed in bold. We remark at this point that the box-local and global minimal points are those for the discrete version of the functions. The Rosenbrock function in the continuous case has, for example, only the global minimal point $(1, 1)$ and no local minima. The discrete version of the function that we consider here and in the following has the two box-local minima $(-2, 4)$ and $(2, 4)$ in addition to the global minimal point $(1, 1)$.

All functions, which have box-local minima that are not global, cannot be L$^\#$- or subgradient-convex, see Figure 5.18 or the corresponding theorems. However, we check L$^\#$-convexity by definition to get an idea of the runtimes. In addition, we

Name	Dim	$\widehat{X}$	Box-local/**global** minimal points
DENSCHNC	2	$[-5,5]^2$	$(-1,1)$, $\mathbf{(1,1)}$
ELATVIDU	2	$[-5,5]^2$	$(-3,-2)$, $(-2,1)$, $(2,1)$, $\mathbf{(3,-2)}$
ROSENBR	2	$[-5,5]^2$	$(-2,4)$, $\mathbf{(1,1)}$, $(2,4)$
WAYSEA	2	$[-5,5]^2$	$(1,-2)$, $\mathbf{(1,2)}$
BOX3	3	$[-5,5]^3$	$\mathbf{(-5,-5,0)}$, $\mathbf{(-4,-4,0)}$, $\mathbf{(-3,-3,0)}$, $\mathbf{(-2,-2,0)}$, $\mathbf{(-1,-1,0)}$, $\mathbf{(0,0,0)}$, $\mathbf{(1,1,0)}$, $(1,5,1)$, $\mathbf{(2,2,0)}$, $\mathbf{(3,3,0)}$, $\mathbf{(4,4,0)}$, $(5,1,-1)$, $\mathbf{(5,5,0)}$
DENSCHND	3	$[-5,5]^3$	$(-4,0,-2)$, $(-4,0,2)$, $\mathbf{(0,0,0)}$, $(4,0,-2)$, $(4,0,2)$
HAFTLDE	3	$[-5,5]^3$	$(0,2,-5)$, $(0,3,-5)$, $(0,4,-5)$, $(0,5,-5)$, $(1,2,2)$, $(1,4,4)$, $(2,-1,-5)$, $\mathbf{(3,-1,-1)}$, $(4,-2,-3)$
HELIX	3	$[-5,5]^3$	$(-1,0,5)$, $\mathbf{(1,0,0)}$
SCHMVETT	3	$[-5,5]^3$	$(-5,-5,-5)$, $(-4,-4,3)$, $(-2,-2,-2)$, $(2,2,-3)$, $\mathbf{(4,4,3)}$, $(5,2,-2)$
HILBERTA	4	$[-3,3]^4$	$(0,-1,3,-2)$, $\mathbf{(0,0,0,0)}$, $(0,1,-3,2)$
HIMMELBF	4	$[-3,3]^4$	$\mathbf{(-3,-3,-3,0)}$, $\mathbf{(-3,-3,3,0)}$, $\mathbf{(-3,3,-3,0)}$, $\mathbf{(-3,3,3,0)}$, $\mathbf{(3,-3,-3,0)}$, $\mathbf{(3,-3,3,0)}$, $\mathbf{(3,3,-3,0)}$, $\mathbf{(3,3,3,0)}$
PENALTY1	4	$[-3,3]^4$	$\mathbf{(0,0,0,0)}$
PENALTY2	4	$[-3,3]^4$	$(0,0,0,-1)$, $\mathbf{(0,0,0,1)}$
POWELLSG	4	$[-3,3]^4$	$\mathbf{(0,0,0,0)}$
STREG	4	$[-3,3]^4$	$\mathbf{(1,1,0,0)}$

TABLE 5.16: Discrete test problems based on the PyCUTEst library; with dimension, considered domain $X = \widehat{X} \cap \mathbb{Z}^n$ and box-local and global (in bold) optimal points

consider again as in the previous subsection subgradient-convexity and the direction locality approach. The results can be found in Table 5.17. Neither L$^\#$-convexity nor subgradient-convexity can be proven for the two- and three-dimensional test functions. This is fundamentally good for us, as we also want to test the behavior of numerical methods, which we consider in Chapter 6, when no convexity is given and almost all of the functions considered before were subgradient-convex. For the four-dimensional test functions, some instances are subgradient-convex. Here, we also see that when the domains of the discrete functions become larger, verifying subgradient-convexity is possible in less time for the four-dimensional functions. The runtime for checking subgradient-convexity is almost half as long as the one required for checking L$^\#$-convexity.

In addition to the discrete functions based on the CUTEst library, we consider two more functions for which we know the exact description of the function. These two functions are also separable.

Separable Test Functions

We consider two discrete functions f_3^n and f_4^n of arbitrary dimension n for the moment with discrete boxes $X_3 = [0,6]^n \cap Z^n$ and $X_4 = [0,8]^n \cap \mathbb{Z}^n$. $\widehat{X}_3$ and $\widehat{X}_4$ are the continuous relaxations of X_3 and X_4. We have the discrete functions $f_3^n : X_3 \to \mathbb{R}$, $f_4^n : X_4 \to \mathbb{R}$ and the continuous ones $\overline{f}_3^n : \widehat{X}_3 \to \mathbb{R}$, $\overline{f}_4^n : \widehat{X}_4 \to \mathbb{R}$ defined through

$$f_3^n(x) = \sum_{i=1}^{n} (x_i - 3)^2, \quad \overline{f}_3^n(x) = \sum_{i=1}^{n} (x_i - 3)^2$$

and

$$f_4^n(x) = \sum_{i=1}^{n} (x_i - 3)^2 \left((x_i - 5)^2 + 0.01\right), \quad \overline{f}_4^n(x) = \sum_{i=1}^{n} (x_i - 3)^2 \left((x_i - 5)^2 + 0.01\right).$$

Since we want to vary the dimension of the domain, we include n as a parameter in the function name. The function $\overline{f}_3^n$ is obviously convex on $\widehat{X}_3$. The function $\overline{f}_4^n$ is not convex on $\widehat{X}_4$ in general. It is easy to verify that $x^\star = (3, \dots, 3)$ is a global minimal point for $\overline{f}_3^n$ and $\overline{f}_4^n$ (on the corresponding sets $\widehat{X}_3/\widehat{X}_4$) with the optimal value zero. Thus, this point is also global minimal on X_3/X_4 for f_3^n and f_4^n. The function f_4^n is constructed so that, in addition to the global minimal point, it also has box-local minimal points on X_4. All points with entries three or five in the corresponding coordinates are box-local minimal points. For completeness, we summarize this in Table 5.18.

Since we can prove convexity for the function $\overline{f}_3^n$ on $\widehat{X}_3$, we can conclude subgradient-convexity for f_3^n on X_3. The L$^\#$-convexity can also be easily proven for f_3^n, since the function is separable and the particular functions are parabolas. The function f_4^n is neither L$^\#$-convex nor subgradient-convex. The verification is very time consuming. All results and runtimes for the separable test functions can be found in Table 5.19.

Name	$L^{\#}$-convex	Runtime $L^{\#}$-convex	Subgradient- -convex	Runtime Subgradient-convex	Box-local =Global	Visibility-local =Global
DENSCHNC	False	0.081	False	0.376	False	False
ELATVIDU	False	0.081	False	0.354	False	False
ROSENBR	False	0.082	False	0.354	False	**True**
WAYSEA1	False	0.081	False	0.355	False	**True**
BOX3	False	45.209	False	43.565	False	**True**
DENSCHND	False	45.036	False	43.468	False	**True**
HATFLDE	False	45.278	False	43.305	False	**True**
HELIX	False	45.290	False	43.884	False	**True**
SCHMVETT	False	45.250	False	43.119	False	**True**
HILBERTA	False	260.244	**True**	145.268	False	**True**
HIMMELBF	False	262.601	False	145.718	**True**	**True**
PENALTY1	False	260.113	**True**	145.162	**True**	**True**
PENALTY2	False	259.912	False	145.581	False	False
POWELLSG	False	259.702	**True**	146.108	**True**	**True**
STREG	False	259.855	False	146.210	**True**	**True**

TABLE 5.17: Convexity analysis for the discrete test functions from the PyCUTEst library

Name	Dim	$\widehat{X}$	Box-local/**global** minimal points
f_3^2	2	$[0,6]^2$	$\mathbf{(3,3)}$
f_3^3	2	$[0,6]^3$	$\mathbf{(3,3,3)}$
f_3^4	2	$[0,6]^4$	$\mathbf{(3,3,3,3)}$
f_4^2	2	$[0,8]^2$	$\mathbf{(3,3)}, (3,5), (5,3), (5,5)$
f_4^3	3	$[0,8]^3$	$\mathbf{(3,3,3)}, (3,3,5), (3,5,3), (3,5,5), (5,3,3),$ $(5,3,5), (5,5,3), (5,5,5)$
f_4^4	4	$[0,8]^4$	$\mathbf{(3,3,3,3)}, (3,3,3,5), (3,3,5,3), (3,3,5,5),$ $(3,5,3,3), (3,5,3,5), (3,5,5,3), (3,5,5,5),$ $(5,3,3,3), (5,3,3,5), (5,3,5,3), (5,3,5,5),$ $(5,5,3,3), (5,5,3,5), (5,5,5,3), (5,5,5,5)$

TABLE 5.18: Discrete test problems for the functions f_3^n and f_4^n; with dimension, domain $X = \widehat{X} \cap \mathbb{Z}^n$ and box-local and global (in bold) optimal points

Name	L$^\#$-convex	Runtime L$^\#$-convex	Subgradient--convex	Runtime Subgradient-convex	Box-local =Global	Visibility-local =Global
f_3^2	**True**	0.010	**True**	0.088	**True**	**True**
f_3^3	**True**	1.127	**True**	2.870	**True**	**True**
f_3^4	**True**	259.212	**True**	147.025	**True**	**True**
f_4^2	False	0.032	False	0.188	False	False
f_4^3	False	8.481	False	12.457	False	False
f_4^4	False	4835.829	False	1070.985	False	False

TABLE 5.19: Convexity analysis for the discrete separable test functions

5.4 Conclusions

Altogether we presented in this chapter with Algorithm 1 in Section 5.1 an alternative solution approach to solving the optimization problem $\overline{P^{gen}}$ of Chapter 4. The approach presented there is based on a decomposition approach. For some examples, the approach proved to be very time-consuming. In the next step, we could establish a connection to discrete functions for this framework. The discrete functions were introduced in Section 5.3 and we considered convexity concepts in more detail in Subsection 5.3.2. Some of the concepts are known from literature, but we also introduced with the subgradient-convexity a new concept that is based on geometrical considerations. In subsequent numerical investigations in Subsection 5.3.3, we determined that many of the functions belonging to the modular system problems do not satisfy any discrete convexity concepts but often do not have any minimal points that are only local and not global.

For the modular system problems considered, all required optimal values were first determined by solving the corresponding optimization problem. Strictly speaking, the optimal value function was always determined first (this is the discrete function), and then convexity was considered. It remains a desirable goal to investigate convexity directly without specifying the optimal value function.

For numerical testing purposes for this and the following chapter, we introduced some new discrete test functions at the end of this chapter in Subsection 5.3.4. These are not related to modular systems, but will prove helpful.

In the next chapter, we now discuss how discrete functions can be efficiently minimized since many examples were time-consuming in Chapter 5. Since we consider again discrete functions, the decomposition idea remains the same, but the minimization approaches will differ. Instead of calculating all function values for the discrete function, we will proceed differently. Thus, we denote the concepts in the following chapter by “extended decomposition approaches”.

Chapter 6

Extended Decomposition Approaches

In Chapter 4, we first derived an optimization problem to determine an optimal configuration for a modular system problem. This optimization problem is generally very difficult to solve because many variables occur and many combinatorial structures need to be modeled. In this context, binary decision variables occur that are part of the combinatorial problem. In operations research, decomposition methods, in which the complicating variables are fixed and the resulting subproblem is solved, are a classic approach for an alternative solution technique.

We presented this approach in Chapter 5 in Subsection 5.1. There, we derived a numerical method based on the decomposition approach. Since there is only a finite number of complicating binary variables, in theory only a finite number of subproblems need to be solved. The subproblems are generally easier to solve, but as soon as the problem dimension increases in binpacking problems, and, in general, in the crane bridge problems, a large amount of computing time is still required.

We remarked that we can also view the presented algorithm from a different perspective. Determining an optimal configuration for a modular system problem corresponds to minimizing a discrete function. Discrete functions and their properties were presented in Section 5.3. The domain of the discrete function corresponds to the fixation of the binary variables. In the application, this is a concrete fixation of a certain number of variants in which a component is available in the modular system. The corresponding function value of the discrete function then corresponds to the total cost of the modular system for this particular fixation. Thus, the discrete function is an optimal value function that we can only evaluate on a part of a discrete grid.

In order to determine the global minimal point of the discrete function, we can determine all function values (if possible) and search for the lowest one. This is very time-consuming since all function values need to be calculated in advance. This procedure corresponds exactly to the decomposition approach. Since in continuous optimization the calculation of local minimal points is sufficient (and less time-consuming) if a function is convex, we introduced convexity concepts for the discrete functions introduced in Subsection 5.3.2. Then, in Subsection 5.3.3, it was shown that many of the discrete functions belonging to the modular systems are at least subgradient-convex. In continuous optimization, local search methods are often used for unconstrained convex optimization problems, since local minimal points are also global minimal points. We also obtained the correspondence between visibility-local and global minimal points for subgradient-convex discrete functions (Corollary 5.3.27). However, the neighborhood around the local minimal point must be chosen in a suitable way. The suitable neighborhood is the so-called direction neighborhood there. Box-local minimal points are on the other hand not necessarily global minimal points if f is subgradient convex (Corollary 5.3.28). This is only the case if f is L$^{\#}$-convex. However, if we consider the direction neighborhood, we have the problem that there are not, as hoped, few points in the neighborhood, but rather almost the total discrete box. The question now arises as to whether and how these results and determinations can be used, preferably efficiently, to minimize discrete functions.

In general, very few references to discrete functions and convexity can be found in the literature. A brief overview is given in the previous chapter. With regard to the minimization of discrete convex functions, the situation is no better. As mentioned in the previous chapter, concepts for discrete functions with domain $\mathbb{Z}^n$ are introduced. In our application, we consider domains that correspond to a box. Therefore we continue to search for efficient methods for minimizing discrete functions on a discrete box. Since the basic idea is still based on the concept of decomposition (i.e., we fix binary variables), in this chapter we refer to extended decomposition approaches.

Instead of this, in Section 6.1 we present a first approach known as the Steepest Descent method. In the next section (Section 6.2) we present two additional methods, the Coordinate Search in Subsection 6.2.1 and a variant of the Nelder-Mead method in Subsection 6.2.2. In Section 6.3 we consider a combined method and make a conclusion.

6.1 Steepest Descent Methods for Discrete Functions

In [44, Chapter 10], two algorithms are presented to minimize L$^{\#}$-convex functions that are based on a steepest descent idea. In continuous optimization, Steepest Descent algorithms are very popular. The gradient method is a popular Steepest Descent method. Many other numerical methods for minimizing unconstrained optimization problems are based on the idea from it. The convexity of the objective function is not a prerequisite. The functions only need to be continuously differentiable. Details can be found in [7, Sections 9.3, 9.4]. The idea in the continuous case is basically simple: starting from an initial point, we determine a direction in which the function values of the objective function decrease the most, and take a step in this direction. The negative gradient of the function itself is a good choice for the direction. Additionally, we need to determine how large the step is. In the next paragraph, we present two Steepest Descent methods for discrete functions, which are presented in [44, Chapter 10].

In the context of L$^{\#}$-convexity, Murota chooses the box neighborhood from (5.36) to characterize a local minimal point (in our work "box-local minimal point"). The first method we introduce here is introduced in this context in [44, Subsection 10.3.1].

Let $X \subseteq \mathbb{Z}^n$ be, as in the previous chapter, a discrete box, $f : X \to \mathbb{R}$ a discrete function, p a point of X and $N^B(p)$ the box neighborhood of p. The idea is to search for a point $q \in (N^B(p) \cap X)$ such that $f(q) = \min_{s \in (N^B(p) \cap X)} f(s)$. If $f(p) \leq f(q)$ holds, p is a box-local minimal point of f. In the other case ($f(p) > f(q)$) the direction from p to q is then the steepest descent direction in the box neighborhood of p. We replace p with q and redo the last steps until we find a box-local minimal point.

In some applications, the problem has occurred that the domain of the discrete functions is only a subset of the discrete box, see Remark 5.3.29. The "missing" points are always located at the "boundary" of the box, i.e., no "holes" are created. However, if the missing points are located at the boundary, this does not cause any further problems, as the set $N^B(p) \cap X$ then consists of fewer points. To keep the notation consistent, we still write in the prerequisites of the methods that X is a discrete box. If this is not exactly the case, we know how to deal with it.

Altogether we get Algorithm 2:

Algorithm 2: Steepest Descent Algorithm

```
   Input: Discrete box $X \subseteq \mathbb{Z}^n$, discrete function $f : X \to \mathbb{R}$, initial point
          $p^0 \in X$
   Output: Box-local minimal point $\overline{q}$
 1 Set $p = p^0$.
 2 Determine $q = \arg \min_{s \in (N^B(p) \cap X)} f(s)$.
 3 while $f(p) > f(q)$ do
 4     Set $p = q$.
 5     Determine $q = \arg \min_{s \in (N^B(p) \cap X)} f(s)$.
 6 end
 7 Set $\overline{q} = q$.
 8 return $\overline{q}$.
```

If, instead of any domain $\mathbb{Z}^n$, we consider the discrete box X as the domain of f, the algorithm presented by Murota corresponds to Algorithm 2. Algorithm 2 terminates after a finite number of steps, since X has a finite number of elements. If f is additionally L$^\#$-convex, the output of Algorithm 2 is also a global minimal point, see Figure 5.18. We summarize this in the following corollary:

Corollary 6.1.1. *Let $X \subseteq \mathbb{Z}^n$ be a discrete box, and $f : X \to \mathbb{R}$ be a discrete L$^\#$-convex function. Let $\overline{q} \in X$ be the output of the Steepest Descent algorithm (Algorithm 2). Then $\overline{q}$ is a global minimal point of f on X.*

In general, it does not matter which neighborhood definition is chosen. The output is always a local minimal point corresponding to the neighborhood choice. It is time-efficient for us if not every function value $f(s)$ with $s \in X$ is considered until the algorithm terminates. If we choose the direction neighborhood N^D from (5.37), this will probably not be the case, as a large number of points s must be considered in line 5. Another important point is that points q and function values $f(q)$ should be saved during the algorithm, since if we go one step further, some function values are considered again, since we shift the box only "one step further". Especially in the application of modular systems, the evaluation of the discrete function corresponds to a large amount of work (calculation of the optimal value of an optimization problem).

In [44, Subsection 10.3.2] and [42] it is mentioned that the so-called "scaling algorithms" are commonly used to minimize a discrete convex function efficiently.

Basically, the idea is to minimize a function f^α instead of the discrete function $f : \mathbb{Z}^n \to \mathbb{R}$, where $f^\alpha(x)$ is only defined for parts of the set $\mathbb{Z}^n$ and we have $f^\alpha(x) := f(\alpha x)$ with $x \in \mathbb{Z}^n$. If we transfer this to a domain $X \subseteq \mathbb{Z}^n$, then $\alpha x \in X$ must also be fulfilled. The scaled function f^α "is simpler in shape, and hence easier to minimize"([42, Chapter 1]). Using "proximity theorems", the quality of a local optimal point of f^α can be estimated. Such results are available for L$^\#$-convex functions, see e.g. [44, Theorem 7.18]. Let x^α be a local minimal point of the scaled function and $x^\star$ a (local) minimal point of f. Then the proximity theorem provides for a L$^\#$-convex function an upper bound for $||x^\alpha - x^\star||_\infty$. A proximity scaling approach is then given if we decrease α, e.g. by halving it, and calculate a minimal point of f^α with the Steepest Descent method. If the minimal point is better than the current incumbent, we replace the old point. Then α is halved until we have $\alpha = 1$. According to the proximity theorem, this procedure terminates at some point with a global optimal point of f.

In this work, we will not consider the scaling method approach further. There are two reasons for this. First, the papers considered do not specify precise algorithms for L$^\#$-convex functions. A method for L-convex functions is specified, which can probably be applied in a similar way to L$^\#$-convex functions as we describe above. However, the papers do not explain how this should work in detail. Second, these methods are likely to be most effective when the size of X is large in some dimension. However, since this is not really the case in our applications, we are pursuing a different approach. This also involves evaluating or considering fewer points.

In the numerical experiments that follow in this and the next sections, we want to use as few function evaluations as possible. For this reason, we will now only consider the box neighborhood and continue with the numerical results of Algorithm 2.

Python Results for the Steepest Descent Method

We now consider the examples introduced in the previous chapters. These are first the modular systems for the binpacking and crane bridge problems. We distinguish between two variants. In one case, we use the result files already generated by Algorithm 1. In this case, all function values of the function f entered in Algorithm 2 on X are known. In the other case, we determine the function values for the required points (in lines 2 and 5) by solving the optimization problem $\overline{P}^p_{bin}(s)$ or $\overline{P}^{crane}(s)$ and then execute lines 2 and 5 accordingly. In the second case, we can compare whether the optimization approach first introduced in Chapter 4 really performs worse in terms of runtime than the method based on a decomposition and Steepest Descent approach. Although the discrete functions of the modular system problems are not L$^\#$-convex, we have already seen in Subsection 5.3.3 that box-local and global minimal points coincide, see Table 5.10. Therefore, the outputs of the Steepest Descent method are also global optimal points of the discrete function. In the last column, we also show the number of function evaluations in relation to all possible ones (in %). This number naturally depends on the initial point p^0. We consider two variants, one with a initial point close to the origin and one close to the center of X.

For the binpacking problems for which we randomly generated the input data,

Name	p^0	$\overline{q}$	Runtime [s]	Points [%]
BP_DIM1_1	(1)	(4)	0.783	50.000
BP_DIM1_1	(5)	(4)	1.179	40.000
BP_DIM2_1	(0,1)	(2,5)	7.077	17.500
BP_DIM2_1	(5,5)	(2,5)	6.882	18.333
BP_DIM3_2	(0,0,1)	(1,2,4)	35.792	10.081
BP_DIM3_2	(5,4,3)	(1,2,4)	53.694	14.797
BP_DIM4_2	(0,0,0,1)	(0,1,2,4)	167.196	11.737
BP_DIM4_2	(2,2,2,2)	(0,1,2,4)	176.141	12.046
∅				21.812

TABLE 6.1: Results of the Steepest Descent method (with optimization) for the binpacking modular system

for the PyCUTEst test functions, and for the separable test functions, we directly include the data of the results files from Algorithm 1. We explicitly look at whether the global minimal point is found. This is interesting as in some of these examples the box-local and global optimal points do not coincide, see Tables 5.13, 5.14, 5.15, or 5.16. In these examples, we focus on how many evaluations of the discrete function are necessary.

Since we are now considering many different modular system problems and various discrete functions, and in this chapter we are looking at other numerical methods and concepts in addition to Algorithm 1 and 2, Tables 6.2 and 6.3 provide an overview of the main results that are collected during this work. These refer in part to previous chapters and also to the content that follows.

Firstly we consider the modular system examples with solving the optimization problem in lines 2 and 5 of Algorithm 2.

Results Steepest Descent Including Solving Optimization Problems

In Table 6.1 the results for the different binpacking modular systems are shown. The corresponding initial points p^0, the output of the algorithm $\overline{q}$, the runtime, and how many of the points of the discrete box are "visited". For example BP_DIM3_2, compared to Algorithm 1, only 14% of the function evaluations are required, for example. For the problem instances with dimension one, the percentage is quite high, but this is obvious since there are relatively few points in the domains of the discrete functions. For the two-dimensional examples, the solution time of $\overline{P}^2_{bin}$ was low at approximately 14 seconds due to the simplicity of the example, but the Steepest Descent algorithm delivers slightly better runtimes (around 7 seconds). The problems $\overline{P}^3_{bin}$ and $\overline{P}^4_{bin}$ for the input data of examples BP_DIM3_2 and BP_DIM4_2 are solved surprisingly fast (in 12 and 6 seconds). Algorithm 2 performs worse for these problems. Presumably, problems $\overline{P}^3_{bin}$ and $\overline{P}^4_{bin}$ can still be solved well by Gurobi instead of considering many individual problems.

Method / Algorithm	Modular System		Binpacking Random		
	Crane Bridge	Binpacking	dim2	dim3	dim4
Model Chapter 4	CRANE_N5_4 (Table 4.8) CRANE_N20_1 (Table 4.10)	BP_DIM1_1 (Table 4.18) BP_DIM2_1 (Table 4.25) BP_DIM3_2 (Table A.18) BP_DIM4_2 (Table A.10)	-	-	-
Decomposition Approach (Algo. 1)	CRANE_N5_4 (Table 5.1) CRANE_N20_1 (Table 5.2)	BP_DIM1_1 (Table 5.4) BP_DIM2_1 (Table 5.8) BP_DIM3_2 (Table A.24) BP_DIM4_2 (Table A.26-A.28)	-	-	-
Convexity Considerations	Table 5.11	Table 5.10	Table 5.13	Table 5.14	Table 5.15
Steepest Descent Algorithm (Algo. 2) with optimization: from file:	 Table 6.4 Table 6.5	 Table 6.1 Table 6.5	 - Table 6.6	 - Table A.29	 - Table A.30
Coordinate Search (Algo. 3)	Table 6.9		Table 6.10	Table A.31	Table A.32
Nelder Mead (Figure 6.7)	Table 6.15		Table 6.16	Table A.41	Table A.42
NM-SD	Table 6.17		-	Table A.43	Table A.45
CS-SD	Table 6.18		-	Table A.44	Table A.46

TABLE 6.2: Overview results tables for the numerical methods for the modular system examples

Method / Algorithm	PyCUTEst Testfunctions			Separable Testfunctions	
	dim2	dim3	dim4	f_3^n	f_4^n
Description	Table 5.16			Table 5.18	
Convexity Considerations	Table 5.17			Table 5.19	
Steepest Descent Algorithm (Algo. 2)	Table 6.8			Table 6.7	
Coordinate Search (Algo. 3)	Table A.35	Table A.36	Table A.36	Table A.33	Table A.34
Nelder Mead (Figure 6.7)	Table 6.14	Table A.39	Table A.40	Table 6.12	Table 6.13

TABLE 6.3: Overview results tables for the numerical methods for the discrete test functions

Name	p^0	$\overline{q}$	Runtime [s]	Visited Points [%]
CRANE_N5_4	(1,1)	(2,3)	102.947	48.000
CRANE_N5_4	(3,3)	(2,3)	116.944	48.000
CRANE_N20_1	(1,1)	(1,3)	185.781	16.000
CRANE_N20_1	(6,3)	(1,3)	10047.783	50.000
∅				40.500

TABLE 6.4: Results of the Steepest Descent method (with optimization) for the crane bridge modular system

Next, we consider the crane bridge modular system and the problem instances CRANE_N5_4 and CRANE_N20_1 and solve the corresponding optimization problems $\overline{P}^{crane}$ during Algorithm 2. The results of the Steepest Descent algorithm are shown in Table 6.4. The runtime for the small problem (CRANE_N5_4) is larger for both initial points than in the first optimization approach (10 s). For the larger example (CRANE_N20_1) we found out in Chapter 5 that some problems with a fixed number of variants had an extremely high solution time. The reason remained unclear at this point, probably numerical issues within Gurobi. Since Algorithm 2 now also solves problems with fixed variants, this can also occur. For this reason, we set a time limit of 30 minutes for solving the corresponding optimization problem $\overline{P}^{crane}$. If the time limit is reached, the best known current objective function value (with a known feasible point) is returned and used. For the initial point $p^0 = (1, 1)$ no optimization procedure had to be stopped and the runtime is smaller than in the optimization approach of Chapter 4 (266 s). For $p^0 = (6, 3)$ some optimization processes had to be stopped, which nevertheless leads to a high total runtime of the Steepest Descent algorithm. It is also noticeable here that the number of problems considered varies considerably depending on the initial point (16 vs. 50%).

At this point, it is clear that when we use Algorithm 2 to find an optimal configuration for the modular system problem, significantly fewer optimization problems need to be solved than it is the case for Algorithm 1. However, the runtimes of the Steepest Descent algorithm are often higher than those of the first optimization approach we considered in Chapter 4. Of course, the runtimes depend very much on the input data, in particular on how many variants are potentially allowed in the modular system and how many products are in demand. The Steepest Descent algorithm does not seem to offer any significant time advantage for the examples if the function values of the discrete function are unknown and must be determined in the algorithm. Before we consider another approach for minimizing discrete functions, we look at the examples in which we assume that the function values of the discrete functions are known.

Results Steepest Descent from Result Files

First we consider again the modular system examples (for completeness), but pick the function values of the discrete function from the result files from Subsection 5.2.1 and 5.2.2. The results are shown in Table 6.5. In this paragraph a comparison of runtimes with results from Chapter 4 is not interesting. The first two columns of the result tables are again the name and the initial point p^0. We again look at the

Name	p^0	Global MP SD	Points [%]
CRANE_N5_4	(1,1)	**True**	48.000
CRANE_N5_4	(3,3)	**True**	48.000
CRANE_N20_1	(1,1)	**True**	16.000
CRANE_N20_1	(6,3)	**True**	50.000
BP_DIM2_1	(0,1)	**True**	24.138
BP_DIM2_1	(5,4)	**True**	26.437
BP_DIM3_2	(0,0,1)	**True**	9.660
BP_DIM3_2	(5,4,1)	**True**	16.995
BP_DIM4_2	(0,0,0,1)	**True**	12.189
BP_DIM4_2	(2,2,2,2)	**True**	12.510
∅ [%]		100.000	26.393

TABLE 6.5: Results of the Steepest Descent method (from resultfile) for modular system examples

number of considered points during the algorithm (column "Points [%]"). In addition, we also tested if the global minimal point was found (column "Global MP SD"). This was the case for all examples, which we also found out in the convexity considerations for the modular system examples in Subsection 5.3.3. We found out that almost all discrete functions are not L$^\#$-convex, but global and local minimal points coincide nevertheless. The initial points and visited points differ slightly from the previous examples, as ".5" are randomly rounded up or down, and the selection of the point q in line 5 of the algorithm is not always unique.

We continue with the randomly generated binpacking examples. The input data were produced for testing issues of L$^\#$- and subgradient-convexity in Subsection 5.3.3. We considered examples from dimension two up to four. Not for all problem instances, the box-local and global minimal points coincide, so it is interesting here if the Steepest Descent method finds the global minimal point. The results of the two-dimensional examples are shown in Table 6.6. For two problem instances (BP_RD_DIM2_4 and BP_RD_DIM2_13) only a box-local optimal point is found that is not global, independent of which initial point was chosen. For the other examples, the global minimal point is found. On average, only 36% of the possible variants are considered. The results for the three- and four dimensional examples are presented in Tables A.29 and A.30. Here, too, only box-local minimal points that are not global are found in some cases. It is noticeable that the number of points visited decreases significantly as the dimension increases (20% and 11%). However, this was to be expected.

For the PyCUTEst test data, the results are shown in Table 6.8. For most of the data, the global minimal points are found, and especially for the three- and four dimensional problems (lower two parts of the table), only a few points are visited during the Steepest Descent method (5% and 8%). For the separable test functions for all initial points, the global minimal point is found by the Steepest Descent method, although only f_3 is L$^\#$-convex. The results are shown in Table 6.7. The average of the number of visited points during the algorithm is also very low (15% and 10%).

Through our research on discrete functions, discrete convexity, and subsequently also the minimization of discrete functions, we observed that there is very little lit-

Name	p^0	Global MP SD	Points [%]
BP_RD_DIM2_1	(0,1)	**True**	25.926
BP_RD_DIM2_1	(2,3)	**True**	51.852
BP_RD_DIM2_2	(0,1)	**True**	62.500
BP_RD_DIM2_2	(2,1)	**True**	70.833
BP_RD_DIM2_3	(0,1)	**True**	15.152
BP_RD_DIM2_3	(4,3)	**True**	24.242
BP_RD_DIM2_4	(0,1)	False	31.111
BP_RD_DIM2_4	(3,2)	False	31.111
BP_RD_DIM2_5	(0,1)	**True**	24.615
BP_RD_DIM2_5	(5,2)	**True**	24.615
BP_RD_DIM2_6	(0,1)	**True**	25.000
BP_RD_DIM2_6	(4,2)	**True**	44.231
BP_RD_DIM2_7	(0,1)	**True**	15.306
BP_RD_DIM2_7	(5,2)	**True**	23.469
BP_RD_DIM2_8	(0,1)	**True**	21.538
BP_RD_DIM2_8	(4,2)	**True**	26.154
BP_RD_DIM2_9	(0,1)	**True**	20.732
BP_RD_DIM2_9	(4,2)	**True**	32.927
BP_RD_DIM2_10	(0,1)	**True**	21.277
BP_RD_DIM2_10	(4,2)	**True**	46.809
BP_RD_DIM2_11	(0,1)	**True**	48.276
BP_RD_DIM2_11	(2,2)	**True**	55.172
BP_RD_DIM2_12	(0,1)	**True**	19.512
BP_RD_DIM2_12	(4,2)	**True**	17.073
BP_RD_DIM2_13	(0,1)	False	22.368
BP_RD_DIM2_13	(4,2)	False	18.421
BP_RD_DIM2_14	(0,1)	**True**	60.000
BP_RD_DIM2_14	(2,0)	**True**	70.000
BP_RD_DIM2_15	(0,1)	**True**	62.963
BP_RD_DIM2_15	(3,0)	**True**	77.778
∅ [%]		86.667	36.365

TABLE 6.6: Results of the Steepest Descent method (from resultfile) for the randomly generated input data for the binpacking example in dimension two

Name	p^0	Global MP SD	Points [%]
f_3^2	(0,0)	**True**	38.776
f_3^2	(3,3)	**True**	16.327
f_3^3	(0,0,0)	**True**	18.950
f_3^3	(3,3,3)	**True**	7.580
f_3^4	(0,0,0,0)	**True**	8.788
f_3^4	(3,3,3,3)	**True**	3.332
∅ [%]		100.000	15.625
f_4^2	(0,0)	**True**	23.457
f_4^2	(4,4)	**True**	17.284
f_4^3	(0,0,0)	**True**	8.916
f_4^3	(4,4,4)	**True**	6.310
f_4^4	(0,0,0,0)	**True**	3.216
f_4^4	(4,4,4,4)	**True**	2.225
∅ [%]		100.000	10.235

TABLE 6.7: Results of the Steepest Descent method (from resultfile) for the separable test functions

Name	p^0	Global MP SD	Points [%]
DENSCHNC	(-5,-5)	False	24.793
DENSCHNC	(2,2)	**True**	11.570
ELATVIDU	(-5,-5)	False	13.223
ELATVIDU	(2,2)	False	9.917
ROSENBR	(-5,-5)	**True**	28.099
ROSENBR	(2,2)	**True**	11.570
WAYSEA1	(-5,-5)	False	23.140
WAYSEA1	(2,2)	**True**	9.917
∅ [%]		50.000	16.529
BOX3	(-5,-5,-5)	**True**	5.485
BOX3	(2,2,2)	**True**	4.508
DENSCHND	(-5,-5,-5)	**True**	7.739
DENSCHND	(2,2,2)	**True**	4.884
HATFLDE	(-5,-5,-5)	False	4.959
HATFLDE	(2,2,2)	False	2.705
HELIX	(-5,-5,-5)	**True**	9.467
HELIX	(2,2,2)	**True**	4.583
∅ [%]		75.000	5.541
HILBERTA	(-3,-3,-3,-3)	**True**	8.788
HILBERTA	(2,2,2,2)	**True**	8.788
HIMMELBF	(-3,-3,-3,-3)	**True**	1.666
HIMMELBF	(2,2,2,2)	**True**	4.040
PENALTY1	(-3,-3,-3,-3)	**True**	8.788
PENALTY1	(2,2,2,2)	**True**	8.788
PENALTY2	(-3,-3,-3,-3)	False	8.455
PENALTY2	(2,2,2,2)	**True**	8.455
POWELLSG	(-3,-3,-3,-3)	**True**	8.788
POWELLSG	(2,2,2,2)	**True**	8.788
STREG	(-3,-3,-3,-3)	**True**	10.662
STREG	(2,2,2,2)	**True**	7.955
∅ [%]		91.667	7.830

TABLE 6.8: Results of the Steepest Descent method (from resultfile) for the PyCUTEst test functions

erature available on these topics. The situation is different for optimization problems with continuous objective functions and, in particular, discrete feasible sets. As already mentioned in Subsection 5.3.1, the main difference between discrete functions and the associated minimization problems is that the functions are only defined for integer points. We cannot therefore define something like differentiability for discrete functions, as we cannot simply move a little bit away from a discrete point [15, Chapter 6]. At these points, the discrete function is not defined. In the context of optimization, especially the minimization of functions, there are, however, also numerical methods that work without derivatives. These will be considered further in the following.

6.2 Derivative-Free Optimization Approaches for Discrete Functions

“In earlier days of nonlinear optimization perhaps one of the most common reasons for using derivative-free methods was the lack of sophistication or perseverance of the user.”([10, Section 1.1]) In many applications, derivatives are not available or are not easy to calculate, so other methods are of interest. The books [10] and [1] present a wide-ranging introduction to derivative-free optimization, to which we mainly refer in this section. According to [1, Chapter 1], “derivative-free optimization (DFO) is the mathematical study of optimization algorithms that [. . .] not use derivatives”. This does not mean that the derivatives do not exist, as in our case, but that they are not used. So, the methods considered in these books and in our following work, “do not rely on derivative information of the objective function [...] nor are the methods designed explicitly to approximate [this] derivative.”([10, Section 1.1]).

Most of the algorithms that are presented in the book are for unconstrained optimization problems where a continuous objective function has to be minimized, see [10, Section 1.4]. The goal is to transfer suitable methods to our problem, where we minimize a discrete function on a discrete box. Normally, it is not a problem to go from an unconstrained problem to a box-constrained problem, as we have already done with the Steepest Descent algorithm. The bigger challenge will be that our objective function is not defined for arbitrary points.

The methods we have already considered (Algorithm 1 and 2) can also be classified in the literature on derivative-free optimization. The idea behind evaluating all function values of the discrete function in Algorithm 1 is also presented in a similar way in [1, Chapter 3.2], where they consider the so-called grid search. The difference is that we choose the discrete box directly as the grid and leave it fixed rather than refining it. As in Murota’s work, the authors of [10] pursue the idea of determining the direction in which to continue searching based on function evaluations. In the Steepest Descent method (SD), which is based on Murota’s ideas, we consider all points in the box around our current one and continue searching in the direction of the neighbor with the lowest function value. In [10, Chapter 7] and [1, Chapter 3.3], methods are presented that follow a similar approach. However, not only boxes are used, but also other so-called patterns that contain fewer points than the box. Thus, the number of function evaluations can be reduced. These methods are called directional direct-search methods in [10] and we consider one of those, the Coordinate search, in Subsection 6.2.1.

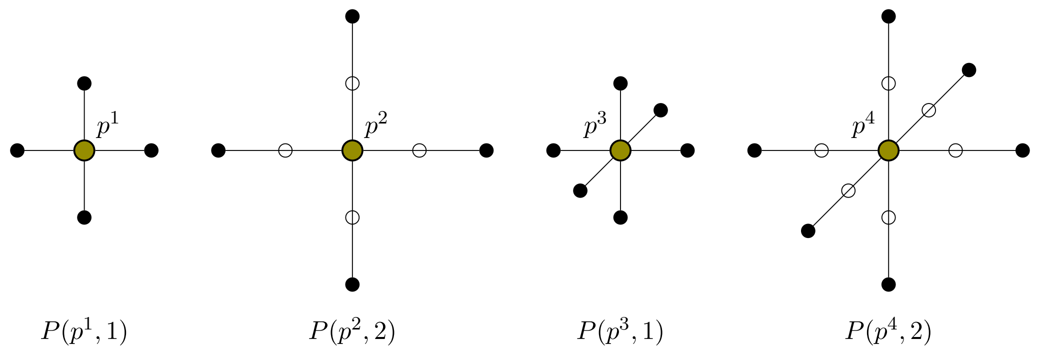

FIGURE 6.1: Patterns $P(p^j, \alpha)$ (filled black points) for the Coordinate search with points $p^1, p^2 \in \mathbb{Z}^2$ (in olive) and $p^3, p^4 \in \mathbb{Z}^3$ (in olive) and $\alpha = 1$ or $\alpha = 2$

The other fundamental idea is not to search in the direction of the best functional value but away from the worst. These methods also work with the geometry of points, for example with a simplex. For this reason, these methods are also referred to as simplicial direct-search methods. The well-known Nelder-Mead method is one of them that "has become one of the most widely applied and researched optimization algorithms in the world"([1, Chapter 5]. In Subsection 6.2.2, we briefly introduce the ideas of the original Nelder-Mead method and consider than a version of the method adapted to our problems. It should be noted that no literature relevant to our application in the context of Nelder-Mead for the minimization of discrete function was found.

As described above we first start with a directional direct-search method, the so-called Coordinate Search method.

6.2.1 Coordinate Search

In this subsection we consider one directional direct-search method, the Coordinate Search (CS). The CS is one of the easiest directional direct-search methods, which is also known as compass search, see [10, Section 7.1]. We briefly introduce the method as it is presented in the literature and consider afterwards our application and how the CS has to be adjusted to our setting. We follow [10, Section 7.1] and [1, Chapter 3.3]. We remark at this point that the methods in the two books are introduced for unconstrained continuous optimization problems.

Let the point $p \in \mathbb{R}^n$ be a current iterate and $\alpha \in \mathbb{R}_{>0}$ a current step size. The idea is now that we first consider the function values of all points in the set

$$P(p, \alpha) = \{p + \alpha d | \; d \in D\}$$

with $D = [I, -I]$. The term $d \in D$ means that d is one column of the matrix $D = [I, -I]$, which here means a unit vector or a negative unit vector. We call the set $P(p, \alpha)$ a search pattern around the point p or only a pattern. In Figure 6.1 different patterns $P(p^j, \alpha)$ $(j = 1, \dots, 4)$ with $\alpha = 1$ or $\alpha = 2$ for dimension two and three are illustrated.

In general, different directional direct-search methods differ in the matrices D. The goal is to find a point in the pattern $P(p, \alpha)$ that decreases the objective function value. Depending on the literature we either choose as next iterate the first point in $P(p, \alpha)$ that does this during consideration (for example in [10]) or some point in the pattern not further specified (as in [1]) as the new iterate. If a better point q is found in the pattern, we set $p = q$ and repeat the last step with the same step size. If there is no improvement, we half the step size and repeat the last step. The steps are repeated until a specified minimal step size is reached. The corresponding detailed algorithm can be found in [10, Algorithm 7.1] or [1, Algorithm 3.3]. It is clear that the CS is a method that is part of local search methods and not global ones, see [1, Section 3.4]. The point at which termination occurs obviously depends mainly on the starting point and also on the starting step size.

In general, the procedure described above can be applied to our application, the minimization of a discrete function f on a discrete box X, if we can evaluate f at the points of the pattern. The restriction imposed by the discrete box does not pose a major problem at this stage. We only require in the following algorithm, that the new candidate has to lie in X. In any case, the step size can only be halved until we reach $\alpha = 1$ and must be a power of two, at the beginning so that we remain on the integer grid. In Section 6.1, we had a similar situation when we briefly described the scaled method for the Steepest Descent algorithm. As in Algorithm 2, it does not matter in this numerical method whether we consider a subset of a discrete box X with missing "boundary points" instead of the discrete box itself. For this reason, we continue to write that X is a discrete box.

The following algorithm presents the Coordinate Search for discrete functions.

Algorithm 3: Coordinate Search

```
Input: Discrete box $X \subseteq \mathbb{Z}^n$, discrete function $f : X \to \mathbb{R}$, initial point
       $p^0 \in X$ and initial step size $\alpha^0 \in 2^{\mathbb{N}}$ such that $(X \cap P(p^0, \alpha^0)) \neq \emptyset$
Output: Best known point $\overline{q}$
1  Set $p = p^0$ and $\alpha = \alpha^0$.
2  while $\alpha \geq 1$ do
3      Determine $P(p, \alpha) = \{p + \alpha d |\ d \in D\}$ with $D = [I, -I]$.
4      Determine $q = \arg \min_{s \in (P(p,\alpha) \cap X)} f(s)$.
5      if $f(q) < f(p)$ then
6          Set $p = q$.
7      else
8          Set $\alpha = \frac{\alpha}{2}$.
9      end
10 end
11 Set $\overline{q} = p$.
12 return $\overline{q}$.
```

Since α^0 has to be a power of two, we briefly write $\alpha^0 \in 2^{\mathbb{N}}$ with $2^{\mathbb{N}} = \{2^n |\ n \in \mathbb{N}\}$. In our setting, the algorithm terminates at some point with a point $\overline{q}$, since X has a finite number of elements. It is not clear whether $\overline{q}$ is a box-local minimal point since we only have $P(\overline{q}, 1) \subsetneq N^B(\overline{q})$ when $n > 1$. For this reason, the Coordinate Search, as we consider it here, falls into the category of heuristics. This is not necessarily the case when we are dealing with a continuous function. There, under certain conditions and assumptions, convergence results to global minimal points are known. Details can be found in [10, Section 7.2 et seq.]

There is hope that fewer function evaluations will be necessary compared to Algorithm 2, and that we will end up at least close to a box-local minimal point. Since we know the box-local and global minimal points for our test functions, we can test Algorithm 3 and examine how it behaves on average.

Python Results for the Coordinate Search

Since we have already established in the results of the Steepest Descent method that the running times for some fixed problems were surprisingly high, we will now only consider modular systems for which we already know the results. The main interest in this subsection is to evaluate how well the Coordinate Search performs, and for this it is of course useful to know where the box-local and global minimal points lie. We measure the success of the method again in the percentage of how often box-local and global optimal points are found and in the number of considered points.

We start similar to the experiments of the Steepest Descent method from Section 6.1 with the modular system examples. The results are given in Table 6.9. We tested different initial points p^0, one near the origin and one near the center of X. They can be found in the column p^0. Since our discrete boxes do not have a large diameter, we only considered $\alpha = 1$ and $\alpha = 2$. In total, for only 50% of the examples a global (and box-local) minimal point was found. This is surprisingly a very small number, since all problems have only box-local minimal points that are global minimal (see Tables 5.11 and 5.10). If a point is identified as a global minimal point (since we know them from the considerations above), we mark it with **True** in the column "Global MP CS". This point is obviously then also a box-local minimal point. For this reason, this is no new information, and we mark it therefore by True. If a point is only a box-local minimal point, we mark it with **True** in the column "Box-Local MP CS". The number of considered points reduces to approximately half compared to that of the Steepest Descent method (Table 6.5). This is not surprising, but of course, it is only helpful if the method works well.

We continue with the randomly generated two-dimensional binpacking problems. Here also in only 56% of the problems, a global minimal point was found, but it is surprising that for some more problems nonetheless a box-local minimal point was found (70%). The Steepest Descent Method (SD) performed significantly better than Coordinate Search (CS) by finding 86% of the global minimal points, but at the price of the number of points that have to be evaluated, more precisely 36% (SD) instead of 24% (CS). For the three- and four-dimensional examples, the results are not better. They are presented in Tables A.31 and A.32. These results motivate another new local search method.

Lastly, we consider the PyCUTEst test functions and the separable functions f_3^n and f_4^n. We start with the PyCUTEst test functions. The results for the two-dimensional examples are quite good, since in 80% of the experiments a box-local minimal point is found. The number is a bit smaller for the three-dimensional case with 62% and for the four-dimensional case with 75%. Similarly to the Steepest Descent method, the Coordinate Search always finds a global (and box-local) minimal point for all cases for the separable test functions. The results of these examples can be found in Tables A.33-A.37.

Name	p^0	α	Global MP CS	Box-Local MP CS	Points [%]
CRANE_N5_4	(1,1)	1	**True**	True	36.000
CRANE_N5_4	(1,1)	2	False	False	36.000
CRANE_N5_4	(3,3)	1	**True**	True	32.000
CRANE_N5_4	(3,3)	2	False	False	40.000
CRANE_N20_1	(1,1)	1	**True**	True	14.000
CRANE_N20_1	(1,1)	2	**True**	True	16.000
CRANE_N20_1	(6,3)	1	False	False	30.000
CRANE_N20_1	(6,3)	2	False	False	36.000
BP_DIM2_1	(0,1)	1	**True**	True	18.391
BP_DIM2_1	(0,1)	2	**True**	True	14.943
BP_DIM2_1	(5,4)	1	False	False	12.644
BP_DIM2_1	(5,4)	2	False	False	13.793
BP_DIM3_2	(0,0,1)	1	**True**	True	4.472
BP_DIM3_2	(0,0,1)	2	**True**	True	4.472
BP_DIM3_2	(5,4,1)	1	False	False	3.936
BP_DIM3_2	(5,4,1)	2	False	False	4.651
BP_DIM4_2	(0,0,0,1)	1	**True**	True	2.566
BP_DIM4_2	(0,0,0,1)	2	False	False	2.005
BP_DIM4_2	(2,2,2,2)	1	False	False	1.283
BP_DIM4_2	(2,2,2,2)	2	**True**	True	2.486
∅ [%]			50.000	50.000	16.282

TABLE 6.9: Results of the Coordinate Search (from resultfile) for modular system examples

Name	p^0	α	Global MP CS	Box-Local MP CS	Points [%]
BP_RD_DIM2_1	(0,1)	1	**True**	True	18.519
BP_RD_DIM2_1	(0,1)	2	**True**	True	33.333
BP_RD_DIM2_1	(2,3)	1	**True**	True	40.741
BP_RD_DIM2_1	(2,3)	2	**True**	True	37.037
BP_RD_DIM2_2	(0,1)	1	**True**	True	50.000
BP_RD_DIM2_2	(0,1)	2	False	False	37.500
BP_RD_DIM2_2	(2,1)	1	**True**	True	50.000
BP_RD_DIM2_2	(2,1)	2	**True**	True	41.667
BP_RD_DIM2_3	(0,1)	1	**True**	True	9.091
BP_RD_DIM2_3	(0,1)	2	**True**	True	11.111
BP_RD_DIM2_3	(4,3)	1	False	False	14.141
BP_RD_DIM2_3	(4,3)	2	False	False	14.141
BP_RD_DIM2_4	(0,1)	1	False	**True**	20.000
BP_RD_DIM2_4	(0,1)	2	False	**True**	20.000
BP_RD_DIM2_4	(3,2)	1	False	**True**	22.222
BP_RD_DIM2_4	(3,2)	2	False	False	24.444
BP_RD_DIM2_5	(0,1)	1	False	False	13.846
BP_RD_DIM2_5	(0,1)	2	False	False	13.846
BP_RD_DIM2_5	(5,2)	1	**True**	True	20.000
BP_RD_DIM2_5	(5,2)	2	**True**	True	16.923
BP_RD_DIM2_6	(0,1)	1	**True**	True	17.308
BP_RD_DIM2_6	(0,1)	2	**True**	True	25.000
BP_RD_DIM2_6	(4,2)	1	False	False	21.154
BP_RD_DIM2_6	(4,2)	2	False	False	23.077
BP_RD_DIM2_7	(0,1)	1	**True**	True	12.245
BP_RD_DIM2_7	(0,1)	2	**True**	True	11.224
BP_RD_DIM2_7	(5,2)	1	False	False	11.224
BP_RD_DIM2_7	(5,2)	2	False	False	12.245
BP_RD_DIM2_8	(0,1)	1	**True**	True	13.846
BP_RD_DIM2_8	(0,1)	2	**True**	True	13.846
BP_RD_DIM2_8	(4,2)	1	False	**True**	15.385
BP_RD_DIM2_8	(4,2)	2	False	**True**	15.385
BP_RD_DIM2_9	(0,1)	1	**True**	True	13.415
BP_RD_DIM2_9	(0,1)	2	**True**	True	12.195
BP_RD_DIM2_9	(4,2)	1	False	False	13.415
BP_RD_DIM2_9	(4,2)	2	False	False	14.634
BP_RD_DIM2_10	(0,1)	1	**True**	True	17.021
BP_RD_DIM2_10	(0,1)	2	False	False	19.149
BP_RD_DIM2_10	(4,2)	1	**True**	True	29.787
BP_RD_DIM2_10	(4,2)	2	**True**	True	29.787
BP_RD_DIM2_11	(0,1)	1	**True**	True	31.034
BP_RD_DIM2_11	(0,1)	2	**True**	True	31.034
BP_RD_DIM2_11	(2,2)	1	**True**	True	41.379
BP_RD_DIM2_11	(2,2)	2	**True**	True	51.724
BP_RD_DIM2_12	(0,1)	1	False	False	10.976
BP_RD_DIM2_12	(0,1)	2	False	False	10.976
BP_RD_DIM2_12	(4,2)	1	**True**	True	13.415
BP_RD_DIM2_12	(4,2)	2	**True**	True	13.415
BP_RD_DIM2_13	(0,1)	1	False	**True**	15.789
BP_RD_DIM2_13	(0,1)	2	False	**True**	17.105
BP_RD_DIM2_13	(4,2)	1	False	**True**	13.158
BP_RD_DIM2_13	(4,2)	2	**True**	True	15.789
BP_RD_DIM2_14	(0,1)	1	**True**	True	50.000
BP_RD_DIM2_14	(0,1)	2	**True**	True	35.000
BP_RD_DIM2_14	(2,0)	1	**True**	True	55.000
BP_RD_DIM2_14	(2,0)	2	**True**	True	45.000
BP_RD_DIM2_15	(0,1)	1	**True**	True	40.741
BP_RD_DIM2_15	(0,1)	2	False	False	33.333
BP_RD_DIM2_15	(3,0)	1	False	False	37.037
BP_RD_DIM2_15	(3,0)	2	False	False	33.333
∅ [%]			56.667	70.000	24.086

TABLE 6.10: Results of the Coordinate Search (from resultfile) for randomly generated two-dimensional binpacking problems

As already mentioned in the introduction of this chapter, we consider in the following a second direct-search method that is based on a slightly different idea. The idea is that this method will perform better for the problems that the Coordinate Search did not handle well.

6.2.2 An Adaptation for the Nelder-Mead Method

In this subsection we consider an adaptation of the Nelder-Mead method. According to the classification of numerical methods in the book [10], the Nelder-Mead method is a numerical method that, like Coordinate Search, belongs to the direct-search methods. As already described in the introduction of this section, the idea is not to continue searching in the direction of the best point within a suitable pattern but to move away from the worst point. To do this, the Nelder-Mead method uses the geometry between points in a simplex. For this reason, the Nelder-Mead method is denoted as a simplicial direct-search method or is also known as the simplex method. However, since a method called the simplex method (by Dantzig) is very well known in the area of linear optimization, we do not use this term. The Nelder-Mead method and the simplex method from linear optimization have nothing to do with each other.

The Nelder-Mead Method was published 1965 by John Nelder and Roger Mead in [48] for unconstrained continuous optimization problems. The Nelder-Mead method "is probably the most widely cited of the direct-search methods"([10, Chapter 8]). As in the previous part of the work, we describe the original method before adapting it to our framework.

Original Nelder Mead Method

As remarked before, we first consider the original method for an unconstrained continuous optimization problem. Every iteration is based on simplices. In the following definition, we present two definitions. Firstly, we present the definition for a simplex as known in the literature, see, for example, [10, Definition 2.15, Section 8.1]. Secondly, we introduce the definition of an ordered simplex, which we need for the Nelder-Mead method.

Definition 6.2.1 (Simplex). *Let Y be a set of $n+1$ affinely independent points $y^i \in \mathbb{R}^n$ $(i = 0, \dots, n)$. Then, the convex hull of Y that is defined through*

$$\operatorname{conv}(Y) = \left\{ \sum_{i=0}^{n} \lambda_i y^i \,\middle|\, \lambda_i \geq 0,\ \sum_{i=0}^{n} \lambda_i = 1 \right\},$$

is called a simplex. We shorten this to the simplex $\mathbb{Y}$ or to $\Delta\{y^0, \dots, y^n\}$ if the exact vertices of the simplex are needed.

The simplex $\mathbb{Y} = \Delta\{y^0, \dots, y^n\}$ has dimension n with Definition 6.2.1 and is spanned through the vertices $y^0, \dots, y^n$ [10]. For notational reasons for the following we also need an object, where the points y^i are ordered by function value for a given (continuous or discrete) function f. Geometrically, the simplices are the same as before, but the indices of the points y^i can differ. We set the indices of the points such that $f(y^0) \leq \cdots \leq f(y^n)$ holds. The corresponding ordered simplex is denoted by $\mathbb{Y}^f = \Delta^f\left(y^0, \dots, y^n\right)$. As is known from set theory and algebra, we use round brackets in places where an order is involved and curly brackets in the other case.

For the sake of completeness, we have the following definition:

Definition 6.2.2 (Ordered Simplex). *Let $\mathbb{Y} = \Delta\{y^0, \dots, y^n\}$ be a simplex as defined in Definition 6.2.1 and f be a discrete or continuous function. If $f(y^0) \leq \dots \leq f(y^n)$ holds, we call the simplex an ordered simplex and we write $\mathbb{Y}^f = \Delta^f\left(y^0, \dots, y^n\right)$.*

For this chapter, elements with the "doubled lined" letters (except the known sets $\mathbb{R}, \mathbb{Z}, ...$) are simplices, and we write $\mathbb{Y} = \Delta\{y^0, \dots, y^n\}$ or $\mathbb{Y}^f = \Delta^f(y^0, \dots, y^n)$ when we mean that the points $y^0, \dots, y^n$ are exactly the vertices of the simplex.

Let f be a function with $f : \mathbb{R}^n \to \mathbb{R}$ and $\mathbb{Y}^f = \Delta^f(y^0, \dots, y^n)$ an ordered simplex with the vertices $y^0, \dots, y^n$. We want to minimize f. Since $\mathbb{Y}^f$ is an ordered simplex, we have $y^n = \arg \max\limits_{i=0,\dots,n} f(y^i)$. The idea is now to replace the vertex of the simplex with the worst function value, i.e. y^n, through a point y that lies in the line between a point y^c and y^n. The point y^c is the center of all points except for y^n, so we have $y^c = \frac{1}{n} \sum\limits_{i=0}^{n-1} y^i$. In particular, we have

$$y = y^c + \delta(y^c - y^n).$$

The points on the line are of several type. Depending on the choice of δ, we speak of a reflection ($\delta = 1$), an expansion ($\delta > 1$), an outside contraction ($\delta \in (0, 1)$) or an inside contraction ($\delta \in (-1, 0)$). To illustrate these terms, we consider a two-dimensional example and choose typical values for δ, see [10, Section 8.1].

Example 6.2.3. We have $n = 2$ and the ordered simplex is defined through $\mathbb{Y}^f = \Delta^f\left(\begin{pmatrix}2\\4\end{pmatrix}, \begin{pmatrix}3\\2\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}\right)$, with a not further described function f. The simplex is illustrated with black edges in Figure 6.2.

With $y^c = \dfrac{1}{2}\left(\begin{pmatrix}2\\4\end{pmatrix} + \begin{pmatrix}3\\2\end{pmatrix}\right) = \begin{pmatrix}2.5\\3\end{pmatrix}$, we obtain the reflection $y^r = \begin{pmatrix}4\\5\end{pmatrix}$ ($\delta = 1$, olive), the expansion $y^e = \begin{pmatrix}5.5\\7\end{pmatrix}$ ($\delta = 2$, teal), the outside contraction $y^{oc} = \begin{pmatrix}3.25\\4\end{pmatrix}$ ($\delta = \frac{1}{2}$, orange) and the inside contraction $y^{ic} = \begin{pmatrix}1.75\\2\end{pmatrix}$ ($\delta = -\frac{1}{2}$, red). The names of the specific points are also clear with Figure 6.2, and the resulting simplices are illustrated.

Depending on the function value $f(y^r)$ of the reflection, a decision is now made by which point the worst point ($y^2 = \left(1, 1\right)^\top$ in Example 6.2.3) in the current simplex is replaced. We briefly outline the procedure below in any dimension n. The procedure can be found, for example, in [10, Algorithm 8.1] or [1, Algorithm 5.1] and is known as the Nelder-Mead method.

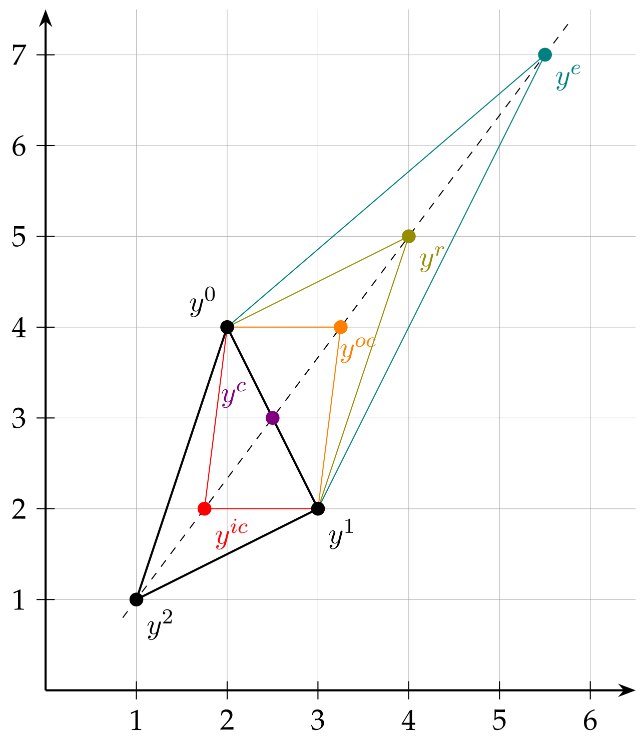

FIGURE 6.2: Illustration of Example 6.2.3 for reflection, expansion and contraction

Algorithm Nelder-Mead Original

We assume that we are in an iteration k. Let $\mathbb{Y}_k^f = \Delta^f\left(y^0,\ldots y^n\right)$ be the current ordered simplex.

1. We determine the reflection y^r of y^n as described above with $\delta = 1$. Then we compare $f(y^r)$ with $f(y^0)$ (best function value), $f(y^{n-1})$ (second worst function value) and $f(y^n)$ (worst function value). Depending on the result, the worst point y^n is replaced by the reflection point, by the expansion, or by a contraction point.

2. If the reflection is better than the best vertex, i.e., $f(y^r) < f(y^0)$, we determine the expansion y^e and calculate $f(x^e)$. Then we replace y^n by the best of y^r and y^e and get the simplex $\mathbb{Y}_{k+1}$. After sorting we get $\mathbb{Y}_{k+1}^f$ and terminate the iteration and go back to 1. with $k = k+1$.

3. If y^r is better than the current second worst point and worse than y^0, i.e., $f(y^0) \leq f(y^r) < f(y^{n-1})$, we replace y^n by y^r to get $\mathbb{Y}_{k+1}$. After sorting we get $\mathbb{Y}_{k+1}^f$ and terminate the iteration and go back to 1. with $k = k+1$.

4. If y^r is worse than y^{n-1}, i.e. $f(y^r) \geq f(y^{n-1})$, we contract the simplex through an outside or inside contraction, depending on whether y^r is the second worst or worst point:

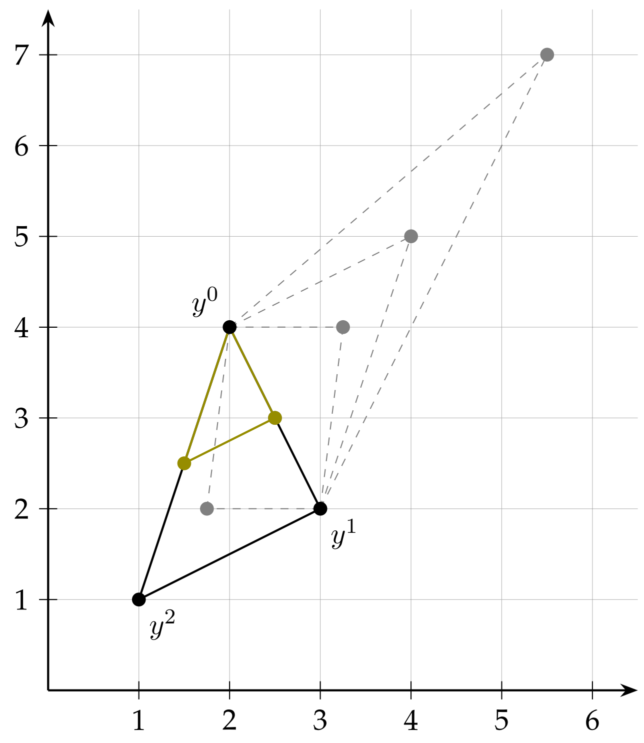

FIGURE 6.3: Illustration simplex with a shrinkage for Example 6.2.3

4.1. If the reflection is the second worst point, i.e., $f(y^r) < f(y^n)$, we determine the outside contraction y^{oc} and calculate $f(x^{oc})$. Then we replace y^n by the best of y^r and y^{oc} and get the simplex $\mathbb{Y}_{k+1}$. After sorting we get $\mathbb{Y}^f_{k+1}$ and terminate the iteration and go back to 1. with $k = k + 1$.

4.2. If the reflection is the worst point, i.e., $f(y^r) \geq f(y^n)$, we determine the inside contraction y^{ic} and calculate $f(y^{ic})$. If $f(y^{ic}) < f(y^n)$ holds, we replace y^n by y^{ic} and get the simplex $\mathbb{Y}_{k+1}$. After sorting, we get $\mathbb{Y}^f_{k+1}$ and terminate the iteration and go back to 1. with $k = k + 1$. If $f(y^{ic}) \geq f(y^r)$ holds, we go directly to 5. and shrink the simplex.

5. We only keep the vertex y^0 and determine the other points by $y^0 + \gamma(y^i - y^0)$ for $i = 1, \ldots, n$ with a given shrink parameter $\gamma > 0$. After sorting, we get the new shrunk simplex $\mathbb{Y}^f_{k+1}$. Then we terminate the iteration and go back to 1. with $k = k + 1$.

A typical choice for the shrinkage parameter γ is $\gamma = \frac{1}{2}$. A shrunk simplex for the data of Example 6.2.3 is shown in Figure 6.3 with olive edges.

To summarize this, with reflection, expansion, or contraction we try to get a new vertex of the simplex. We compare the reflection point y^r with the best, the second worst, and the worst vertex of the current simplex. “Either the new simplex differs by a single vertex [(2.,3. or 4.)], or has only one vertex in common with the previous simplex[(5.)]”([1, Section 5.1]). The idea is to take the reflection as a new vertex if it is good enough (3.) and not better than the best point in the current simplex. If the reflection is better than the current best point (2.), we try to “go more into this good direction” and try to make the simplex larger, so we consider the expansion. If the expansion is still better, we choose the expansion as the new vertex.

Something similar is done in the contraction step. If the reflection is bad, we check whether how bad it is, that means, whether y^r is either the worst (4.2.) or the second worst (4.1.) point. In general, we do not want to go too far in the direction of y^r in this case. If y^r is the second worst point, we try if the outside contraction y^{oc} is better than the reflection. If it is, we choose y^r as new vertex (4.1). In the other case, we try a point closer to y^n and so we choose the inside contraction y^{ic} if it is an improvement (4.2). If the inside contraction does not provide a better point, we shrink the simplex to refine the search area. The decision as to when a simplex is shrunk (i.e. when 5. is executed) varies somewhat in the literature. We have followed [1, Algorithm 5.1] in this regard. Since we will not be considering any shrink steps in the following anyway, we will not go into more detail for now. The basic idea of this paragraph is to give only a rough understanding of how Nelder-Mead works.

According to the literature, one termination criterion is if the diameter of $\mathbb{Y}^f_{k+1}$ is small enough. The diameter of a simplex $\mathbb{Y} = \Delta \left\{y^0, \dots, y^n\right\}$ is given by

$$\text{diam}(\mathbb{Y}) = \max_{0 \le i \le j \le n} ||y^i - y^j||$$

for some norm $||\cdot||$ [10, Section 2.5]. According to [1, Theorem 5.3] the Nelder-Mead method terminates after a finite number of steps with a point $\overline{y}$, if f is bounded from below. It is not guaranteed that $\overline{y}$ is a (local) minimal point. For this reason, the Nelder-Mead method is in general a heuristic method. Under more assumptions and “minor variations”([Chapter 5][1]) of the presented method, the convergence to a local minimal point can be shown. Details and a modified method can be found in [10, Section 8.3] and in Algorithm 8.2 of [10]. According to [10, Introduction Chapter 8], the Nelder-Mead method performs so well because the simplices considered take into account the curvature of the function. Neither continuity nor derivative information is required for this.

After the introduction of the original Nelder-Mead method, we want to adapt the ideas of the method to our framework. We require that all points for which we need to determine the function values are integers and that these should be within the given discrete set X. If the points $y^0, \dots, y^n$ that span the simplex are integers, there is no guaranty that the potentially new simplex points y^r, y^e, y^{oc} or y^{ic} are integers or that they lie in the set X. Even if the simplex is shrunk, this is not necessarily the case. The point y^e from Figure 6.2, for example, is not an integer, although y^0, y^1 and y^2 are integers, and even a $\delta \in \mathbb{Z}$ is chosen. Depending on the choice of δ, a point on the line-segment can, of course, be an integer, e.g. y^r of Figure 6.2 is integer. Since we want to derive a method that can be transferred to arbitrary dimension, we do not start by determining the parameter δ exactly so that the possible new simplex points are integers. This may be possible for a few special cases, but it is probably not arbitrarily transferable to other points and, above all, to higher dimensions.

Adaptation of the Nelder-Mead Method to Discrete Functions

We try to transfer the ideas from the original Nelder-Mead method to our framework. No literature was found in context. The derivation of the method was very experimental. Fortunately, in the end, a method was developed that performed very well for certain test functions.

The exact calculation of new simplex points leads to problems with integer values, as can already be seen in the small example in Figure 6.2. Since our discrete boxes have a rather small diameter in our applications, we limit ourselves to only "small" simplices. We choose as the starting simplex a simplex $\mathbb{S} = \Delta\{y^0, \dots, y^n\}$ (or later $\mathbb{S}_0$) for which

$$||y^i - y^j||_\infty = 1, \quad i, j = 0, \dots, n, \ (i \neq j) \tag{6.1}$$

holds. We denote simplices that fulfill (6.1) with a doubled-line $\mathbb{S}$ from now on. All vertices of $\mathbb{S}$ need to be in the discrete box X, which we denote by $\mathbb{S} \in X$. The unit or standard simplex $\mathbb{S}_1 = \Delta\{0, e_1, \dots, e_n\}$, with unit vectors e_j, is such a simplex, for example.

In the two-dimensional case, it is easy to verify that a reflection in an arbitrary simplex with integer vertices is always an integer. However, this can no longer be transferred to the three-dimensional case if we calculate the reflection as above. Let $\mathbb{S}_1 = \Delta\{0, e_1, e_2, e_3\}$ be a three-dimensional simplex. If we want to replace the point 0 for example, we obtain $y^c = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)^\top$ and so $y^r = \left(\frac{2}{3}, \frac{2}{3}, \frac{2}{3}\right)$, which is obviously not integer.

We will now pursue a more geometric idea and continue to consider only simplices that satisfy (6.1). We will not consider the concepts of expansion, contraction, and shrinkage any further, as they cannot be easily applied to our case. We will continue to consider the idea of reflection and adapt it slightly. To derive the procedure, we consider another example with a discrete function that is already known.

Example 6.2.4. We consider the discrete separable function $f_3^2 : X \to \mathbb{R}$ with $X = [0, 6]^2 \cap \mathbb{Z}^2$ and $f_3^2(x) = (x_1 - 3)^2 + (x_2 - 3)^2$. The function f_3^2 has the global optimal point $(3, 3)^\top$ on X. We assume that we are in iteration k and we consider the simplex $\mathbb{S}_k^f = \Delta^f \left(\begin{pmatrix} 2 \\ 2 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \end{pmatrix} \right)$ with $f = f_3^2$. The ordering is given, since we have

$$f\left(\begin{pmatrix} 2 \\ 2 \end{pmatrix}\right) = 2 \leq 5 = f\left(\begin{pmatrix} 2 \\ 1 \end{pmatrix}\right) = f\left(\begin{pmatrix} 1 \\ 2 \end{pmatrix}\right).$$

Thus, we want to replace the point $y^2 = (1, 2)^\top$. The simplex (black) and a hyperplane (here the gray line) through the best two points are shown in Figure 6.4.

The idea is now to replace y^2 through a point on the opposite side of the hyperplane through y^0 and y^1 as y^2. The new vertex s, of course, needs to lie in X and must satisfy

$$||s - y^0||_\infty = 1, \quad ||s - y^1||\infty = 1.$$

Here, two points fulfill these properties. They are illustrated in the Figure. Firstly, that is obviously the point $y^r = (3, 1)^\top$ (violet), which is exactly the reflection of y^2. The other point is $\tilde{y} = (3, 2)^\top$ (olive). If there is more than one candidate, we choose one that is farthest away from y^2 measured in the Euclidean norm. In the illustrated example, this is clearly the point y^r.

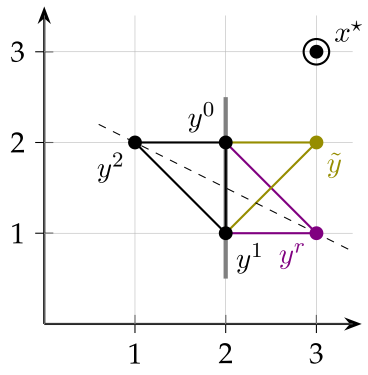

FIGURE 6.4: Illustration simplices for the discrete Nelder-Mead adaptation from Example 6.2.4

Now we want to consider the general case, so we consider again the simplex $\mathbb{S}_k^f = \Delta^f(y^0, \ldots, y^n)$. We want to replace the worst vertex y^n. Then a new vertex candidate s has to

1. fulfill

$$||s - y^j||_\infty = 1, \quad j = 0, \ldots, n-1, \; s \in X$$

   and

2. lie on the opposite side as y^n of the hyperplane that passes through the points $y^0, \ldots, y^{n-1}$.

Here, 1. ensures that we only consider points of the set X and the new simplex is also a small simplex, while we hope to get with 2. a good search direction. We denote the hyperplane by $H(I)$, where I is the index set of the corresponding indices of the vertices that lie on the hyperplane. So we have $I = \{0, \ldots, n-1\}$. The index of the point to be replaced becomes the index i^-, thus here we have $i^- = n$. We collect all candidates for new simplex points in a set C^k and sort them by the Euclidean distance from y^n. The first element of C^k should be the point with the longest distance from y^n. If the set C^k is empty, which can occur on the "boundary" of the discrete box, we try to find candidates which only fulfill 1. If that is also not possible, we stop the method. This can occur if, instead of a discrete box, we consider only a subset of it. We will briefly discuss this again in the experiments and explain when this occurs. In the running example, we have:

Example 6.2.5. *(Sequel 1 of Example 6.2.4)*
Here we have $C^k = \{y^r, \tilde{y}\}$. So we choose the candidate $y^r = (3, 1)^\top$.

This, of course, raises the question of why we are now taking a different approach, but still end up at the point y^r in the example. The key point is that we can apply this procedure to any dimension without getting issues with the integrality and feasibility of the points. In addition, the candidates play another role later in the termination. Since we want to avoid function evaluations in our application because they are very time-consuming, we do not determine the function value of all candidates to find out which one is really the best. We simply go as far as the simplex construction allows us to move away from the worst point in the expectation that this direction will be effective. Let s^0 be the first element of C^k, s^1 the second (if existing), and so on.

Through the method, we store the already considered vertices of the simplices as sets in a set $\mathcal{S}$. Before defining the new simplex with s^0 instead of y^n, we need to check whether this combination of vertices was already considered in a previous iteration. We collect the vertices of the new simplex candidate in a set S, here $S = \left\{y^j : j \in I\right\} \cup \left\{s^0\right\}$ and need to check whether $S \in \mathcal{S}$ holds. As is usual with sets, the order of the individual vertices in S is irrelevant. If we would consider a simplex in the next step that we have already considered, i.e., $s \in S$, we may enter an infinite loop. Of course, we must avoid this. If $S \notin \mathcal{S}$ we determine $f(s^0)$ and determine the simplex $\mathbb{S}_{k+1} = \Delta\{y^0, \ldots, y^{n-1}, s^0\}$ and sort it to $\mathbb{S}^f_{k+1}$. If $S \in \mathcal{S}$, i.e., the simplex is considered in an earlier iteration, we consider two obvious possibilities. We denote the variants by replacement rules, and they are determined from the start.

In variant 1, we choose the second point, i.e., s^1, of the set of candidates C^k instead of the first (if it exists). Then we set $S = \{y^0, \ldots, y^{n-1}, s^1\}$. If $S \in \mathcal{S}$ we take the third element of C^k, and so on. If there are no more points left, we terminate the method. The corresponding index of the points we try is denoted by q. So, at the beginning, we have $q = 0$, then $q = 1$ and so on.

In variant 2, instead of the worst point (y^n) we replace the second worst point, i.e., y^{n-1}. Thus, we set $i^- = n - 1$ and the index set to $I = \{0, \ldots, n\} \setminus \{i^-\}$ and go back to the calculation of new candidates for the new vertex. The new vertex must lie on the opposite side of $H(I)$ as y^{i^-} (2.) and the new simplex must be small (1.). If there are no candidates, we try to find a point that satisfies only 1. We overwrite the set C^k, sort the points again, now according to their Euclidean distance from y^{n-1}. The point with the longest distance is again s^0 (if it exists), and we set $S = \{y^j : j \in I\} \cup \{s^0\}$. If S is not considered yet (i.e., $S \notin \mathcal{S}$), we calculate $f(s^0)$ and define the new simplex $\mathbb{S}^f_{k+1}$ accordingly. If $S \in \mathcal{S}$ we set $i^- = n - 2$ and repeat the last steps. If there is no vertex of $\mathbb{S}^f_k$ left to replace but the best point y^0, we stop.

At the beginning, we decide which replacement rule (variant 1 or 2) to choose and then follow it throughout the entire method. If a new vertex is found, we set $k = k + 1$, add the new simplex vertices to $\mathcal{S}$ and reset the corresponding indices, i.e., we start again with $q = 0$, $I = \{1, \ldots, n-1\}$ and $i^- = n$.

So far, only termination criteria based on the inability to find a possible simplex candidate have been presented in the procedure. If we find a point that is very good, we must ensure that the procedure terminates. In the original Nelder-Mead procedure, this is ensured by the simplices shrinking around such a point. It stops as soon as a simplex has a too small diameter. Since our simplices are not generally shrinking, we choose a different but similar termination criterion. The problem is that, especially in high dimensions, many new simplices can be considered in our procedure, even if the best point (i.e., the minimal point) remains identical. The simplex rotates around the best point y^0, so to speak. The process would eventually stop without any other termination criteria when all simplices are considered, but this can take a long time. To avoid considering all possible simplices, we stop when the best known point does not change for too long. We denote this point by $\overline{y}$ and count the number of occurrences of this point. The possible number of simplices

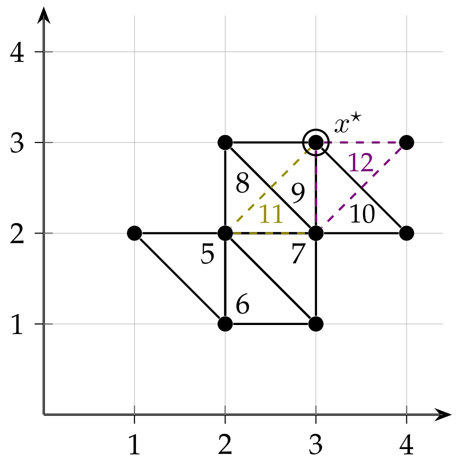

FIGURE 6.5: Illustration Example 6.2.6 for iteration $k = 5$ to $k = 10$ (black), and $k = 11, 12$ with replace rule 1

that can be considered depends in any case on how many points are located in the immediate neighborhood of y^0. If we choose the box neighborhood, which fits to our simplex construction, this is exactly $3^n - 1$ points. So we choose the following termination criterion: If the best point has already occurred more than $\frac{3^n}{2}$ times, we stop. We denote this termination criterion by ($\star$). Since we never replace the best point in the other stop criteria, we return the best point known in the simplex at the end in these termination cases.

We continue with the running example of this Subsection.

Example 6.2.6. *(Sequel 2 of Example 6.2.4)* Let the simplex of Example 6.2.4 be the simplex of iteration $k = 5$. We get there if we choose the initial simplex $\mathbb{S}_0 = \Delta\left\{\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}\right\}$. The simplices of iterations $k = 5$ to $k = 10$ are shown in Figure 6.5 (in black). Until $k = 10$ there are no special occurrences. The iteration index k is also shown in the figure, by marking within the corresponding simplex, or here, triangle. We have $\mathbb{S}^f_{10} = \Delta^f\left(\begin{pmatrix}3\\3\end{pmatrix}, \begin{pmatrix}3\\2\end{pmatrix}, \begin{pmatrix}4\\2\end{pmatrix}\right)$. In Iteration 11, we would get again the simplex $\mathbb{S}^f_9$, so we have to find another new vertex. Depending on the variants 1 and 2, this choice differs.

Variant 1: With the replacement rule 1, we choose a point that also lies on the other side of the hyperplane through $y^0 = (3,3)^\top$ and $y^1 = (3,2)^\top$ with the second-longest distance from $y^2 = (4,2)^\top$. This is the point $(2,2)^\top$ and the new simplex is not considered yet, so we obtain $\mathbb{S}^f_{11} = \Delta^f\left(\begin{pmatrix}3\\3\end{pmatrix}, \begin{pmatrix}3\\2\end{pmatrix}, \begin{pmatrix}2\\2\end{pmatrix}\right)$ (olive). With that we get $\mathbb{S}^f_{12} = \Delta^f\left\{\begin{pmatrix}3\\3\end{pmatrix}, \begin{pmatrix}3\\2\end{pmatrix}, \begin{pmatrix}4\\3\end{pmatrix}\right\}$ (violet). In $k = 13$ we have $C_{13} = \left\{\begin{pmatrix}2\\2\end{pmatrix}, \begin{pmatrix}2\\3\end{pmatrix}\right\}$ (sorted by distance from $(4,3)^\top$). The two simplices that are now being considered are already known. For this reason, the method stops. The output is the point $\overline{y} = (3,3)^\top$.

Variant 2: If we choose replacement rule 2, we try to replace instead of $y^2 = (4,2)^\top$ the second worst point of the current simplex ($\mathbb{S}^f_{10}$), here $y^1 = (3,2)^\top$.

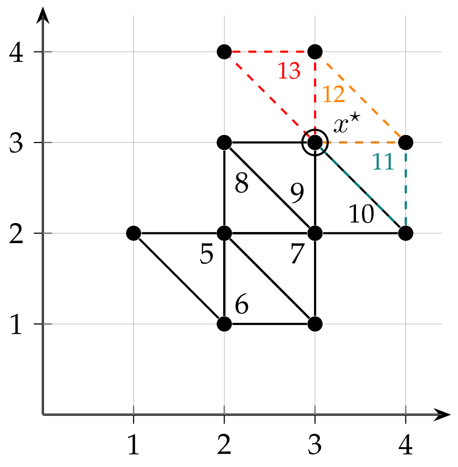

FIGURE 6.6: Illustration Example 6.2.6 for iteration $k = 5$ to $k = 10$ (black), and $k = 11, 12, 13$ with replace rule 2

The resulting simplex has not been considered yet, so we obtain $\mathbb{S}^f_{11} = \Delta^f\left(\begin{pmatrix}3\\3\end{pmatrix}, \begin{pmatrix}4\\3\end{pmatrix}, \begin{pmatrix}4\\2\end{pmatrix}\right)$ (teal). The illustration for this case is shown in Figure 6.6. Then we obtain directly $\mathbb{S}^f_{12} = \Delta\left(\begin{pmatrix}3\\3\end{pmatrix}, \begin{pmatrix}3\\3\end{pmatrix}, \begin{pmatrix}4\\3\end{pmatrix}\right)$ (orange). By replacing $y^2 = (4,3)^\top$ we get $\mathbb{S}^f_{13} = \Delta^f\left(\begin{pmatrix}3\\3\end{pmatrix}, \begin{pmatrix}3\\4\end{pmatrix}, \begin{pmatrix}2\\4\end{pmatrix}\right)$ (red). It is already noticeable here that we have already circled around point $(3,3)^\top$, which is always the best point of the simplices considered. The termination criterion is (not mentioned yet because it has not been applied): If the best point in the simplex has already occurred more than $\frac{9}{2} = 4.5$ times, we stop. In our case, $(3,3)^\top$ has already occurred 5 times in iteration 13, i.e., the method terminates in this step and the output is the point $\overline{y} = (3,3)^\top$.

With both variants, the output of the method is the global minimal point of f_3 on X. The termination criterion differs depending on the replace rule in this example. In some cases, the order of the points in the simplex or in the set C^k is important. If the values of the objective function are the same, this order is, of course, not unique.

In Figure 6.7 the flow chart of the complete adaptation of the Nelder-Mead algorithm is illustrated. The main steps of the method are depicted in the olive boxes. The different termination criteria are illustrated with **Stop**^{1} for the criterion $(\star)$, **Stop**^{2}, if no candidates can be found that fulfill the conditions 1.-3. and **Stop**^{3}, if there is no replacement possibility in variants 1 or 2.

The method is similar to Coordinate Search and, unlike the Steepest Descent method, clearly a heuristic method. We cannot ensure that the method will terminate with a local or global minimal point. We hope that this method will work well despite the modifications, as Nelder-Mead works very well in the continuous case.

Python Results for Nelder-Mead for the Discrete Test Functions

To verify whether our method works well, we start with the test functions. The reason for introducing them in Subsection 5.3.4 are the considerations we make

Adaptation of the Nelder-Mead Method

Prerequisites: $X \subseteq \mathbb{Z}^n$ discrete box, $f : X \to \mathbb{R}$,
initial ordered simplex $\mathbb{S}_0^f = \Delta^f(y^0, \ldots, y^n) \in X$,
$\mathcal{S} = \{\{y^0, \ldots, y^n\}\}$

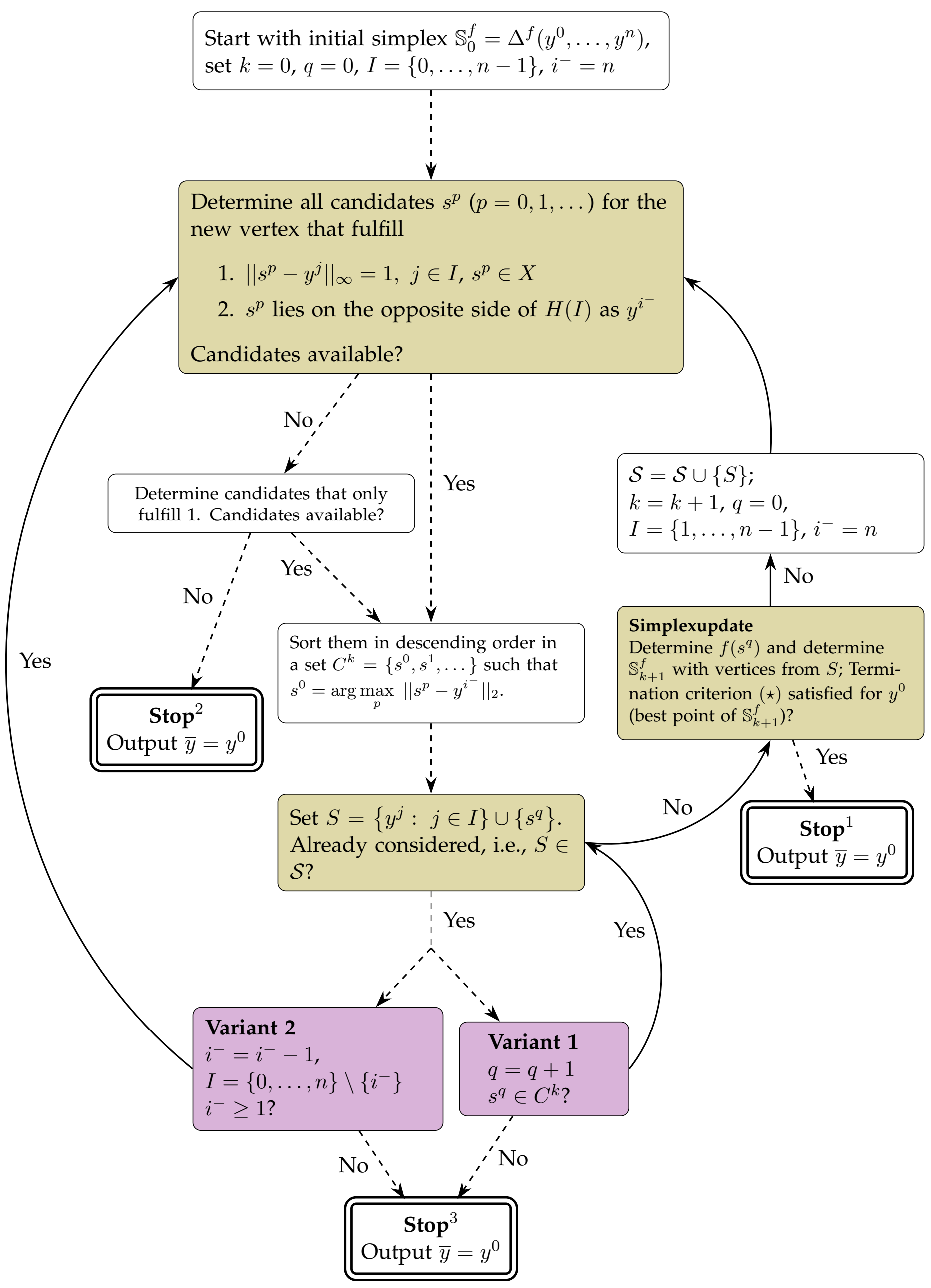
Start with initial simplex $\mathbb{S}_0^f = \Delta^f(y^0, \ldots, y^n)$, set $k = 0$, $q = 0$, $I = \{0, \ldots, n-1\}$, $i^- = n$
Determine all candidates s^p $(p = 0, 1, \ldots)$ for the new vertex that fulfill
1. $\|s^p - y^j\|_\infty = 1,\ j \in I,\ s^p \in X$
2. s^p lies on the opposite side of $H(I)$ as y^{i^-}
Candidates available?
No
Yes
Determine candidates that only fulfill 1. Candidates available?
No
Yes
Stop^2 Output $\overline{y} = y^0$
Sort them in descending order in a set $C^k = \{s^0, s^1, \ldots\}$ such that $s^0 = \arg\max_p \|s^p - y^{i^-}\|_2$.
$\mathcal{S} = \mathcal{S} \cup \{S\}$; $k = k+1$, $q = 0$, $I = \{1, \ldots, n-1\}$, $i^- = n$
No
Simplexupdate
Determine $f(s^q)$ and determine $\mathbb{S}_{k+1}^f$ with vertices from S; Termination criterion $(\star)$ satisfied for y^0 (best point of $\mathbb{S}_{k+1}^f$)?
Yes
Stop^1 Output $\overline{y} = y^0$
No
Set $S = \{y^j : j \in I\} \cup \{s^q\}$. Already considered, i.e., $S \in \mathcal{S}$?
Yes
Yes
Yes
Variant 2
$i^- = i^- - 1$, $I = \{0, \ldots, n\} \setminus \{i^-\}$ $i^- \geq 1$?
Variant 1
$q = q + 1$ $s^q \in C^k$?
No
No
Stop^3 Output $\overline{y} = y^0$

FIGURE 6.7: Flowchart for the adaptation of the Nelder-Mead method to the minimization of discrete functions

Name	start index	$\mathbb{S}_0$
f_3^2	0	$\Delta\{(0,0),(1,0),(0,1)\}$
	1	$\Delta\{(3,3),(4,3),(3,4)\}$
f_3^3	0	$\Delta\{(0,0,0),(1,0,0),(0,1,0),(0,0,1)\}$
	1	$\Delta\{(3,3,3),(4,3,3),(3,4,3),(3,3,4)\}$
f_3^4	0	$\Delta\{(0,0,0,0),(1,0,0,0),(0,1,0,0),(0,0,1,0),(0,0,0,1)\}$
	1	$\Delta\{(3,3,3,3),(4,3,3,3),(3,4,3,3),(3,3,4,3),(3,3,3,4)\}$

TABLE 6.11: Initial simplices Nelder-Mead Adaptation for discrete test functions f_3 and f_4

here in this part of the work.

We start with the discrete separable functions f_3^n and f_4^n for $n = 2,3,4$. We consider different initial simplices (which have different start indices) and both available replacement rules. The initial simplices can be found in Table 6.11. For simplicity, we omit the transpose symbol at the points. We choose for each dimension n a starting simplex near the origin and one in the center of the discrete box X.

Firstly, we consider f_3^n. This discrete function is "very symmetric" and we have already considered it in Example 6.2.4 and its sequels (Examples 6.2.5 and 6.2.6). It is very positive that the global minimal point is found for all initial points. The results can be found in Table 6.12. For the upcoming numerical tests, we also check if the output is a box-local minimal point. If the output is a global optimal point, then it is also a box-local minimal point. In this case, we also print the "True", but in gray. In addition to the start node index (column name "start index"), the replacement rule we choose from the start, there is also the termination criterion index (column name "Stop NM") which is determined in the flow chart in Figure 6.7. In the last column ("Points NM [%]"), we show how many of the points of X were evaluated as a percentage. For the dimensions three and four these are surprisingly few points. The reason for termination differs depending on the initial simplex and the replacement rule. For this example, the initial simplex with the index 1 contains already the global minimal point. However, the point must still be verified accordingly, but the number of points evaluated is slightly lower than for the starting point near 0. It is also noticeable that slightly fewer points are evaluated than with the Coordinate Search (in percentage) and significantly fewer than with the Steepest Descent method (Tables 6.7 and A.33).

Secondly, we consider f_4^n. This discrete function has box-local minimal points that are not global minimal. The results are shown in Table 6.13. Also for the functions f_4^n the Nelder-Mead Adaptation works quite well since for every initial simplex and replacement rule, at least a box-local minimal point is found. The number of evaluated points is also slightly lower than in the results from the Coordinate Search (Table A.34). However, in the Coordinate search, the global minimal points were always found, which is not the case here. However, due to the simplex construction and the fact that we have no expansion, it is reasonable that local minimal points are also found, depending on where the algorithm gets stuck.

We continue with the other discrete test functions from the PyCUTEst library. We choose again two different initial simplices, one at the boundary and one near the center of X. The discrete boxes are identical for all problems within one dimension. We can therefore choose identical initial simplices. The initial simplices are

Name	start index	replace rule	Stop NM	Global MP NM	Box-Local MP NM	Points NM [%]
f_3^2	0	1	3	**True**	True	28.571
f_3^2	0	2	1	**True**	True	32.653
f_3^2	1	1	3	**True**	True	12.245
f_3^2	1	2	1	**True**	True	14.286
f_3^3	0	1	3	**True**	True	8.455
f_3^3	0	2	1	**True**	True	8.746
f_3^3	1	1	3	**True**	True	3.207
f_3^3	1	2	1	**True**	True	4.665
f_3^4	0	1	3	**True**	True	2.624
f_3^4	0	2	1	**True**	True	2.166
f_3^4	1	1	3	**True**	True	0.833
f_3^4	1	2	1	**True**	True	1.249
∅ [%]				100.000	100.000	9.975

TABLE 6.12: Results of the Nelder-Mead Adaptation for separable test function f_3^n for $n = 2, 3, 4$

Name	start index	replace rule	Stop NM	Global MP NM	Box-Local MP NM	Points NM [%]
f_4^2	0	1	1	**True**	True	18.519
f_4^2	0	2	1	**True**	True	19.753
f_4^2	1	1	3	False	**True**	7.407
f_4^2	1	2	1	False	**True**	9.877
f_4^3	0	1	1	**True**	True	4.115
f_4^3	0	2	1	**True**	True	4.390
f_4^3	1	1	1	False	**True**	1.920
f_4^3	1	2	1	False	**True**	2.195
f_4^4	0	1	1	**True**	True	0.991
f_4^4	0	2	1	**True**	True	0.945
f_4^4	1	1	3	False	**True**	0.381
f_4^4	1	2	1	False	**True**	0.488
∅ [%]				50.000	100.000	5.915

TABLE 6.13: Results of the Nelder-Mead Adaptation for separable test function f_4^n for $n = 2, 3, 4$

Name	start index	replace rule	Stop NM	Global MP NM	Box-Local MP NM	Points NM [%]
DENSCHNC	0	1	1	False	**True**	19.835
DENSCHNC	0	2	1	False	**True**	19.835
DENSCHNC	1	1	3	**True**	True	4.959
DENSCHNC	1	2	1	**True**	True	6.612
DENSCHNC	2	1	1	**True**	True	10.744
DENSCHNC	2	2	1	**True**	True	10.744
ELATVIDU	0	1	3	False	**True**	9.091
ELATVIDU	0	2	1	False	**True**	9.917
ELATVIDU	1	1	3	False	**True**	6.612
ELATVIDU	1	2	1	False	**True**	8.264
ELATVIDU	2	1	1	False	**True**	12.397
ELATVIDU	2	2	1	False	**True**	13.223
ROSENBR	0	1	3	**True**	True	19.835
ROSENBR	0	2	1	False	False	23.140
ROSENBR	1	1	3	**True**	True	4.132
ROSENBR	1	2	1	False	False	5.785
ROSENBR	2	1	1	False	**True**	7.438
ROSENBR	2	2	1	False	**True**	8.264
WAYSEA1	0	1	3	False	False	15.702
WAYSEA1	0	2	1	False	**True**	19.835
WAYSEA1	1	1	3	False	False	4.959
WAYSEA1	1	2	1	**True**	True	9.091
WAYSEA1	2	1	1	**True**	True	9.917
WAYSEA1	2	2	1	**True**	True	9.917
∅ [%]				37.500	83.333	11.260

TABLE 6.14: Results of the Nelder-Mead Adaptation for PyCUTEst test functions of dimension two

shown in Table A.38, divided into the different dimensions.

We start with the two-dimensional problems. The results are shown in Table 6.14. It is noticeable here that a global minimal point is found for only about one-third of the problems, but in approximately 83% of the cases at least a box-local minimal point is found. In the Coordinate Search, slightly more global optimal points are found, but fewer box-local minimal points (Table A.35). The average number of function evaluations required is again slightly lower here, at 11% instead of 13%.

For the three-dimensional examples, the results are shown in Table A.39. It is noticeable that only for 50% of the problems a global optimal point can be found, where both termination criteria apply. For the other problems, not even a box-local minimal point is found. With the Coordinate Search, we got similar results (Table A.36). Slightly more problems are solved there to global optimality (56%)

The results for the four-dimensional examples are shown in Table A.40 and they are more promising than in the three-dimensional case, since more than 70% of the problems are solved to global optimality and nearby 80% to box-local optimality. The number of function evaluations is extremely small with around 2% of all possible ones. Compared to the Coordinate Search slightly more problems are solved to global optimality here with a similar number of function evaluations.

Altogether, we have many problems in which our Nelder Mead Adaptation performs very well. Nevertheless, there are some problems (two- and three-di-

Name	start index	replace rule	Stop NM	Global MP NM	Box-Local MP NM	Points NM [%]
CRANE_N5_4	0	1	3	**True**	True	32.000
CRANE_N5_4	0	2	1	**True**	True	40.000
CRANE_N5_4	1	1	3	**True**	True	28.000
CRANE_N5_4	1	2	1	**True**	True	36.000
CRANE_N20_1	0	1	3	False	False	8.000
CRANE_N20_1	0	2	1	**True**	True	16.000
CRANE_N20_1	1	1	3	False	False	14.000
CRANE_N20_1	1	2	1	**True**	True	40.000
BP_DIM2_1	0	1	3	**True**	True	17.241
BP_DIM2_1	0	2	1	**True**	True	19.540
BP_DIM2_1	1	1	3	**True**	True	10.345
BP_DIM2_1	1	2	1	**True**	True	13.793
BP_DIM3_2	0	1	3	False	False	3.936
BP_DIM3_2	0	2	1	**True**	True	5.188
BP_DIM3_2	1	1	3	False	False	3.041
BP_DIM3_2	1	2	1	**True**	True	8.408
BP_DIM4_2	0	1	3	False	False	2.085
BP_DIM4_2	0	2	1	**True**	True	3.609
BP_DIM4_2	1	1	3	False	False	1.363
BP_DIM4_2	1	2	1	**True**	True	3.128
∅ [%]				70.000	70.000	15.284

TABLE 6.15: Results of the Nelder-Mead Adaptation for the modular system examples

mensional PyCUTEst and f_4^n) where a global minimal point was found in less than half of the cases. For some of these problems (three-dimensional PyCUTEst and f_4^n) the resulting points are usually box-local minimal points. It is also noticeable that the process terminates either because the best point has not changed for a long time (**Stop**^{1}), or because there are no more points to replace (**Stop**^{3}). Since the set X corresponds to the complete discrete box in all these examples and we always find enough suitable neighboring points, **Stop**^{2} does not occur. We continue with the modular system problems.

Python Results for Nelder-Mead for the Modular System Examples

We again only consider the methods without solving optimization models in each iteration for the same reason as in the Coordinate Search. The main interest in this subsection is to evaluate how well the Nelder-Mead adaptation performs, and for this it is of course useful to know where the box-local and global minimal points lie.

We start with the first-considered modular system examples. The initial simplices are again near the origin and near the center of X. They can be found in [21]. The set X is for the binpacking examples not a discrete box, since the point 0 is not part of it but all combinations of 0 and 1 per component instead. It turns out that this is not a problem here at this point. The results of the Nelder-Mead Adaptation are shown in Table 6.15. Here, too, the method performs acceptably, as 70% of the problems are solved globally. In the Coordinate Search, this was the case for significantly fewer problems (50%).

We continue with the randomly generated binpacking examples. Here we have the situation that for large numbers of variants, some problems cannot be solved, which means that f is not defined there. For this reason, the method also terminates in individual cases with the termination criterion **Stop**^{2}. The results for the two-dimensional examples are given in Table 6.16. The results are very good since about 80% of the problems are solved to global optimality, and in 90% of the examples, a box-local optimal point is found. For some of the problems, the method, as already mentioned, stops since there are no suitable neighboring points in the calculation of the set C^k. Nevertheless, in these problems, the global optimal point is found. Compared to the Coordinate Search (Table 6.10), we have significantly better results here, with around 20% more problems that are solved to global (and box-local) minimality.

When increasing the dimension to three and four, unfortunately this the results are worse. For the three-dimensional examples, only one third of the global minimal points are found. For the four-dimensional examples, this number is slightly higher, but also only 45%. For details, see Tables A.41 and A.42.

Basically, the results are not bad. However, there are some examples with comparatively poor results. Problems whose discrete functions are based on the modular system application perform particularly poorly. We suppose that this is because the sets X are not complete discrete boxes, especially in the randomly generated modular system problems. For a high number of variants, many problems cannot be solved, which means that the discrete function cannot be evaluated. Therefore, these points are not considered further and there are many cases in which the procedure terminates because no simplex can be determined (**Stop**^{3}). However, with PyCUTEst problems, the situation is different, and Nelder-Mead is more promising. Here we defined X as a discrete box from the beginning. Discrete functions $f : X \to \mathbb{R}$ for which the set $X \subseteq \mathbb{Z}^n$ is not suitable should therefore not be considered in connection with the Nelder-Mead method.

The Steepest Descent method performed significantly better in these examples. This raises the question whether it is efficient to combine the methods that we have considered so far. With the Steepest Descent method, the process is guaranteed to terminate at a box-local minimal point, but the discrete function must be evaluated at many points during the process. The idea is to first execute the Nelder-Mead adaptation (as a heuristic approach) or the Coordinate Search and then the Steepest Descent method. We hope that this will result in fewer function calls than if we only choose SD.

6.3 Combined Methods for Minimizing Discrete Functions

In this section, we conclude our work with the combination of the methods considered above. We denote the variant with first Nelder-Mead and then Steepest Descent by NM-SD and first Coordinate Search and then Steepest Descent accordingly with CS-SD. We consider the modular system examples (the first ones introduced) and the randomly generated binpacking problems in dimensions three and four, since there is the highest need of improvement.

Firstly, we consider NM-SD and NM-CS for the initial modular systems. The results are shown in Tables 6.17 and 6.18. In the end, for all problems and both combined methods, we found the global optimal point. This is not surprising, since

Name	start index	replace rule	Stop NM	Global MP NM	Box-Local MP NM	Points NM [%]
BP_RD_DIM2_1	0	1	3	**True**	True	18.519
BP_RD_DIM2_1	0	2	1	**True**	True	25.926
BP_RD_DIM2_1	1	1	3	False	False	25.926
BP_RD_DIM2_1	1	2	1	**True**	True	40.741
BP_RD_DIM2_2	0	1	2	**True**	True	50.000
BP_RD_DIM2_2	0	2	2	**True**	True	50.000
BP_RD_DIM2_2	1	1	2	**True**	True	29.167
BP_RD_DIM2_2	1	2	2	**True**	True	29.167
BP_RD_DIM2_3	0	1	3	**True**	True	9.091
BP_RD_DIM2_3	0	2	1	**True**	True	11.111
BP_RD_DIM2_3	1	1	3	**True**	True	11.111
BP_RD_DIM2_3	1	2	1	**True**	True	13.131
BP_RD_DIM2_4	0	1	1	False	**True**	17.778
BP_RD_DIM2_4	0	2	1	False	**True**	20.000
BP_RD_DIM2_4	1	1	3	**True**	True	15.556
BP_RD_DIM2_4	1	2	1	**True**	True	20.000
BP_RD_DIM2_5	0	1	3	**True**	True	10.769
BP_RD_DIM2_5	0	2	1	**True**	True	13.846
BP_RD_DIM2_5	1	1	3	False	False	10.769
BP_RD_DIM2_5	1	2	1	**True**	True	20.000
BP_RD_DIM2_6	0	1	3	**True**	True	17.308
BP_RD_DIM2_6	0	2	1	**True**	True	21.154
BP_RD_DIM2_6	1	1	3	**True**	True	17.308
BP_RD_DIM2_6	1	2	1	**True**	True	23.077
BP_RD_DIM2_7	0	1	3	**True**	True	11.224
BP_RD_DIM2_7	0	2	1	**True**	True	13.265
BP_RD_DIM2_7	1	1	3	**True**	True	9.184
BP_RD_DIM2_7	1	2	1	**True**	True	12.245
BP_RD_DIM2_8	0	1	1	**True**	True	12.308
BP_RD_DIM2_8	0	2	1	**True**	True	13.846
BP_RD_DIM2_8	1	1	3	False	**True**	10.769
BP_RD_DIM2_8	1	2	1	False	**True**	13.846
BP_RD_DIM2_9	0	1	3	False	False	10.976
BP_RD_DIM2_9	0	2	1	**True**	True	15.854
BP_RD_DIM2_9	1	1	3	False	False	10.976
BP_RD_DIM2_9	1	2	1	**True**	True	19.512
BP_RD_DIM2_10	0	1	3	**True**	True	14.894
BP_RD_DIM2_10	0	2	1	**True**	True	19.149
BP_RD_DIM2_10	1	1	3	**True**	True	23.404
BP_RD_DIM2_10	1	2	1	**True**	True	27.660
BP_RD_DIM2_11	0	1	3	**True**	True	31.034
BP_RD_DIM2_11	0	2	1	**True**	True	37.931
BP_RD_DIM2_11	1	1	3	**True**	True	27.586
BP_RD_DIM2_11	1	2	1	**True**	True	34.483
BP_RD_DIM2_12	0	1	3	False	False	8.537
BP_RD_DIM2_12	0	2	1	**True**	True	13.415
BP_RD_DIM2_12	1	1	3	False	False	8.537
BP_RD_DIM2_12	1	2	1	**True**	True	13.415
BP_RD_DIM2_13	0	1	3	False	**True**	11.842
BP_RD_DIM2_13	0	2	1	False	**True**	14.474
BP_RD_DIM2_13	1	1	3	**True**	True	9.211
BP_RD_DIM2_13	1	2	1	**True**	True	11.842
BP_RD_DIM2_14	0	1	2	**True**	True	50.000
BP_RD_DIM2_14	0	2	2	**True**	True	50.000
BP_RD_DIM2_14	1	1	2	**True**	True	35.000
BP_RD_DIM2_14	1	2	2	**True**	True	35.000
BP_RD_DIM2_15	0	1	3	**True**	True	40.741
BP_RD_DIM2_15	0	2	1	**True**	True	48.148
BP_RD_DIM2_15	1	1	3	False	False	22.222
BP_RD_DIM2_15	1	2	1	**True**	True	55.556
∅ [%]				78.333	88.333	21.992

TABLE 6.16: Results of the Nelder-Mead Adaptation for the randomly generated two-dimensional binpacking problems

Name	start index	replace rule	Stop NM	Global MP NM	Box-Local MP NM	Points NM [%]	Global MP NM-SD	Points NM-SD [%]
CRANE_N5_4	0	1	3	**True**	True	32.000	True	44.000
CRANE_N5_4	0	2	1	**True**	True	40.000	True	44.000
CRANE_N5_4	1	1	3	**True**	True	28.000	True	40.000
CRANE_N5_4	1	2	1	**True**	True	36.000	True	40.000
CRANE_N20_1	0	1	3	False	False	8.000	**True**	16.000
CRANE_N20_1	0	2	1	**True**	True	16.000	True	16.000
CRANE_N20_1	1	1	3	False	False	14.000	**True**	46.000
CRANE_N20_1	1	2	1	**True**	True	40.000	True	40.000
BP_DIM2_1	0	1	3	**True**	True	17.241	True	20.690
BP_DIM2_1	0	2	1	**True**	True	19.540	True	20.690
BP_DIM2_1	1	1	3	**True**	True	10.345	True	16.092
BP_DIM2_1	1	2	1	**True**	True	13.793	True	16.092
BP_DIM3_2	0	1	3	False	False	3.936	**True**	8.766
BP_DIM3_2	0	2	1	**True**	True	5.188	True	7.156
BP_DIM3_2	1	1	3	False	False	3.041	**True**	14.848
BP_DIM3_2	1	2	1	**True**	True	8.408	True	10.555
BP_DIM4_2	0	1	3	False	False	2.085	**True**	7.618
BP_DIM4_2	0	2	1	**True**	True	3.609	True	6.014
BP_DIM4_2	1	1	3	False	False	1.363	**True**	7.137
BP_DIM4_2	1	2	1	**True**	True	3.208	True	5.533
∅ [%]				70.000	70.000	15.288	100.000	21.360

TABLE 6.17: NM-SD for the modular system examples

these problems only have box-local optimal points that are global optimal points. For these examples, it is an advantage that we start with the heuristic method and only consider the Steepest Descent method after scheduling, as we only have to evaluate a total of 21% (NM-SD) or 24% (CS-SD) of the points instead of 30% (SD). For the three-dimensional randomly generated binpacking problem, we have a improvement from 20% (SD) to 14% (NM-SD, CS-SD). For the four-dimensional examples, we have similar results with an improvement of 5%, that means from 11% (SD) to 6% (NM-SD, CS-SD). The corresponding results can be found in Tables A.43-A.46.

There are minor improvements available. What is certainly the case in all problems, however, is that by executing the Steepest Descent method in the second step, we have the guaranteed output of a box-local minimal point, and in most cases we do not need more steps than if we only used the Steepest Descent method.

6.4 Conclusions

In this chapter, we have considered methods for minimizing discrete functions. The first was the Steepest Descent method in Section 6.1, which is based on the idea known from continuous optimization. This could be easily transferred to discrete functions. In addition, we adapted and examined in detail two methods known from derivative-free optimization for continuous functions. The Coordinate Search from Subsection 6.2.1 can easily be adapted to our case, whereas the Nelder-Mead method in Subsection 6.2.2 cannot be directly transferred. We have developed an adaptation for our case and tested it in many test instances. Problems in which the set X is a discrete box work surprisingly well (PyCUTEst with dimensions 2 and 4 and f_4^n), even though these discrete functions do not satisfy the discrete convexity requirements. Therefore, these methods are promising. For problems where the set

Name	p^0	α	Global MP CS	Box-Local MP CS	Points CS [%]	Global MP CS-SD	Points CS-SD [%]
CRANE_N5_4	(1,1)	1	**True**	True	36.000	True	44.000
CRANE_N5_4	(1,1)	2	False	False	36.000	**True**	56.000
CRANE_N5_4	(3,3)	1	**True**	True	32.000	True	40.000
CRANE_N5_4	(3,3)	2	False	False	40.000	**True**	60.000
CRANE_N20_1	(1,1)	1	**True**	True	14.000	True	16.000
CRANE_N20_1	(1,1)	2	**True**	True	16.000	True	20.000
CRANE_N20_1	(6,3)	1	False	False	30.000	**True**	44.000
CRANE_N20_1	(6,3)	2	False	False	36.000	**True**	50.000
BP_DIM2_1	(0,1)	1	**True**	True	18.391	True	20.690
BP_DIM2_1	(0,1)	2	**True**	True	14.943	True	19.540
BP_DIM2_1	(5,4)	1	False	False	12.644	**True**	20.690
BP_DIM2_1	(5,4)	2	False	False	13.793	**True**	21.839
BP_DIM3_2	(0,0,1)	1	**True**	True	4.472	True	6.798
BP_DIM3_2	(0,0,1)	2	**True**	True	4.472	True	6.619
BP_DIM3_2	(5,4,1)	1	False	False	3.936	**True**	14.848
BP_DIM3_2	(5,4,1)	2	False	False	4.651	**True**	10.197
BP_DIM4_2	(0,0,0,1)	1	**True**	True	2.566	True	5.453
BP_DIM4_2	(0,0,0,1)	2	False	False	2.005	**True**	7.939
BP_DIM4_2	(2,2,2,2)	1	False	False	1.283	**True**	11.708
BP_DIM4_2	(2,2,2,2)	2	**True**	True	2.486	True	5.774
∅ [%]			50.000	50.000	16.282	100.000	24.105

TABLE 6.18: CS-SD for the modular system examples

X is "very thin" at the boundary, the Nelder-Mead variant does not work very well. However, the results could be improved by subsequently executing the Steepest Descent method with a warm start. It appears that the heuristic methods (NM and CS) are more suitable for problems where the feasible set is a discrete box. For other cases, further research is needed to determine how to better deal with "problematic boundaries". In summary, however, in this chapter we were able to present an adaptation of the well-known Nelder-Mead method for discrete functions, which is promising overall.

Chapter 7

Conclusions and Directions for Future Research

This work demonstrates the utility of mathematical optimization in modeling modular systems and determining their optimal configurations. To the best of our knowledge, this approach has not previously been applied in this scope, particularly not in settings where the number of variants for components remains open. Subsequent to a moderate amount of modeling effort, the resulting problem can be formulated as a standard mixed-integer optimization problem and passed to a suitable solver without further difficulty.

However, it has been observed that certain problem instances pose substantial challenges to modern solvers. This phenomenon was particularly evident in the crane bridge application, where functions, such as the load capacity function, initially had to be reformulated in a manner that was compatible with the solver. This finding suggests a significant opportunity for additional enhancements, whether through the development of alternative formulations or through a more extensive investigation of solver techniques and parameter tuning.

The decomposition approach investigated in this work proved to be useful as well. Although this approach does not necessarily result in significantly accelerated runtimes, it does provide a more clear and understandable structural interpretation of the original problem. Furthermore, this perspective enables the examination of the setting through the framework of parametric optimization and discrete functions that emerge from it. This connection has not been previously explored in the existing literature and appears to promise significant advancements.

A major advantage for the subsequent minimization of discrete functions and thus for identifying optimal configurations of the modular system would arise if these discrete functions satisfied discrete convexity properties. In our setting, however, the function values are not given directly, they have to be obtained by solving a parametric optimization problem at every point in the domain. As a result, verifying discrete convexity is computationally demanding and no immediate benefit is gained. A central idea of this work was therefore to identify discrete convexity without explicitly solving all parametric subproblems. This could not be achieved, as even the simplest problems posed considerable difficulties and the idea was not pursued further. The binpacking examples were intended to create an elementary test case that illustrates the underlying mechanisms, yet even there small changes in the input parameters turned a discrete convex function into a nonconvex one. This area therefore offers the greatest potential for further research within the scope of this work, for instance, through the development of even simpler examples or by further investigating the theory of parametric optimization.

Despite these challenges, the problems studied here possess interesting “almost convex” structures, which naturally motivate the use of numerical search methods. In many cases, these methods performed well. In particular, it was interesting to observe that the Nelder–Mead method, which was originally designed for continuous unconstrained optimization problems and requiring no derivative information, could be adapted in a straightforward manner to the minimization of box-constrained discrete functions and yielded good results. Since convexity could not be established in general, some of the considered methods remain heuristic, but they nevertheless provided good results.

Throughout this work, a large number of experiments were conducted, and many of the fundamental concepts and insights emerged directly from these experimental results. We were able to establish a solid mathematical framework to understand the observed phenomena. Several aspects identified along the way promise valuable opportunities for continued research and further development.

Appendix A

Complementary Material for the Computational Studies

A.1 Complementary Material for the Crane Bridge Modular System

A.1.1 Implementable Optimization Problem

In addition to $\overline{P}^{crane}$, we will give here all the details of the Python implementation with the package *gurobipy*. We start with the reformulations for the rounding constraints.

Rounding

Terms in the form of $f = \lfloor x \rfloor$ with variables $f \in \mathbb{Z}$ and $x \in \mathbb{R}$ can be modeled by

$$f \in \mathbb{Z}, \quad x - 1 + \varepsilon_{round} \leq f \leq x$$

with a tolerance $\varepsilon_{round} > 0$. We need this tolerance since "$>$" is not a feasible constraint in gurobipy. In the set $Z(v, \xi)$ of $\overline{P}^{crane}$ we have to reformulate $\left\lfloor \frac{L^\ell}{2l_j^S} \right\rfloor$, so we introduce the variables $f_j^\ell \in \mathbb{Z}$ with

$$\frac{L^\ell}{2l_j^S} - 1 + \varepsilon_{round} \leq f_j^\ell \leq \frac{L^\ell}{2l_j^S}.$$

If we multiply the equations with $2l_j^S$, which is nonnegative by other constraints, we obtain quadratic inequalities.

Next, we introduce necessary additional binary variables:

Additional Binary Variables

We need additional binary variables for the indicator constraints and for the logical constraints in the objective function. In these cases, the same variables can be used, as in both cases conditions $z_{i,\ell}^P > 0$ and $z_{j,\ell}^S > 0$ and their combination is needed.

To use indicator constraints in Gurobi, the following structure must be required (see [11]):

$$b = 1 \Rightarrow a^\top x \overset{\leq}{\underset{\geq}{=}} c$$

with variables $b \in \mathbb{B}$, $x \in \mathbb{R}^n$ and input data $a \in \mathbb{R}^n$, $c \in \mathbb{R}$. So we introduce binary variables $b^P_{i,\ell}$ and $b^S_{j,\ell}$ with $i = i = 1, \dots, \overline{n}$, $j = 1, \dots, \overline{m}$, $\ell = 1, \dots, N$ that should fulfill the following:

$$b^P_{i,\ell} = \begin{cases} 1, & \text{if } z^P_{i,\ell} > 0, \\ 0, & \text{else,} \end{cases}$$

$$b^S_{j,\ell} = \begin{cases} 1, & \text{if } z^S_{j,\ell} > 0, \\ 0, & \text{else.} \end{cases}$$

With these variables, conditions $z^P_{i,\ell} > 0$ and $z^S_{j,\ell}$ can be replaced and we also replace the number variable in (4.3) and (4.4) by the corresponding binary variable. We also need a binary variable, which is one if and only if we choose the combination of profile and sheet (i, j) for the crane bridge ℓ, which we denote with $b^{PS}_{i,j,\ell}$. To model the required property, we set

$$b^{PS}_{i,j,\ell} = b^P_{i,\ell} b^S_{j,\ell},$$

which is a feasible constraint in gurobipy. The constraints (4.7) can be reformulated in

$$\sum_{i=1}^{\overline{n}} b^P_{i,\ell} = 1, \quad \ell = 1, \dots, N, \tag{A.1}$$

$$\sum_{j=1}^{\overline{m}} b^S_{j,\ell} = 1, \quad \ell = 1, \dots, N, \tag{A.2}$$

and the sum with the logical constraints in the objective function can be replaced by

$$\sum_{i,j,\ell=1}^{\overline{n},\overline{m},N} b^{PS}_{i,j,\ell}(M(P^i, S^j) - M^\ell).$$

At this point, it is clear that we also need a linear reformulation of the load capacity function M, so that the above function can be modeled as a quadratic function.

Equations (A.1) and (A.2) ensure that only one profile variant i and one sheet variant j is chosen for crane bridge ℓ. With that, only one $b^{PS}_{i,j,\ell}$ is one for the crane bridge ℓ. Since we multiply the binary variable $b^{PS}_{i,j,\ell}$ in the objective function (in the part with the load capacity and weight function), we do not have to explicitly require that the number variables or the geometry variables of the variant combinations not used have to be zero, since the term is multiplied by zero anyway.

We continue with the reformulation of the load capacity function M.

Load Capacity Function

The load capacity function was introduced in (3.8) by

$$M(P^i, S^j) = \frac{c_1}{L^\ell} \left(c_2 h^S_j + c_3 h^P_i + c_4 w^P_i + c_5 w^S_j - c_6 \left(\frac{h^S_j - 2h^P_i}{l^S_j} - \sqrt{3} \right)^2 \right),$$

which is not quadratic.

As in the indicator constraints, the right-hand side of the constraint has to be linear, and as we need a linear expression for M as written above, we reformulate M as a linear function. In general, this is possible for each rational function through suitable replacements. Of course, this procedure complicates the solution process, since we have many new variables and equality constraints. We introduce the two new variables $t^1_{i,j}$ and $t^2_{i,j} \in \mathbb{R}$ and require the following:

$$t^1_{i,j} l^S_j = h^S_j - 2h^P_i, \quad i = 1, \dots, \overline{n},\ j = 1, \dots, \overline{m},$$

and

$$t^2_{i,j} = \left(t^1_{i,j} - \sqrt{3}\right)^2, \quad i = 1, \dots, \overline{n},\ j = 1, \dots, \overline{m}.$$

This gives us the following linear-quadratic reformulation:

$$M(P^i, S^j, t^2_{i,j}) = \frac{c_1}{L^\ell}\left(c_2 h^S_j + c_3 h^P_i + c_4 w^P_i + c_5 w^S_j - c_6 t^2_{i,j}\right).$$

Then, we continue with the weight function.

Weight Function

Profiles and sheets are made of steel, which has a density of $\rho_{steel} = 7,85$ g/cm^3. The volume of a specific profile depends on the chosen sheet for the crane bridge, as can be seen in Figure 3.1. Thus, we consider a crane bridge with profile variant i and sheet variant j and obtain the volume of the profile:

$$vol^P_{i,j} = 2l^S_j \Bigg(\underbrace{w^P_i h^P_i}_{=:g^1_i} - \underbrace{(w^P_i - 2t^P_i)}_{=:g^2_i}\underbrace{(h^P_i - 2t^P_i)}_{=:g^3_i}\Bigg) \tag{A.3}$$

We get a difference because the profiles are hollow profiles. We can reformulate this with g^1_i, g^2_i and g^3_i as defined in (A.3) and $g^4_i := g^2_i g^3_i$ in the quadratic expression

$$vol^P_{i,j} = 2l^S_j (g^1_i - g^4_i).$$

For the sheets, we get (approximately) a volume of

$$vol^S_{i,j} = \underbrace{w^S_j t^S_j}_{=:g^8_j}\left(2h^P_i + \sqrt{\underbrace{(l^S_j)^2}_{=:g^6_j} + \underbrace{(h^S_j - 2h^P_i)^2}_{=:g^5_{i,j}}}\right), \tag{A.4}$$

which can be reformulated with new variables $g^5_{i,j}$, g^6_j, g^8_j as defined in (A.4) and together with

$$(g^7_{i,j})^2 = g^6_j + g^5_{i,j}$$

into a quadratic formulation

$$vol^S_{i,j} = g^8_j (2h^P_i + g^7_{i,j}).$$

If we assume all new variables as nonnegative, which is allowed for a feasible profile-sheet combination through the set F in Equation (4.12), $vol^S_{i,j}$ corresponds approximately to the volume of a sheet. To be precise, we ignore the very small overlap here and assume that the profile consists of three parts in the cross-section, the two short straight ones and the slanted one.

The weight of the crane bridge ℓ, which consists of profiles of variant i and sheets of variant j, is then given by

$$w(vol^P_{i,j}, vol^S_{i,j}, z^P_{i,\ell}, z^S_{j,\ell}) = \rho_{steel}\left(z^P_{i,\ell} vol^P_{i,j} + z^S_{j,\ell} vol^S_{i,j}\right). \tag{A.5}$$

We can reformulate the weight function w from (A.5) by replacement in a function, which can be handled by Gurobi. As in the reformulation of the load capacity function, many new variables and equation conditions arise here, which are very likely to affect the solution speed. As in the other parts of the objective function, we multiply the weight of the bridges with a cost coefficient $c_w > 0$.

Altogether, we obtain optimization problems that can be solved by Gurobi:

Complete Optimization Problems

The reformulations can be found in the following optimization problems $\overline{P}^{crane}_{complete}$ for the case without weight optimization. $P^{crane,w}_{complete}$ is the problem for the case with weight optimization.

We collect the binary variables $(b^P_{i,\ell})_{\substack{i=1,\dots,\overline{n},\\ \ell=1,\dots,N}}$, $(b^S_{j,\ell})_{\substack{j=1,\dots,\overline{m},\\ \ell=1,\dots,N}}$, and $(b^{PS}_{i,j,\ell})_{\substack{i=1,\dots,\overline{n},\ j=1,\dots,\overline{m},\\ \ell=1,\dots,N}}$ in a variable vector b

Then we collect the variables $(t^1_{i,j})_{\substack{i=1,\dots,\overline{n},\\ j=1,\dots,\overline{m}}}$ and $(t^2_{i,j})_{\substack{i=1,\dots,\overline{n},\\ j=1,\dots,\overline{m}}}$ in t.

All variables for the reformulation of the weight function are collected in a variable g of the corresponding length.

The problems are mixed-integer with a quadratic objective function and quadratic constraints and can be implemented with Gurobi.

$$
\begin{aligned}
\overline{P}^{crane,w}_{complete}: \quad \min_{b,f,t,v,\xi,z} \quad & c^P \sum_{i=1}^{\overline{n}} v_i^P + c^S \sum_{j=1}^{\overline{m}} v_j^S + c_d \sum_{i,j,\ell=1}^{\overline{n},\overline{m},N} b^{PS}_{i,j,\ell}(M(P^i, S^j, t^2_{i,j}) - M^\ell) + c_w \sum_{i,j,\ell=1}^{\overline{n},\overline{m},N} b^{PS}_{i,j,\ell} w(vol^P_{i,j}, vol^S_{i,j}, z^P_{i,\ell}, z^S_{j,\ell}) \\
\text{s.t.} \quad & \\
& \left.\begin{aligned}
&\left(h_i^P, t_i^P, w_i^P\right) \in [40,100] \times [6,6] \times [100,200], \\
&\left(h_j^S, l_j^S, t_j^S, w_j^S\right) \in [400,1000] \times [150,600] \times [6,6] \times [300,400],
\end{aligned}\right] =: X. \\
& \left.\begin{aligned}
&b^P_{i,\ell} \le v_i^P,\ b^S_{j,\ell} \le v_j^S, \\
&\sum_{i=1}^{\overline{n}} b^P_{i,\ell} = 1,\ \sum_{j=1}^{\overline{m}} b^S_{j,\ell} = 1, \\
&z^P_{i,\ell} \ge 0,\ z^S_{j,\ell} \ge 0, \\
&b^{PS}_{i,j,\ell} = 1 \ \Rightarrow\ z^P_{i,\ell} = 4 f_j^\ell - 2 \\
&b^S_{j\ell} = 1 \ \Rightarrow\ z^S_{j,\ell} = 2 f_j^\ell - 2 \\
&b^{PS}_{i,j,\ell} = 1 \ \Rightarrow\ M(P^i, S^j, t^2_{i,j}) \ge M^\ell \\
&b^{PS}_{i,j,\ell} = 1 \ \Rightarrow\ (P^i, S^j) \in F, \\
&L^\ell - 2l_j^S + \varepsilon_{round} 2 l_j^S \le 2 l_j^S f_j^\ell,\ 2 l_j^S f_j^\ell \le L^\ell, \\
&b^{PS}_{i,j,\ell} = b^P_{i,\ell} b^S_{j,\ell}, \\
&t^1_{i,j} l_j^S = h_j^S - 2h_i^P, \\
&t^2_{i,j} = \left(t^1_{i,j} - \sqrt{3}\right)^2, \\
&vol^P_{i,j} = 2 l_j^S (g_i^1 - g_i^4), \\
&g_i^1 = w_i^P h_i^P, \quad g_i^4 = g_i^2 g_i^3, \quad g_i^2 = w_i^P - 2t_i^P, \quad g_i^3 = h_i^P - 2t_i^P, \\
&vol^S_{i,j} = g_j^8 (2h_i^P + g^7_{i,j}) \\
&(g^7_{i,j})^2 = g_j^6 + g^5_{i,j}, \quad g_j^8 = w_j^S t_j^S, \quad g_j^6 = (l_j^S)^2, \quad g^5_{i,j} = (h_j^S - 2h_i^P)^2,
\end{aligned}\right] =: Z(b,f,g,t,v,\xi,z). \\
& b \in \mathbb{B}^{\overline{n}N + \overline{m}N + \overline{nn}N},\ f \in \mathbb{Z}^{\overline{m}N},\ g \in \mathbb{R}^{4\overline{n}+2\overline{m}+2\overline{nm}}_{\ge 0},\ t \in \mathbb{R}^{2\overline{nm}},\ v \in \mathbb{B}^{\overline{n}+\overline{m}},\ \xi \in \mathbb{R}^{3\overline{n}+4\overline{m}},\ z \in \mathbb{Z}^{\overline{n}N+\overline{m}N}
\end{aligned}
$$

$$
\begin{aligned}
\overline{P}^{crane}_{complete}: \quad \min_{b,t,v,\xi} \; & c^P \sum_{i=1}^{\overline{n}} v_i^P + c^S \sum_{j=1}^{\overline{m}} v_j^S + c_d \sum_{i,j,\ell=1}^{\overline{n},\overline{m},N} b^{PS}_{i,j,\ell}(M(P^i, S^j, t^2_{i,j}) - M^\ell) \\
& \text{s.t.} \\
& \left.\begin{array}{l}
\left(h_i^P, t_i^P, w_i^P\right) \in [40,100] \times [6,6] \times [100,200], \\
\left(h_j^S, l_j^S, t_j^S, w_j^S\right) \in [400,1000] \times [150,600] \times [6,6] \times [300,400]
\end{array}\right] =: X. \\
& \left.\begin{array}{l}
b^P_{i,\ell} \le v_i^P, \; b^S_{j,\ell} \le v_j^S, \\
\sum_{i=1}^{\overline{n}} b^P_{i,\ell} = 1, \; \sum_{j=1}^{\overline{m}} b^S_{j,\ell} = 1, \\
b^{PS}_{i,j,\ell} = 1 \;\Rightarrow\; M(P^i, S^j, t^2_{i,j}) \ge M^\ell \\
b^{PS}_{i,j,\ell} = 1 \;\Rightarrow\; (P^i, S^j) \in F, \\
b^{PS}_{i,j,\ell} = b^P_{i,\ell} b^S_{j,\ell}, \\
t^1_{i,j} l_j^S = h_j^S - 2h_i^P, \\
t^2_{i,j} = \left(t^1_{i,j} - \sqrt{3}\right)^2
\end{array}\right] =: Z(b,f,t,v,\xi). \\
& b \in \mathbb{B}^{\overline{n}N + \overline{m}N + \overline{nn}N}, \; t \in \mathbb{R}^{2\overline{nm}}, \; v \in \mathbb{B}^{\overline{n}+\overline{m}}, \; \xi \in \mathbb{R}^{3\overline{n}+4\overline{m}}
\end{aligned}
$$

A.1.2 Additional Input Data and Statistics

Small Examples

CRANE_N5_1	
ε_{round}	0.000010
optimality tol	0.001000
feasibility tol	0.001000
integer feasibility tol	0.001000
solution status	optimal

TABLE A.1: CRANE_N5_1: additional data

CRANE_N5_2	
ε_{round}	0.000010
optimality tol	0.010000
feasibility tol	0.010000
integer feasibility tol	0.001000
solution status	optimal

TABLE A.2: CRANE_N5_2: additional data

CRANE_N5_3	
ε_{round}	0.000010
optimality tol	0.001000
feasibility tol	0.001000
integer feasibility tol	0.001000
solution status	optimal

TABLE A.3: CRANE_N5_3: additional data

CRANE_N5_4	
ε_{round}	0.000010
optimality tol	0.001000
feasibility tol	0.001000
integer feasibility tol	0.005000
solution status	optimal

TABLE A.4: CRANE_N5_4: additional data

CRANE_N2_1	
$[\overline{n}, \overline{m}]$	[2, 2]
N	2
$[c^P, c^S]$	[10, 5]
c_d	10
c_w	100
runtime	2.83
total costs	330.6460
variant costs	30.00
deviation costs	201.70
weight costs	98.95
weight	0.99
# cont. var.	66
# integer var.	24

	i	j	$M\,[t]$	$M^\ell\,[t]$	$L^\ell\,[m]$	$w\,[t]$
ℓ						
1	1	2	5.00	5.00	13.0	0.97
2	2	1	25.17	5.00	1.0	0.02

TABLE A.5: CRANE_N2_1: input data, optimization results, mapping

	w^P	h^P
i		
1	100.00	69.17
2	100.00	40.00

	w^S	h^S	l^S
j			
1	300.00	400.00	490.63
2	300.00	1,000.00	590.91

TABLE A.6: CRANE_N2_1: details profiles and sheets in millimeters

CRANE_N2_1	
ε_{round}	0.000010
optimality tol	0.000100
feasibility tol	0.000100
integer feasibility tol	0.000100
solution status	optimal

TABLE A.7: CRANE_N2_1: additional data

Larger Crane Bridge Examples

CRANE_N20_1	
ε_{round}	0.000010
optimality tol	0.001000
feasibility tol	0.001000
integer feasibility tol	0.000001
solution status	optimal

TABLE A.8: CRANE_N20_1: additional data

CRANE_N20_2	
ε_{round}	0.000010
optimality tol	0.001000
feasibility tol	0.001000
integer feasibility tol	0.000001
solution status	optimal

TABLE A.9: CRANE_N20_2: additional data

CRANE_N20_3	
ε_{round}	0.000010
optimality tol	0.001000
feasibility tol	0.001000
integer feasibility tol	0.000001
solution status	optimal

TABLE A.10: CRANE_N20_3: additional data

CRANE_N20_4	
ε_{round}	0.000010
optimality tol	0.001000
feasibility tol	0.001000
integer feasibility tol	0.001000
solution status	optimal

TABLE A.11: CRANE_N20_4: additional data

A.2 Complementary Material for the Binpacking Modular System

A.2.1 One-Dimensional Binpacking Problem

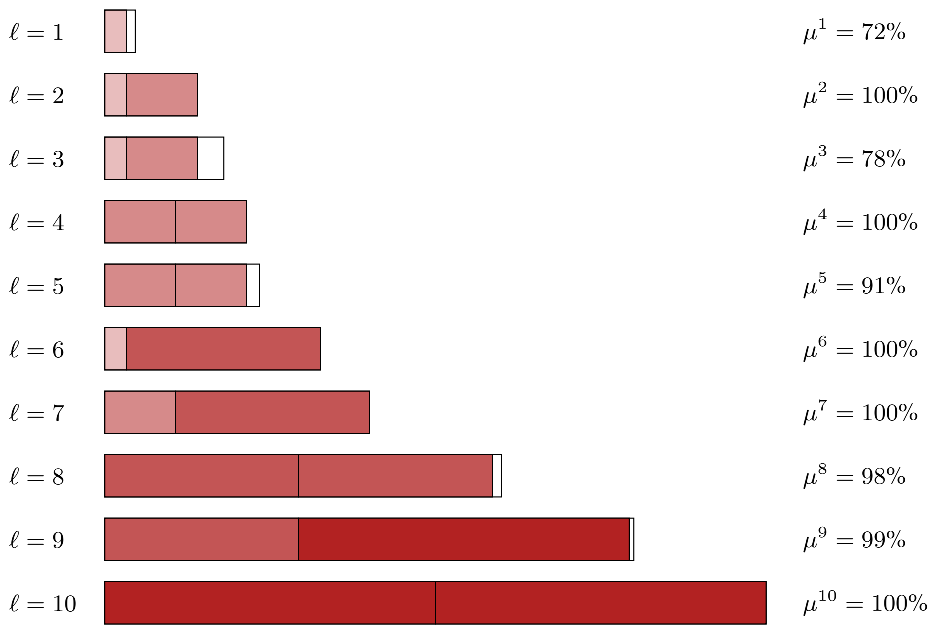

FIGURE A.1: BP_DIM1_2: mapping

ℓ	1	2	3	4	5	6	7	8	9	10
L^ℓ	23	70	90	107	117	163	200	300	400	500

TABLE A.12: BP_DIM1_2: lengths of bins

ℓ	1	2	3	4	5	6	7	8	9	10
L^ℓ	23	70	90	107	117	163	200	300	400	500

TABLE A.13: BP_DIM1_3: lengths of bins

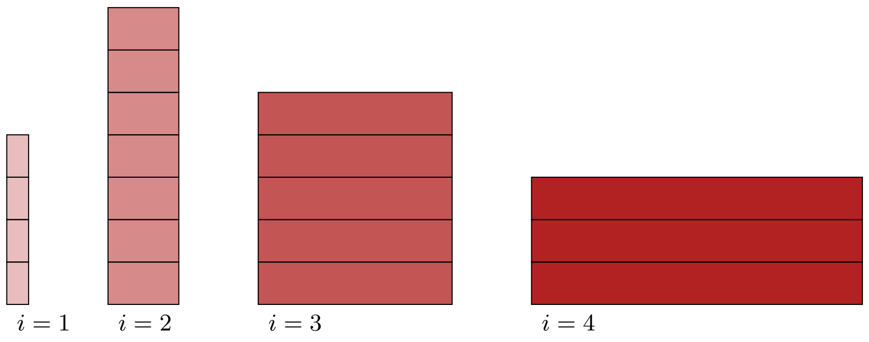

FIGURE A.2: BP_DIM1_2: modular system

ℓ	1	2	3	4	5	6	7	8	9	10
L^ℓ	23	70	90	107	117	163	200	300	400	500

TABLE A.14: BP_DIM1_4: lengths of bins

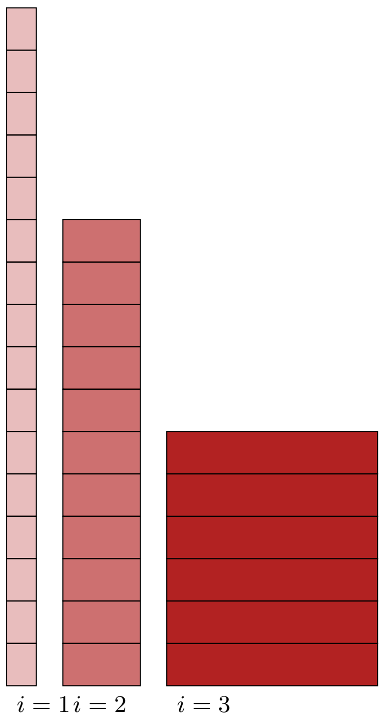

FIGURE A.3: BP_DIM1_3: modular system

ℓ	1	2	3	4	5	6	7	8	9	10
L^ℓ	23	70	90	107	117	163	200	300	400	500

TABLE A.15: BP_DIM1_5: lengths of bins

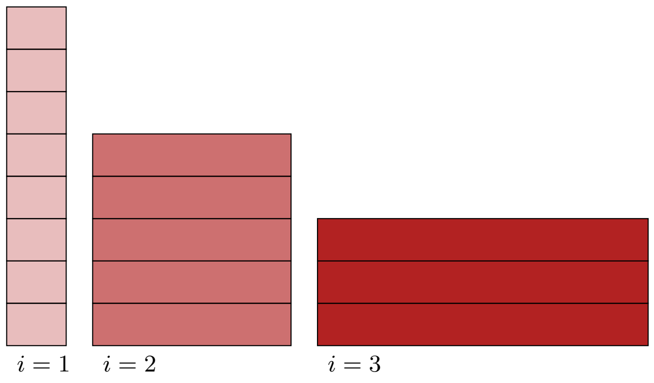

FIGURE A.4: BP_DIM1_4: modular system

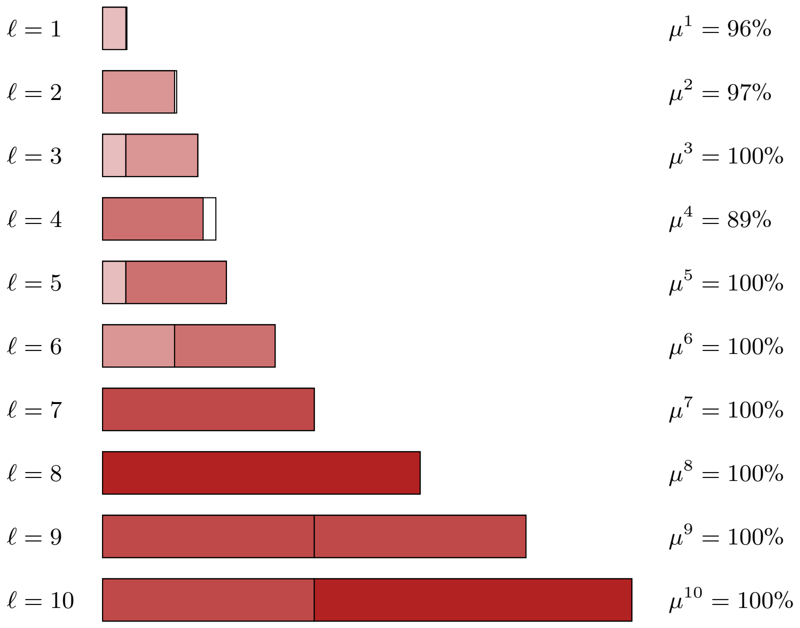

FIGURE A.5: BP_DIM1_5: mapping

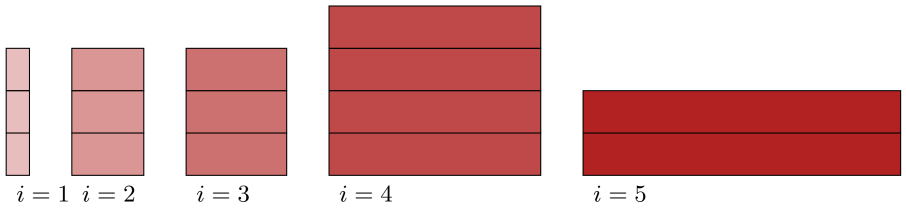

FIGURE A.6: BP_DIM1_5: modular system

A.2.2 Multi-Dimensional Binpacking Problem

ℓ	1	2	3	4	5	6	7	8	9	10
L^ℓ	23	70	90	107	117	163	200	300	400	500

TABLE A.16: BP_DIM2_1: lengths of bins

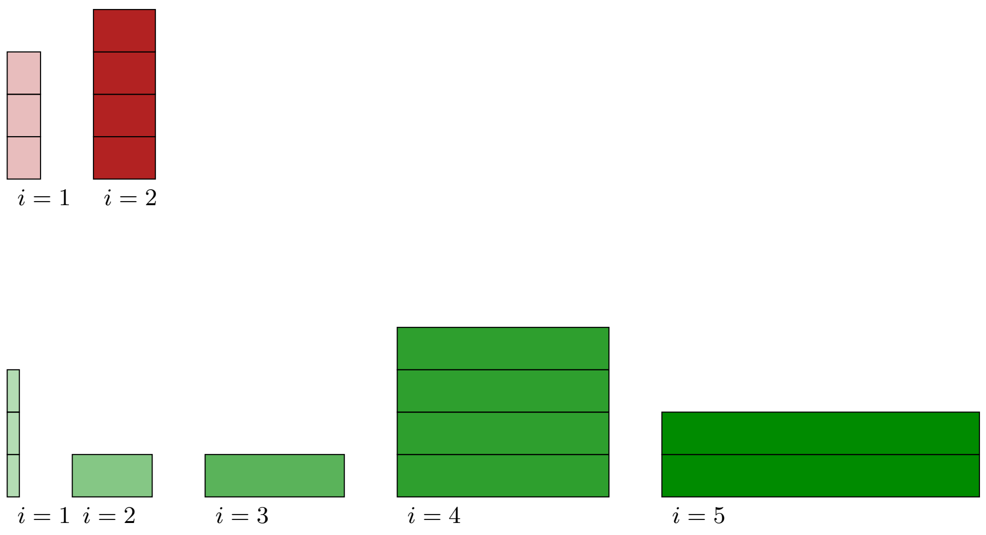

FIGURE A.7: BP_DIM2_1: modular system

ℓ	1	2	3	4	5	6	7	8	9	10
L^ℓ	23	70	90	107	117	163	200	300	400	500

TABLE A.17: BP_DIM3_1: lengths of bins

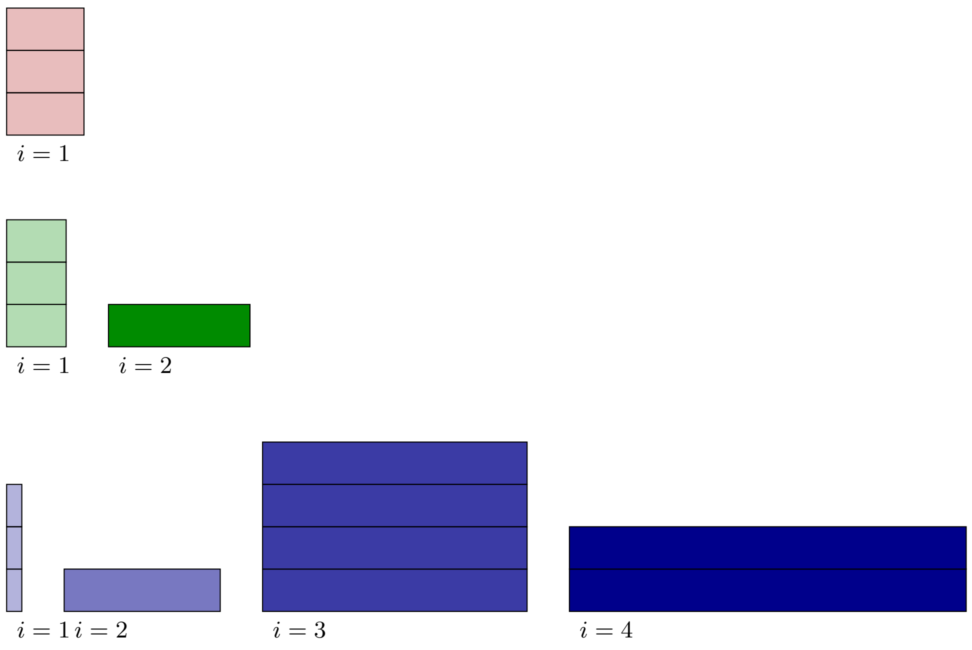

FIGURE A.8: BP_DIM3_1: modular system

BP_DIM3_2	
N	10
c_v	[10, 5, 1]
c_d	5
$\overline{k}$	[10, 7, 6]
τ	[15, 50, 75]
$\overline{o}$	2
runtime	12.63
total costs	24.00
variant costs	24
deviation costs	0.00

BP_DIM3_2	x_i^1	x_i^2	x_i^3
1	58.50	17.00	11.50
2	-	81.50	90.00
3	-	-	200.00
4	-	-	300.00

TABLE A.18: BP_DIM3_2: optimization results free model

ℓ	1	2	3	4	5	6	7	8	9	10
L^ℓ	23	70	90	107	117	163	200	300	400	500

TABLE A.19: BP_DIM4_1: lengths of bins

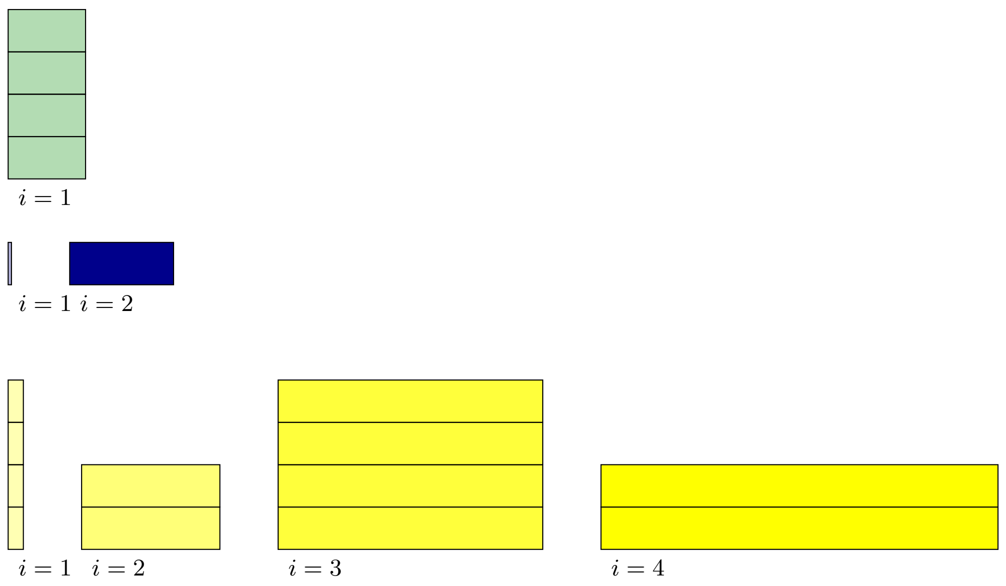

FIGURE A.9: BP_DIM4_1: modular system

BP_DIM4_1	
N	10
c_v	[10, 5, 1, 0.5]
c_d	5
$\overline{k}$	[5, 5, 5, 5]
τ	[15, 50, 75, 80]
$\overline{o}$	2
runtime	6.01
total costs	9.00
variant costs	9.00
deviation costs	0.00

BP_DIM4_1	x_i^1	x_i^2	x_i^3	x_i^4
1	-	58.50	31.50	11.50
2	-	-	107.00	104.50
3	-	-	-	200.00
4	-	-	-	300.00

FIGURE A.10: BP_DIM4_2: optimization results free model

A.2.3 Complementary Material for Subsection 5.2.2

ℓ	1	2	3	4	5
$Len(\ell)$	50	100	150	200	300

TABLE A.20: BP_DIM1_6: lengths of bins

BP_DIM1_6	
N	5
c_v	[10]
c_d	0.2
$\overline{k}$	[5]
τ	[60]
$\overline{o}$	2

TABLE A.21: BP_DIM1_6: Input data

ℓ	1	2	3	4	5	6	7	8
$Len(\ell)$	5	6	7	8	9	18	19	20

TABLE A.22: BP_DIM1_7-BP_DIM1_13: lengths of bins

BP_DIM1_7	
N	8
c_v	[15]
c_d	10
$\overline{k}$	[8]
τ	[2]
$\overline{o}$	[2]

TABLE A.23: BP_DIM1_7: Input data

BP_DIM3_2

$\widehat{k}$	$\widehat{obj}$	deviation costs	runtime $[s]$
(0,0,1)	3851.0	3850.0	0.01
(0,0,2)	1402.0	1400.0	0.03
(0,0,3)	553.0	550.0	0.05
(0,0,4)	424.0	420.0	0.05
(0,0,5)	425.0	420.0	0.04
(0,0,6)	426.0	420.0	0.07
(0,1,0)	3855.0	3850.0	0.01
(0,1,1)	1406.0	1400.0	0.03
(0,1,2)	557.0	550.0	0.11
(0,1,3)	243.0	235.0	0.18
(0,1,4)	99.0	90.0	0.13
(0,1,5)	100.0	90.0	0.14
(0,1,6)	101.0	90.0	0.05
(0,2,0)	1410.0	1400.0	0.03
(0,2,1)	561.0	550.0	0.12
(0,2,2)	247.0	235.0	0.3
(0,2,3)	70.5	57.5	0.28
(0,2,4)	29.0	15.0	0.14
(0,2,5)	30.0	15.0	0.22
(0,2,6)	31.0	15.0	0.12
(0,3,0)	565.0	550.0	0.05
(0,3,1)	251.0	235.0	0.26
(0,3,2)	74.5	57.5	0.48
(0,3,3)	33.0	15.0	0.34
(0,3,4)	26.5	7.5	0.31
(0,3,5)	27.5	7.5	0.38
(0,3,6)	28.5	7.5	0.14
(0,4,0)	257.5	237.5	0.04
(0,4,1)	78.5	57.5	0.43
(0,4,2)	37.0	15.0	0.52
(0,4,3)	30.5	7.5	0.29
(0,4,4)	31.5	7.5	0.27
(0,4,5)	32.5	7.5	0.33
(0,4,6)	41.0	15.0	0.14
(0,5,0)	225.0	200.0	0.05
(0,5,1)	56.0	30.0	0.34
(0,5,2)	34.5	7.5	0.18
(0,5,3)	35.5	7.5	0.24
(0,5,4)	36.5	7.5	0.19
(0,5,5)	37.5	7.5	0.21
(0,5,6)	-	-	-
(0,6,0)	230.0	200.0	0.07
(0,6,1)	61.0	30.0	0.31
(0,6,2)	39.5	7.5	0.42
(0,6,3)	40.5	7.5	0.21
(0,6,4)	41.5	7.5	0.44
(0,6,5)	50.0	15.0	0.12
(0,6,6)	-	-	-
(0,7,0)	235.0	200.0	0.05
(0,7,1)	66.0	30.0	0.19
(0,7,2)	44.5	7.5	0.16
(0,7,3)	45.5	7.5	0.28
(0,7,4)	54.0	15.0	0.09
(0,7,5)	-	-	-
(0,7,6)	-	-	-
(1,0,0)	3860.0	3850.0	0.01
(1,0,1)	1411.0	1400.0	0.03
(1,0,2)	562.0	550.0	0.12
(1,0,3)	248.0	235.0	0.17
(1,0,4)	104.0	90.0	0.14
(1,0,5)	105.0	90.0	0.16
(1,0,6)	106.0	90.0	0.05
(1,1,0)	1415.0	1400.0	0.03
(1,1,1)	566.0	550.0	0.17
(1,1,2)	252.0	235.0	0.69
(1,1,3)	68.0	50.0	0.54
(1,1,4)	31.5	12.5	0.8
(1,1,5)	32.5	12.5	0.79
(1,1,6)	33.5	12.5	0.3
(1,2,0)	570.0	550.0	0.11
(1,2,1)	256.0	235.0	0.6
(1,2,2)	72.0	50.0	1.19
(1,2,3)	33.0	10.0	1.49
(1,2,4)	24.0	0.0	0.21
(1,2,5)	25.0	0.0	0.12
(1,2,6)	26.0	0.0	0.12
(1,3,0)	260.0	235.0	0.31
(1,3,1)	76.0	50.0	0.79
(1,3,2)	37.0	10.0	2.33
(1,3,3)	28.0	0.0	0.54
(1,3,4)	29.0	0.0	0.1
(1,3,5)	30.0	0.0	0.08
(1,3,6)	31.0	0.0	0.02
(1,4,0)	87.5	57.5	0.38
(1,4,1)	41.0	10.0	1.01
(1,4,2)	32.0	0.0	0.3
(1,4,3)	33.0	0.0	0.11
(1,4,4)	34.0	0.0	0.03
(1,4,5)	35.0	0.0	0.07
(1,4,6)	36.0	0.0	0.1
(1,5,0)	65.0	30.0	0.3
(1,5,1)	36.0	0.0	0.15
(1,5,2)	37.0	0.0	0.09
(1,5,3)	38.0	0.0	0.08
(1,5,4)	39.0	0.0	0.06
(1,5,5)	40.0	0.0	0.07
(1,5,6)	-	-	-
(1,6,0)	70.0	30.0	0.15
(1,6,1)	41.0	0.0	0.32
(1,6,2)	42.0	0.0	0.12
(1,6,3)	43.0	0.0	0.11
(1,6,4)	44.0	0.0	0.04
(1,6,5)	45.0	0.0	0.1
(1,6,6)	-	-	-
(1,7,0)	75.0	30.0	0.33
(1,7,1)	46.0	0.0	0.33
(1,7,2)	47.0	0.0	0.11
(1,7,3)	48.0	0.0	0.05
(1,7,4)	49.0	0.0	0.07
(1,7,5)	-	-	-
(1,7,6)	-	-	-
(2,0,0)	1420.0	1400.0	0.03
(2,0,1)	571.0	550.0	0.1
(2,0,2)	257.0	235.0	0.35
(2,0,3)	73.0	50.0	0.43
(2,0,4)	36.5	12.5	0.53
(2,0,5)	37.5	12.5	0.19
(2,0,6)	38.5	12.5	0.27
(2,1,0)	575.0	550.0	0.1
(2,1,1)	261.0	235.0	0.67
(2,1,2)	77.0	50.0	0.98
(2,1,3)	38.0	10.0	2.16
(2,1,4)	29.0	0.0	0.18
(2,1,5)	30.0	0.0	0.11
(2,1,6)	31.0	0.0	0.09
(2,2,0)	265.0	235.0	0.58
(2,2,1)	81.0	50.0	1.05
(2,2,2)	42.0	10.0	5.97
(2,2,3)	33.0	0.0	0.84
(2,2,4)	34.0	0.0	0.11
(2,2,5)	35.0	0.0	0.1
(2,2,6)	36.0	0.0	0.07
(2,3,0)	85.0	50.0	0.67
(2,3,1)	46.0	10.0	3.89
(2,3,2)	37.0	0.0	0.5
(2,3,3)	38.0	0.0	0.17
(2,3,4)	39.0	0.0	0.07
(2,3,5)	40.0	0.0	0.1
(2,3,6)	41.0	0.0	0.06
(2,4,0)	50.0	10.0	0.61
(2,4,1)	41.0	0.0	0.41
(2,4,2)	42.0	0.0	0.08
(2,4,3)	43.0	0.0	0.08
(2,4,4)	44.0	0.0	0.07
(2,4,5)	45.0	0.0	0.05
(2,4,6)	46.0	0.0	0.13
(2,5,0)	45.0	0.0	0.55
(2,5,1)	46.0	0.0	0.12
(2,5,2)	47.0	0.0	0.1
(2,5,3)	48.0	0.0	0.04
(2,5,4)	49.0	0.0	0.09
(2,5,5)	50.0	0.0	0.06
(2,5,6)	-	-	-
(2,6,0)	50.0	0.0	0.31
(2,6,1)	51.0	0.0	0.07
(2,6,2)	52.0	0.0	0.06
(2,6,3)	53.0	0.0	0.07
(2,6,4)	54.0	0.0	0.05
(2,6,5)	55.0	0.0	0.05
(2,6,6)	-	-	-
(2,7,0)	55.0	0.0	0.25
(2,7,1)	56.0	0.0	0.08
(2,7,2)	57.0	0.0	0.05
(2,7,3)	58.0	0.0	0.07
(2,7,4)	59.0	0.0	0.03
(2,7,5)	-	-	-
(2,7,6)	-	-	-
(3,0,0)	580.0	550.0	0.05
(3,0,1)	266.0	235.0	0.45
(3,0,2)	82.0	50.0	0.6
(3,0,3)	43.0	10.0	0.91
(3,0,4)	34.0	0.0	0.2
(3,0,5)	35.0	0.0	0.11
(3,0,6)	36.0	0.0	0.04
(3,1,0)	270.0	235.0	0.34
(3,1,1)	86.0	50.0	0.99
(3,1,2)	47.0	10.0	6.34
(3,1,3)	38.0	0.0	0.55
(3,1,4)	39.0	0.0	0.08
(3,1,5)	40.0	0.0	0.08
(3,1,6)	41.0	0.0	0.05
(3,2,0)	90.0	50.0	0.59
(3,2,1)	51.0	10.0	4.42
(3,2,2)	42.0	0.0	0.59
(3,2,3)	43.0	0.0	0.06
(3,2,4)	44.0	0.0	0.13
(3,2,5)	45.0	0.0	0.1
(3,2,6)	46.0	0.0	0.08
(3,3,0)	55.0	10.0	1.45
(3,3,1)	46.0	0.0	0.13
(3,3,2)	47.0	0.0	0.13
(3,3,3)	48.0	0.0	0.09
(3,3,4)	49.0	0.0	0.06
(3,3,5)	50.0	0.0	0.09
(3,3,6)	51.0	0.0	0.06
(3,4,0)	50.0	0.0	0.29
(3,4,1)	51.0	0.0	0.16
(3,4,2)	52.0	0.0	0.12
(3,4,3)	53.0	0.0	0.07
(3,4,4)	54.0	0.0	0.09
(3,4,5)	55.0	0.0	0.05
(3,4,6)	56.0	0.0	0.03
(3,5,0)	55.0	0.0	0.21
(3,5,1)	56.0	0.0	0.09
(3,5,2)	57.0	0.0	0.11
(3,5,3)	58.0	0.0	0.07
(3,5,4)	59.0	0.0	0.03
(3,5,5)	60.0	0.0	0.11
(3,5,6)	-	-	-
(3,6,0)	60.0	0.0	0.06
(3,6,1)	61.0	0.0	0.06
(3,6,2)	62.0	0.0	0.09
(3,6,3)	63.0	0.0	0.05
(3,6,4)	64.0	0.0	0.06
(3,6,5)	65.0	0.0	0.1
(3,6,6)	-	-	-
(3,7,0)	65.0	0.0	0.07
(3,7,1)	66.0	0.0	0.08
(3,7,2)	67.0	0.0	0.09
(3,7,3)	68.0	0.0	0.08
(3,7,4)	69.0	0.0	0.06
(3,7,5)	-	-	-
(3,7,6)	-	-	-
(4,0,0)	275.0	235.0	0.14
(4,0,1)	91.0	50.0	0.41
(4,0,2)	52.0	10.0	1.56
(4,0,3)	43.0	0.0	0.58
(4,0,4)	44.0	0.0	0.14
(4,0,5)	45.0	0.0	0.06
(4,0,6)	46.0	0.0	0.08
(4,1,0)	95.0	50.0	0.67
(4,1,1)	56.0	10.0	4.53
(4,1,2)	47.0	0.0	0.67
(4,1,3)	48.0	0.0	0.15
(4,1,4)	49.0	0.0	0.09
(4,1,5)	50.0	0.0	0.1
(4,1,6)	51.0	0.0	0.03
(4,2,0)	60.0	10.0	1.53
(4,2,1)	51.0	0.0	0.54
(4,2,2)	52.0	0.0	0.11
(4,2,3)	53.0	0.0	0.08
(4,2,4)	54.0	0.0	0.08
(4,2,5)	55.0	0.0	0.1
(4,2,6)	56.0	0.0	0.03
(4,3,0)	55.0	0.0	0.74
(4,3,1)	56.0	0.0	0.21
(4,3,2)	57.0	0.0	0.08
(4,3,3)	58.0	0.0	0.1
(4,3,4)	59.0	0.0	0.1
(4,3,5)	60.0	0.0	0.05
(4,3,6)	61.0	0.0	0.06
(4,4,0)	60.0	0.0	0.15
(4,4,1)	61.0	0.0	0.09
(4,4,2)	62.0	0.0	0.09
(4,4,3)	63.0	0.0	0.04
(4,4,4)	64.0	0.0	0.05
(4,4,5)	65.0	0.0	0.12
(4,4,6)	66.0	0.0	0.03
(4,5,0)	65.0	0.0	0.11
(4,5,1)	66.0	0.0	0.09
(4,5,2)	67.0	0.0	0.03
(4,5,3)	68.0	0.0	0.04
(4,5,4)	69.0	0.0	0.04
(4,5,5)	70.0	0.0	0.08
(4,5,6)	-	-	-
(4,6,0)	70.0	0.0	0.06
(4,6,1)	71.0	0.0	0.08
(4,6,2)	72.0	0.0	0.06
(4,6,3)	73.0	0.0	0.11
(4,6,4)	74.0	0.0	0.08
(4,6,5)	75.0	0.0	0.02
(4,6,6)	-	-	-
(4,7,0)	75.0	0.0	0.08
(4,7,1)	76.0	0.0	0.07
(4,7,2)	77.0	0.0	0.09
(4,7,3)	78.0	0.0	0.08
(4,7,4)	79.0	0.0	0.03
(4,7,5)	-	-	-
(4,7,6)	-	-	-
(5,0,0)	125.0	75.0	0.21
(5,0,1)	61.0	10.0	1.35
(5,0,2)	52.0	0.0	0.81
(5,0,3)	53.0	0.0	0.18
(5,0,4)	54.0	0.0	0.1
(5,0,5)	55.0	0.0	0.08
(5,0,6)	56.0	0.0	0.03
(5,1,0)	65.0	10.0	1.24
(5,1,1)	56.0	0.0	0.33
(5,1,2)	57.0	0.0	0.15
(5,1,3)	58.0	0.0	0.09
(5,1,4)	59.0	0.0	0.07
(5,1,5)	60.0	0.0	0.1
(5,1,6)	61.0	0.0	0.04
(5,2,0)	60.0	0.0	0.88
(5,2,1)	61.0	0.0	0.32
(5,2,2)	62.0	0.0	0.15
(5,2,3)	63.0	0.0	0.06
(5,2,4)	64.0	0.0	0.04
(5,2,5)	65.0	0.0	0.06
(5,2,6)	66.0	0.0	0.01
(5,3,0)	65.0	0.0	0.14
(5,3,1)	66.0	0.0	0.13
(5,3,2)	67.0	0.0	0.06
(5,3,3)	68.0	0.0	0.11
(5,3,4)	69.0	0.0	0.07
(5,3,5)	70.0	0.0	0.08
(5,3,6)	71.0	0.0	0.11
(5,4,0)	70.0	0.0	0.08
(5,4,1)	71.0	0.0	0.09
(5,4,2)	72.0	0.0	0.1
(5,4,3)	73.0	0.0	0.04
(5,4,4)	74.0	0.0	0.05
(5,4,5)	75.0	0.0	0.04
(5,4,6)	76.0	0.0	0.11
(5,5,0)	75.0	0.0	0.08
(5,5,1)	76.0	0.0	0.1
(5,5,2)	77.0	0.0	0.05
(5,5,3)	78.0	0.0	0.05
(5,5,4)	79.0	0.0	0.04
(5,5,5)	80.0	0.0	0.07
(5,5,6)	-	-	-
(5,6,0)	80.0	0.0	0.06
(5,6,1)	81.0	0.0	0.1
(5,6,2)	82.0	0.0	0.1
(5,6,3)	83.0	0.0	0.04
(5,6,4)	84.0	0.0	0.03
(5,6,5)	85.0	0.0	0.16
(5,6,6)	-	-	-
(5,7,0)	85.0	0.0	0.07
(5,7,1)	86.0	0.0	0.12
(5,7,2)	87.0	0.0	0.02
(5,7,3)	88.0	0.0	0.16
(5,7,4)	89.0	0.0	0.07
(5,7,5)	-	-	-
(5,7,6)	-	-	-
(6,0,0)	70.0	10.0	0.61
(6,0,1)	61.0	0.0	0.2
(6,0,2)	62.0	0.0	0.12
(6,0,3)	63.0	0.0	0.13
(6,0,4)	64.0	0.0	0.02
(6,0,5)	65.0	0.0	0.09
(6,0,6)	66.0	0.0	0.08
(6,1,0)	65.0	0.0	0.25
(6,1,1)	66.0	0.0	0.09
(6,1,2)	67.0	0.0	0.06
(6,1,3)	68.0	0.0	0.11
(6,1,4)	69.0	0.0	0.09
(6,1,5)	70.0	0.0	0.03
(6,1,6)	71.0	0.0	0.04
(6,2,0)	70.0	0.0	0.23
(6,2,1)	71.0	0.0	0.11
(6,2,2)	72.0	0.0	0.05
(6,2,3)	73.0	0.0	0.1
(6,2,4)	74.0	0.0	0.04
(6,2,5)	75.0	0.0	0.04
(6,2,6)	76.0	0.0	0.05
(6,3,0)	75.0	0.0	0.12
(6,3,1)	76.0	0.0	0.1
(6,3,2)	77.0	0.0	0.06
(6,3,3)	78.0	0.0	0.05
(6,3,4)	79.0	0.0	0.05
(6,3,5)	80.0	0.0	0.05
(6,3,6)	81.0	0.0	0.08
(6,4,0)	80.0	0.0	0.1
(6,4,1)	81.0	0.0	0.07
(6,4,2)	82.0	0.0	0.12
(6,4,3)	83.0	0.0	0.1
(6,4,4)	84.0	0.0	0.11
(6,4,5)	85.0	0.0	0.05
(6,4,6)	86.0	0.0	0.07
(6,5,0)	85.0	0.0	0.11
(6,5,1)	86.0	0.0	0.06
(6,5,2)	87.0	0.0	0.08
(6,5,3)	88.0	0.0	0.05
(6,5,4)	89.0	0.0	0.07
(6,5,5)	90.0	0.0	0.09
(6,5,6)	-	-	-
(6,6,0)	90.0	0.0	0.11
(6,6,1)	91.0	0.0	0.06
(6,6,2)	92.0	0.0	0.01
(6,6,3)	93.0	0.0	0.08
(6,6,4)	94.0	0.0	0.05
(6,6,5)	95.0	0.0	0.07
(6,6,6)	-	-	-
(6,7,0)	95.0	0.0	0.06
(6,7,1)	96.0	0.0	0.11
(6,7,2)	97.0	0.0	0.1
(6,7,3)	98.0	0.0	0.05
(6,7,4)	99.0	0.0	0.11
(6,7,5)	-	-	-
(6,7,6)	-	-	-
(7,0,0)	70.0	0.0	0.31
(7,0,1)	71.0	0.0	0.38
(7,0,2)	72.0	0.0	0.15
(7,0,3)	73.0	0.0	0.08
(7,0,4)	74.0	0.0	0.1

TABLE A.24: BP_DIM3_2: Fixed Variants - Part 1

BP_DIM3_2			
$\widehat{k}$	$\widehat{obj}$	deviation costs	runtime $[s]$
(7,0,5)	75.0	0.0	0.09
(7,0,6)	76.0	0.0	0.11
(7,1,0)	75.0	0.0	0.2
(7,1,1)	76.0	0.0	0.11
(7,1,2)	77.0	0.0	0.09
(7,1,3)	78.0	0.0	0.07
(7,1,4)	79.0	0.0	0.05
(7,1,5)	80.0	0.0	0.08
(7,1,6)	81.0	0.0	0.1
(7,2,0)	80.0	0.0	0.12
(7,2,1)	81.0	0.0	0.12
(7,2,2)	82.0	0.0	0.08
(7,2,3)	83.0	0.0	0.04
(7,2,4)	84.0	0.0	0.03
(7,2,5)	85.0	0.0	0.04
(7,2,6)	86.0	0.0	0.05
(7,3,0)	85.0	0.0	0.11
(7,3,1)	86.0	0.0	0.14
(7,3,2)	87.0	0.0	0.05
(7,3,3)	88.0	0.0	0.01
(7,3,4)	89.0	0.0	0.13
(7,3,5)	90.0	0.0	0.07
(7,3,6)	91.0	0.0	0.07
(7,4,0)	90.0	0.0	0.1
(7,4,1)	91.0	0.0	0.04
(7,4,2)	92.0	0.0	0.12
(7,4,3)	93.0	0.0	0.03
(7,4,4)	94.0	0.0	0.11
(7,4,5)	95.0	0.0	0.08
(7,4,6)	96.0	0.0	0.08
(7,5,0)	95.0	0.0	0.1
(7,5,1)	96.0	0.0	0.04
(7,5,2)	97.0	0.0	0.04
(7,5,3)	98.0	0.0	0.08
(7,5,4)	99.0	0.0	0.04
(7,5,5)	100.0	0.0	0.07
(7,5,6)	-	-	-
(7,6,0)	100.0	0.0	0.04
(7,6,1)	101.0	0.0	0.09
(7,6,2)	102.0	0.0	0.1
(7,6,3)	103.0	0.0	0.11
(7,6,4)	104.0	0.0	0.07
(7,6,5)	105.0	0.0	0.07
(7,6,6)	-	-	-
(7,7,0)	105.0	0.0	0.07
(7,7,1)	106.0	0.0	0.08
(7,7,2)	107.0	0.0	0.1
(7,7,3)	108.0	0.0	0.06
(7,7,4)	109.0	0.0	0.18
(7,7,5)	-	-	-
(7,7,6)	-	-	-
(8,0,0)	80.0	0.0	0.13
(8,0,1)	81.0	0.0	0.17
(8,0,2)	82.0	0.0	0.1
(8,0,3)	83.0	0.0	0.07
(8,0,4)	84.0	0.0	0.04
(8,0,5)	85.0	0.0	0.03
(8,0,6)	86.0	0.0	0.19
(8,1,0)	85.0	0.0	0.22
(8,1,1)	86.0	0.0	0.08
(8,1,2)	87.0	0.0	0.06
(8,1,3)	88.0	0.0	0.13
(8,1,4)	89.0	0.0	0.01
(8,1,5)	90.0	0.0	0.14
(8,1,6)	91.0	0.0	0.11
(8,2,0)	90.0	0.0	0.11
(8,2,1)	91.0	0.0	0.14
(8,2,2)	92.0	0.0	0.1
(8,2,3)	93.0	0.0	0.05
(8,2,4)	94.0	0.0	0.06
(8,2,5)	95.0	0.0	0.11
(8,2,6)	96.0	0.0	0.13
(8,3,0)	95.0	0.0	0.11
(8,3,1)	96.0	0.0	0.11
(8,3,2)	97.0	0.0	0.07
(8,3,3)	98.0	0.0	0.11
(8,3,4)	99.0	0.0	0.07
(8,3,5)	100.0	0.0	0.08
(8,3,6)	101.0	0.0	0.18
(8,4,0)	100.0	0.0	0.1
(8,4,1)	101.0	0.0	0.06
(8,4,2)	102.0	0.0	0.08
(8,4,3)	103.0	0.0	0.05
(8,4,4)	104.0	0.0	0.11
(8,4,5)	105.0	0.0	0.05
(8,4,6)	131.0	25.0	0.3
(8,5,0)	105.0	0.0	0.09
(8,5,1)	106.0	0.0	0.03
(8,5,2)	107.0	0.0	0.07
(8,5,3)	108.0	0.0	0.09
(8,5,4)	109.0	0.0	0.09
(8,5,5)	110.0	0.0	0.14
(8,5,6)	-	-	-
(8,6,0)	110.0	0.0	0.06
(8,6,1)	111.0	0.0	0.08
(8,6,2)	112.0	0.0	0.08
(8,6,3)	113.0	0.0	0.13
(8,6,4)	114.0	0.0	0.17
(8,6,5)	-	-	-
(8,6,6)	-	-	-
(8,7,0)	115.0	0.0	0.11
(8,7,1)	116.0	0.0	0.05
(8,7,2)	117.0	0.0	0.08
(8,7,3)	118.0	0.0	0.06
(8,7,4)	-	-	-
(8,7,5)	-	-	-
(8,7,6)	-	-	-
(9,0,0)	90.0	0.0	0.13
(9,0,1)	91.0	0.0	0.13
(9,0,2)	92.0	0.0	0.09
(9,0,3)	93.0	0.0	0.11
(9,0,4)	94.0	0.0	0.04
(9,0,5)	95.0	0.0	0.09
(9,0,6)	96.0	0.0	0.04
(9,1,0)	95.0	0.0	0.13
(9,1,1)	96.0	0.0	0.08
(9,1,2)	97.0	0.0	0.14
(9,1,3)	98.0	0.0	0.08
(9,1,4)	99.0	0.0	0.07
(9,1,5)	100.0	0.0	0.11
(9,1,6)	101.0	0.0	0.06
(9,2,0)	100.0	0.0	0.05
(9,2,1)	101.0	0.0	0.12
(9,2,2)	102.0	0.0	0.14
(9,2,3)	103.0	0.0	0.09
(9,2,4)	104.0	0.0	0.05
(9,2,5)	105.0	0.0	0.09
(9,2,6)	106.0	0.0	0.1
(9,3,0)	105.0	0.0	0.12
(9,3,1)	106.0	0.0	0.15
(9,3,2)	107.0	0.0	0.14
(9,3,3)	108.0	0.0	0.1
(9,3,4)	109.0	0.0	0.05
(9,3,5)	110.0	0.0	0.07
(9,3,6)	111.0	0.0	0.22
(9,4,0)	110.0	0.0	0.13
(9,4,1)	111.0	0.0	0.11
(9,4,2)	112.0	0.0	0.03
(9,4,3)	113.0	0.0	0.01
(9,4,4)	114.0	0.0	0.07
(9,4,5)	115.0	0.0	0.18
(9,4,6)	-	-	-
(9,5,0)	115.0	0.0	0.14
(9,5,1)	116.0	0.0	0.07
(9,5,2)	117.0	0.0	0.07
(9,5,3)	118.0	0.0	0.04
(9,5,4)	119.0	0.0	0.08
(9,5,5)	120.0	0.0	0.2
(9,5,6)	-	-	-
(9,6,0)	120.0	0.0	0.09
(9,6,1)	121.0	0.0	0.1
(9,6,2)	122.0	0.0	0.04
(9,6,3)	123.0	0.0	0.11
(9,6,4)	124.0	0.0	0.1
(9,6,5)	-	-	-
(9,6,6)	-	-	-
(9,7,0)	125.0	0.0	0.12
(9,7,1)	126.0	0.0	0.03
(9,7,2)	127.0	0.0	0.1
(9,7,3)	133.0	5.0	0.27
(9,7,4)	-	-	-
(9,7,5)	-	-	-
(9,7,6)	-	-	-
(10,0,0)	100.0	0.0	0.16
(10,0,1)	101.0	0.0	0.13
(10,0,2)	102.0	0.0	0.09
(10,0,3)	103.0	0.0	0.06
(10,0,4)	104.0	0.0	0.12
(10,0,5)	105.0	0.0	0.11
(10,0,6)	291.0	185.0	0.3
(10,1,0)	105.0	0.0	0.13
(10,1,1)	106.0	0.0	0.09
(10,1,2)	107.0	0.0	0.14
(10,1,3)	108.0	0.0	0.05
(10,1,4)	109.0	0.0	0.12
(10,1,5)	110.0	0.0	0.08
(10,1,6)	221.0	110.0	0.28
(10,2,0)	110.0	0.0	0.1
(10,2,1)	111.0	0.0	0.1
(10,2,2)	112.0	0.0	0.06
(10,2,3)	113.0	0.0	0.1
(10,2,4)	114.0	0.0	0.07
(10,2,5)	115.0	0.0	0.04
(10,2,6)	176.0	60.0	0.23
(10,3,0)	115.0	0.0	0.12
(10,3,1)	116.0	0.0	0.11
(10,3,2)	117.0	0.0	0.06
(10,3,3)	118.0	0.0	0.05
(10,3,4)	119.0	0.0	0.11
(10,3,5)	120.0	0.0	0.02
(10,3,6)	-	-	-
(10,4,0)	120.0	0.0	0.08
(10,4,1)	121.0	0.0	0.04
(10,4,2)	122.0	0.0	0.12
(10,4,3)	123.0	0.0	0.05
(10,4,4)	124.0	0.0	0.13
(10,4,5)	125.0	0.0	0.14
(10,4,6)	-	-	-
(10,5,0)	125.0	0.0	0.11
(10,5,1)	126.0	0.0	0.01
(10,5,2)	127.0	0.0	0.1
(10,5,3)	128.0	0.0	0.06
(10,5,4)	129.0	0.0	0.16
(10,5,5)	-	-	-
(10,5,6)	-	-	-
(10,6,0)	130.0	0.0	0.15
(10,6,1)	131.0	0.0	0.09
(10,6,2)	132.0	0.0	0.14
(10,6,3)	133.0	0.0	0.08
(10,6,4)	-	-	-
(10,6,5)	-	-	-
(10,6,6)	-	-	-
(10,7,0)	135.0	0.0	0.21
(10,7,1)	136.0	0.0	0.16
(10,7,2)	137.0	0.0	0.21
(10,7,3)	-	-	-
(10,7,4)	-	-	-
(10,7,5)	-	-	-
(10,7,6)	-	-	-

TABLE A.25: BP_DIM3_2: Fixed Variants - Part 2

BP_DIM4_2			
$\widehat{k}$	$\widehat{obj}$	deviation costs	runtime $[s]$
(0,0,0,1)	3850.5	3850.0	0.01
(0,0,0,2)	1401.0	1400.0	0.03
(0,0,0,3)	551.5	550.0	0.05
(0,0,0,4)	457.0	455.0	0.04
(0,0,0,5)	457.5	455.0	0.05
(0,0,1,0)	3851.0	3850.0	0.01
(0,0,1,1)	1401.5	1400.0	0.03
(0,0,1,2)	552.0	550.0	0.09
(0,0,1,3)	237.5	235.0	0.16
(0,0,1,4)	103.0	100.0	0.14
(0,0,1,5)	103.5	100.0	0.11
(0,0,2,0)	1402.0	1400.0	0.02
(0,0,2,1)	552.5	550.0	0.11
(0,0,2,2)	238.0	235.0	0.41
(0,0,2,3)	93.5	90.0	0.13
(0,0,2,4)	54.0	50.0	0.19
(0,0,2,5)	54.5	50.0	0.11
(0,0,3,0)	553.0	550.0	0.05
(0,0,3,1)	238.5	235.0	0.18
(0,0,3,2)	94.0	90.0	0.24
(0,0,3,3)	32.0	27.5	0.13
(0,0,3,4)	32.5	27.5	0.19
(0,0,3,5)	33.0	27.5	0.15
(0,0,4,0)	424.0	420.0	0.05
(0,0,4,1)	94.5	90.0	0.11
(0,0,4,2)	32.5	27.5	0.18
(0,0,4,3)	33.0	27.5	0.16
(0,0,4,4)	33.5	27.5	0.13
(0,0,4,5)	54.0	47.5	0.13
(0,0,5,0)	425.0	420.0	0.04
(0,0,5,1)	95.5	90.0	0.12
(0,0,5,2)	33.5	27.5	0.18
(0,0,5,3)	34.0	27.5	0.1
(0,0,5,4)	54.5	47.5	0.13
(0,0,5,5)	-	-	-
(0,1,0,0)	3855.0	3850.0	0.01
(0,1,0,1)	1405.5	1400.0	0.03
(0,1,0,2)	556.0	550.0	0.09
(0,1,0,3)	241.5	235.0	0.15
(0,1,0,4)	107.0	100.0	0.16
(0,1,0,5)	107.5	100.0	0.11
(0,1,1,0)	1406.0	1400.0	0.03
(0,1,1,1)	556.5	550.0	0.17
(0,1,1,2)	242.0	235.0	0.73
(0,1,1,3)	57.5	50.0	0.45
(0,1,1,4)	20.5	12.5	0.34
(0,1,1,5)	21.0	12.5	0.34
(0,1,2,0)	557.0	550.0	0.11
(0,1,2,1)	242.5	235.0	0.67
(0,1,2,2)	58.0	50.0	0.72
(0,1,2,3)	21.0	12.5	0.99
(0,1,2,4)	9.0	0.0	0.52
(0,1,2,5)	9.5	0.0	0.13
(0,1,3,0)	243.0	235.0	0.18
(0,1,3,1)	58.5	50.0	0.72
(0,1,3,2)	21.5	12.5	0.98
(0,1,3,3)	9.5	0.0	0.45
(0,1,3,4)	10.0	0.0	0.06
(0,1,3,5)	10.5	0.0	0.1
(0,1,4,0)	99.0	90.0	0.13
(0,1,4,1)	22.0	12.5	0.6
(0,1,4,2)	10.0	0.0	0.32
(0,1,4,3)	10.5	0.0	0.14
(0,1,4,4)	11.0	0.0	0.08
(0,1,4,5)	11.5	0.0	0.08
(0,1,5,0)	100.0	90.0	0.15
(0,1,5,1)	23.0	12.5	0.87
(0,1,5,2)	11.0	0.0	0.67
(0,1,5,3)	11.5	0.0	0.19
(0,1,5,4)	12.0	0.0	0.07
(0,1,5,5)	-	-	-
(0,2,0,0)	1410.0	1400.0	0.03
(0,2,0,1)	560.5	550.0	0.1
(0,2,0,2)	246.0	235.0	0.54
(0,2,0,3)	69.0	57.5	0.29
(0,2,0,4)	27.0	15.0	0.21
(0,2,0,5)	27.5	15.0	0.29
(0,2,1,0)	561.0	550.0	0.11
(0,2,1,1)	246.5	235.0	0.83
(0,2,1,2)	62.0	50.0	1.02
(0,2,1,3)	22.5	10.0	1.55
(0,2,1,4)	13.0	0.0	0.51
(0,2,1,5)	13.5	0.0	0.2
(0,2,2,0)	247.0	235.0	0.32
(0,2,2,1)	62.5	50.0	1.0
(0,2,2,2)	23.0	10.0	2.46
(0,2,2,3)	13.5	0.0	0.35
(0,2,2,4)	14.0	0.0	0.23
(0,2,2,5)	14.5	0.0	0.08
(0,2,3,0)	70.5	57.5	0.29
(0,2,3,1)	23.5	10.0	1.25
(0,2,3,2)	14.0	0.0	0.09
(0,2,3,3)	14.5	0.0	0.13
(0,2,3,4)	15.0	0.0	0.11
(0,2,3,5)	15.5	0.0	0.08
(0,2,4,0)	29.0	15.0	0.14
(0,2,4,1)	14.5	0.0	0.3
(0,2,4,2)	15.0	0.0	0.12
(0,2,4,3)	15.5	0.0	0.14
(0,2,4,4)	16.0	0.0	0.08
(0,2,4,5)	16.5	0.0	0.07
(0,2,5,0)	30.0	15.0	0.2
(0,2,5,1)	15.5	0.0	0.37
(0,2,5,2)	16.0	0.0	0.08
(0,2,5,3)	16.5	0.0	0.06
(0,2,5,4)	17.0	0.0	0.03
(0,2,5,5)	-	-	-
(0,3,0,0)	565.0	550.0	0.05
(0,3,0,1)	250.5	235.0	0.24
(0,3,0,2)	73.5	57.5	0.48
(0,3,0,3)	31.5	15.0	0.2
(0,3,0,4)	24.5	7.5	0.26
(0,3,0,5)	25.0	7.5	0.17
(0,3,1,0)	251.0	235.0	0.28
(0,3,1,1)	66.5	50.0	1.19
(0,3,1,2)	27.0	10.0	2.22
(0,3,1,3)	17.5	0.0	0.63
(0,3,1,4)	18.0	0.0	0.24
(0,3,1,5)	18.5	0.0	0.07
(0,3,2,0)	74.5	57.5	0.55
(0,3,2,1)	27.5	10.0	1.58
(0,3,2,2)	18.0	0.0	0.39
(0,3,2,3)	18.5	0.0	0.11
(0,3,2,4)	19.0	0.0	0.04
(0,3,2,5)	19.5	0.0	0.08
(0,3,3,0)	33.0	15.0	0.35
(0,3,3,1)	18.5	0.0	0.48
(0,3,3,2)	19.0	0.0	0.24
(0,3,3,3)	19.5	0.0	0.08
(0,3,3,4)	20.0	0.0	0.06
(0,3,3,5)	20.5	0.0	0.06
(0,3,4,0)	26.5	7.5	0.31
(0,3,4,1)	19.5	0.0	0.1
(0,3,4,2)	20.0	0.0	0.08
(0,3,4,3)	20.5	0.0	0.06
(0,3,4,4)	21.0	0.0	0.07
(0,3,4,5)	21.5	0.0	0.05
(0,3,5,0)	27.5	7.5	0.38
(0,3,5,1)	20.5	0.0	0.07
(0,3,5,2)	21.0	0.0	0.05
(0,3,5,3)	21.5	0.0	0.08
(0,3,5,4)	22.0	0.0	0.04
(0,3,5,5)	-	-	-
(0,4,0,0)	257.5	237.5	0.04
(0,4,0,1)	78.0	57.5	0.42
(0,4,0,2)	36.0	15.0	0.35
(0,4,0,3)	29.0	7.5	0.31
(0,4,0,4)	29.5	7.5	0.41
(0,4,0,5)	30.0	7.5	0.29
(0,4,1,0)	78.5	57.5	0.43
(0,4,1,1)	31.5	10.0	1.28
(0,4,1,2)	22.0	0.0	0.16
(0,4,1,3)	22.5	0.0	0.14
(0,4,1,4)	23.0	0.0	0.09
(0,4,1,5)	23.5	0.0	0.07
(0,4,2,0)	37.0	15.0	0.55
(0,4,2,1)	22.5	0.0	0.6
(0,4,2,2)	23.0	0.0	0.11
(0,4,2,3)	23.5	0.0	0.07
(0,4,2,4)	24.0	0.0	0.06
(0,4,2,5)	24.5	0.0	0.09
(0,4,3,0)	30.5	7.5	0.26
(0,4,3,1)	23.5	0.0	0.2
(0,4,3,2)	24.0	0.0	0.11
(0,4,3,3)	24.5	0.0	0.08
(0,4,3,4)	25.0	0.0	0.07
(0,4,3,5)	25.5	0.0	0.03
(0,4,4,0)	31.5	7.5	0.26
(0,4,4,1)	24.5	0.0	0.14
(0,4,4,2)	25.0	0.0	0.08
(0,4,4,3)	25.5	0.0	0.08
(0,4,4,4)	26.0	0.0	0.06
(0,4,4,5)	26.5	0.0	0.1
(0,4,5,0)	32.5	7.5	0.34
(0,4,5,1)	25.5	0.0	0.04
(0,4,5,2)	26.0	0.0	0.09
(0,4,5,3)	26.5	0.0	0.06
(0,4,5,4)	27.0	0.0	0.04
(0,4,5,5)	-	-	-
(0,5,0,0)	225.0	200.0	0.06
(0,5,0,1)	55.5	30.0	0.47
(0,5,0,2)	33.5	7.5	0.29
(0,5,0,3)	34.0	7.5	0.38
(0,5,0,4)	34.5	7.5	0.38
(0,5,0,5)	35.0	7.5	0.16
(0,5,1,0)	56.0	30.0	0.37
(0,5,1,1)	26.5	0.0	0.24
(0,5,1,2)	27.0	0.0	0.19
(0,5,1,3)	27.5	0.0	0.12
(0,5,1,4)	28.0	0.0	0.07
(0,5,1,5)	28.5	0.0	0.06
(0,5,2,0)	34.5	7.5	0.17
(0,5,2,1)	27.5	0.0	0.19
(0,5,2,2)	28.0	0.0	0.09
(0,5,2,3)	28.5	0.0	0.09
(0,5,2,4)	29.0	0.0	0.03
(0,5,2,5)	29.5	0.0	0.09
(0,5,3,0)	35.5	7.5	0.23
(0,5,3,1)	28.5	0.0	0.06
(0,5,3,2)	29.0	0.0	0.04
(0,5,3,3)	29.5	0.0	0.07
(0,5,3,4)	30.0	0.0	0.09
(0,5,3,5)	30.5	0.0	0.09
(0,5,4,0)	36.5	7.5	0.2
(0,5,4,1)	29.5	0.0	0.1
(0,5,4,2)	30.0	0.0	0.1
(0,5,4,3)	30.5	0.0	0.09
(0,5,4,4)	31.0	0.0	0.09
(0,5,4,5)	-	-	-
(0,5,5,0)	37.5	7.5	0.2
(0,5,5,1)	30.5	0.0	0.05
(0,5,5,2)	31.0	0.0	0.06
(0,5,5,3)	31.5	0.0	0.06
(0,5,5,4)	-	-	-
(0,5,5,5)	-	-	-
(1,0,0,0)	3860.0	3850.0	0.01
(1,0,0,1)	1410.5	1400.0	0.03
(1,0,0,2)	561.0	550.0	0.09
(1,0,0,3)	246.5	235.0	0.16
(1,0,0,4)	112.0	100.0	0.13
(1,0,0,5)	112.5	100.0	0.1
(1,0,1,0)	1411.0	1400.0	0.03
(1,0,1,1)	561.5	550.0	0.17
(1,0,1,2)	247.0	235.0	0.81
(1,0,1,3)	62.5	50.0	0.58
(1,0,1,4)	25.5	12.5	0.34
(1,0,1,5)	26.0	12.5	0.36
(1,0,2,0)	562.0	550.0	0.12
(1,0,2,1)	247.5	235.0	0.64
(1,0,2,2)	63.0	50.0	0.77
(1,0,2,3)	26.0	12.5	0.93
(1,0,2,4)	14.0	0.0	0.75
(1,0,2,5)	14.5	0.0	0.12
(1,0,3,0)	248.0	235.0	0.18
(1,0,3,1)	63.5	50.0	0.46
(1,0,3,2)	26.5	12.5	1.44
(1,0,3,3)	14.5	0.0	0.64
(1,0,3,4)	15.0	0.0	0.06
(1,0,3,5)	15.5	0.0	0.13
(1,0,4,0)	104.0	90.0	0.13
(1,0,4,1)	27.0	12.5	0.88
(1,0,4,2)	15.0	0.0	0.18
(1,0,4,3)	15.5	0.0	0.17
(1,0,4,4)	16.0	0.0	0.08
(1,0,4,5)	16.5	0.0	0.08
(1,0,5,0)	105.0	90.0	0.14
(1,0,5,1)	28.0	12.5	0.81
(1,0,5,2)	16.0	0.0	0.63
(1,0,5,3)	16.5	0.0	0.15
(1,0,5,4)	17.0	0.0	0.07
(1,0,5,5)	-	-	-
(1,1,0,0)	1415.0	1400.0	0.03
(1,1,0,1)	565.5	550.0	0.18
(1,1,0,2)	251.0	235.0	0.7
(1,1,0,3)	66.5	50.0	0.6
(1,1,0,4)	29.5	12.5	0.41
(1,1,0,5)	30.0	12.5	0.39
(1,1,1,0)	566.0	550.0	0.18
(1,1,1,1)	251.5	235.0	0.94
(1,1,1,2)	67.0	50.0	1.97
(1,1,1,3)	27.5	10.0	3.32
(1,1,1,4)	18.0	0.0	0.14
(1,1,1,5)	18.5	0.0	0.13
(1,1,2,0)	252.0	235.0	0.8
(1,1,2,1)	67.5	50.0	1.74
(1,1,2,2)	28.0	10.0	6.94
(1,1,2,3)	18.5	0.0	0.66
(1,1,2,4)	19.0	0.0	0.11
(1,1,2,5)	19.5	0.0	0.09
(1,1,3,0)	68.0	50.0	0.56
(1,1,3,1)	28.5	10.0	3.46
(1,1,3,2)	19.0	0.0	0.72
(1,1,3,3)	19.5	0.0	0.09
(1,1,3,4)	20.0	0.0	0.09
(1,1,3,5)	20.5	0.0	0.08
(1,1,4,0)	31.5	12.5	0.83
(1,1,4,1)	19.5	0.0	0.62
(1,1,4,2)	20.0	0.0	0.11
(1,1,4,3)	20.5	0.0	0.04
(1,1,4,4)	21.0	0.0	0.11
(1,1,4,5)	21.5	0.0	0.05
(1,1,5,0)	32.5	12.5	0.83
(1,1,5,1)	20.5	0.0	0.14
(1,1,5,2)	21.0	0.0	0.14
(1,1,5,3)	21.5	0.0	0.08
(1,1,5,4)	22.0	0.0	0.02
(1,1,5,5)	-	-	-
(1,2,0,0)	570.0	550.0	0.12
(1,2,0,1)	255.5	235.0	0.6
(1,2,0,2)	71.0	50.0	1.22
(1,2,0,3)	31.5	10.0	1.27
(1,2,0,4)	22.0	0.0	0.57
(1,2,0,5)	22.5	0.0	0.16
(1,2,1,0)	256.0	235.0	0.66
(1,2,1,1)	71.5	50.0	2.03
(1,2,1,2)	32.0	10.0	14.36
(1,2,1,3)	22.5	0.0	0.23
(1,2,1,4)	23.0	0.0	0.11
(1,2,1,5)	23.5	0.0	0.07
(1,2,2,0)	72.0	50.0	1.16
(1,2,2,1)	32.5	10.0	18.49
(1,2,2,2)	23.0	0.0	0.39
(1,2,2,3)	23.5	0.0	0.14
(1,2,2,4)	24.0	0.0	0.1
(1,2,2,5)	24.5	0.0	0.09
(1,2,3,0)	33.0	10.0	1.6
(1,2,3,1)	23.5	0.0	0.15
(1,2,3,2)	24.0	0.0	0.14
(1,2,3,3)	24.5	0.0	0.1
(1,2,3,4)	25.0	0.0	0.07
(1,2,3,5)	25.5	0.0	0.01
(1,2,4,0)	24.0	0.0	0.56
(1,2,4,1)	24.5	0.0	0.17
(1,2,4,2)	25.0	0.0	0.08
(1,2,4,3)	25.5	0.0	0.08
(1,2,4,4)	26.0	0.0	0.07
(1,2,4,5)	26.5	0.0	0.08
(1,2,5,0)	25.0	0.0	0.15
(1,2,5,1)	25.5	0.0	0.13
(1,2,5,2)	26.0	0.0	0.07
(1,2,5,3)	26.5	0.0	0.05
(1,2,5,4)	27.0	0.0	0.07
(1,2,5,5)	-	-	-
(1,3,0,0)	260.0	235.0	0.32
(1,3,0,1)	75.5	50.0	1.04
(1,3,0,2)	36.0	10.0	1.7
(1,3,0,3)	26.5	0.0	0.88
(1,3,0,4)	27.0	0.0	0.14
(1,3,0,5)	27.5	0.0	0.12
(1,3,1,0)	76.0	50.0	0.86
(1,3,1,1)	36.5	10.0	5.63
(1,3,1,2)	27.0	0.0	0.22
(1,3,1,3)	27.5	0.0	0.14
(1,3,1,4)	28.0	0.0	0.1
(1,3,1,5)	28.5	0.0	0.11
(1,3,2,0)	37.0	10.0	2.59
(1,3,2,1)	27.5	0.0	0.32
(1,3,2,2)	28.0	0.0	0.1
(1,3,2,3)	28.5	0.0	0.06
(1,3,2,4)	29.0	0.0	0.09
(1,3,2,5)	29.5	0.0	0.07
(1,3,3,0)	28.0	0.0	0.61
(1,3,3,1)	28.5	0.0	0.15
(1,3,3,2)	29.0	0.0	0.08
(1,3,3,3)	29.5	0.0	0.08
(1,3,3,4)	30.0	0.0	0.08
(1,3,3,5)	30.5	0.0	0.05
(1,3,4,0)	29.0	0.0	0.1
(1,3,4,1)	29.5	0.0	0.07
(1,3,4,2)	30.0	0.0	0.09
(1,3,4,3)	30.5	0.0	0.06
(1,3,4,4)	31.0	0.0	0.03
(1,3,4,5)	31.5	0.0	0.05
(1,3,5,0)	30.0	0.0	0.08
(1,3,5,1)	30.5	0.0	0.03
(1,3,5,2)	31.0	0.0	0.09
(1,3,5,3)	31.5	0.0	0.07
(1,3,5,4)	32.0	0.0	0.04
(1,3,5,5)	-	-	-
(1,4,0,0)	87.5	57.5	0.38
(1,4,0,1)	40.5	10.0	1.19
(1,4,0,2)	31.0	0.0	0.75
(1,4,0,3)	31.5	0.0	0.25
(1,4,0,4)	32.0	0.0	0.17
(1,4,0,5)	32.5	0.0	0.11
(1,4,1,0)	41.0	10.0	1.12
(1,4,1,1)	31.5	0.0	0.45
(1,4,1,2)	32.0	0.0	0.14
(1,4,1,3)	32.5	0.0	0.12
(1,4,1,4)	33.0	0.0	0.06
(1,4,1,5)	33.5	0.0	0.07
(1,4,2,0)	32.0	0.0	0.29
(1,4,2,1)	32.5	0.0	0.14
(1,4,2,2)	33.0	0.0	0.1
(1,4,2,3)	33.5	0.0	0.07
(1,4,2,4)	34.0	0.0	0.09
(1,4,2,5)	34.5	0.0	0.02
(1,4,3,0)	33.0	0.0	0.11
(1,4,3,1)	33.5	0.0	0.09
(1,4,3,2)	34.0	0.0	0.09
(1,4,3,3)	34.5	0.0	0.07
(1,4,3,4)	35.0	0.0	0.09
(1,4,3,5)	35.5	0.0	0.01
(1,4,4,0)	34.0	0.0	0.03
(1,4,4,1)	34.5	0.0	0.09
(1,4,4,2)	35.0	0.0	0.04
(1,4,4,3)	35.5	0.0	0.08
(1,4,4,4)	36.0	0.0	0.07
(1,4,4,5)	36.5	0.0	0.03
(1,4,5,0)	35.0	0.0	0.07
(1,4,5,1)	35.5	0.0	0.09
(1,4,5,2)	36.0	0.0	0.05
(1,4,5,3)	36.5	0.0	0.04
(1,4,5,4)	37.0	0.0	0.07
(1,4,5,5)	-	-	-
(1,5,0,0)	65.0	30.0	0.33
(1,5,0,1)	35.5	0.0	0.17
(1,5,0,2)	36.0	0.0	0.18
(1,5,0,3)	36.5	0.0	0.11
(1,5,0,4)	37.0	0.0	0.08
(1,5,0,5)	37.5	0.0	0.07
(1,5,1,0)	36.0	0.0	0.14
(1,5,1,1)	36.5	0.0	0.1
(1,5,1,2)	37.0	0.0	0.07
(1,5,1,3)	37.5	0.0	0.06
(1,5,1,4)	38.0	0.0	0.02
(1,5,1,5)	38.5	0.0	0.05
(1,5,2,0)	37.0	0.0	0.09
(1,5,2,1)	37.5	0.0	0.09
(1,5,2,2)	38.0	0.0	0.08
(1,5,2,3)	38.5	0.0	0.08
(1,5,2,4)	39.0	0.0	0.04
(1,5,2,5)	39.5	0.0	0.01
(1,5,3,0)	38.0	0.0	0.08
(1,5,3,1)	38.5	0.0	0.08
(1,5,3,2)	39.0	0.0	0.03
(1,5,3,3)	39.5	0.0	0.03
(1,5,3,4)	40.0	0.0	0.01
(1,5,3,5)	40.5	0.0	0.03
(1,5,4,0)	39.0	0.0	0.07
(1,5,4,1)	39.5	0.0	0.08
(1,5,4,2)	40.0	0.0	0.04
(1,5,4,3)	40.5	0.0	0.03
(1,5,4,4)	41.0	0.0	0.08
(1,5,4,5)	-	-	-
(1,5,5,0)	40.0	0.0	0.07
(1,5,5,1)	40.5	0.0	0.03
(1,5,5,2)	41.0	0.0	0.04
(1,5,5,3)	41.5	0.0	0.04
(1,5,5,4)	-	-	-
(1,5,5,5)	-	-	-
(2,0,0,0)	1420.0	1400.0	0.03
(2,0,0,1)	570.5	550.0	0.12
(2,0,0,2)	256.0	235.0	0.52
(2,0,0,3)	71.5	50.0	0.56
(2,0,0,4)	34.5	12.5	0.49
(2,0,0,5)	35.0	12.5	0.45
(2,0,1,0)	571.0	550.0	0.15
(2,0,1,1)	256.5	235.0	0.89
(2,0,1,2)	72.0	50.0	1.27
(2,0,1,3)	32.5	10.0	1.84
(2,0,1,4)	23.0	0.0	0.17
(2,0,1,5)	23.5	0.0	0.2
(2,0,2,0)	257.0	235.0	0.38
(2,0,2,1)	72.5	50.0	1.1
(2,0,2,2)	33.0	10.0	4.75
(2,0,2,3)	23.5	0.0	0.56
(2,0,2,4)	24.0	0.0	0.2
(2,0,2,5)	24.5	0.0	0.29
(2,0,3,0)	73.0	50.0	0.47
(2,0,3,1)	33.5	10.0	2.6
(2,0,3,2)	24.0	0.0	0.56
(2,0,3,3)	24.5	0.0	0.13
(2,0,3,4)	25.0	0.0	0.08
(2,0,3,5)	25.5	0.0	0.08
(2,0,4,0)	36.5	12.5	0.58
(2,0,4,1)	24.5	0.0	0.14
(2,0,4,2)	25.0	0.0	0.13
(2,0,4,3)	25.5	0.0	0.08
(2,0,4,4)	26.0	0.0	0.06
(2,0,4,5)	26.5	0.0	0.05
(2,0,5,0)	37.5	12.5	0.2
(2,0,5,1)	25.5	0.0	0.23
(2,0,5,2)	26.0	0.0	0.11
(2,0,5,3)	26.5	0.0	0.07
(2,0,5,4)	27.0	0.0	0.07
(2,0,5,5)	-	-	-
(2,1,0,0)	575.0	550.0	0.11
(2,1,0,1)	260.5	235.0	0.7
(2,1,0,2)	76.0	50.0	0.94
(2,1,0,3)	36.5	10.0	1.58
(2,1,0,4)	27.0	0.0	0.17
(2,1,0,5)	27.5	0.0	0.12
(2,1,1,0)	261.0	235.0	0.74
(2,1,1,1)	76.5	50.0	2.23
(2,1,1,2)	37.0	10.0	15.43
(2,1,1,3)	27.5	0.0	0.44
(2,1,1,4)	28.0	0.0	0.09
(2,1,1,5)	28.5	0.0	0.1
(2,1,2,0)	77.0	50.0	1.07
(2,1,2,1)	37.5	10.0	15.4
(2,1,2,2)	28.0	0.0	0.38
(2,1,2,3)	28.5	0.0	0.07
(2,1,2,4)	29.0	0.0	0.08
(2,1,2,5)	29.5	0.0	0.07
(2,1,3,0)	38.0	10.0	2.83
(2,1,3,1)	28.5	0.0	0.69
(2,1,3,2)	29.0	0.0	0.14
(2,1,3,3)	29.5	0.0	0.04
(2,1,3,4)	30.0	0.0	0.07
(2,1,3,5)	30.5	0.0	0.05
(2,1,4,0)	29.0	0.0	0.18
(2,1,4,1)	29.5	0.0	0.13
(2,1,4,2)	30.0	0.0	0.08
(2,1,4,3)	30.5	0.0	0.07
(2,1,4,4)	31.0	0.0	0.07
(2,1,4,5)	31.5	0.0	0.05
(2,1,5,0)	30.0	0.0	0.11
(2,1,5,1)	30.5	0.0	0.08
(2,1,5,2)	31.0	0.0	0.08
(2,1,5,3)	31.5	0.0	0.07
(2,1,5,4)	32.0	0.0	0.05
(2,1,5,5)	-	-	-
(2,2,0,0)	265.0	235.0	0.62
(2,2,0,1)	80.5	50.0	1.21
(2,2,0,2)	41.0	10.0	6.06
(2,2,0,3)	31.5	0.0	0.29
(2,2,0,4)	32.0	0.0	0.11
(2,2,0,5)	32.5	0.0	0.09
(2,2,1,0)	81.0	50.0	1.22
(2,2,1,1)	41.5	10.0	17.04
(2,2,1,2)	32.0	0.0	0.44
(2,2,1,3)	32.5	0.0	0.2
(2,2,1,4)	33.0	0.0	0.07
(2,2,1,5)	33.5	0.0	0.04
(2,2,2,0)	42.0	10.0	6.22
(2,2,2,1)	32.5	0.0	0.31
(2,2,2,2)	33.0	0.0	0.09
(2,2,2,3)	33.5	0.0	0.04
(2,2,2,4)	34.0	0.0	0.05

TABLE A.26: BP_DIM4_2: Fixed Variants - Part 1

BP_DIM4_2			
$\widehat{k}$	$\widehat{obj}$	deviation costs	runtime $[s]$
(2,2,2,5)	34.5	0.0	0.04
(2,2,3,0)	33.0	0.0	0.84
(2,2,3,1)	33.5	0.0	0.1
(2,2,3,2)	34.0	0.0	0.11
(2,2,3,3)	34.5	0.0	0.11
(2,2,3,4)	35.0	0.0	0.07
(2,2,3,5)	35.5	0.0	0.05
(2,2,4,0)	34.0	0.0	0.11
(2,2,4,1)	34.5	0.0	0.06
(2,2,4,2)	35.0	0.0	0.08
(2,2,4,3)	35.5	0.0	0.04
(2,2,4,4)	36.0	0.0	0.06
(2,2,4,5)	36.5	0.0	0.06
(2,2,5,0)	35.0	0.0	0.09
(2,2,5,1)	35.5	0.0	0.08
(2,2,5,2)	36.0	0.0	0.07
(2,2,5,3)	36.5	0.0	0.08
(2,2,5,4)	37.0	0.0	0.1
(2,2,5,5)	-	-	-
(2,3,0,0)	85.0	50.0	0.65
(2,3,0,1)	45.5	10.0	4.49
(2,3,0,2)	36.0	0.0	0.45
(2,3,0,3)	36.5	0.0	0.15
(2,3,0,4)	37.0	0.0	0.07
(2,3,0,5)	37.5	0.0	0.02
(2,3,1,0)	46.0	10.0	3.91
(2,3,1,1)	36.5	0.0	0.73
(2,3,1,2)	37.0	0.0	0.13
(2,3,1,3)	37.5	0.0	0.09
(2,3,1,4)	38.0	0.0	0.08
(2,3,1,5)	38.5	0.0	0.05
(2,3,2,0)	37.0	0.0	0.48
(2,3,2,1)	37.5	0.0	0.18
(2,3,2,2)	38.0	0.0	0.08
(2,3,2,3)	38.5	0.0	0.09
(2,3,2,4)	39.0	0.0	0.04
(2,3,2,5)	39.5	0.0	0.07
(2,3,3,0)	38.0	0.0	0.17
(2,3,3,1)	38.5	0.0	0.08
(2,3,3,2)	39.0	0.0	0.09
(2,3,3,3)	39.5	0.0	0.04
(2,3,3,4)	40.0	0.0	0.04
(2,3,3,5)	40.5	0.0	0.03
(2,3,4,0)	39.0	0.0	0.07
(2,3,4,1)	39.5	0.0	0.09
(2,3,4,2)	40.0	0.0	0.07
(2,3,4,3)	40.5	0.0	0.04
(2,3,4,4)	41.0	0.0	0.04
(2,3,4,5)	41.5	0.0	0.07
(2,3,5,0)	40.0	0.0	0.1
(2,3,5,1)	40.5	0.0	0.08
(2,3,5,2)	41.0	0.0	0.04
(2,3,5,3)	41.5	0.0	0.04
(2,3,5,4)	42.0	0.0	0.04
(2,3,5,5)	-	-	-
(2,4,0,0)	50.0	10.0	0.57
(2,4,0,1)	40.5	0.0	0.69
(2,4,0,2)	41.0	0.0	0.31
(2,4,0,3)	41.5	0.0	0.15
(2,4,0,4)	42.0	0.0	0.09
(2,4,0,5)	42.5	0.0	0.02
(2,4,1,0)	41.0	0.0	0.4
(2,4,1,1)	41.5	0.0	0.38
(2,4,1,2)	42.0	0.0	0.09
(2,4,1,3)	42.5	0.0	0.08
(2,4,1,4)	43.0	0.0	0.07
(2,4,1,5)	43.5	0.0	0.05
(2,4,2,0)	42.0	0.0	0.08
(2,4,2,1)	42.5	0.0	0.12
(2,4,2,2)	43.0	0.0	0.06
(2,4,2,3)	43.5	0.0	0.08
(2,4,2,4)	44.0	0.0	0.09
(2,4,2,5)	44.5	0.0	0.04
(2,4,3,0)	43.0	0.0	0.08
(2,4,3,1)	43.5	0.0	0.08
(2,4,3,2)	44.0	0.0	0.09
(2,4,3,3)	44.5	0.0	0.1
(2,4,3,4)	45.0	0.0	0.03
(2,4,3,5)	45.5	0.0	0.04
(2,4,4,0)	44.0	0.0	0.08
(2,4,4,1)	44.5	0.0	0.02
(2,4,4,2)	45.0	0.0	0.1
(2,4,4,3)	45.5	0.0	0.04
(2,4,4,4)	46.0	0.0	0.08
(2,4,4,5)	46.5	0.0	0.12
(2,4,5,0)	45.0	0.0	0.05
(2,4,5,1)	45.5	0.0	0.08
(2,4,5,2)	46.0	0.0	0.07
(2,4,5,3)	46.5	0.0	0.01
(2,4,5,4)	47.0	0.0	0.03
(2,4,5,5)	-	-	-
(2,5,0,0)	45.0	0.0	0.53
(2,5,0,1)	45.5	0.0	0.12
(2,5,0,2)	46.0	0.0	0.11
(2,5,0,3)	46.5	0.0	0.08
(2,5,0,4)	47.0	0.0	0.08
(2,5,0,5)	47.5	0.0	0.08
(2,5,1,0)	46.0	0.0	0.12
(2,5,1,1)	46.5	0.0	0.04
(2,5,1,2)	47.0	0.0	0.11
(2,5,1,3)	47.5	0.0	0.05
(2,5,1,4)	48.0	0.0	0.03
(2,5,1,5)	48.5	0.0	0.04
(2,5,2,0)	47.0	0.0	0.14
(2,5,2,1)	47.5	0.0	0.03
(2,5,2,2)	48.0	0.0	0.05
(2,5,2,3)	48.5	0.0	0.11
(2,5,2,4)	49.0	0.0	0.11
(2,5,2,5)	49.5	0.0	0.02
(2,5,3,0)	48.0	0.0	0.04
(2,5,3,1)	48.5	0.0	0.07
(2,5,3,2)	49.0	0.0	0.03
(2,5,3,3)	49.5	0.0	0.09
(2,5,3,4)	50.0	0.0	0.08
(2,5,3,5)	50.5	0.0	0.05
(2,5,4,0)	49.0	0.0	0.12
(2,5,4,1)	49.5	0.0	0.03
(2,5,4,2)	50.0	0.0	0.13
(2,5,4,3)	50.5	0.0	0.04
(2,5,4,4)	51.0	0.0	0.04
(2,5,4,5)	-	-	-
(2,5,5,0)	50.0	0.0	0.06
(2,5,5,1)	50.5	0.0	0.05
(2,5,5,2)	51.0	0.0	0.01
(2,5,5,3)	51.5	0.0	0.02
(2,5,5,4)	-	-	-
(2,5,5,5)	-	-	-
(3,0,0,0)	580.0	550.0	0.05
(3,0,0,1)	265.5	235.0	0.46
(3,0,0,2)	81.0	50.0	0.6
(3,0,0,3)	41.5	10.0	0.81
(3,0,0,4)	32.0	0.0	0.67
(3,0,0,5)	32.5	0.0	0.23
(3,0,1,0)	266.0	235.0	0.47
(3,0,1,1)	81.5	50.0	1.26
(3,0,1,2)	42.0	10.0	7.69
(3,0,1,3)	32.5	0.0	0.57
(3,0,1,4)	33.0	0.0	0.14
(3,0,1,5)	33.5	0.0	0.07
(3,0,2,0)	82.0	50.0	0.76
(3,0,2,1)	42.5	10.0	7.53
(3,0,2,2)	33.0	0.0	0.18
(3,0,2,3)	33.5	0.0	0.13
(3,0,2,4)	34.0	0.0	0.11
(3,0,2,5)	34.5	0.0	0.07
(3,0,3,0)	43.0	10.0	0.92
(3,0,3,1)	33.5	0.0	0.1
(3,0,3,2)	34.0	0.0	0.1
(3,0,3,3)	34.5	0.0	0.12
(3,0,3,4)	35.0	0.0	0.08
(3,0,3,5)	35.5	0.0	0.1
(3,0,4,0)	34.0	0.0	0.2
(3,0,4,1)	34.5	0.0	0.19
(3,0,4,2)	35.0	0.0	0.17
(3,0,4,3)	35.5	0.0	0.07
(3,0,4,4)	36.0	0.0	0.07
(3,0,4,5)	36.5	0.0	0.08
(3,0,5,0)	35.0	0.0	0.11
(3,0,5,1)	35.5	0.0	0.1
(3,0,5,2)	36.0	0.0	0.09
(3,0,5,3)	36.5	0.0	0.09
(3,0,5,4)	37.0	0.0	0.07
(3,0,5,5)	-	-	-
(3,1,0,0)	270.0	235.0	0.33
(3,1,0,1)	85.5	50.0	1.28
(3,1,0,2)	46.0	10.0	5.79
(3,1,0,3)	36.5	0.0	0.67
(3,1,0,4)	37.0	0.0	0.14
(3,1,0,5)	37.5	0.0	0.07
(3,1,1,0)	86.0	50.0	1.0
(3,1,1,1)	46.5	10.0	16.64
(3,1,1,2)	37.0	0.0	0.27
(3,1,1,3)	37.5	0.0	0.1
(3,1,1,4)	38.0	0.0	0.06
(3,1,1,5)	38.5	0.0	0.04
(3,1,2,0)	47.0	10.0	5.68
(3,1,2,1)	37.5	0.0	0.2
(3,1,2,2)	38.0	0.0	0.13
(3,1,2,3)	38.5	0.0	0.11
(3,1,2,4)	39.0	0.0	0.08
(3,1,2,5)	39.5	0.0	0.08
(3,1,3,0)	38.0	0.0	0.56
(3,1,3,1)	38.5	0.0	0.4
(3,1,3,2)	39.0	0.0	0.12
(3,1,3,3)	39.5	0.0	0.11
(3,1,3,4)	40.0	0.0	0.1
(3,1,3,5)	40.5	0.0	0.12
(3,1,4,0)	39.0	0.0	0.08
(3,1,4,1)	39.5	0.0	0.22
(3,1,4,2)	40.0	0.0	0.1
(3,1,4,3)	40.5	0.0	0.08
(3,1,4,4)	41.0	0.0	0.13
(3,1,4,5)	41.5	0.0	0.03
(3,1,5,0)	40.0	0.0	0.08
(3,1,5,1)	40.5	0.0	0.09
(3,1,5,2)	41.0	0.0	0.1
(3,1,5,3)	41.5	0.0	0.1
(3,1,5,4)	42.0	0.0	0.04
(3,1,5,5)	-	-	-
(3,2,0,0)	90.0	50.0	0.57
(3,2,0,1)	50.5	10.0	5.29
(3,2,0,2)	41.0	0.0	0.25
(3,2,0,3)	41.5	0.0	0.17
(3,2,0,4)	42.0	0.0	0.12
(3,2,0,5)	42.5	0.0	0.09
(3,2,1,0)	51.0	10.0	4.45
(3,2,1,1)	41.5	0.0	0.09
(3,2,1,2)	42.0	0.0	0.16
(3,2,1,3)	42.5	0.0	0.12
(3,2,1,4)	43.0	0.0	0.09
(3,2,1,5)	43.5	0.0	0.06
(3,2,2,0)	42.0	0.0	0.59
(3,2,2,1)	42.5	0.0	0.33
(3,2,2,2)	43.0	0.0	0.05
(3,2,2,3)	43.5	0.0	0.08
(3,2,2,4)	44.0	0.0	0.1
(3,2,2,5)	44.5	0.0	0.04
(3,2,3,0)	43.0	0.0	0.06
(3,2,3,1)	43.5	0.0	0.07
(3,2,3,2)	44.0	0.0	0.1
(3,2,3,3)	44.5	0.0	0.1
(3,2,3,4)	45.0	0.0	0.09
(3,2,3,5)	45.5	0.0	0.04
(3,2,4,0)	44.0	0.0	0.11
(3,2,4,1)	44.5	0.0	0.07
(3,2,4,2)	45.0	0.0	0.04
(3,2,4,3)	45.5	0.0	0.08
(3,2,4,4)	46.0	0.0	0.04
(3,2,4,5)	46.5	0.0	0.07
(3,2,5,0)	45.0	0.0	0.1
(3,2,5,1)	45.5	0.0	0.08
(3,2,5,2)	46.0	0.0	0.09
(3,2,5,3)	46.5	0.0	0.04
(3,2,5,4)	47.0	0.0	0.01
(3,2,5,5)	-	-	-
(3,3,0,0)	55.0	10.0	1.2
(3,3,0,1)	45.5	0.0	0.13
(3,3,0,2)	46.0	0.0	0.13
(3,3,0,3)	46.5	0.0	0.1
(3,3,0,4)	47.0	0.0	0.1
(3,3,0,5)	47.5	0.0	0.11
(3,3,1,0)	46.0	0.0	0.13
(3,3,1,1)	46.5	0.0	0.19
(3,3,1,2)	47.0	0.0	0.12
(3,3,1,3)	47.5	0.0	0.1
(3,3,1,4)	48.0	0.0	0.07
(3,3,1,5)	48.5	0.0	0.03
(3,3,2,0)	47.0	0.0	0.13
(3,3,2,1)	47.5	0.0	0.08
(3,3,2,2)	48.0	0.0	0.09
(3,3,2,3)	48.5	0.0	0.08
(3,3,2,4)	49.0	0.0	0.11
(3,3,2,5)	49.5	0.0	0.08
(3,3,3,0)	48.0	0.0	0.08
(3,3,3,1)	48.5	0.0	0.07
(3,3,3,2)	49.0	0.0	0.04
(3,3,3,3)	49.5	0.0	0.1
(3,3,3,4)	50.0	0.0	0.04
(3,3,3,5)	50.5	0.0	0.05
(3,3,4,0)	49.0	0.0	0.06
(3,3,4,1)	49.5	0.0	0.08
(3,3,4,2)	50.0	0.0	0.06
(3,3,4,3)	50.5	0.0	0.08
(3,3,4,4)	51.0	0.0	0.06
(3,3,4,5)	51.5	0.0	0.08
(3,3,5,0)	50.0	0.0	0.09
(3,3,5,1)	50.5	0.0	0.08
(3,3,5,2)	51.0	0.0	0.05
(3,3,5,3)	51.5	0.0	0.09
(3,3,5,4)	52.0	0.0	0.1
(3,3,5,5)	-	-	-
(3,4,0,0)	50.0	0.0	0.29
(3,4,0,1)	50.5	0.0	0.25
(3,4,0,2)	51.0	0.0	0.08
(3,4,0,3)	51.5	0.0	0.11
(3,4,0,4)	52.0	0.0	0.04
(3,4,0,5)	52.5	0.0	0.08
(3,4,1,0)	51.0	0.0	0.15
(3,4,1,1)	51.5	0.0	0.05
(3,4,1,2)	52.0	0.0	0.08
(3,4,1,3)	52.5	0.0	0.07
(3,4,1,4)	53.0	0.0	0.03
(3,4,1,5)	53.5	0.0	0.04
(3,4,2,0)	52.0	0.0	0.11
(3,4,2,1)	52.5	0.0	0.08
(3,4,2,2)	53.0	0.0	0.09
(3,4,2,3)	53.5	0.0	0.04
(3,4,2,4)	54.0	0.0	0.02
(3,4,2,5)	54.5	0.0	0.09
(3,4,3,0)	53.0	0.0	0.07
(3,4,3,1)	53.5	0.0	0.08
(3,4,3,2)	54.0	0.0	0.04
(3,4,3,3)	54.5	0.0	0.05
(3,4,3,4)	55.0	0.0	0.03
(3,4,3,5)	55.5	0.0	0.06
(3,4,4,0)	54.0	0.0	0.09
(3,4,4,1)	54.5	0.0	0.09
(3,4,4,2)	55.0	0.0	0.1
(3,4,4,3)	55.5	0.0	0.05
(3,4,4,4)	56.0	0.0	0.06
(3,4,4,5)	56.5	0.0	0.04
(3,4,5,0)	55.0	0.0	0.05
(3,4,5,1)	55.5	0.0	0.06
(3,4,5,2)	56.0	0.0	0.08
(3,4,5,3)	56.5	0.0	0.01
(3,4,5,4)	57.0	0.0	0.07
(3,4,5,5)	-	-	-
(3,5,0,0)	55.0	0.0	0.21
(3,5,0,1)	55.5	0.0	0.09
(3,5,0,2)	56.0	0.0	0.09
(3,5,0,3)	56.5	0.0	0.06
(3,5,0,4)	57.0	0.0	0.06
(3,5,0,5)	57.5	0.0	0.1
(3,5,1,0)	56.0	0.0	0.09
(3,5,1,1)	56.5	0.0	0.08
(3,5,1,2)	57.0	0.0	0.08
(3,5,1,3)	57.5	0.0	0.1
(3,5,1,4)	58.0	0.0	0.07
(3,5,1,5)	58.5	0.0	0.07
(3,5,2,0)	57.0	0.0	0.08
(3,5,2,1)	57.5	0.0	0.04
(3,5,2,2)	58.0	0.0	0.04
(3,5,2,3)	58.5	0.0	0.05
(3,5,2,4)	59.0	0.0	0.06
(3,5,2,5)	59.5	0.0	0.04
(3,5,3,0)	58.0	0.0	0.07
(3,5,3,1)	58.5	0.0	0.04
(3,5,3,2)	59.0	0.0	0.04
(3,5,3,3)	59.5	0.0	0.05
(3,5,3,4)	60.0	0.0	0.01
(3,5,3,5)	60.5	0.0	0.06
(3,5,4,0)	59.0	0.0	0.03
(3,5,4,1)	59.5	0.0	0.05
(3,5,4,2)	60.0	0.0	0.08
(3,5,4,3)	60.5	0.0	0.08
(3,5,4,4)	61.0	0.0	0.03
(3,5,4,5)	-	-	-
(3,5,5,0)	60.0	0.0	0.11
(3,5,5,1)	60.5	0.0	0.04
(3,5,5,2)	61.0	0.0	0.09
(3,5,5,3)	61.5	0.0	0.03
(3,5,5,4)	-	-	-
(3,5,5,5)	-	-	-
(4,0,0,0)	275.0	235.0	0.14
(4,0,0,1)	90.5	50.0	0.44
(4,0,0,2)	51.0	10.0	1.98
(4,0,0,3)	41.5	0.0	0.59
(4,0,0,4)	42.0	0.0	0.13
(4,0,0,5)	42.5	0.0	0.21
(4,0,1,0)	91.0	50.0	0.45
(4,0,1,1)	51.5	10.0	4.22
(4,0,1,2)	42.0	0.0	0.92
(4,0,1,3)	42.5	0.0	0.11
(4,0,1,4)	43.0	0.0	0.08
(4,0,1,5)	43.5	0.0	0.09
(4,0,2,0)	52.0	10.0	1.59
(4,0,2,1)	42.5	0.0	0.17
(4,0,2,2)	43.0	0.0	0.07
(4,0,2,3)	43.5	0.0	0.08
(4,0,2,4)	44.0	0.0	0.12
(4,0,2,5)	44.5	0.0	0.03
(4,0,3,0)	43.0	0.0	0.58
(4,0,3,1)	43.5	0.0	0.11
(4,0,3,2)	44.0	0.0	0.1
(4,0,3,3)	44.5	0.0	0.14
(4,0,3,4)	45.0	0.0	0.08
(4,0,3,5)	45.5	0.0	0.09
(4,0,4,0)	44.0	0.0	0.13
(4,0,4,1)	44.5	0.0	0.19
(4,0,4,2)	45.0	0.0	0.08
(4,0,4,3)	45.5	0.0	0.09
(4,0,4,4)	46.0	0.0	0.03
(4,0,4,5)	46.5	0.0	0.03
(4,0,5,0)	45.0	0.0	0.06
(4,0,5,1)	45.5	0.0	0.03
(4,0,5,2)	46.0	0.0	0.07
(4,0,5,3)	46.5	0.0	0.03
(4,0,5,4)	47.0	0.0	0.04
(4,0,5,5)	-	-	-
(4,1,0,0)	95.0	50.0	0.65
(4,1,0,1)	55.5	10.0	4.37
(4,1,0,2)	46.0	0.0	0.4
(4,1,0,3)	46.5	0.0	0.11
(4,1,0,4)	47.0	0.0	0.08
(4,1,0,5)	47.5	0.0	0.08
(4,1,1,0)	56.0	10.0	4.59
(4,1,1,1)	46.5	0.0	0.42
(4,1,1,2)	47.0	0.0	0.15
(4,1,1,3)	47.5	0.0	0.1
(4,1,1,4)	48.0	0.0	0.04
(4,1,1,5)	48.5	0.0	0.05
(4,1,2,0)	47.0	0.0	0.67
(4,1,2,1)	47.5	0.0	0.16
(4,1,2,2)	48.0	0.0	0.1
(4,1,2,3)	48.5	0.0	0.09
(4,1,2,4)	49.0	0.0	0.08
(4,1,2,5)	49.5	0.0	0.1
(4,1,3,0)	48.0	0.0	0.12
(4,1,3,1)	48.5	0.0	0.1
(4,1,3,2)	49.0	0.0	0.09
(4,1,3,3)	49.5	0.0	0.04
(4,1,3,4)	50.0	0.0	0.04
(4,1,3,5)	50.5	0.0	0.05
(4,1,4,0)	49.0	0.0	0.1
(4,1,4,1)	49.5	0.0	0.08
(4,1,4,2)	50.0	0.0	0.06
(4,1,4,3)	50.5	0.0	0.06
(4,1,4,4)	51.0	0.0	0.05
(4,1,4,5)	51.5	0.0	0.04
(4,1,5,0)	50.0	0.0	0.1
(4,1,5,1)	50.5	0.0	0.05
(4,1,5,2)	51.0	0.0	0.05
(4,1,5,3)	51.5	0.0	0.1
(4,1,5,4)	52.0	0.0	0.03
(4,1,5,5)	-	-	-
(4,2,0,0)	60.0	10.0	1.39
(4,2,0,1)	50.5	0.0	0.22
(4,2,0,2)	51.0	0.0	0.05
(4,2,0,3)	51.5	0.0	0.08
(4,2,0,4)	52.0	0.0	0.04
(4,2,0,5)	52.5	0.0	0.04
(4,2,1,0)	51.0	0.0	0.55
(4,2,1,1)	51.5	0.0	0.13
(4,2,1,2)	52.0	0.0	0.06
(4,2,1,3)	52.5	0.0	0.09
(4,2,1,4)	53.0	0.0	0.08
(4,2,1,5)	53.5	0.0	0.09
(4,2,2,0)	52.0	0.0	0.1
(4,2,2,1)	52.5	0.0	0.08
(4,2,2,2)	53.0	0.0	0.08
(4,2,2,3)	53.5	0.0	0.07
(4,2,2,4)	54.0	0.0	0.05
(4,2,2,5)	54.5	0.0	0.07
(4,2,3,0)	53.0	0.0	0.07
(4,2,3,1)	53.5	0.0	0.1
(4,2,3,2)	54.0	0.0	0.08
(4,2,3,3)	54.5	0.0	0.07
(4,2,3,4)	55.0	0.0	0.1
(4,2,3,5)	55.5	0.0	0.06
(4,2,4,0)	54.0	0.0	0.08
(4,2,4,1)	54.5	0.0	0.08
(4,2,4,2)	55.0	0.0	0.03
(4,2,4,3)	55.5	0.0	0.05
(4,2,4,4)	56.0	0.0	0.04
(4,2,4,5)	56.5	0.0	0.03
(4,2,5,0)	55.0	0.0	0.1
(4,2,5,1)	55.5	0.0	0.11
(4,2,5,2)	56.0	0.0	0.06
(4,2,5,3)	56.5	0.0	0.05
(4,2,5,4)	57.0	0.0	0.05
(4,2,5,5)	-	-	-
(4,3,0,0)	55.0	0.0	0.69
(4,3,0,1)	55.5	0.0	0.3
(4,3,0,2)	56.0	0.0	0.09
(4,3,0,3)	56.5	0.0	0.11
(4,3,0,4)	57.0	0.0	0.04
(4,3,0,5)	57.5	0.0	0.04
(4,3,1,0)	56.0	0.0	0.34
(4,3,1,1)	56.5	0.0	0.11
(4,3,1,2)	57.0	0.0	0.11
(4,3,1,3)	57.5	0.0	0.02
(4,3,1,4)	58.0	0.0	0.04
(4,3,1,5)	58.5	0.0	0.07
(4,3,2,0)	57.0	0.0	0.11
(4,3,2,1)	57.5	0.0	0.11
(4,3,2,2)	58.0	0.0	0.11
(4,3,2,3)	58.5	0.0	0.07
(4,3,2,4)	59.0	0.0	0.06
(4,3,2,5)	59.5	0.0	0.03
(4,3,3,0)	58.0	0.0	0.1
(4,3,3,1)	58.5	0.0	0.02
(4,3,3,2)	59.0	0.0	0.07
(4,3,3,3)	59.5	0.0	0.05
(4,3,3,4)	60.0	0.0	0.02
(4,3,3,5)	60.5	0.0	0.04
(4,3,4,0)	59.0	0.0	0.1
(4,3,4,1)	59.5	0.0	0.12
(4,3,4,2)	60.0	0.0	0.04
(4,3,4,3)	60.5	0.0	0.05
(4,3,4,4)	61.0	0.0	0.07
(4,3,4,5)	61.5	0.0	0.12
(4,3,5,0)	60.0	0.0	0.05
(4,3,5,1)	60.5	0.0	0.08
(4,3,5,2)	61.0	0.0	0.03
(4,3,5,3)	61.5	0.0	0.03
(4,3,5,4)	62.0	0.0	0.06
(4,3,5,5)	-	-	-
(4,4,0,0)	60.0	0.0	0.15
(4,4,0,1)	60.5	0.0	0.1
(4,4,0,2)	61.0	0.0	0.08
(4,4,0,3)	61.5	0.0	0.06
(4,4,0,4)	62.0	0.0	0.1
(4,4,0,5)	62.5	0.0	0.07
(4,4,1,0)	61.0	0.0	0.1
(4,4,1,1)	61.5	0.0	0.1
(4,4,1,2)	62.0	0.0	0.05
(4,4,1,3)	62.5	0.0	0.02
(4,4,1,4)	63.0	0.0	0.06
(4,4,1,5)	63.5	0.0	0.02
(4,4,2,0)	62.0	0.0	0.09
(4,4,2,1)	62.5	0.0	0.1
(4,4,2,2)	63.0	0.0	0.11
(4,4,2,3)	63.5	0.0	0.06
(4,4,2,4)	64.0	0.0	0.03
(4,4,2,5)	64.5	0.0	0.1
(4,4,3,0)	63.0	0.0	0.04
(4,4,3,1)	63.5	0.0	0.1
(4,4,3,2)	64.0	0.0	0.08
(4,4,3,3)	64.5	0.0	0.03
(4,4,3,4)	65.0	0.0	0.02
(4,4,3,5)	65.5	0.0	0.06
(4,4,4,0)	64.0	0.0	0.05
(4,4,4,1)	64.5	0.0	0.05
(4,4,4,2)	65.0	0.0	0.11
(4,4,4,3)	65.5	0.0	0.03
(4,4,4,4)	66.0	0.0	0.06
(4,4,4,5)	66.5	0.0	0.05
(4,4,5,0)	65.0	0.0	0.11
(4,4,5,1)	65.5	0.0	0.04
(4,4,5,2)	66.0	0.0	0.06

TABLE A.27: BP_DIM4_2: Fixed Variants - Part 2

BP_DIM4_2			
$\widehat{k}$	$\widehat{obj}$	deviation costs	runtime $[s]$
(4,4,5,3)	66.5	0.0	0.04
(4,4,5,4)	67.0	0.0	0.03
(4,4,5,5)	-	-	-
(4,5,0,0)	65.0	0.0	0.1
(4,5,0,1)	65.5	0.0	0.09
(4,5,0,2)	66.0	0.0	0.04
(4,5,0,3)	66.5	0.0	0.07
(4,5,0,4)	67.0	0.0	0.06
(4,5,0,5)	67.5	0.0	0.06
(4,5,1,0)	66.0	0.0	0.09
(4,5,1,1)	66.5	0.0	0.1
(4,5,1,2)	67.0	0.0	0.1
(4,5,1,3)	67.5	0.0	0.06
(4,5,1,4)	68.0	0.0	0.04
(4,5,1,5)	68.5	0.0	0.05
(4,5,2,0)	67.0	0.0	0.03
(4,5,2,1)	67.5	0.0	0.1
(4,5,2,2)	68.0	0.0	0.1
(4,5,2,3)	68.5	0.0	0.06
(4,5,2,4)	69.0	0.0	0.05
(4,5,2,5)	69.5	0.0	0.03
(4,5,3,0)	68.0	0.0	0.04
(4,5,3,1)	68.5	0.0	0.02
(4,5,3,2)	69.0	0.0	0.05
(4,5,3,3)	69.5	0.0	0.07
(4,5,3,4)	70.0	0.0	0.08
(4,5,3,5)	70.5	0.0	0.03
(4,5,4,0)	69.0	0.0	0.04
(4,5,4,1)	69.5	0.0	0.04
(4,5,4,2)	70.0	0.0	0.03
(4,5,4,3)	70.5	0.0	0.04
(4,5,4,4)	71.0	0.0	0.06
(4,5,4,5)	-	-	-
(4,5,5,0)	70.0	0.0	0.08
(4,5,5,1)	70.5	0.0	0.03
(4,5,5,2)	71.0	0.0	0.04
(4,5,5,3)	71.5	0.0	0.05
(4,5,5,4)	-	-	-
(4,5,5,5)	-	-	-
(5,0,0,0)	125.0	75.0	0.22
(5,0,0,1)	60.5	10.0	1.18
(5,0,0,2)	51.0	0.0	0.68
(5,0,0,3)	51.5	0.0	0.1
(5,0,0,4)	52.0	0.0	0.09
(5,0,0,5)	52.5	0.0	0.08
(5,0,1,0)	61.0	10.0	1.36
(5,0,1,1)	51.5	0.0	0.35
(5,0,1,2)	52.0	0.0	0.16
(5,0,1,3)	52.5	0.0	0.11
(5,0,1,4)	53.0	0.0	0.1
(5,0,1,5)	53.5	0.0	0.1
(5,0,2,0)	52.0	0.0	0.8
(5,0,2,1)	52.5	0.0	0.11
(5,0,2,2)	53.0	0.0	0.08
(5,0,2,3)	53.5	0.0	0.09
(5,0,2,4)	54.0	0.0	0.05
(5,0,2,5)	54.5	0.0	0.11
(5,0,3,0)	53.0	0.0	0.17
(5,0,3,1)	53.5	0.0	0.1
(5,0,3,2)	54.0	0.0	0.06
(5,0,3,3)	54.5	0.0	0.1
(5,0,3,4)	55.0	0.0	0.1
(5,0,3,5)	55.5	0.0	0.04
(5,0,4,0)	54.0	0.0	0.1
(5,0,4,1)	54.5	0.0	0.06
(5,0,4,2)	55.0	0.0	0.05
(5,0,4,3)	55.5	0.0	0.11
(5,0,4,4)	56.0	0.0	0.05
(5,0,4,5)	56.5	0.0	0.08
(5,0,5,0)	55.0	0.0	0.08
(5,0,5,1)	55.5	0.0	0.06
(5,0,5,2)	56.0	0.0	0.08
(5,0,5,3)	56.5	0.0	0.09
(5,0,5,4)	57.0	0.0	0.04
(5,0,5,5)	-	-	-
(5,1,0,0)	65.0	10.0	1.27
(5,1,0,1)	55.5	0.0	0.19
(5,1,0,2)	56.0	0.0	0.21
(5,1,0,3)	56.5	0.0	0.11
(5,1,0,4)	57.0	0.0	0.1
(5,1,0,5)	57.5	0.0	0.11
(5,1,1,0)	56.0	0.0	0.33
(5,1,1,1)	56.5	0.0	0.15
(5,1,1,2)	57.0	0.0	0.13
(5,1,1,3)	57.5	0.0	0.09
(5,1,1,4)	58.0	0.0	0.09
(5,1,1,5)	58.5	0.0	0.04
(5,1,2,0)	57.0	0.0	0.16
(5,1,2,1)	57.5	0.0	0.13
(5,1,2,2)	58.0	0.0	0.03
(5,1,2,3)	58.5	0.0	0.1
(5,1,2,4)	59.0	0.0	0.06
(5,1,2,5)	59.5	0.0	0.07
(5,1,3,0)	58.0	0.0	0.1
(5,1,3,1)	58.5	0.0	0.08
(5,1,3,2)	59.0	0.0	0.06
(5,1,3,3)	59.5	0.0	0.09
(5,1,3,4)	60.0	0.0	0.09
(5,1,3,5)	60.5	0.0	0.04
(5,1,4,0)	59.0	0.0	0.08
(5,1,4,1)	59.5	0.0	0.04
(5,1,4,2)	60.0	0.0	0.07
(5,1,4,3)	60.5	0.0	0.06
(5,1,4,4)	61.0	0.0	0.03
(5,1,4,5)	61.5	0.0	0.04
(5,1,5,0)	60.0	0.0	0.11
(5,1,5,1)	60.5	0.0	0.11
(5,1,5,2)	61.0	0.0	0.05
(5,1,5,3)	61.5	0.0	0.02
(5,1,5,4)	62.0	0.0	0.03
(5,1,5,5)	-	-	-
(5,2,0,0)	60.0	0.0	0.84
(5,2,0,1)	60.5	0.0	0.3
(5,2,0,2)	61.0	0.0	0.22
(5,2,0,3)	61.5	0.0	0.08
(5,2,0,4)	62.0	0.0	0.12
(5,2,0,5)	62.5	0.0	0.1
(5,2,1,0)	61.0	0.0	0.31
(5,2,1,1)	61.5	0.0	0.11
(5,2,1,2)	62.0	0.0	0.09
(5,2,1,3)	62.5	0.0	0.04
(5,2,1,4)	63.0	0.0	0.13
(5,2,1,5)	63.5	0.0	0.02
(5,2,2,0)	62.0	0.0	0.16
(5,2,2,1)	62.5	0.0	0.13
(5,2,2,2)	63.0	0.0	0.06
(5,2,2,3)	63.5	0.0	0.04
(5,2,2,4)	64.0	0.0	0.07
(5,2,2,5)	64.5	0.0	0.05
(5,2,3,0)	63.0	0.0	0.06
(5,2,3,1)	63.5	0.0	0.05
(5,2,3,2)	64.0	0.0	0.08
(5,2,3,3)	64.5	0.0	0.05
(5,2,3,4)	65.0	0.0	0.06
(5,2,3,5)	65.5	0.0	0.05
(5,2,4,0)	64.0	0.0	0.04
(5,2,4,1)	64.5	0.0	0.05
(5,2,4,2)	65.0	0.0	0.1
(5,2,4,3)	65.5	0.0	0.03
(5,2,4,4)	66.0	0.0	0.01
(5,2,4,5)	66.5	0.0	0.08
(5,2,5,0)	65.0	0.0	0.05
(5,2,5,1)	65.5	0.0	0.04
(5,2,5,2)	66.0	0.0	0.03
(5,2,5,3)	66.5	0.0	0.03
(5,2,5,4)	67.0	0.0	0.03
(5,2,5,5)	-	-	-
(5,3,0,0)	65.0	0.0	0.13
(5,3,0,1)	65.5	0.0	0.11
(5,3,0,2)	66.0	0.0	0.1
(5,3,0,3)	66.5	0.0	0.07
(5,3,0,4)	67.0	0.0	0.03
(5,3,0,5)	67.5	0.0	0.04
(5,3,1,0)	66.0	0.0	0.13
(5,3,1,1)	66.5	0.0	0.1
(5,3,1,2)	67.0	0.0	0.09
(5,3,1,3)	67.5	0.0	0.03
(5,3,1,4)	68.0	0.0	0.05
(5,3,1,5)	68.5	0.0	0.05
(5,3,2,0)	67.0	0.0	0.06
(5,3,2,1)	67.5	0.0	0.05
(5,3,2,2)	68.0	0.0	0.1
(5,3,2,3)	68.5	0.0	0.04
(5,3,2,4)	69.0	0.0	0.02
(5,3,2,5)	69.5	0.0	0.08
(5,3,3,0)	68.0	0.0	0.12
(5,3,3,1)	68.5	0.0	0.06
(5,3,3,2)	69.0	0.0	0.07
(5,3,3,3)	69.5	0.0	0.06
(5,3,3,4)	70.0	0.0	0.04
(5,3,3,5)	70.5	0.0	0.08
(5,3,4,0)	69.0	0.0	0.07
(5,3,4,1)	69.5	0.0	0.02
(5,3,4,2)	70.0	0.0	0.04
(5,3,4,3)	70.5	0.0	0.02
(5,3,4,4)	71.0	0.0	0.04
(5,3,4,5)	71.5	0.0	0.04
(5,3,5,0)	70.0	0.0	0.07
(5,3,5,1)	70.5	0.0	0.04
(5,3,5,2)	71.0	0.0	0.08
(5,3,5,3)	71.5	0.0	0.06
(5,3,5,4)	72.0	0.0	0.04
(5,3,5,5)	-	-	-
(5,4,0,0)	70.0	0.0	0.08
(5,4,0,1)	70.5	0.0	0.08
(5,4,0,2)	71.0	0.0	0.05
(5,4,0,3)	71.5	0.0	0.07
(5,4,0,4)	72.0	0.0	0.07
(5,4,0,5)	72.5	0.0	0.12
(5,4,1,0)	71.0	0.0	0.08
(5,4,1,1)	71.5	0.0	0.11
(5,4,1,2)	72.0	0.0	0.03
(5,4,1,3)	72.5	0.0	0.1
(5,4,1,4)	73.0	0.0	0.09
(5,4,1,5)	73.5	0.0	0.07
(5,4,2,0)	72.0	0.0	0.1
(5,4,2,1)	72.5	0.0	0.05
(5,4,2,2)	73.0	0.0	0.06
(5,4,2,3)	73.5	0.0	0.06
(5,4,2,4)	74.0	0.0	0.04
(5,4,2,5)	74.5	0.0	0.04
(5,4,3,0)	73.0	0.0	0.04
(5,4,3,1)	73.5	0.0	0.09
(5,4,3,2)	74.0	0.0	0.06
(5,4,3,3)	74.5	0.0	0.01
(5,4,3,4)	75.0	0.0	0.07
(5,4,3,5)	75.5	0.0	0.04
(5,4,4,0)	74.0	0.0	0.05
(5,4,4,1)	74.5	0.0	0.04
(5,4,4,2)	75.0	0.0	0.04
(5,4,4,3)	75.5	0.0	0.05
(5,4,4,4)	76.0	0.0	0.04
(5,4,4,5)	76.5	0.0	0.11
(5,4,5,0)	75.0	0.0	0.04
(5,4,5,1)	75.5	0.0	0.06
(5,4,5,2)	76.0	0.0	0.04
(5,4,5,3)	76.5	0.0	0.04
(5,4,5,4)	77.0	0.0	0.05
(5,4,5,5)	-	-	-
(5,5,0,0)	75.0	0.0	0.08
(5,5,0,1)	75.5	0.0	0.09
(5,5,0,2)	76.0	0.0	0.09
(5,5,0,3)	76.5	0.0	0.08
(5,5,0,4)	77.0	0.0	0.09
(5,5,0,5)	77.5	0.0	0.04
(5,5,1,0)	76.0	0.0	0.09
(5,5,1,1)	76.5	0.0	0.1
(5,5,1,2)	77.0	0.0	0.04
(5,5,1,3)	77.5	0.0	0.05
(5,5,1,4)	78.0	0.0	0.04
(5,5,1,5)	78.5	0.0	0.05
(5,5,2,0)	77.0	0.0	0.05
(5,5,2,1)	77.5	0.0	0.05
(5,5,2,2)	78.0	0.0	0.06
(5,5,2,3)	78.5	0.0	0.07
(5,5,2,4)	79.0	0.0	0.03
(5,5,2,5)	79.5	0.0	0.06
(5,5,3,0)	78.0	0.0	0.05
(5,5,3,1)	78.5	0.0	0.03
(5,5,3,2)	79.0	0.0	0.01
(5,5,3,3)	79.5	0.0	0.08
(5,5,3,4)	80.0	0.0	0.02
(5,5,3,5)	80.5	0.0	0.03
(5,5,4,0)	79.0	0.0	0.04
(5,5,4,1)	79.5	0.0	0.01
(5,5,4,2)	80.0	0.0	0.05
(5,5,4,3)	80.5	0.0	0.04
(5,5,4,4)	81.0	0.0	0.05
(5,5,4,5)	-	-	-
(5,5,5,0)	80.0	0.0	0.07
(5,5,5,1)	80.5	0.0	0.04
(5,5,5,2)	81.0	0.0	0.04
(5,5,5,3)	81.5	0.0	0.06
(5,5,5,4)	-	-	-
(5,5,5,5)	-	-	-

TABLE A.28: BP_DIM4_2: Fixed Variants - Part 3

A.3 Complementary Material for Chapter 6

A.3.1 Steepest Descent Method

Name	p^0	Global MP SD	Points [%]
BP_RD_DIM3_1	(0,0,1)	**True**	5.000
BP_RD_DIM3_1	(5,4,0)	**True**	7.576
BP_RD_DIM3_2	(0,0,1)	**True**	4.209
BP_RD_DIM3_2	(4,2,2)	**True**	11.111
BP_RD_DIM3_3	(0,0,1)	**True**	18.248
BP_RD_DIM3_3	(2,2,2)	**True**	38.686
BP_RD_DIM3_4	(0,0,1)	**True**	13.966
BP_RD_DIM3_4	(2,2,1)	**True**	28.492
BP_RD_DIM3_5	(0,0,1)	**True**	3.026
BP_RD_DIM3_5	(5,2,2)	**True**	10.663
BP_RD_DIM3_6	(0,0,1)	**True**	8.434
BP_RD_DIM3_6	(2,2,4)	**True**	24.498
BP_RD_DIM3_7	(0,0,1)	**True**	7.376
BP_RD_DIM3_7	(4,2,4)	**True**	9.362
BP_RD_DIM3_8	(0,0,1)	**True**	5.385
BP_RD_DIM3_8	(4,4,1)	**True**	18.205
BP_RD_DIM3_9	(0,0,1)	**True**	50.909
BP_RD_DIM3_9	(2,0,0)	**True**	61.818
BP_RD_DIM3_10	(0,0,1)	**True**	46.774
BP_RD_DIM3_10	(3,0,0)	**True**	70.968
BP_RD_DIM3_11	(0,0,1)	**True**	20.379
BP_RD_DIM3_11	(2,2,2)	**True**	19.905
BP_RD_DIM3_12	(0,0,1)	False	10.338
BP_RD_DIM3_12	(4,2,1)	False	7.806
BP_RD_DIM3_13	(0,0,1)	**True**	10.575
BP_RD_DIM3_13	(5,3,1)	**True**	13.544
BP_RD_DIM3_14	(0,0,1)	**True**	13.548
BP_RD_DIM3_14	(2,2,1)	**True**	30.323
BP_RD_DIM3_15	(0,0,1)	**True**	5.524
BP_RD_DIM3_15	(4,4,2)	**True**	9.632
∅ [%]		93.333	19.543

TABLE A.29: Results Steepest Descent method (from resultfile) for the randomly generated input data for the binpacking example in dimension three

Name	p^0	Global MP SD	Points [%]
BP_RD_DIM4_1	(0,0,0,1)	**True**	5.301
BP_RD_DIM4_1	(3,3,0,1)	**True**	7.437
BP_RD_DIM4_2	(0,0,0,1)	**True**	9.075
BP_RD_DIM4_2	(2,2,1,0)	**True**	19.751
BP_RD_DIM4_3	(0,0,0,1)	**True**	26.817
BP_RD_DIM4_3	(2,1,2,0)	**True**	20.551
BP_RD_DIM4_4	(0,0,0,1)	False	2.924
BP_RD_DIM4_4	(4,4,1,0)	False	1.849
BP_RD_DIM4_5	(0,0,0,1)	**True**	9.738
BP_RD_DIM4_5	(2,2,1,0)	**True**	10.756
BP_RD_DIM4_6	(0,0,0,1)	**True**	25.072
BP_RD_DIM4_6	(3,2,0,0)	**True**	8.646
BP_RD_DIM4_7	(0,0,0,1)	False	2.786
BP_RD_DIM4_7	(4,2,2,0)	False	3.411
BP_RD_DIM4_8	(0,0,0,1)	**True**	4.357
BP_RD_DIM4_8	(2,2,1,1)	**True**	12.361
BP_RD_DIM4_9	(0,0,0,1)	**True**	1.972
BP_RD_DIM4_9	(2,4,2,1)	**True**	2.971
BP_RD_DIM4_10	(0,0,0,1)	**True**	2.731
BP_RD_DIM4_10	(4,4,1,0)	**True**	6.008
BP_RD_DIM4_11	(0,0,0,1)	**True**	4.798
BP_RD_DIM4_11	(2,4,2,0)	**True**	6.134
BP_RD_DIM4_12	(0,0,0,1)	**True**	3.514
BP_RD_DIM4_12	(4,3,2,1)	**True**	4.836
BP_RD_DIM4_13	(0,0,0,1)	**True**	23.739
BP_RD_DIM4_13	(4,1,0,0)	**True**	19.139
BP_RD_DIM4_14	(0,0,0,1)	**True**	13.720
BP_RD_DIM4_14	(2,2,0,0)	**True**	17.822
BP_RD_DIM4_15	(0,0,0,1)	**True**	24.747
BP_RD_DIM4_15	(4,1,0,0)	**True**	26.667
∅ [%]		86.667	10.988

TABLE A.30: Results Steepest Descent method (from resultfile) for the randomly generated input data for the binpacking example in dimension four

A.3.2 Coordinate Search

Name	p^0	α	Global MP CS	Box-Local MP CS	Points [%]
BP_RD_DIM3_1	(0,0,1)	1	False	False	1.818
BP_RD_DIM3_1	(0,0,1)	2	False	False	1.818
BP_RD_DIM3_1	(5,4,0)	1	**True**	True	3.939
BP_RD_DIM3_1	(5,4,0)	2	**True**	True	2.879
BP_RD_DIM3_2	(0,0,1)	1	**True**	True	1.684
BP_RD_DIM3_2	(0,0,1)	2	**True**	True	1.852
BP_RD_DIM3_2	(4,2,2)	1	False	False	4.882
BP_RD_DIM3_2	(4,2,2)	2	False	False	4.040
BP_RD_DIM3_3	(0,0,1)	1	False	False	8.029
BP_RD_DIM3_3	(0,0,1)	2	**True**	True	8.029
BP_RD_DIM3_3	(2,2,2)	1	False	False	13.869
BP_RD_DIM3_3	(2,2,2)	2	False	False	12.409
BP_RD_DIM3_4	(0,0,1)	1	**True**	True	5.587
BP_RD_DIM3_4	(0,0,1)	2	**True**	True	6.145
BP_RD_DIM3_4	(2,2,1)	1	False	False	8.939
BP_RD_DIM3_4	(2,2,1)	2	False	False	7.821
BP_RD_DIM3_5	(0,0,1)	1	**True**	True	1.441
BP_RD_DIM3_5	(0,0,1)	2	False	False	1.729
BP_RD_DIM3_5	(5,2,2)	1	**True**	True	4.899
BP_RD_DIM3_5	(5,2,2)	2	False	False	3.314
BP_RD_DIM3_6	(0,0,1)	1	False	False	3.213
BP_RD_DIM3_6	(0,0,1)	2	**True**	True	6.024
BP_RD_DIM3_6	(2,2,4)	1	**True**	True	12.048
BP_RD_DIM3_6	(2,2,4)	2	**True**	True	8.032
BP_RD_DIM3_7	(0,0,1)	1	**True**	True	3.404
BP_RD_DIM3_7	(0,0,1)	2	**True**	True	2.837
BP_RD_DIM3_7	(4,2,4)	1	**True**	True	4.823
BP_RD_DIM3_7	(4,2,4)	2	**True**	True	3.404
BP_RD_DIM3_8	(0,0,1)	1	**True**	True	2.821
BP_RD_DIM3_8	(0,0,1)	2	False	False	2.564
BP_RD_DIM3_8	(4,4,1)	1	False	False	8.462
BP_RD_DIM3_8	(4,4,1)	2	False	False	5.128
BP_RD_DIM3_9	(0,0,1)	1	**True**	True	25.455
BP_RD_DIM3_9	(0,0,1)	2	**True**	True	16.364
BP_RD_DIM3_9	(2,0,0)	1	**True**	True	41.818
BP_RD_DIM3_9	(2,0,0)	2	**True**	True	36.364
BP_RD_DIM3_10	(0,0,1)	1	**True**	True	24.194
BP_RD_DIM3_10	(0,0,1)	2	False	False	25.806
BP_RD_DIM3_10	(3,0,0)	1	False	False	14.516
BP_RD_DIM3_10	(3,0,0)	2	**True**	True	25.806
BP_RD_DIM3_11	(0,0,1)	1	**True**	True	7.583
BP_RD_DIM3_11	(0,0,1)	2	False	False	6.635
BP_RD_DIM3_11	(2,2,2)	1	**True**	True	7.583
BP_RD_DIM3_11	(2,2,2)	2	False	False	7.583
BP_RD_DIM3_12	(0,0,1)	1	False	False	3.376
BP_RD_DIM3_12	(0,0,1)	2	False	False	3.797
BP_RD_DIM3_12	(4,2,1)	1	False	**True**	3.586
BP_RD_DIM3_12	(4,2,1)	2	False	False	3.586
BP_RD_DIM3_13	(0,0,1)	1	False	False	2.968
BP_RD_DIM3_13	(0,0,1)	2	**True**	True	3.711
BP_RD_DIM3_13	(5,3,1)	1	False	False	4.824
BP_RD_DIM3_13	(5,3,1)	2	False	False	4.267
BP_RD_DIM3_14	(0,0,1)	1	**True**	True	7.097
BP_RD_DIM3_14	(0,0,1)	2	False	False	7.742
BP_RD_DIM3_14	(2,2,1)	1	False	False	10.323
BP_RD_DIM3_14	(2,2,1)	2	False	False	9.032
BP_RD_DIM3_15	(0,0,1)	1	False	False	2.266
BP_RD_DIM3_15	(0,0,1)	2	**True**	True	2.550
BP_RD_DIM3_15	(4,4,2)	1	**True**	True	4.533
BP_RD_DIM3_15	(4,4,2)	2	**True**	True	3.116
∅ [%]			50.000	51.667	8.073

TABLE A.31: Results Coordinate Search (from resultfile) for the randomly generated three-dimensional binpacking problems

Name	p^0	α	Global MP CS	Box-Local MP CS	Points [%]
BP_RD_DIM4_1	(0,0,0,1)	1	False	False	1.187
BP_RD_DIM4_1	(0,0,0,1)	2	False	False	1.187
BP_RD_DIM4_1	(3,3,0,1)	1	**True**	True	2.215
BP_RD_DIM4_1	(3,3,0,1)	2	False	False	1.582
BP_RD_DIM4_2	(0,0,0,1)	1	False	False	2.491
BP_RD_DIM4_2	(0,0,0,1)	2	**True**	True	2.491
BP_RD_DIM4_2	(2,2,1,0)	1	False	False	3.381
BP_RD_DIM4_2	(2,2,1,0)	2	False	False	2.669
BP_RD_DIM4_3	(0,0,0,1)	1	False	False	5.013
BP_RD_DIM4_3	(0,0,0,1)	2	**True**	True	5.514
BP_RD_DIM4_3	(2,1,2,0)	1	False	False	5.764
BP_RD_DIM4_3	(2,1,2,0)	2	**True**	True	5.764
BP_RD_DIM4_4	(0,0,0,1)	1	False	False	0.538
BP_RD_DIM4_4	(0,0,0,1)	2	False	False	0.495
BP_RD_DIM4_4	(4,4,1,0)	1	False	**True**	0.559
BP_RD_DIM4_4	(4,4,1,0)	2	False	False	0.559
BP_RD_DIM4_5	(0,0,0,1)	1	False	False	2.180
BP_RD_DIM4_5	(0,0,0,1)	2	False	False	2.180
BP_RD_DIM4_5	(2,2,1,0)	1	False	False	2.471
BP_RD_DIM4_5	(2,2,1,0)	2	False	False	2.471
BP_RD_DIM4_6	(0,0,0,1)	1	**True**	True	6.052
BP_RD_DIM4_6	(0,0,0,1)	2	False	False	3.890
BP_RD_DIM4_6	(3,2,0,0)	1	False	False	3.170
BP_RD_DIM4_6	(3,2,0,0)	2	False	False	3.026
BP_RD_DIM4_7	(0,0,0,1)	1	False	False	0.480
BP_RD_DIM4_7	(0,0,0,1)	2	False	False	0.552
BP_RD_DIM4_7	(4,2,2,0)	1	False	False	0.745
BP_RD_DIM4_7	(4,2,2,0)	2	False	False	0.601
BP_RD_DIM4_8	(0,0,0,1)	1	**True**	True	1.418
BP_RD_DIM4_8	(0,0,0,1)	2	**True**	True	1.520
BP_RD_DIM4_8	(2,2,1,1)	1	**True**	True	2.533
BP_RD_DIM4_8	(2,2,1,1)	2	**True**	True	2.330
BP_RD_DIM4_9	(0,0,0,1)	1	False	False	0.475
BP_RD_DIM4_9	(0,0,0,1)	2	False	False	0.452
BP_RD_DIM4_9	(2,4,2,1)	1	**True**	True	0.832
BP_RD_DIM4_9	(2,4,2,1)	2	False	False	0.689
BP_RD_DIM4_10	(0,0,0,1)	1	False	False	0.630
BP_RD_DIM4_10	(0,0,0,1)	2	False	False	0.630
BP_RD_DIM4_10	(4,4,1,0)	1	False	False	1.681
BP_RD_DIM4_10	(4,4,1,0)	2	False	False	1.050
BP_RD_DIM4_11	(0,0,0,1)	1	False	False	0.790
BP_RD_DIM4_11	(0,0,0,1)	2	False	False	0.820
BP_RD_DIM4_11	(2,4,2,0)	1	False	False	0.577
BP_RD_DIM4_11	(2,4,2,0)	2	False	False	0.577
BP_RD_DIM4_12	(0,0,0,1)	1	False	False	0.696
BP_RD_DIM4_12	(0,0,0,1)	2	**True**	True	0.800
BP_RD_DIM4_12	(4,3,2,1)	1	**True**	True	1.427
BP_RD_DIM4_12	(4,3,2,1)	2	False	False	0.905
BP_RD_DIM4_13	(0,0,0,1)	1	**True**	True	5.341
BP_RD_DIM4_13	(0,0,0,1)	2	False	False	3.264
BP_RD_DIM4_13	(4,1,0,0)	1	False	False	4.006
BP_RD_DIM4_13	(4,1,0,0)	2	False	False	2.226
BP_RD_DIM4_14	(0,0,0,1)	1	False	False	3.112
BP_RD_DIM4_14	(0,0,0,1)	2	False	False	2.687
BP_RD_DIM4_14	(2,2,0,0)	1	False	False	4.385
BP_RD_DIM4_14	(2,2,0,0)	2	**True**	True	4.243
BP_RD_DIM4_15	(0,0,0,1)	1	**True**	True	4.040
BP_RD_DIM4_15	(0,0,0,1)	2	**True**	True	2.929
BP_RD_DIM4_15	(4,1,0,0)	1	False	False	5.657
BP_RD_DIM4_15	(4,1,0,0)	2	**True**	True	2.828
∅ [%]			28.333	30.000	2.246

TABLE A.32: Results Coordinate Search (from resultfile) for the randomly generated four-dimensional binpacking problems

Name	p^0	α	Global MP CS	Box-Local MP CS	Points [%]
f_3^2	(0,0)	1	**True**	True	30.612
f_3^2	(0,0)	2	**True**	True	30.612
f_3^2	(3,3)	1	**True**	True	10.204
f_3^2	(3,3)	2	**True**	True	18.367
f_3^3	(0,0,0)	1	**True**	True	11.079
f_3^3	(0,0,0)	2	**True**	True	9.329
f_3^3	(3,3,3)	1	**True**	True	2.041
f_3^3	(3,3,3)	2	**True**	True	3.790
f_3^4	(0,0,0,0)	1	**True**	True	2.999
f_3^4	(0,0,0,0)	2	**True**	True	2.332
f_3^4	(3,3,3,3)	1	**True**	True	0.375
f_3^4	(3,3,3,3)	2	**True**	True	0.708
∅ [%]			100.000	100.000	10.204

TABLE A.33: Results Coordinate Search for the separable test function f_3^n

Name	p^0	α	Global MP CS	Box-Local MP CS	Points [%]
f_4^2	(0,0)	1	**True**	True	18.519
f_4^2	(0,0)	2	**True**	True	23.457
f_4^2	(4,4)	1	**True**	True	12.346
f_4^2	(4,4)	2	**True**	True	16.049
f_4^3	(0,0,0)	1	**True**	True	5.213
f_4^3	(0,0,0)	2	**True**	True	6.036
f_4^3	(4,4,4)	1	**True**	True	2.743
f_4^3	(4,4,4)	2	**True**	True	3.429
f_4^4	(0,0,0,0)	1	**True**	True	1.097
f_4^4	(0,0,0,0)	2	**True**	True	1.219
f_4^4	(4,4,4,4)	1	**True**	True	0.518
f_4^4	(4,4,4,4)	2	**True**	True	0.625
∅ [%]			100.000	100.000	7.604

TABLE A.34: Results Coordinate Search for the separable test function f_4^n

Name	p^0	α	Global MP CS	Box-Local MP CS	Points [%]
DENSCHNC	(-5,-5)	1	False	**True**	19.835
DENSCHNC	(-5,-5)	2	**True**	True	16.529
DENSCHNC	(0,0)	1	**True**	True	8.264
DENSCHNC	(0,0)	2	**True**	True	13.223
ELATVIDU	(-5,-5)	1	False	**True**	10.744
ELATVIDU	(-5,-5)	2	False	**True**	11.570
ELATVIDU	(0,0)	1	False	**True**	10.744
ELATVIDU	(0,0)	2	False	**True**	11.570
ROSENBR	(-5,-5)	1	False	False	22.314
ROSENBR	(-5,-5)	2	**True**	True	17.355
ROSENBR	(0,0)	1	False	False	4.132
ROSENBR	(0,0)	2	False	False	7.438
WAYSEA1	(-5,-5)	1	False	**True**	20.661
WAYSEA1	(-5,-5)	2	**True**	True	18.182
WAYSEA1	(0,0)	1	**True**	True	10.744
WAYSEA1	(0,0)	2	**True**	True	11.570
∅ [%]			43.750	81.250	13.430

TABLE A.35: Results Coordinate Search for PyCUTEst test functions of dimension two

Name	p^0	α	Global MP CS	Box-Local MP CS	Points [%]
BOX3	(-5,-5,-5)	1	**True**	True	1.427
BOX3	(-5,-5,-5)	2	**True**	True	1.202
BOX3	(0,0,0)	1	**True**	True	0.526
BOX3	(0,0,0)	2	**True**	True	0.977
DENSCHND	(-5,-5,-5)	1	**True**	True	4.358
DENSCHND	(-5,-5,-5)	2	False	False	2.705
DENSCHND	(0,0,0)	1	**True**	True	0.526
DENSCHND	(0,0,0)	2	**True**	True	0.977
HATFLDE	(-5,-5,-5)	1	False	**True**	3.005
HATFLDE	(-5,-5,-5)	2	False	False	1.878
HATFLDE	(0,0,0)	1	False	False	1.277
HATFLDE	(0,0,0)	2	False	False	1.352
HELIX	(-5,-5,-5)	1	False	False	2.479
HELIX	(-5,-5,-5)	2	False	False	2.554
HELIX	(0,0,0)	1	**True**	True	0.902
HELIX	(0,0,0)	2	**True**	True	1.277
∅ [%]			56.250	62.500	1.714

TABLE A.36: Results Coordinate Search for PyCUTEst test functions of dimension three

Name	p^0	α	Global MP CS	Box-Local MP CS	Points [%]
HILBERTA	(-3,-3,-3,-3)	1	False	False	3.207
HILBERTA	(-3,-3,-3,-3)	2	False	**True**	2.082
HILBERTA	(0,0,0,0)	1	**True**	True	0.375
HILBERTA	(0,0,0,0)	2	**True**	True	0.708
HIMMELBF	(-3,-3,-3,-3)	1	**True**	True	0.708
HIMMELBF	(-3,-3,-3,-3)	2	**True**	True	0.708
HIMMELBF	(0,0,0,0)	1	**True**	True	2.416
HIMMELBF	(0,0,0,0)	2	**True**	True	1.791
PENALTY1	(-3,-3,-3,-3)	1	**True**	True	2.999
PENALTY1	(-3,-3,-3,-3)	2	**True**	True	3.332
PENALTY1	(0,0,0,0)	1	**True**	True	0.375
PENALTY1	(0,0,0,0)	2	**True**	True	0.708
PENALTY2	(-3,-3,-3,-3)	1	False	**True**	2.582
PENALTY2	(-3,-3,-3,-3)	2	**True**	True	3.082
PENALTY2	(0,0,0,0)	1	**True**	True	0.666
PENALTY2	(0,0,0,0)	2	**True**	True	0.958
POWELLSG	(-3,-3,-3,-3)	1	False	False	2.249
POWELLSG	(-3,-3,-3,-3)	2	False	False	2.416
POWELLSG	(0,0,0,0)	1	**True**	True	0.375
POWELLSG	(0,0,0,0)	2	**True**	True	0.708
STREG	(-3,-3,-3,-3)	1	False	False	2.749
STREG	(-3,-3,-3,-3)	2	**True**	True	2.291
STREG	(0,0,0,0)	1	False	False	0.375
STREG	(0,0,0,0)	2	False	False	0.708
∅ [%]			66.667	75.000	1.607

TABLE A.37: Results Coordinate Search for PyCUTEst test functions of dimension four

A.3.3 Nelder-Mead

n	start index	$\mathbb{S}_0$
2	0	$\Delta\{(-5,-5),(-4,-5),(-5,-4)\}$
	1	$\Delta\{(0,0),(1,0),(0,1)\}$
3	0	$\Delta\{(-5,-5,-5),(-5,-5,-4),(-5,-4,-5),(-4,-4,-4)\}$
	1	$\Delta\{(0,0,0),(1,0,0),(0,1,0),(0,0,1)\}$
4	0	$\Delta\{(-3,-3,-3,-3),(-3,-3,-3,-2),(-3,-3,-2,-3),$ $(-3,-2,-3,-3),(-2,-3,-3,-3)\}$
	1	$\Delta\{(0,0,0,0),(1,0,0,0),(0,1,0,0),(0,0,1,0),(0,0,0,1)\}$

TABLE A.38: Initial simplices Nelder-Mead adaptation for discrete test functions from the PyCUTEst test library

Name	start index	replace rule	Stop NM	Global MP NM	Box-Local MP NM	Points NM [%]
BOX3	0	1	3	False	False	0.526
BOX3	0	2	1	False	False	1.052
BOX3	1	1	3	**True**	True	0.826
BOX3	1	2	1	**True**	True	0.977
BOX3	2	1	3	**True**	True	1.653
BOX3	2	2	1	**True**	True	1.653
DENSCHND	0	1	3	**True**	True	3.080
DENSCHND	0	2	1	**True**	True	3.231
DENSCHND	1	1	3	**True**	True	0.826
DENSCHND	1	2	1	**True**	True	1.202
DENSCHND	2	1	3	**True**	True	2.705
DENSCHND	2	2	1	**True**	True	3.005
HATFLDE	0	1	3	False	False	0.601
HATFLDE	0	2	1	False	False	3.156
HATFLDE	1	1	3	False	False	1.127
HATFLDE	1	2	1	False	False	1.277
HATFLDE	2	1	3	False	False	0.676
HATFLDE	2	2	1	False	False	1.127
HELIX	0	1	3	False	False	1.578
HELIX	0	2	1	False	False	2.329
HELIX	1	1	3	**True**	True	0.902
HELIX	1	2	1	**True**	True	1.202
HELIX	2	1	3	False	False	2.404
HELIX	2	2	1	False	False	3.080
∅ [%]				50.000	50.000	1.675

TABLE A.39: Results Nelder-Mead adaptation for PyCUTEst test functions of dimension three

Name	start index	replace rule	Stop NM	Global MP NM	Box-Local MP NM	Points NM [%]
HILBERTA	0	1	3	False	False	1.833
HILBERTA	0	2	1	**True**	True	3.415
HILBERTA	1	1	3	**True**	True	0.833
HILBERTA	1	2	1	**True**	True	1.125
HILBERTA	2	1	3	False	False	1.833
HILBERTA	2	2	1	**True**	True	3.415
HIMMELBF	0	1	3	**True**	True	1.624
HIMMELBF	0	2	1	**True**	True	1.291
HIMMELBF	1	1	3	False	False	1.416
HIMMELBF	1	2	1	False	False	1.791
HIMMELBF	2	1	3	**True**	True	1.624
HIMMELBF	2	2	1	**True**	True	1.374
PENALTY1	0	1	3	**True**	True	2.666
PENALTY1	0	2	1	**True**	True	2.332
PENALTY1	1	1	3	**True**	True	0.833
PENALTY1	1	2	1	**True**	True	1.208
PENALTY1	2	1	3	**True**	True	2.249
PENALTY1	2	2	1	**True**	True	2.207
PENALTY2	0	1	3	False	**True**	2.499
PENALTY2	0	2	1	False	**True**	2.707
PENALTY2	1	1	3	**True**	True	0.833
PENALTY2	1	2	1	**True**	True	0.791
PENALTY2	2	1	3	**True**	True	2.207
PENALTY2	2	2	1	**True**	True	2.457
POWELLSG	0	1	3	False	False	1.458
POWELLSG	0	2	1	**True**	True	2.791
POWELLSG	1	1	3	**True**	True	0.833
POWELLSG	1	2	1	**True**	True	1.208
POWELLSG	2	1	3	False	False	1.499
POWELLSG	2	2	1	**True**	True	3.040
STREG	0	1	3	False	False	1.583
STREG	0	2	1	**True**	True	2.791
STREG	1	1	3	**True**	True	0.750
STREG	1	2	1	**True**	True	1.833
STREG	2	1	3	False	False	0.666
STREG	2	2	1	**True**	True	2.499
∅ [%]				72.222	77.778	1.820

TABLE A.40: Results Nelder-Mead adaptation for PyCUTEst test functions of dimension four

Name	start index	replace rule	Stop NM	Global MP NM	Box-Local MP NM	Points NM [%]
BP_RD_DIM3_1	0	1	3	False	False	1.818
BP_RD_DIM3_1	0	2	1	False	False	1.970
BP_RD_DIM3_1	1	1	3	False	False	2.727
BP_RD_DIM3_1	1	2	1	**True**	True	3.788
BP_RD_DIM3_2	0	1	3	False	False	1.515
BP_RD_DIM3_2	0	2	1	False	False	2.189
BP_RD_DIM3_2	1	1	3	False	False	3.367
BP_RD_DIM3_2	1	2	1	**True**	True	5.892
BP_RD_DIM3_3	0	1	3	False	False	6.569
BP_RD_DIM3_3	0	2	1	False	False	9.489
BP_RD_DIM3_3	1	1	3	False	False	8.759
BP_RD_DIM3_3	1	2	1	**True**	True	18.978
BP_RD_DIM3_4	0	1	3	False	False	5.028
BP_RD_DIM3_4	0	2	1	False	False	7.263
BP_RD_DIM3_4	1	1	3	False	False	6.704
BP_RD_DIM3_4	1	2	1	False	False	8.939
BP_RD_DIM3_5	0	1	3	**True**	True	1.297
BP_RD_DIM3_5	0	2	1	False	False	1.873
BP_RD_DIM3_5	1	1	3	False	False	3.458
BP_RD_DIM3_5	1	2	1	**True**	True	4.467
BP_RD_DIM3_6	0	1	3	**True**	True	3.614
BP_RD_DIM3_6	0	2	1	**True**	True	4.819
BP_RD_DIM3_6	1	1	3	False	False	4.819
BP_RD_DIM3_6	1	2	1	False	False	12.450
BP_RD_DIM3_7	0	1	3	False	False	2.270
BP_RD_DIM3_7	0	2	3	False	False	3.262
BP_RD_DIM3_7	1	1	3	False	False	2.553
BP_RD_DIM3_7	1	2	1	**True**	True	4.113
BP_RD_DIM3_8	0	1	3	**True**	True	2.308
BP_RD_DIM3_8	0	2	1	False	False	3.077
BP_RD_DIM3_8	1	1	3	False	False	5.128
BP_RD_DIM3_8	1	2	1	**True**	True	10.513
BP_RD_DIM3_9	0	1	3	False	False	23.636
BP_RD_DIM3_9	0	2	2	**True**	True	38.182
BP_RD_DIM3_9	1	1	3	False	False	23.636
BP_RD_DIM3_9	1	2	2	**True**	True	38.182
BP_RD_DIM3_10	0	1	3	**True**	True	25.806
BP_RD_DIM3_10	0	2	1	**True**	True	22.581
BP_RD_DIM3_10	1	1	3	False	False	20.968
BP_RD_DIM3_10	1	2	1	**True**	True	35.484
BP_RD_DIM3_11	0	1	3	**True**	True	5.213
BP_RD_DIM3_11	0	2	1	**True**	True	8.531
BP_RD_DIM3_11	1	1	3	False	False	5.687
BP_RD_DIM3_11	1	2	1	**True**	True	8.531
BP_RD_DIM3_12	0	1	3	False	False	2.321
BP_RD_DIM3_12	0	2	1	False	**True**	5.063
BP_RD_DIM3_12	1	1	3	False	False	1.899
BP_RD_DIM3_12	1	2	1	False	**True**	4.430
BP_RD_DIM3_13	0	1	3	False	False	2.041
BP_RD_DIM3_13	0	2	3	False	False	5.195
BP_RD_DIM3_13	1	1	3	False	False	1.670
BP_RD_DIM3_13	1	2	1	False	False	5.380
BP_RD_DIM3_14	0	1	3	**True**	True	5.806
BP_RD_DIM3_14	0	2	1	False	False	8.387
BP_RD_DIM3_14	1	1	3	False	False	7.742
BP_RD_DIM3_14	1	2	1	False	False	11.613
BP_RD_DIM3_15	0	1	3	**True**	True	2.550
BP_RD_DIM3_15	0	2	1	False	False	2.975
BP_RD_DIM3_15	1	1	3	False	False	2.550
BP_RD_DIM3_15	1	2	1	False	False	3.824
∅ [%]				33.333	36.667	8.248

TABLE A.41: Results Nelder-Mead adaptation for the randomly generated three-dimensional binpacking problems

Name	start index	replace rule	Stop NM	Global MP NM	Box-Local MP NM	Points NM [%]
BP_RD_DIM4_1	0	1	3	False	False	0.949
BP_RD_DIM4_1	0	2	1	False	False	2.136
BP_RD_DIM4_1	1	1	3	False	False	1.661
BP_RD_DIM4_1	1	2	1	**True**	True	2.611
BP_RD_DIM4_2	0	1	3	False	False	2.135
BP_RD_DIM4_2	0	2	1	False	False	5.338
BP_RD_DIM4_2	1	1	3	False	False	4.270
BP_RD_DIM4_2	1	2	3	False	False	6.584
BP_RD_DIM4_3	0	1	3	**True**	True	7.769
BP_RD_DIM4_3	0	2	3	**True**	True	9.023
BP_RD_DIM4_3	1	1	3	False	False	5.764
BP_RD_DIM4_3	1	2	3	**True**	True	6.767
BP_RD_DIM4_4	0	1	3	False	False	0.473
BP_RD_DIM4_4	0	2	1	False	False	0.710
BP_RD_DIM4_4	1	1	3	False	False	0.194
BP_RD_DIM4_4	1	2	1	False	**True**	0.839
BP_RD_DIM4_5	0	1	3	False	False	1.744
BP_RD_DIM4_5	0	2	3	**True**	True	5.814
BP_RD_DIM4_5	1	1	3	False	False	2.035
BP_RD_DIM4_5	1	2	1	**True**	True	4.360
BP_RD_DIM4_6	0	1	3	False	False	5.187
BP_RD_DIM4_6	0	2	1	**True**	True	6.772
BP_RD_DIM4_6	1	1	3	**True**	True	2.017
BP_RD_DIM4_6	1	2	1	**True**	True	2.882
BP_RD_DIM4_7	0	1	3	False	False	0.577
BP_RD_DIM4_7	0	2	1	False	**True**	1.009
BP_RD_DIM4_7	1	1	3	False	**True**	0.913
BP_RD_DIM4_7	1	2	3	False	**True**	1.057
BP_RD_DIM4_8	0	1	3	**True**	True	2.736
BP_RD_DIM4_8	0	2	1	**True**	True	2.533
BP_RD_DIM4_8	1	1	3	**True**	True	2.026
BP_RD_DIM4_8	1	2	1	**True**	True	2.938
BP_RD_DIM4_9	0	1	3	False	False	0.404
BP_RD_DIM4_9	0	2	1	**True**	True	0.808
BP_RD_DIM4_9	1	1	3	False	False	0.665
BP_RD_DIM4_9	1	2	1	**True**	True	0.784
BP_RD_DIM4_10	0	1	3	False	False	0.504
BP_RD_DIM4_10	0	2	3	**True**	True	1.134
BP_RD_DIM4_10	1	1	3	False	False	1.429
BP_RD_DIM4_10	1	2	1	**True**	True	2.227
BP_RD_DIM4_11	0	1	3	False	False	0.668
BP_RD_DIM4_11	0	2	1	**True**	True	1.275
BP_RD_DIM4_11	1	1	3	False	False	0.607
BP_RD_DIM4_11	1	2	3	**True**	True	2.581
BP_RD_DIM4_12	0	1	3	False	False	0.765
BP_RD_DIM4_12	0	2	1	**True**	True	1.496
BP_RD_DIM4_12	1	1	3	False	False	0.592
BP_RD_DIM4_12	1	2	3	**True**	True	1.287
BP_RD_DIM4_13	0	1	3	False	False	5.786
BP_RD_DIM4_13	0	2	3	False	False	5.490
BP_RD_DIM4_13	1	1	3	False	False	2.819
BP_RD_DIM4_13	1	2	2	False	False	2.819
BP_RD_DIM4_14	0	1	3	False	False	3.112
BP_RD_DIM4_14	0	2	3	**True**	True	5.516
BP_RD_DIM4_14	1	1	3	False	False	1.980
BP_RD_DIM4_14	1	2	3	**True**	True	7.072
BP_RD_DIM4_15	0	1	3	**True**	True	3.535
BP_RD_DIM4_15	0	2	1	**True**	True	5.051
BP_RD_DIM4_15	1	1	3	**True**	True	5.657
BP_RD_DIM4_15	1	2	1	**True**	True	6.768
∅ [%]				45.000	51.667	2.911

TABLE A.42: Results Nelder-Mead for the randomly generated four-dimensional binpacking problems

A.3.4 NM-SD and CS-SD

Name	start index	replace rule	Stop NM	Global MP NM	Box-Local MP NM	Points NM [%]	Global MP NM-SD	Points NM-SD [%]
BP_RD_DIM3_1	0	1	3	False	False	1.818	**True**	4.091
BP_RD_DIM3_1	0	2	1	False	False	1.970	**True**	3.636
BP_RD_DIM3_1	1	1	3	False	False	2.727	**True**	3.939
BP_RD_DIM3_1	1	2	1	**True**	True	3.788	True	3.939
BP_RD_DIM3_2	0	1	3	False	False	1.515	**True**	4.209
BP_RD_DIM3_2	0	2	1	False	False	2.189	**True**	4.040
BP_RD_DIM3_2	1	1	3	False	False	3.367	**True**	6.734
BP_RD_DIM3_2	1	2	1	**True**	True	5.892	True	5.892
BP_RD_DIM3_3	0	1	3	False	False	6.569	**True**	18.248
BP_RD_DIM3_3	0	2	1	False	False	9.489	**True**	17.518
BP_RD_DIM3_3	1	1	3	False	False	8.759	**True**	23.358
BP_RD_DIM3_3	1	2	1	**True**	True	18.978	True	19.708
BP_RD_DIM3_4	0	1	3	False	False	5.028	**True**	13.966
BP_RD_DIM3_4	0	2	1	False	False	7.263	**True**	13.408
BP_RD_DIM3_4	1	1	3	False	False	6.704	**True**	20.112
BP_RD_DIM3_4	1	2	1	False	False	8.939	**True**	20.112
BP_RD_DIM3_5	0	1	3	**True**	True	1.297	True	3.026
BP_RD_DIM3_5	0	2	1	False	False	1.873	**True**	3.746
BP_RD_DIM3_5	1	1	3	False	False	3.458	**True**	5.764
BP_RD_DIM3_5	1	2	1	**True**	True	4.467	True	5.476
BP_RD_DIM3_6	0	1	3	**True**	True	3.614	True	5.622
BP_RD_DIM3_6	0	2	1	**True**	True	4.819	True	5.622
BP_RD_DIM3_6	1	1	3	False	False	4.819	**True**	18.876
BP_RD_DIM3_6	1	2	1	False	False	12.450	**True**	15.663
BP_RD_DIM3_7	0	1	3	False	False	2.270	**True**	4.681
BP_RD_DIM3_7	0	2	3	False	False	3.262	**True**	4.113
BP_RD_DIM3_7	1	1	3	False	False	2.553	**True**	5.957
BP_RD_DIM3_7	1	2	1	**True**	True	4.113	True	4.397
BP_RD_DIM3_8	0	1	3	**True**	True	2.308	True	5.385
BP_RD_DIM3_8	0	2	1	False	False	3.077	**True**	6.667
BP_RD_DIM3_8	1	1	3	False	False	5.128	**True**	13.333
BP_RD_DIM3_8	1	2	1	**True**	True	10.513	True	12.051
BP_RD_DIM3_9	0	1	3	False	False	23.636	**True**	41.818
BP_RD_DIM3_9	0	2	2	**True**	True	38.182	True	40.000
BP_RD_DIM3_9	1	1	3	False	False	23.636	**True**	45.455
BP_RD_DIM3_9	1	2	2	**True**	True	38.182	True	40.000
BP_RD_DIM3_10	0	1	3	**True**	True	25.806	True	45.161
BP_RD_DIM3_10	0	2	1	**True**	True	22.581	True	45.161
BP_RD_DIM3_10	1	1	3	False	False	20.968	**True**	61.290
BP_RD_DIM3_10	1	2	1	**True**	True	35.484	True	56.452
BP_RD_DIM3_11	0	1	3	**True**	True	5.213	True	13.744
BP_RD_DIM3_11	0	2	1	**True**	True	8.531	True	13.744
BP_RD_DIM3_11	1	1	3	False	False	5.687	**True**	16.588
BP_RD_DIM3_11	1	2	1	**True**	True	8.531	True	15.640
BP_RD_DIM3_12	0	1	3	False	False	2.321	False	6.118
BP_RD_DIM3_12	0	2	1	False	**True**	5.063	False	5.907
BP_RD_DIM3_12	1	1	3	False	False	1.899	False	6.118
BP_RD_DIM3_12	1	2	1	False	**True**	4.430	False	5.696
BP_RD_DIM3_13	0	1	3	False	False	2.041	**True**	5.751
BP_RD_DIM3_13	0	2	3	False	False	5.195	**True**	6.308
BP_RD_DIM3_13	1	1	3	False	False	1.670	**True**	12.059
BP_RD_DIM3_13	1	2	1	False	False	5.380	**True**	7.792
BP_RD_DIM3_14	0	1	3	**True**	True	5.806	True	13.548
BP_RD_DIM3_14	0	2	1	False	False	8.387	**True**	16.774
BP_RD_DIM3_14	1	1	3	False	False	7.742	**True**	20.645
BP_RD_DIM3_14	1	2	1	False	False	11.613	**True**	20.645
BP_RD_DIM3_15	0	1	3	**True**	True	2.550	True	3.116
BP_RD_DIM3_15	0	2	1	False	False	2.975	**True**	4.108
BP_RD_DIM3_15	1	1	3	False	False	2.550	**True**	5.382
BP_RD_DIM3_15	1	2	1	False	False	3.824	**True**	5.099
∅ [%]				33.333	36.667	8.248	93.333	14.724

TABLE A.43: NM-SD for the three-dimensional randomly generated binpacking modular system examples

Name	p^0	α	Global MP CS	Box-Local MP CS	Points CS [%]	Global MP CS-SD	Points CS-SD [%]
BP_RD_DIM3_1	(0,0,1)	1	False	False	1.818	**True**	3.788
BP_RD_DIM3_1	(0,0,1)	2	False	False	1.818	**True**	4.091
BP_RD_DIM3_1	(5,4,0)	1	**True**	True	3.939	True	4.697
BP_RD_DIM3_1	(5,4,0)	2	**True**	True	2.879	True	3.939
BP_RD_DIM3_2	(0,0,1)	1	**True**	True	1.684	True	2.525
BP_RD_DIM3_2	(0,0,1)	2	**True**	True	1.852	True	3.030
BP_RD_DIM3_2	(4,2,2)	1	False	False	4.882	**True**	6.734
BP_RD_DIM3_2	(4,2,2)	2	False	False	4.040	**True**	6.229
BP_RD_DIM3_3	(0,0,1)	1	False	False	8.029	**True**	16.788
BP_RD_DIM3_3	(0,0,1)	2	**True**	True	8.029	True	13.139
BP_RD_DIM3_3	(2,2,2)	1	False	False	13.869	**True**	21.898
BP_RD_DIM3_3	(2,2,2)	2	False	False	12.409	**True**	21.898
BP_RD_DIM3_4	(0,0,1)	1	**True**	True	5.587	True	8.380
BP_RD_DIM3_4	(0,0,1)	2	**True**	True	6.145	True	10.056
BP_RD_DIM3_4	(2,2,1)	1	False	False	8.939	**True**	21.229
BP_RD_DIM3_4	(2,2,1)	2	False	False	7.821	**True**	21.229
BP_RD_DIM3_5	(0,0,1)	1	**True**	True	1.441	True	2.594
BP_RD_DIM3_5	(0,0,1)	2	False	False	1.729	**True**	5.187
BP_RD_DIM3_5	(5,2,2)	1	**True**	True	4.899	True	5.908
BP_RD_DIM3_5	(5,2,2)	2	False	False	3.314	**True**	5.764
BP_RD_DIM3_6	(0,0,1)	1	False	False	3.213	**True**	8.434
BP_RD_DIM3_6	(0,0,1)	2	**True**	True	6.024	True	7.631
BP_RD_DIM3_6	(2,2,4)	1	**True**	True	12.048	True	13.655
BP_RD_DIM3_6	(2,2,4)	2	**True**	True	8.032	True	10.843
BP_RD_DIM3_7	(0,0,1)	1	**True**	True	3.404	True	3.972
BP_RD_DIM3_7	(0,0,1)	2	**True**	True	2.837	True	3.262
BP_RD_DIM3_7	(4,2,4)	1	**True**	True	4.823	True	5.390
BP_RD_DIM3_7	(4,2,4)	2	**True**	True	3.404	True	4.113
BP_RD_DIM3_8	(0,0,1)	1	**True**	True	2.821	True	4.872
BP_RD_DIM3_8	(0,0,1)	2	False	False	2.564	**True**	6.154
BP_RD_DIM3_8	(4,4,1)	1	False	False	8.462	**True**	13.333
BP_RD_DIM3_8	(4,4,1)	2	False	False	5.128	**True**	10.256
BP_RD_DIM3_9	(0,0,1)	1	**True**	True	25.455	True	30.909
BP_RD_DIM3_9	(0,0,1)	2	**True**	True	16.364	True	27.273
BP_RD_DIM3_9	(2,0,0)	1	**True**	True	41.818	True	45.455
BP_RD_DIM3_9	(2,0,0)	2	**True**	True	36.364	True	40.000
BP_RD_DIM3_10	(0,0,1)	1	**True**	True	24.194	True	43.548
BP_RD_DIM3_10	(0,0,1)	2	False	False	25.806	**True**	66.129
BP_RD_DIM3_10	(3,0,0)	1	False	False	14.516	**True**	64.516
BP_RD_DIM3_10	(3,0,0)	2	**True**	True	25.806	True	50.000
BP_RD_DIM3_11	(0,0,1)	1	**True**	True	7.583	True	14.218
BP_RD_DIM3_11	(0,0,1)	2	False	False	6.635	False	14.692
BP_RD_DIM3_11	(2,2,2)	1	**True**	True	7.583	True	14.218
BP_RD_DIM3_11	(2,2,2)	2	False	False	7.583	**True**	18.957
BP_RD_DIM3_12	(0,0,1)	1	False	False	3.376	False	6.540
BP_RD_DIM3_12	(0,0,1)	2	False	False	3.797	False	6.540
BP_RD_DIM3_12	(4,2,1)	1	False	**True**	3.586	False	5.063
BP_RD_DIM3_12	(4,2,1)	2	False	False	3.586	False	6.329
BP_RD_DIM3_13	(0,0,1)	1	False	False	2.968	**True**	6.122
BP_RD_DIM3_13	(0,0,1)	2	**True**	True	3.711	True	4.267
BP_RD_DIM3_13	(5,3,1)	1	False	False	4.824	**True**	7.978
BP_RD_DIM3_13	(5,3,1)	2	False	False	4.267	**True**	7.050
BP_RD_DIM3_14	(0,0,1)	1	**True**	True	7.097	True	12.258
BP_RD_DIM3_14	(0,0,1)	2	False	False	7.742	**True**	20.000
BP_RD_DIM3_14	(2,2,1)	1	False	False	10.323	**True**	21.935
BP_RD_DIM3_14	(2,2,1)	2	False	False	9.032	**True**	21.935
BP_RD_DIM3_15	(0,0,1)	1	False	False	2.266	**True**	4.108
BP_RD_DIM3_15	(0,0,1)	2	**True**	True	2.550	True	3.116
BP_RD_DIM3_15	(4,4,2)	1	**True**	True	4.533	True	5.099
BP_RD_DIM3_15	(4,4,2)	2	**True**	True	3.116	True	4.108
∅ [%]			50.000	51.667	8.073	91.667	14.223

TABLE A.44: CS-SD for the three-dimensional randomly generated binpacking modular system examples

Name	start index	replace rule	Stop NM	Global MP NM	Box-Local MP NM	Points NM [%]	Global MP NM-SD	Points NM-SD [%]
BP_RD_DIM4_1	0	1	3	False	False	0.949	**True**	5.775
BP_RD_DIM4_1	0	2	1	False	False	2.136	**True**	3.956
BP_RD_DIM4_1	1	1	3	False	False	1.661	**True**	3.877
BP_RD_DIM4_1	1	2	1	**True**	True	2.611	True	3.085
BP_RD_DIM4_2	0	1	3	False	False	2.135	**True**	12.989
BP_RD_DIM4_2	0	2	1	False	False	5.338	**True**	9.431
BP_RD_DIM4_2	1	1	3	False	False	4.270	**True**	13.167
BP_RD_DIM4_2	1	2	3	False	False	6.584	**True**	11.388
BP_RD_DIM4_3	0	1	3	**True**	True	7.769	True	10.025
BP_RD_DIM4_3	0	2	3	**True**	True	9.023	True	11.278
BP_RD_DIM4_3	1	1	3	False	False	5.764	**True**	13.033
BP_RD_DIM4_3	1	2	3	**True**	True	6.767	True	9.774
BP_RD_DIM4_4	0	1	3	False	False	0.473	False	1.570
BP_RD_DIM4_4	0	2	1	False	False	0.710	False	1.570
BP_RD_DIM4_4	1	1	3	False	False	0.194	False	1.849
BP_RD_DIM4_4	1	2	1	False	**True**	0.839	False	1.140
BP_RD_DIM4_5	0	1	3	False	False	1.744	**True**	10.610
BP_RD_DIM4_5	0	2	3	**True**	True	5.814	True	6.977
BP_RD_DIM4_5	1	1	3	False	False	2.035	**True**	8.285
BP_RD_DIM4_5	1	2	1	**True**	True	4.360	True	5.233
BP_RD_DIM4_6	0	1	3	False	False	5.187	**True**	10.951
BP_RD_DIM4_6	0	2	1	**True**	True	6.772	True	9.942
BP_RD_DIM4_6	1	1	3	**True**	True	2.017	True	5.764
BP_RD_DIM4_6	1	2	1	**True**	True	2.882	True	5.764
BP_RD_DIM4_7	0	1	3	False	False	0.577	False	1.441
BP_RD_DIM4_7	0	2	1	False	**True**	1.009	False	1.297
BP_RD_DIM4_7	1	1	3	False	**True**	0.913	False	1.441
BP_RD_DIM4_7	1	2	3	False	**True**	1.057	False	1.489
BP_RD_DIM4_8	0	1	3	**True**	True	2.736	True	3.951
BP_RD_DIM4_8	0	2	1	**True**	True	2.533	True	4.053
BP_RD_DIM4_8	1	1	3	**True**	True	2.026	True	4.357
BP_RD_DIM4_8	1	2	1	**True**	True	2.938	True	4.357
BP_RD_DIM4_9	0	1	3	False	False	0.404	**True**	1.830
BP_RD_DIM4_9	0	2	1	**True**	True	0.808	True	1.046
BP_RD_DIM4_9	1	1	3	False	False	0.665	**True**	1.260
BP_RD_DIM4_9	1	2	1	**True**	True	0.784	True	0.903
BP_RD_DIM4_10	0	1	3	False	False	0.504	**True**	2.731
BP_RD_DIM4_10	0	2	3	**True**	True	1.134	True	1.765
BP_RD_DIM4_10	1	1	3	False	False	1.429	**True**	4.160
BP_RD_DIM4_10	1	2	1	**True**	True	2.227	True	2.899
BP_RD_DIM4_11	0	1	3	False	False	0.668	**True**	2.581
BP_RD_DIM4_11	0	2	1	**True**	True	1.275	True	1.458
BP_RD_DIM4_11	1	1	3	False	False	0.607	**True**	4.616
BP_RD_DIM4_11	1	2	3	**True**	True	2.581	True	2.855
BP_RD_DIM4_12	0	1	3	False	False	0.765	**True**	1.879
BP_RD_DIM4_12	0	2	1	**True**	True	1.496	True	1.635
BP_RD_DIM4_12	1	1	3	False	False	0.592	**True**	3.445
BP_RD_DIM4_12	1	2	3	**True**	True	1.287	True	1.635
BP_RD_DIM4_13	0	1	3	False	False	5.786	**True**	9.941
BP_RD_DIM4_13	0	2	3	False	False	5.490	**True**	9.792
BP_RD_DIM4_13	1	1	3	False	False	2.819	**True**	13.501
BP_RD_DIM4_13	1	2	2	False	False	2.819	**True**	12.760
BP_RD_DIM4_14	0	1	3	False	False	3.112	**True**	10.467
BP_RD_DIM4_14	0	2	3	**True**	True	5.516	True	5.941
BP_RD_DIM4_14	1	1	3	False	False	1.980	**True**	15.276
BP_RD_DIM4_14	1	2	3	**True**	True	7.072	True	7.638
BP_RD_DIM4_15	0	1	3	**True**	True	3.535	True	6.869
BP_RD_DIM4_15	0	2	1	**True**	True	5.051	True	6.667
BP_RD_DIM4_15	1	1	3	**True**	True	5.657	True	8.889
BP_RD_DIM4_15	1	2	1	**True**	True	6.768	True	8.788
∅ [%]				45.000	51.667	2.911	86.667	5.884

TABLE A.45: NM-SD for the four-dimensional randomly generated binpacking modular system examples

Name	p^0	α	Global MP CS	Box-Local MP CS	Points CS [%]	Global MP CS-SD	Points CS-SD [%]
BP_RD_DIM4_1	(0,0,0,1)	1	False	False	1.187	**True**	3.797
BP_RD_DIM4_1	(0,0,0,1)	2	False	False	1.187	**True**	4.035
BP_RD_DIM4_1	(3,3,0,1)	1	**True**	True	2.215	True	3.085
BP_RD_DIM4_1	(3,3,0,1)	2	False	False	1.582	**True**	6.646
BP_RD_DIM4_2	(0,0,0,1)	1	False	False	2.491	**True**	8.007
BP_RD_DIM4_2	(0,0,0,1)	2	**True**	True	2.491	True	5.694
BP_RD_DIM4_2	(2,2,1,0)	1	False	False	3.381	**True**	14.769
BP_RD_DIM4_2	(2,2,1,0)	2	False	False	2.669	**True**	14.769
BP_RD_DIM4_3	(0,0,0,1)	1	False	False	5.013	**True**	13.283
BP_RD_DIM4_3	(0,0,0,1)	2	**True**	True	5.514	True	9.023
BP_RD_DIM4_3	(2,1,2,0)	1	False	False	5.764	**True**	13.283
BP_RD_DIM4_3	(2,1,2,0)	2	**True**	True	5.764	True	9.273
BP_RD_DIM4_4	(0,0,0,1)	1	False	False	0.538	False	1.613
BP_RD_DIM4_4	(0,0,0,1)	2	False	False	0.495	**True**	1.269
BP_RD_DIM4_4	(4,4,1,0)	1	False	**True**	0.559	False	1.054
BP_RD_DIM4_4	(4,4,1,0)	2	False	False	0.559	**True**	1.312
BP_RD_DIM4_5	(0,0,0,1)	1	False	False	2.180	**True**	6.977
BP_RD_DIM4_5	(0,0,0,1)	2	False	False	2.180	**True**	7.413
BP_RD_DIM4_5	(2,2,1,0)	1	False	False	2.471	**True**	6.831
BP_RD_DIM4_5	(2,2,1,0)	2	False	False	2.471	**True**	7.703
BP_RD_DIM4_6	(0,0,0,1)	1	**True**	True	6.052	True	9.510
BP_RD_DIM4_6	(0,0,0,1)	2	False	False	3.890	**True**	13.977
BP_RD_DIM4_6	(3,2,0,0)	1	False	False	3.170	**True**	9.078
BP_RD_DIM4_6	(3,2,0,0)	2	False	False	3.026	**True**	9.798
BP_RD_DIM4_7	(0,0,0,1)	1	False	False	0.480	False	1.369
BP_RD_DIM4_7	(0,0,0,1)	2	False	False	0.552	False	1.393
BP_RD_DIM4_7	(4,2,2,0)	1	False	False	0.745	False	1.802
BP_RD_DIM4_7	(4,2,2,0)	2	False	False	0.601	False	1.730
BP_RD_DIM4_8	(0,0,0,1)	1	**True**	True	1.418	True	3.749
BP_RD_DIM4_8	(0,0,0,1)	2	**True**	True	1.520	True	4.458
BP_RD_DIM4_8	(2,2,1,1)	1	**True**	True	2.533	True	4.965
BP_RD_DIM4_8	(2,2,1,1)	2	**True**	True	2.330	True	4.762
BP_RD_DIM4_9	(0,0,0,1)	1	False	False	0.475	**True**	1.260
BP_RD_DIM4_9	(0,0,0,1)	2	False	False	0.452	**True**	1.236
BP_RD_DIM4_9	(2,4,2,1)	1	**True**	True	0.832	True	1.093
BP_RD_DIM4_9	(2,4,2,1)	2	False	False	0.689	**True**	1.473
BP_RD_DIM4_10	(0,0,0,1)	1	False	False	0.630	**True**	2.185
BP_RD_DIM4_10	(0,0,0,1)	2	False	False	0.630	**True**	2.311
BP_RD_DIM4_10	(4,4,1,0)	1	False	False	1.681	**True**	3.571
BP_RD_DIM4_10	(4,4,1,0)	2	False	False	1.050	**True**	3.025
BP_RD_DIM4_11	(0,0,0,1)	1	False	False	0.790	**True**	1.792
BP_RD_DIM4_11	(0,0,0,1)	2	False	False	0.820	**True**	1.822
BP_RD_DIM4_11	(2,4,2,0)	1	False	False	0.577	**True**	4.646
BP_RD_DIM4_11	(2,4,2,0)	2	False	False	0.577	**True**	4.737
BP_RD_DIM4_12	(0,0,0,1)	1	False	False	0.696	**True**	1.844
BP_RD_DIM4_12	(0,0,0,1)	2	**True**	True	0.800	True	1.287
BP_RD_DIM4_12	(4,3,2,1)	1	**True**	True	1.427	True	1.809
BP_RD_DIM4_12	(4,3,2,1)	2	False	False	0.905	**True**	3.166
BP_RD_DIM4_13	(0,0,0,1)	1	**True**	True	5.341	True	7.567
BP_RD_DIM4_13	(0,0,0,1)	2	False	False	3.264	**True**	8.605
BP_RD_DIM4_13	(4,1,0,0)	1	False	False	4.006	**True**	13.353
BP_RD_DIM4_13	(4,1,0,0)	2	False	False	2.226	**True**	12.166
BP_RD_DIM4_14	(0,0,0,1)	1	False	False	3.112	**True**	6.648
BP_RD_DIM4_14	(0,0,0,1)	2	False	False	2.687	**True**	9.335
BP_RD_DIM4_14	(2,2,0,0)	1	False	False	4.385	**True**	10.325
BP_RD_DIM4_14	(2,2,0,0)	2	**True**	True	4.243	True	5.375
BP_RD_DIM4_15	(0,0,0,1)	1	**True**	True	4.040	True	7.374
BP_RD_DIM4_15	(0,0,0,1)	2	**True**	True	2.929	True	6.566
BP_RD_DIM4_15	(4,1,0,0)	1	False	False	5.657	**True**	10.404
BP_RD_DIM4_15	(4,1,0,0)	2	**True**	True	2.828	True	6.061
∅ [%]			28.333	30.000	2.246	90.000	5.791

TABLE A.46: CS-SD for the four-dimensional randomly generated binpacking modular system examples

Bibliography

[1] Charles Audet and Warren Hare. *Derivative-Free and Blackbox Optimization*. Cham: Springer International Publishing, 2017. ISBN: 978-3-319-68912-8. DOI: 10.1007/978-3-319-68913-5.

[2] *Baukästen im Automobilbau*. visited on 12/01/2025. URL: https://www.mobile.de/magazin/artikel/baukaesten-im-automobilbau-hintergruende-vorteile-risiken-10398.

[3] Maren Beck, Steffen Bolender, and Oliver Stein. "Optimal configurations for modular systems at the example of crane bridges." In: *Optimization and Engineering* (Nov. 2024). DOI: 10.1007/s11081-024-09936-x.

[4] *Begriffsbestimmung: Modulbau oder Containerbau?* visited on 12/01/2025. URL: https://www.industriebau-online.de/aktuelles/begriffsbestimmung-modulbau-oder-containerbau/.

[5] J F Benders. "Partitioning procedures for solving mixed-variables programming problems." In: *Numerische Mathematik* 4.1 (1962), pp. 238–252. DOI: 10.1007/BF01386316.

[6] Steffen Bolender et al. "Skalierbarer Modularer Brückenkranträger in Segmentbauweise." In: *Logistics Journal* (2017), pp. 9–14. DOI: 10.2195/lj_Proc_bolender_de_201710_01.

[7] Stephen Boyd and Vandenberghe Lieven. *Convex Optimization*. Cambridge University Press, 2004. ISBN: 9780521833783.

[8] Giuseppe C. Calafiore and Laurent El Ghaoui. *Optimization Models*. Cambridge University Press, Oct. 2014. ISBN: 9781107050877. DOI: 10.1017/CBO9781107279667.

[9] *Citypark Karlsruhe*. self-taken photo, 12/01/2025.

[10] Andrew R Conn, Katya Scheinberg, and Luis N Vicente. *Introduction to Derivative-Free Optimization*. Tech. rep. Society for Industrial and Applied Mathematics, 2009.

[11] *Constraints Gurobi*. visited on 04/28/2025. URL: https://docs.gurobi.com/projects/optimizer/en/current/concepts/modeling/constraints.html.

[12] *Cranebridge Pixabay*. visited on 03/27/2025. URL: https://pixabay.com/photos/crane-gripping-system-scheffer-3626174/.

[13] Duan Li and Xiaoling Sun. *Nonlinear Integer Programming*. 1st ed. Springer New York, NY, May 2006. ISBN: 0-387-29503-8. DOI: 10.1007/0-387-32995-1.

[14] *Entwicklung eines skalierbaren Brückenkranträgers in Segmentbauweise*. visited on 12/10/2025. URL: https://www.ifl.kit.edu/5011_kranbruecke.php.

[15] Otto Forster. *Analysis 2*. Wiesbaden: Vieweg+Teubner, 2010. ISBN: 978-3-8348-1231-5. DOI: 10.1007/978-3-8348-8103-8.

[16] Jaroslav Fowkes, Lindon Roberts, and Árpád Bűrmen. "PyCUTEst: an open source Python package of optimization test problems." In: *Journal of Open Source Software* 7.78 (Oct. 2022), p. 4377. DOI: 10.21105/joss.04377.

[17] Kikuo Fujita. "Product variety optimization under modular architecture." In: *Computer-Aided Design* 34.12 (Oct. 2002), pp. 953–965. DOI: 10.1016/S0010-4485(01)00149-X.

[18] Kikuo Fujita, Hisato Sakaguchi, and Shinsuke Akagi. "Product Variety Deployment and its Optimization Under Modular Architecture and Module Commonalization." In: *Volume 4: 4th Design for Manufacturing Conference*. American Society of Mechanical Engineers, Sept. 1999, pp. 337–348. ISBN: 978-0-7918-1974-6. DOI: 10.1115/DETC99/DFM-8923.

[19] Kikuo Fujita and Hiroko Yoshida. "Product variety optimization simultaneously designing module combination and module attributes." In: *Concurrent Engineering Research and Applications* 12.2 (2004), pp. 105–118. DOI: 10.1177/1063293X04044758.

[20] A M Geoffrion. "Generalized Benders decomposition." In: *Journal of Optimization Theory and Applications* 10.4 (1972), pp. 237–260. DOI: 10.1007/BF00934810.

[21] *Gitlab Repository Maren Beck*. URL: https://gitlab.com/marenbeck1/marenbeck-dissertation.

[22] Peter Gritzmann. *Grundlagen der Mathematischen Optimierung*. Wiesbaden: Springer Fachmedien Wiesbaden, 2013. ISBN: 978-3-528-07290-2. DOI: 10.1007/978-3-8348-2011-2.

[23] Monique Guignard and Siwhan Kim. "Lagrangean decomposition: A model yielding stronger lagrangean bounds." In: *Mathematical Programming* 39.2 (1987), pp. 215–228. DOI: 10.1007/BF02592954.

[24] *Gurobi documentation*. visited on 06/11/2025. URL: https://docs.gurobi.com/_/downloads/optimizer/en/12.0/pdf/.

[25] *Gurobi history*. visited on 12/10/2025. URL: https://www.gurobi.com/company/our-story/.

[26] *Gurobi titlepage*. visited on 06/11/2025. URL: https://www.gurobi.com/solutions/gurobi-optimizer/.

[27] Fabian Habiger. "Numerische Verfahren in der Optimierung für die Clusteranalyse in der Standortplanung." PhD thesis. Continuous Optimization, 2023.

[28] Hiroshi Hira and Kazuo Murota. "M-convex functions and tree metrics." In: *Japan Journal of Industrial and Applied Mathematics* 21.3 (2004), pp. 391–403. DOI: 10.1007/BF03167590.

[29] Abiodun M. Ikotun et al. "K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data." In: *Information Sciences* 622 (Apr. 2023), pp. 178–210. DOI: `10.1016/j.ins.2022.11.139`.

[30] *Indicator constraints CPLEX*. visited on 04/28/04/2025. URL: `https://www.ibm.com/docs/en/cofz/12.10.0?topic=optimization-indicator-constraints-in`.

[31] Bernhard Korte and Jens Vygen. *Combinatorial Optimization*. 6th ed. Springer Berlin, Heidelberg, 2018. DOI: `https://doi.org/10.1007/978-3-662-56039-6`. URL: `http://www.springer.com/series/13`.

[32] Rainer Lasch. "Standortmanagement." In: *Strategisches und operatives Logistikmanagement: Distribution*. Wiesbaden: Springer Fachmedien Wiesbaden, 2020, pp. 179–239. DOI: `10.1007/978-3-658-31869-7_7`.

[33] Andrea Lodi. "Mixed Integer Programming Computation." In: *50 Years of Integer Programming 1958-2008*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 619–645. DOI: `10.1007/978-3-540-68279-0_16`.

[34] Andrea Lodi, Silvano Martello, and Daniele Vigo. "Recent advances on two-dimensional bin packing problems." In: *Discrete Applied Mathematics* 123 (2002), pp. 379–396.

[35] *Logistics, Operations Management, Algorithms and Design*. visited on 12/10/2025. URL: `https://www.ifl.kit.edu/5948.php`.

[36] Bruce L Miller. "On Minimizing Nonseparable Functions Defined on the Integers with an Inventory Application." In: *SIAM Journal on Applied Mathematics* 21.1 (1971), pp. 166–185.

[37] *Modular construction childcare center*. visited on 12/01/2025. URL: `https://mr-service-gmbh.de/container/kita-container/`.

[38] *Modular Construction Illustration*. visited on 12/01/2025. URL: `https://www.cpbau.de/unternehmen/aktuelles/detail/modulbau-vs-containerbau-worin-liegt-der-unterschied/#lightbox%5B23%5D`.

[39] *Modular construction kindergarden*. visited on 12/01/2025. URL: `https://adapteo.de/unser-angebot/kita`.

[40] *Modular construction school*. visited on 12/01/2025. URL: `https://mr-service-gmbh.de/container/schul-container/`.

[41] *Modular construction vs. container buildings*. visited on 12/01/2025. URL: `https://www.cpbau.de/en/company/news/detail/modular-construction-vs-container-buildings-what-is-the-difference/`.

[42] Satoko Moriguchi et al. "Scaling and proximity properties of integrally convex functions." In: *27th International Symposium on Algorithms and Computation (ISAAC 2016)*. 2016, pp. 51–57.

[43] *Motorisierungen VW*. visited on 12/01/2025. URL: `https://www.volkswagen-newsroom.com/de/der-neue-polo-die-fahrvorstellung-2574/hoechst-effiziente-mpi-tsi-tgi-tdi-und-dsg-2602`.

[44] Kazuo Murota. *Discrete Convex Analysis*. Society for Industrial and Applied Mathematics, 2003. DOI: `10.1137/1.9780898718508`. URL: `https://epubs.siam.org/doi/abs/10.1137/1.9780898718508`.

[45] Kazuo Murota. “Discrete Convex Analysis.” In: *Hausdorff Institute of Mathematics, Summer School*. Sept. 2015, pp. 1–21.

[46] Kazuo Murota. “Discrete convex analysis.” In: *Mathematical Programming* 83.1 (1998), pp. 313–371. DOI: 10.1007/BF02680565.

[47] Kazuo Murota. “Discrete Convex Analysis : A Tool for Economics and Game Theory.” In: *arXiv preprint arXiv:2212.03598* (2022).

[48] John Ashworth Nelder and Roger Mead. *A simplex method for function minimization*. Tech. rep. Jan. 1965, pp. 308–313. URL: https://academic.oup.com/comjnl/article/7/4/308/354237.

[49] Jan Oellerich et al. “Modeling and investigation of a segmented truss structure for overhead cranes.” In: *Logistics Journal* 2018 (2018). DOI: 10.2195/lj_Proc_oellerich_de_201811_01.

[50] *On the Value Function of a Mixed Integer Linear Optimization Problem and an Algorithm for its Construction*. visited on 10/21/2025. URL: https://optimization-online.org/wp-content/uploads/2014/08/4475-1.pdf.

[51] Josef Ponn and Udo Lindemann. *Konzeptentwicklung und Gestaltung technischer Produkte*. Springer Berlin Heidelberg, 2011. DOI: 10.1007/978-3-642-20580-4.

[52] *Process simulation in the automotive industry*. visited on 12/02/2025. URL: https://www.welt.de/wirtschaft/article161106023/Airbus-stellt-Konzept-fuer-modulare-Flugzeugkabinen-vor.html.

[53] *PyCUTEst documentation*. visited on 10/13/2025. URL: https://jfowkes.github.io/pycutest/_build/html/index.html.

[54] R Tyrrell Rockafellar. *Convex Analysis*. Princeton University Press, 1970.

[55] H H Rosenbrock. “An Automatic Method for Finding the Greatest or Least Value of a Function.” In: *The Computer Journal* 3.3 (Jan. 1960), pp. 175–184. DOI: 10.1093/comjnl/3.3.175.

[56] Timothy W. Simpson, Jonathan R. Maier, and Farrokh Mistree. “Product platform design: Method and application.” In: *Research in Engineering Design - Theory, Applications, and Concurrent Engineering* 13.1 (2001), pp. 2–22. DOI: 10.1007/s001630100002.

[57] Oliver Stein. *Basic Concepts of Global Optimization*. Vol. 5. Berlin, Heidelberg: Springer Berlin Heidelberg, 2024. ISBN: 978-3-662-66239-7. DOI: 10.1007/978-3-662-66240-3.

[58] Oliver Stein. *Grundzüge der Parametrischen Optimierung*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2021. ISBN: 978-3-662-61989-6. DOI: 10.1007/978-3-662-61990-2.

[59] Mitchell M. Tseng and Chenjie Wang. “Modular Design.” In: *CIRP Encyclopedia of Production Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 895–897. DOI: 10.1007/978-3-642-20617-7_6460.

[60] Nobuyuki Tsuchimura, Satoko Moriguchi, and Kazuo Murota. "Discrete Convex Optimization Solvers and Demonstration Softwares *." In: *Transactions of the Japan Society for Industrial and Applied Mathematics* 23 (2013), pp. 233–252. DOI: 10.11540/jsiamt.23.2_233.

[61] Conrad S. Tucker and Harrison M. Kim. "Optimal product portfolio formulation by merging predictive data mining with multilevel optimization." In: *Journal of Mechanical Design* 130.4 (2008). DOI: 10.1115/1.2838336.

[62] *VW MQB*. visited on 11/29/2025. URL: https://www.volkswagen-newsroom.com/de/pressemitteilungen/baukastenstrategie-als-erfolgsformel-der-mqb-feiert-zehnjaehriges-jubilaeum-8030.

[63] Dongkuan Xu and Yingjie Tian. "A Comprehensive Survey of Clustering Algorithms." In: *Annals of Data Science* 2.2 (June 2015), pp. 165–193. DOI: 10.1007/s40745-015-0040-1.

[64] Ahmet S. Yigit, A. Galip Ulsoy, and Ali Allahverdi. "Optimizing modular product design for reconfigurable manufacturing." In: *Journal of Intelligent Manufacturing* 13.4 (2002), pp. 309–316. DOI: 10.1023/A:1016032714680.