DS&OR Lab
Department Business Information Systems
Faculty of Business Administration and Economics
University of Paderborn

Master Thesis

Models and Algorithms for Extended Three-Dimensional Bin Packing Problems in Air Cargo Applications

by

Marius Merschformann
6466141
Tegelweg 5, 33102 Paderborn
mmarius@mail.upb.de

submitted to

Prof. Dr. Leena Suhl
Dr. Kostja Siefen

April 12, 2014

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Paderborn, April 12, 2014

_____

(forename surname)

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne unerlaubte fremde Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen und Hilfsmitteln wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Paderborn, 12.04.2014

_____

(Vorname Name)

Abstract

This thesis investigates different model formulations and heuristic algorithms applied to the packing problem arising in air cargo. The main goal is a volume efficient packing of a given set of heterogeneous pieces inside a set of differently sized containers. In addition to simple cuboid shapes, more complex forms, called "Tetris"-shapes, are investigated. Furthermore, special requirements of the application need to be considered. All of this is comprised in the proposal of two MIP-formulations and three constructive heuristic approaches combined with a simple metaheuristic, the Greedy Adaptive Search Procedure. Afterwards, these are evaluated on two sets of instances, including one set with certain "Tetris"-pieces.

Keywords: Container Loading Problem, Load planning, Three-dimensional Bin Packing, Three-dimensional Knapsack, Mixed Integer Program, Heuristic, Metaheuristic, Optimization

Abstract (German)

Diese Arbeit untersucht verschiedene Modellformulierungen und Lösungsverfahren im Kontext eines Packungsproblems aus dem Bereich der Luftfracht. Das Ziel ist das Generieren von gültigen Packplänen, welche eine effiziente Auslastung des verfügbaren Volumens gewährleisten. Dabei muss eine Menge heterogener Güter einer Menge von unterschiedlich großen Containern zugeordnet werden. Weiterhin sollen komplexere Formen durch eine Approximation mittels "Tetris"-Formen berücksichtigt werden. Insgesamt müssen hierbei spezielle Anforderungen des Umfelds beachtet werden. All dies wird untersucht für zwei vorgeschlagene MIP Formulierungen sowie drei verschiedene heuristische Methoden integriert in einer einfachen Metaheuristik, der Greedy Adaptive Search Procedure. Zuletzt werden alle Methoden miteinander auf zwei Mengen von Instanzen verglichen, wovon eine "Tetris"-Formen enthält.

Contents

List of Figures

List of Algorithms

List of Tables

1 Introduction

Nowadays, the delivery of goods and mail by air is crucial to global trade. Due to the fast transportation it connects the global industry even further. Although it only handles 3 per cent of the total world tonnage, "[...] it is the 35 per cent and upwards value of goods that makes it such a vital business" (see [Sal13], p. XVII). One basic step when handling the freight in the air cargo logistic chain is the efficient packing of the goods inside special containers. In reality this is done by highly experienced load planners in a manual process. Thus, a high pressure rests on these persons, especially due to the fact that in air cargo decisions have to be made very quickly and delays are extremely costly. The core problem is to build feasible packing plans that can be executed just in time by loading the goods according to these. The general problem of packing items into given containers while utilizing the available space as good as possible can be seen in many contexts, e.g. trucks or container ships. Even when packing the trunk of a car, before going on a long vacation, one has to use the available space efficiently. These specialists, in contrast, have to do the task over and over again during every shift to handle the huge amount of shipments dispatched daily (see [ICA12], p. 71).
Therefore, a more reliable automated system that supports the process and relieves these agents would be quite valuable. This system has to be able to generate packing plans which utilize the available volume as good as possible while still satisfying all other requirements. These efficient packing plans may also help to tackle big economical and ecological challenges arising due to "[...] the global effect of human activities in our planet's sustainability" (see [PAVTO08], p. 412). The ecological efficiency is especially relevant to the field of air cargo - pollution due to aircrafts is a central topic when discussing global warming in media and administration. Increasing the utilization of the space an aircraft offers potentially reduces the number of necessary flights to ship all goods. However, the number of goods shipped by air is still rising, which renders the efficient usage of the planes cargo space even more crucial. Additionally airports, more specifically cargo hubs, underly heavy competition with other airports (see [Zha03]).

1.1 The problem

The topic of this thesis is to develop a system capable of generating feasible packing plans that can be used to load freight into containers accordingly. These packing plans have to satisfy certain requirements while using the available space of the containers efficiently. The main goal is to supply plans of reasonable quality (i.e. high volume utilization) in an appropriate time horizon that can be executed immediately. Thus,

this work focuses on formulations of the problem and methods capable of solving them, while the concepts of providing these plans in an intuitive way is not discussed in this thesis.

1.2 Goal of this thesis

The objective of this thesis is to investigate optimization techniques capable of solving the discussed problem. For that, the basic requirements applicable in the field of air cargo have to be considered. Furthermore, the limited time-horizon available to generate high quality packing plans has to be respected. Therefore, the concept and implementation of optimization models and heuristics are the subject of this thesis. On the one hand, two model-formulations are proposed in the form of mixed integer programs (MIP). On the other hand, in order to generate high quality solutions even for large problem instances, different heuristic techniques are proposed. These are expected to generate solutions of adequate quality in a much smaller time-horizon. Since air freight containers are substantially smaller than in maritime cargo for example, the heterogeneous shapes of the goods are also part of this investigation. This way the items can be handled more suitably, because the typical assumption of using only the bounding box of complex shaped items is not always practical and simply wastes available space (see [Fas13], p. 280).

1.3 Outline and contributions

This work is divided into seven parts. Following this short introduction the fundamentals are introduced in chapter 2. Besides explaining the basics about air cargo a short scientific classification of this thesis' topic is done. This scientific background is extended in chapter 3, introducing the current state-of-the-art work by a literature review. Next the main part of this work is discussed. At first, the model formulations and the basic concepts of these are proposed in chapter 4. The algorithms conceived and implemented by this work are shown in chapter 5. In order to evaluate and compare the different methods proposed, an evaluation is done and the computational results of it are analyzed in chapter 6. Finally, chapter 7 concludes the work.
The contributions of this thesis are as follows:

1. A MIP formulation of three-dimensional bin packing respecting cuboid items with integrated application specific constraints
2. A MIP formulation of three-dimensional bin packing respecting "Tetris" items with integrated application specific constraints
3. Three primal constructive heuristic algorithms respecting application specific requirements solving the particular packing problem
4. An integration of the constructive techniques into a Greedy Adaptive Search Procedure metaheuristic

2 Fundamentals of freight handling in air cargo

This section provides the fundamentals of the discussed problem and its origins. At first an overview of the field of application (see section 2.1) as well as the basic planning process from which the discussed problem emerges is given (see section 2.2). The second part reviews two related classic combinatorial optimization problems to supply the scientific background of the practical problem (see section 2.3).

2.1 Air cargo at a glance

Logistics in general is a wide field thoroughly discussed in the literature. The term "logistics" consolidates all processes of transportation and storage (see [AIK+08], p. 3). The general objective of economic efficiency in terms of the logistic processes implies to cause minimal costs for the appropriate performance and to have maximal performance for the respective costs. Therefore, the complete logistic chain has to be taken into account. This, immediately, also includes sub-tasks like the packing of freight when handling air cargo. Furthermore, ecological objectives need to be considered, especially in the context of today's air cargo.
The logistic chain in the application of air cargo can be basically subdivided into specific steps. For each step the particular agent handles the cargo to be shipped from a starting location to a place of destination (see [Sal13], p. 5). The agents are defined according to [Sal13]:

- A *shipper*, a person or company, requests the movement of goods. This can be for example the manufacturer or seller of a product.

- A *freight forwarder* plans and negotiates the complete transportation with a carrier. Furthermore, an appropriate packaging and correct and complete documents are supplied by these. The forwarder also picks up the cargo and transports it to the departing airport in time as well as arranging the transport from the destination airport to the end-customer.
  The forwarders typically offer three different service levels, mainly distinguished by the transit time:

  Priority This is the highest level of service for very critical shipments. These are handled by the fastest available connections or even direct flights with a transit time of 1-2 days.

Standard The standard level typically has a transit-time of 3-4 days.

Economy The cheapest level of service usually results in transit times of 5-6 days.

- A *carrier* or airline transports the cargo from the source to the destination airport, possibly over some transition airports on the way. The shipment may be transported with different planes or even trucks on its way. A significant amount of freight is shipped alongside personal baggage in the belly of passenger aircrafts. The term belly refers to the lower compartment beneath the main deck. The other possibilities of transportation are cargo airplanes and the road feeder service (RFS). The latter is used for short distances and airports not served by the airline. In that case a truck is used instead of a plane, driving to the destined location. A cargo airplane offers the largest cargo capacity. Additional to the sub-deck these mainly converted passenger airplanes allow loading in the main-deck. In every case the freight has to be efficiently packed into a container in terms of the utilized volume.

- The *consignee* is the receiver of the goods. This might be a manufacturer, an importer, a retailer or even the shipper itself. In the process the consignee, the shipper or a third party might be the owner of the goods.

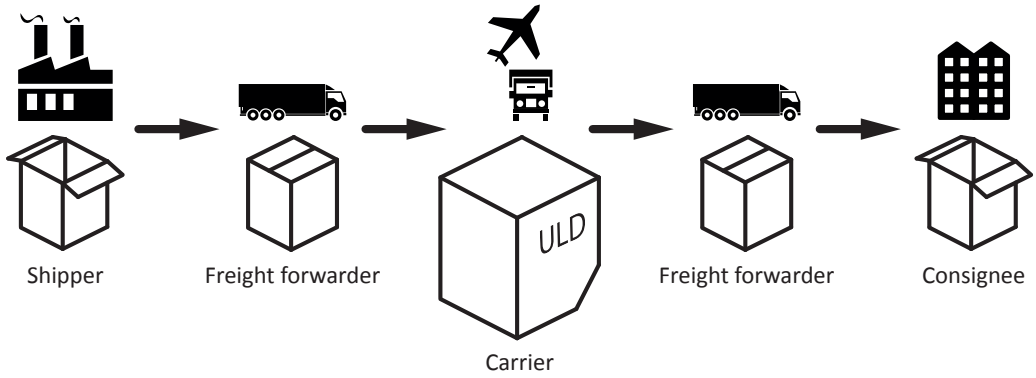


Figure 2.1: Depiction of a typical air cargo process

In the first step the shipper requests the transportation of a set of goods, which is handled by the freight forwarder (see Figure 2.1). The forwarder then organizes the transportation to the respective starting airport. The shipments now get prepared by the carrier at the hub of the airport. This particular step is the fundamental process this thesis concentrates on. Besides other actions, the goods have to be assembled and packed into containers or palettes. The detailed handling of the goods at a cargo hub, especially the assembly and disassembly of the packings, will be discussed in the next section. Eventually, the cargo is transported by an aircraft, possibly along transit airports, to the destination airport. As the last step the freight forwarder transports the cargo to the final destination. The air transportation done by the carrier hereby

may be handled in different ways, like mentioned above. While 25 percent of the goods are transported by commercial passenger aircrafts of the carriers (see [Sal13], p. 4), also cargo airplanes and road-feeder-services (RFS) are used. The complete transportation of the shipment is maintained by a receipt called air waybill (AWB), which contains all information like a customs declaration or a contract of carriage.

2.2 The special air cargo load planning problem

In a cargo hub of an air cargo carrier the daily business mainly consists of transshipping goods from land-side to airside, airside to airside or even from landside to landside, using the so-called RFS. In every case Unit Load Devices (ULD) need to be disassembled for inbound cargo and assembled for outbound cargo. These ULDs define both containers and palettes, which are specifically shaped to fit into an airplane. Many different types of these need to be considered and only compatible ones fit at a loading position of a specific plane (see [LSL11], p. 1271). For a depiction of a typical shape of an ULD see Figure 2.2. In this work the more general term container will be used. Palettes and containers in this case are very similar, because every loading position inside the aircraft is bounded by the planes' hull. So the usable space on a palette looks very similar to the typically angled containers used in air cargo. Instead of having sidewalls the cargo packed on a palette is generally protected and fixed by a loading net. The goods packed into the containers are denoted by the simple term 'pieces'. The main objective in this step is to pack all pieces into a set of available containers depending on particular requirements and goals. Since the number of pieces handled daily is very high, the support of this task by a decision support system (DSS) gets more and more interesting. The term DSS is used to describe an information system capable of automatically processing the data fundamental to a planning problem and suggesting a solution to it (see [SM13], p. 89). The optimization engine, as the core of such a system, shall be investigated here.

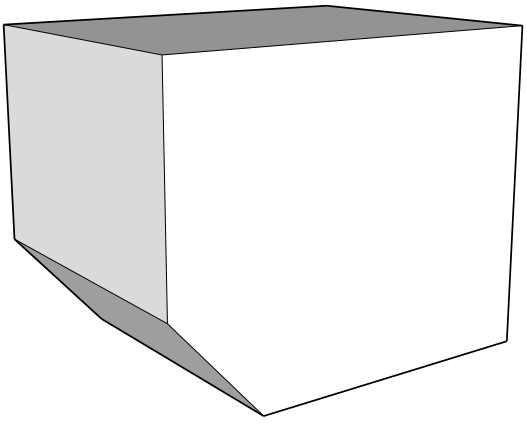


Figure 2.2: Depiction of the shape of a LD3 (a typical ULD)

Nowadays, the allocation of the cargo to the available containers is normally done by loadmasters who use interactive graphical tools with drag and drop capabilities (see [LSL11], p. 1272). The process is mainly a manual activity relying on the ramp master's experience and vague guidelines. These guidelines may be given by simple rules like packing light above heavy freight or to begin with big pieces. To enable more flexibility and react to ad-hoc changes due to late arrivals or other requests, the final packing, the so-called build-up, is done shortly before the plane departs. Thus, the efficiency of the packing's planning and execution is crucial (see [TZL08], p. 1). However, in practice sufficient time is not always available to handle this process manually while also being able to fine tune the packing for given goals (see [LSL11], p. 1272). So, loadplanners underlie heavy workload and need to keep track of many flights planned in parallel.
Therefore, a DSS should be able to generate packing plans in a very short time horizon while considering the different requirements for a valid solution. The resulting packing plans should be visualized and provided in a customizable way, allowing manual adjustments made by the loadplanners. In this way the process potentially gains more robustness, because a first feasible solution on how the pieces should be allocated and packed is always available. At best such a packing for a flight would be available in realtime, supplying plans which can be executed immediately while also reporting information about the packing. This thesis emphasizes on the optimization techniques necessary to supply these plans in a reasonable time-horizon. Besides that, on any given day the cargo load planning problem needs to be solved thousands of times worldwide, which makes a reliable system to support this task very interesting (see [ICA12], p. 71).

The concept used here encapsulates the optimization of the packing plans inside a system with defined data inputs and outputs. Input data of this system is required in the form of piece-information and container-information. All pieces and necessary characteristics of them have to be supplied at a certain time as well as information about the available containers and their extent. At the same time the process needs to be efficient enough to react to late changes and still supply solutions of reasonable quality. After acquiring all information an internal representation of the problem instance is generated. Depending on the supplied method type to use, either a mathematical model formulation of the problem is transformed into a solver representation or a heuristic method is applied to approximate a solution of reasonable quality. This choice mainly depends on the time available for the process and is currently dedicated to an external agent using the system. For the heuristic approach three different techniques are proposed in chapter 5 of this work. The model formulations used to integrate exact algorithms given by commercial solvers are introduced in chapter 4. After a solution is calculated it is transformed into an output representation. This solution consists of the information about the allocation of the pieces as well as their positions and orientations inside the containers.

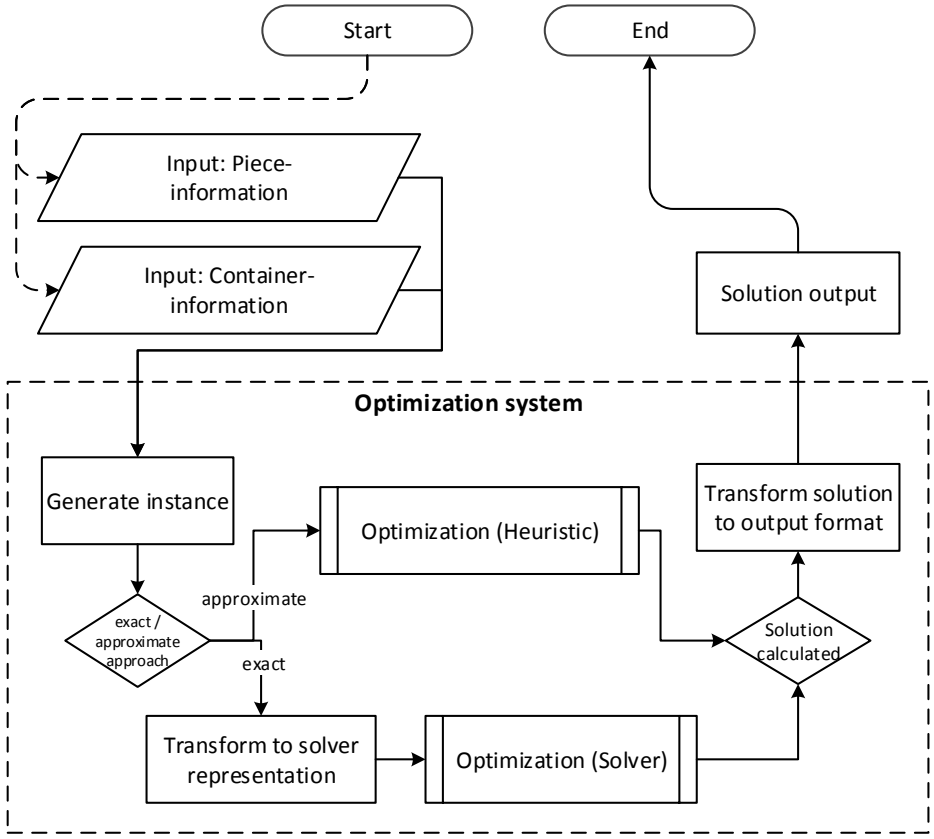


Figure 2.3: The basic optimization system

Problem characteristics

In the following the typical requirements which apply for valid packing plans and the problem's characteristics are summarized. Furthermore, the optimization's objective as well as the degrees of freedom, respectively the information defining a solution are described.
The degrees of freedom existing when building valid packing plans can be split up into three parts. These also implicitly define the structure of a solution to the packing problem:

Allocation For every piece available for packing it has to be decided whether the piece is packed at all. Furthermore, if a piece is packed then it has also to be assigned to exactly one container.

Position The position of the packed piece in the container's domain has to be supplied. This is comprised of three components identifying the exact position in three-dimensional space.

Rotation It should be possible for a piece to be rotated about all axes. At this, 90° rotations are sufficient to describe the most distinctive ones. Therefore, pieces will be assumed to be consisting of only orthogonal and parallel sides.

Although continuous rotations allow more degrees of freedom they are in most cases inefficient, because the sides of two pieces would not be parallel to each other any longer. Thus, both pieces together would consume more volume. Besides that, packing diagonal oriented pieces is not overly practicable, because they are not able to support and stabilize other pieces. On the other side this could be useful in rare cases when handling very long pieces that cannot fit a container without, for example, rotating them by 45°, but such cases are not considered in this thesis.

When generating valid packing plans certain requisitions have to be met. These should be observed by the decision support system to relieve the loadplanner and to build feasible and realizable packing plans. Some of these requirements are very basic like the non-overlapping of packed pieces. Other requirements underlie certain regulations or are simply given by nature, like the stability of the packing (see [PSL12], p. 4; [JMY12b]). Most of these can also be found in other fields of logistic (see [AIK+08], p. 177).

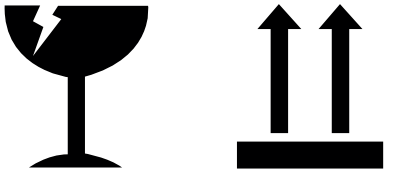
Non-overlapping For every feasible solution no overlaps of two pieces may exist. While in reality being physically impossible, the particular requirement must also be considered by the optimization.

Container integrity Another very basic restriction needs to be respected to keep pieces from protruding beyond the containers domain. This is also straightforward, because a piece allocated to a container should also be completely included inside of it.

Load stability Besides these basic requisitions, some more specific requirements need to be considered. One of them is the stability, i.e. it has to be avoided that the packing will collapse when building it up according to the generated plan. Thus, at least a simplification of gravity has to be respected, preventing pieces from floating in mid-air. In this work the requirement is given in the way that a piece has either to be supported by another one positioned directly below it or it has to lie on the ground. Since the weight is assumed to be uniformly distributed the center of gravity of a piece has to be positioned within the supporting area of the one below. Also its lower face has to match the upper face of the supporting piece regarding the height. Thus, the stability requirement, mainly arising due to the consideration of the three dimensions, introduces a very complex challenge, because the influence of gravity and other forces in transportation, per se, cannot easily be simplified while still enabling the resulting packing to cope with all possible situations. However, in this thesis the requirement will be handled like stated above.

Load bearing Some pieces may be marked as “fragile” and consequently have to be handled with care at all times. For a valid packing plan this should be ensured by limiting the weight which is put onto such pieces. Or it may be even completely

forbidden to load any other piece onto them. A typical marker for this is depicted by icon (a) in Figure 2.4.



(a) "Fragile" (b) "This side up"

Figure 2.4: Examples of handling markers.

Forbidden orientations Load handling advices like "this side up" or similar ones limiting the orientations in which a piece may be packed have to be respected. This means that the specified rotations resulting in such forbidden orientations need to be banned. The icon (b) of Figure 2.4 depicts a typical marker for this.

Compatibility Incompatibilities due to dangerous goods regulations (DGR) and other load handling advices have to be respected. The DGR are specified by the International Air Transport Association IATA and define standards which have to be followed when handling dangerous freight (see [IAT10]). As a result of this some pieces have to be strictly separated from each other. Thus, these cannot be packed into the same container. This has to be satisfied at all times so that safety can be assured. Due to not following these requirement several accidents already happened in the past (see [Sal13], p.79). Examples of dangerous goods are explosive materials, flammable gases, flammable liquids, , infectious substances or radioactive material. These also have to be visually marked at the outside of the packaging (see Figure 2.5).

(a) "Explosive" (b) "Flammable gas" (c) "Flammable liquid" (d) "Infectious substance" (e) "Radioactive"

Figure 2.5: Examples of dangerous goods markers[1]

When packing pieces into a container not all of them can be assumed to have cuboid shapes. So, the intention is to exploit space which would be lost in the case of only

[1] `http://www.iata.org/publications/dgr/Pages/hazard-labels.aspx` (retrieved: 04.03.2014)

handling bounding boxes. Thus, the accuracy of a solution shall be increased. Since this also increases the complexity of the problem, an approximation is necessary to be able to compromise between accuracy and runtime.

Piece shapes Not all goods are simple cuboid packages, but may also have more complex forms. The space lost when only handling the pieces smallest cuboid hull may therefore be crucial. Furthermore, some stability measures may no longer hold if a complex piece is handled like a cuboid.

Container shapes Given by the shape of an aircrafts hull most of the containers used in air cargo have angled side-walls. Like mentioned before even the pallets have to be build to fit exactly into the planes hull. In both cases special areas of the otherwise cuboid shaped container have to be spared. These sections are often not orthogonal (see [PSL12], p. 2 and Figure 2.2).

Despite generating feasible packing plans the goal is to utilize the available space as efficiently as possible while also minimizing the number of used containers. Both goals have in common that more dense packings are likely to provide a better solution quality (see [ZZOL12], p. 452). This thesis will mainly concentrate on the first goal: the best volume utilization. However, both goals are defined as follows:

Volume utilization A common approach is to utilize the volume of a given set of available containers as effectively as possible. Thus, as many goods as possible can be packed and shipped in one flight. This objective is especially interesting, if more piece volume shall be packed as container volume is available.

Container minimization Another approach is to minimize the number of used containers for a fixed set of goods that have to be shipped. This goal becomes more interesting as there is quite lesser volume of the pieces to pack than available container volume.

These listed challenges define the complete list of issues this thesis concentrates on. The goal is to generate packing plans according to the aspects defined above. To achieve this a basic optimization engine is proposed as shown by Figure 2.3. The motivation is that despite models and algorithms already existing and partially covering requirements for container loading, substantial need for research exists (see [AIK+08], p. 178). A more detailed study of the current literature in this area is done in chapter 3.

2.3 Scientific fundamentals of packing problems

In order to bring the practical load planning problem of air cargo and the scientific fundamentals together, the following section comprises two classic combinatorial optimization problems which are the base of three-dimensional packing. The first one is the classic knapsack problem and the second one is called bin packing.
In the case of the knapsack problem a set of items has to be packed into a single container (knapsack), each with a weight and a value. Thereby, the weight defines a specific consumption of the container's capacity when packing the respective item, while the value defines the profit gained when packing the item. Hence, the objective is to pack the maximal sum of values given by the items (see Equation 2.1) without exceeding the limit of the maximum weight of the container (see Equation 2.2). Thus, the mathematical formulation defines a set of items $p \in \mathcal{P}$ which have to be packed into the container with a given limit of the maximal weight W. The binary variable ψ_p indicates whether item p is packed, or not. The requirement of the container's capacity is ensured by Equation 2.2 limiting the sum of the pieces' weight w_p by the container's limit. Hence, the classic knapsack problem only considers one container. The objective function then maximizes the value of the packed pieces (see Equation 2.1). For a scientific definition of the problem see for example [Law76] (p. 62) and more recently [WN99] (p. 5).

Knapsack problem:

$$\max \quad \sum_{p \in \mathcal{P}} v_p \cdot \psi_p \tag{2.1}$$

$$\text{subject to} \quad \sum_{p \in \mathcal{P}} w_p \cdot \psi_p \leq W \tag{2.2}$$

$$p \in \mathcal{P}, \psi_p \in \{0, 1\} \tag{2.3}$$

The bin packing problem is very similar to the knapsack problem as a set of items $p \in \mathcal{P}$ have to be allocated to a set of containers $c \in \mathcal{C}$ without exceeding their respective weight limit W_c. In this case the number of used containers is minimized while all items have to be allocated to one of them. For that purpose, the binary variable η_c depicts whether a container c is used, respectively active. A second binary variable ψ_{cp} indicates whether piece p is allocated to container c. Thus, Equation 2.5 ensures that the maximal weight of each container is not exceeded by summing all pieces' weights allocated to that particular container. Also the allocation of every piece to exactly one container has to be ensured (see Equation 2.6), because otherwise minimizing the number of active containers would result in the trivial optimal solution of no packed pieces at all. The objective function given by Equation 2.4 simply minimizes the number of activated containers. For a scientific definition of the problem see for example [All11] (p. 9).

Bin packing problem:

$$\min \quad \sum_{c \in \mathcal{C}} \eta_c \tag{2.4}$$

$$\text{subject to} \quad \sum_{p \in \mathcal{P}} w_p \cdot \psi_{cp} \leq W_c \cdot \eta_c \qquad \forall c \in \mathcal{C} \tag{2.5}$$

$$\sum_{c \in \mathcal{C}} \psi_{cp} = 1 \qquad \forall p \in \mathcal{P} \tag{2.6}$$

$$c \in \mathcal{C}, p \in \mathcal{P}, \eta_c \in \{0,1\}, \psi_{cp} \in \{0,1\} \tag{2.7}$$

The main differences between those two problems are the objective function, the number of containers and whether all pieces have to be packed or not. The classic knapsack problem only considers a single container while the bin packing problem allows multiple containers, but due to the objective function the actual allocation of pieces has to be fixed.
Both of these problems are known to be $\mathcal{NP}$-complete. For the knapsack problem see [Kar72] (p. 95) and more recently [Woe03] (p.198). For the bin packing problem see [GJ79] (p. 226) and more recently [GJ00] (p. 226). For the three-dimensional case of those problems the complexity still holds, because "[...] 3D-BPP is strongly NP-hard because it is a generalization of the well-known (one-dimensional) bin packing problem (1D-BPP) [...]" (see [MPV00], p. 256). The same applies for the knapsack problem, which can also be shown by a generalization (see [EP09], p. 1026). A similar investigation has also been done for an offline version of the "Tetris"-game which is similar to the three-dimensional packing of "Tetris"-pieces, but instead limited to two dimensions. In that case, the term 'offline' refers to the situation in which all information about the pieces' characteristics are given before starting the optimization, in contrast to the situation of the actual game. As a result it is shown that the problem is $\mathcal{NP}$-complete. This is mentioned, because such "Tetris"-pieces get explained in more detail later in this work (see subsection 3.1.2).
In terms of this thesis' problem a combination of these two with the extension to three dimensions is given. According to [WHS07] the underlying problem of this work would have to be classified as three-dimensional tetris/cuboid (Multiple) Heterogeneous Knapsack Problem, because the authors propose a typology for all cutting and packing problems which at the first step sub-divides the different classes by the objective. Since the main objective of this problem is to maximize the utilized space this term applies. In the case of the second objective type - minimizing the number of used containers - the problem would be classified as three-dimensional tetris/cuboid Residual Bin Packing Problem.

3 Literature review

Since the first publications about multi-dimensional packing problems (especially [GG65]), several works in that field have been published. This chapter provides an overview of the current literature addressing similar problems. Although only few applications discussed by the literature consider air cargo, many ideas and techniques of three-dimensional packing problems are transferable to the application of this thesis, except for special requirements and extensions. But these are missing anyway in most of the current literature (see section 3.3). In the first part of this chapter different mixed integer mathematical formulations are described. Most of the work for the three-dimensional case is quite recent, probably because of the fact that computation power to solve these problems with typical exact algorithms integrated in commercial solvers like CPLEX (see [IBM13]) is critical. Due to the advancements of the hardware as well as the algorithms, complex three-dimensional problems become solvable in reasonable time at all. But since real-world problems of this type often are still unsolvable in the time available because of their size, heuristics are essential to compute solutions. The different methods proposed in this field are introduced in the second part of this chapter. The notation derived from the different papers has been modified to harmonize the formal notation with this work.
At the beginning of the chapter an overview of the different mixed integer program (MIP) formulations used in the literature is given (see section 3.1). The sub-sections 3.1.1 and 3.1.2 then introduce two formulations, which represent the foundation of the ones introduced and investigated within this thesis. Afterwards, an overview of the current literature about heuristic methods applied to similar problems is given (see section 3.2). Furthermore, the concept of Extreme-Points (EP) is introduced in subsection 3.2.1, which also is a fundamental concept of the heuristic approaches proposed in chapter 5. Additionally the basic methods, which are extended by this work, are described. At last subsection 3.2.2 introduces a simple metaheuristic successfully applied to solve similar problems. This method is also extended by this thesis.

3.1 Mixed integer formulations

After undergoing a long development since the late 50s (see [Gom10]) major advancements were made in solving mixed integer programs (see [AW13]). Nowadays, such formulations are used to tackle many different real-world planning-problems. Although the bin-packing-problem or the knapsack-problem are both well-known for a long time, an application of the three-dimensional case of these formulations to solve real-world problems is quite recent. This is probably due to the fact that the three-

dimensional bin packing problem (3DBPP) underlying these formulations is known to be NP-hard in the strong sense (see [MPV00]), while adding more requirements of the real-world increases its complexity even more. As a consequence heuristics are often used to solve real-world sized instances of these problems (see [WLGdS10]).

The work by [CLS95] uses a MIP-formulation to solve the problem of packing non-uniform sized pieces of cuboid shape into a set of containers. The authors consider multiple containers, variable piece-sizes, multiple orientations of the pieces as well as non-overlapping requirements, which are mandatory to these problems. Also some special requirements like weight-balancing are considered with basic constraints added to the model formulation. Like most of the works in this area the authors conclude that a "[...] more efficient solution procedure is needed to solve large scale container loading problems" (see [CLS95], p. 75). [WLGdS10] adopt the approach and formulate it as a single bin packing problem with variable bin height. I.e., the container is of unlimited size regarding one dimension. Instead of maximizing the utilized volume like [CLS95], here the resulting packing's height is minimized. The authors also conclude that solving the model formulation with standard approaches is still not suitable for real-time bin packing problems. To overcome this issue a genetic algorithm is designed, which encodes the insertion order of the pieces as well as their orientations in the chromosome. The algorithm then uses classic mutation and crossover techniques to vary the population in each step. The evaluation of the fitness for each individual of the population employs the technique proposed by [CPT08] using EPs to insert the pieces into the container.
[ABM12] propose a different formulation discretizing the pieces and the space available for packing, thus indexing every discrete sub-space. Every allocation of a piece to a container then occupies a subset of indices of the container's space. The authors state that the described formulation has a strong LP-relaxation, but since the dimension of the discrete space-units is given by the largest common denominator, the formulation strongly depends on the side-lengths of the containers and pieces. In other words, the number of variables increases dramatically when experiencing adverse length ratios even for small problems. Therefore, the authors employ a dynamic programming approach to achieve efficient discretization of the containers and pieces, thus trying to reduce this effect.
The work by [JMY12b] introduces a mathematical formulation for the basic problem considering a single container and cuboid shaped pieces. Initially, rotations are not possible in this formulation, but the authors already consider some real-world requirements like load bearing and stability. As stated before, load bearing describes the requirement not to overload fragile pieces, thus avoiding damaged freight. The stability requirement demands all pieces inside a container to be positioned in a stable way, i.e. every piece has to be supported either by another piece beneath it or the ground. The authors likewise state that only moderate instance sizes can be solved with commercial solvers using this approach.
[JMY12a] focus on a formulation handling multi-drop constraints. This means that the sequence in which the container is unloaded has to be respected. The field of

application addresses the loading of trucks with fixed routes. On its route a truck sequentially has to unload subsets of the pieces loaded. The authors state that solutions properly representing the situations could be obtained, "although only problems of moderate size can be solved optimally" (see [JMY12a], p. 74).

3.1.1 Cuboid formulation

The first model formulation introduced here in more detail is the one proposed by [PSL12]. The underlying problem tackled by the authors' work is residing in the same field of application as the one discussed in this paper: The air-cargo load planning problem. Likewise a set of pieces has to be assigned to a set of containers, respectively ULDs, and positioned in three-dimensional space without any overlaps. The containers in this formulation are all of the same size, thus implementing the three-dimensional single bin-size bin packing problem (3D-SBSBPP). The approach is to implement a MIP-formulation of the three-dimensional bin packing problem in the particular case of air cargo. The basic formulation is oriented by the work of [CLS95]. Since the first model formulation (see section 4.2) investigated in this work relies on the particular paper, this basic model is described in the following. Here a set of pieces denoted by $\mathcal{P}$ has to be packed into containers denoted by $\mathcal{C}$. The whole concept of the formulation relies on two corner points per piece depicting the position of the front-left-bottom (FLB) corner χ_{pd} and rear-right-top (RRT) corner χ'_{pd} of piece p in dimension d. The dimensions considered here are simply given by $d \in \mathcal{D}$ with $\mathcal{D} := \{x, y, z\}$. Constraint 3.1 then ensures that a container c has to be active ($\eta_c = 1$) in the solution when a piece p is assigned to it ($\psi_{pc} = 1$). Since the formulation is based on classical bin packing, thus minimizing the number of containers used, constraint 3.2 ensures that every piece is packed into one container. In constraint 3.3 the maximal capacity C of one container is respected by keeping the summed weights (M_p) of the pieces assigned to it smaller than C. Every container has the same maximal payload in this formulation. Constraint 3.4 denotes the requirement of every piece not to extend the container's domain. Therefore the position of the piece's RRT corner point (χ'_{pd}) has to be less than the container's length (L^{C}_d) regarding every dimension d. Since the FLB corner point (χ_{pd}) is modeled by positive continuous variables, no piece can extend the domain in the opposite direction.

Basic allocation and domain constraints:

$$\forall p \in \mathcal{P}, c \in \mathcal{C} : \psi_{pc} \leq \eta_c \tag{3.1}$$

$$\forall p \in \mathcal{P} : \sum_{c \in \mathcal{C}} \psi_{pc} = 1 \tag{3.2}$$

$$\forall c \in \mathcal{C} : \sum_{p \in \mathcal{P}} \psi_{pc} \cdot M_p \leq C \cdot \eta_c \tag{3.3}$$

$$\forall p \in \mathcal{P}, d \in \mathcal{D} : \chi'_{pd} \leq L^{\mathsf{C}}_d \tag{3.4}$$

The constraints starting from 3.5 up to 3.9 implicitly formulate the rotation matrix, which enables 90° rotations of the pieces. The constraints can be rewritten as shown in Equation 3.10 with the matrix given by R_p (see [PSL12]). Every element of the matrix is a binary variable $\rho_{pr_1r_2}$. Together with the requirement, that the sum of every row and every column of R_p has to be one, the matrix transforms the lengths of the three axes according to the used rotation into the RRT and FLB corner positions. This is a simplified approach of general rotation matrices used to transform a position in euclidean space. Usually sin and cos are necessary to model a complete continuous rotation matrix, but since only 90° rotations are used and the pieces are cuboids this linear formulation is possible.

Rotation constraints:

$$\forall p \in \mathcal{P} : \chi'_{px} - \chi_{px} = \rho_{p11} \cdot L^{\mathtt{P}}_{px} + \rho_{p12} \cdot L^{\mathtt{P}}_{py} + \rho_{p13} \cdot L^{\mathtt{P}}_{pz} \tag{3.5}$$

$$\forall p \in \mathcal{P} : \chi'_{py} - \chi_{py} = \rho_{p21} \cdot L^{\mathtt{P}}_{px} + \rho_{p22} \cdot L^{\mathtt{P}}_{py} + \rho_{p23} \cdot L^{\mathtt{P}}_{pz} \tag{3.6}$$

$$\forall p \in \mathcal{P} : \chi'_{pz} - \chi_{pz} = \rho_{p31} \cdot L^{\mathtt{P}}_{px} + \rho_{p32} \cdot L^{\mathtt{P}}_{py} + \rho_{p33} \cdot L^{\mathtt{P}}_{pz} \tag{3.7}$$

$$\forall p \in \mathcal{P}, r_2 \in \mathcal{R} : \sum_{r_1 \in \mathcal{R}} \rho_{pr_1r_2} = 1 \tag{3.8}$$

$$\forall p \in \mathcal{P}, r_1 \in \mathcal{R} : \sum_{r_2 \in \mathcal{R}} \rho_{pr_1r_2} = 1 \tag{3.9}$$

$$\begin{pmatrix} \chi'_{px} - \chi_{px} \\ \chi'_{py} - \chi_{py} \\ \chi'_{pz} - \chi_{pz} \end{pmatrix} = \underbrace{\begin{pmatrix} \rho_{p11} & \rho_{p12} & \rho_{p13} \\ \rho_{p21} & \rho_{p22} & \rho_{p23} \\ \rho_{p31} & \rho_{p32} & \rho_{p33} \end{pmatrix}}_{R_p} \cdot \begin{pmatrix} L^{\mathtt{P}}_{px} \\ L^{\mathtt{P}}_{py} \\ L^{\mathtt{P}}_{pz} \end{pmatrix} \qquad \forall p \in \mathcal{P} \tag{3.10}$$

To avoid overlapping of two pieces it is sufficient to require that one piece has to be positioned in front of (behind, on the left of, on the right of, above or below) another one. At least one of these relative position markers has to be true. In that case it is ensured that the two considered pieces do not overlap. This has to hold for every combination of two pieces, which are allocated to the same container. The relative position marker for two pieces p_1 and p_2 is modeled as a binary variable $\tau^b_{dp_2p_1}$ for the in front of / left / below case respectively $\tau^a_{dp_2p_1}$ for the behind / right / above case for every dimension $d \in \mathcal{D}$. This results in six variables per piece combination. These variables are linked to the positions of the pieces by constraints 3.11 and 3.12. If a relative position marker $\tau^b_{dp_2p_1}$ respectively $\tau^a_{dp_2p_1}$ attains the value 1 the difference $\chi_{p_1d} - \chi'_{p_2d}$, respectively $\chi_{p_2d} - \chi'_{p_1d}$, has to be positive, thus no overlap exists in dimension d since the pieces lie next to each other. Constraint 3.13 ensures that at least one of the relative position markers is used. Certainly an exception has to be made in the case that two pieces are not packed into the same container. Therefore the constraint only needs to be satisfied if ψ_{p_1c} and ψ_{p_2c} indicate that both pieces are

allocated to the same container c. Otherwise the last parts of constraints 3.11 and 3.12, respectively, relax this requisition.

Non-overlapping constraints:

$$\begin{aligned}\forall p_1, p_2 \in \mathcal{P}, c \in \mathcal{C}, d \in \mathcal{D} : \chi_{p_1 d} - \chi'_{p_2 d} \\ + \left(1 - \tau^b_{dp_2p_1}\right) \cdot M \\ + (2 - (\psi_{p_1 c} + \psi_{p_2 c})) \cdot M \geq 0 \end{aligned} \tag{3.11}$$

$$\begin{aligned}\forall p_1, p_2 \in \mathcal{P}, c \in \mathcal{C}, d \in \mathcal{D} : \chi_{p_2 d} - \chi'_{p_1 d} \\ + \left(1 - \tau^a_{dp_2p_1}\right) \cdot M \\ + (2 - (\psi_{p_1 c} + \psi_{p_2 c})) \cdot M \geq 0 \end{aligned} \tag{3.12}$$

$$\forall p_1, p_2 \in \mathcal{P} : \sum_{d \in \mathcal{D}} \tau^b_{dp_2p_1} + \tau^a_{dp_2p_1} \geq 1 \tag{3.13}$$

The goal of this approach is to minimize the unused volume of the containers as seen in Equation 3.14. This is done by summing the volume $V_c^{\mathtt{C}}$ of all active containers and subtracting the sum of the volume $V_p^{\mathtt{P}}$ of all pieces to be packed. This term is then minimized.

$$\min z = \sum_{c \in \mathcal{C}} \eta_c \cdot V_c^{\mathtt{C}} - \sum_{p \in \mathcal{P}} V_p^{\mathtt{P}} \tag{3.14}$$

Since all the containers have the same size this is equivalent to minimizing the number of used containers (see [PSL12]). This is depicted in Equation 3.15.

$$\min z = \sum_{c \in \mathcal{C}} \eta_c \tag{3.15}$$

In addition the authors emphasize some special air-cargo requirements like the stability of the resulting packing. To ensure the packing does not collapse a first vertical stability criterion is applied, which requires each piece either to lie on the ground, or to be placed on another piece. This means that the vertical position of the piece's lower side has to match the position of the other piece's upper side, while the base area has to be positioned above the other piece regarding x and y. Simultaneously the horizontal stability has to be satisfied, i.e. the piece's center of gravity has to lie inside the support area given by the pieces below it. The pieces' weight is hereby assumed to be uniformly distributed. Furthermore, forbidden rotations are considered by limiting the valid states of the explicitly formulated rotation matrix. Finally, fragile pieces are modeled by prohibiting these to become supporting pieces for stability sakes. Thus, no other piece is allowed to lie on a fragile one so that it is less likely for it to break during transportation. The used formulation employs techniques proposed by [JMY12a]. This MIP is the basic foundation for the model proposed in section 4.2.

3.1.2 "Tetris" formulation

The second paper providing a foundation for this work is the publication by [Fas13]. The author proposes a MIP formulation capable of dealing with "Tetris"-like pieces when packing a single container. This means that any possible rectangular construct consisting of multiple (tied together) cuboid components can be modeled. The approximation of complex shapes is discussed in more detail in subsection 4.1.1. A first formulation was proposed in [Fas99] and [Fas04]. It was then improved in [Fas08] alongside with a MIP based heuristic algorithm which incrementally adds new items to the container, while keeping the relative position markers fixed. In [Fas13] the author also focuses on a non-linear formulation of the problem, the packing of arbitrary polygons.
In the considered paper the field of application, namely the cargo accommodation of space vehicles, is somewhat similar to the air-cargo load planning, which is the field of application of this work. The container to be filled in their work is a so-called Automated Transfer Vehicle (ATV). Such a vehicle is used to transport supplies to outer space and was developed by the European Space Agency (ESA). Due to the very specific application particular requirements have to be satisfied and more complex forms than simple cuboids need to be considered. In the introduced work some balancing constraints are integrated, which assume the weight of each piece to be uniformly distributed and require the packing's center of gravity to stay inside a predefined area. Since no such requirements are considered by this work, the following description will focus only on the basic model formulation.
The set of pieces is again denoted by $\mathcal{P}$, while only one container is considered. The variable ψ_p indicates whether a piece is allocated to the container. In contrast to [PSL12] not the complete rotation matrix is modeled, but a discrete set of orientations ($\mathcal{O}$) is used. Constraint 3.16 states that if a piece $p \in \mathcal{P}$ is packed into the container, one orientation $o \in \mathcal{O}$ of the piece has to be active, indicated by $\vartheta_{po} = 1$. For a "Tetris"-like item, which is not symmetrical, 24 distinct orientations are possible when rotating by 90° about all axes. In subsection 4.1.2 the basics about the orientations and vertices discussed in the following are described in more detail. The position of each is determined by a point in space called the local reference frame (χ^o_{pd}). The position of each vertex v of each cuboid component b of all the "Tetris"-like pieces p is then linked by constraint 3.17 relative to the local reference frame. Again, the relative position of every vertex P^{B}_{pbovd} depends on the orientation index o currently in use. I.e. for all necessary orientations P^{B}_{pbovd} describes the position of the respective vertex, or the center of the component, if $v = 0$. Constraint 3.18 in connection with 3.19 ensure that every component b of each piece p stays inside the containers domain given by its vertices $P^{\mathsf{C}}_{v'd}$ for every dimension d. This is achieved by introducing a non-negative continuous variable $\lambda_{pbv'v}$. The sum of this variable, across all vertices v' of the container, is equal to one, if the corresponding piece is allocated $\psi_p = 1$ (see constraint 3.19). Since the variable is always positive no position of a piece's vertex χ_{pbvd} may extent the container's domain. At most one of these variables may attain the value one ($\lambda_{pbv'v} = 1$) resulting in the position of the piece's vertex χ_{pbvd} equal

to the vertex of the container $P^{\mathtt{C}}_{v'd}$. Thus, the piece borders at the container's sidewall.

Domain and orthogonality constraints:

$$\forall p \in \mathcal{P} : \sum_{o \in \mathcal{O}} \vartheta_{po} = \psi_p \tag{3.16}$$

$$\forall d \in \mathcal{D}, p \in \mathcal{P}, b \in \mathcal{B}_p, v \in \mathcal{V}^* : \chi_{pbvd} = \chi^o_{pd} + \sum_{o \in \mathcal{O}} P^{\mathtt{P}}_{pbovd} \cdot \vartheta_{po} \tag{3.17}$$

$$\forall d \in \mathcal{D}, p \in \mathcal{P}, b \in \mathcal{B}_p, v \in \mathcal{V} : \chi_{pbvd} = \sum_{v' \in \mathcal{V}} P^{\mathtt{C}}_{v'd} \cdot \lambda_{pbv'v} \tag{3.18}$$

$$\forall p \in \mathcal{P}, b \in \mathcal{B}_p, v \in \mathcal{V} : \sum_{v' \in \mathcal{V}} \lambda_{pbv'v} = \psi_p \tag{3.19}$$

Moreover, overlapping has to be avoided. Similar to the relative position markers discussed before, $\sigma^+_{p_1p_2b_1b_2d}$ and $\sigma^-_{p_1p_2b_1b_2d}$ are defined for every piece p_1 and each of its components b_1 in combination with each other component b_2 of every other piece p_2. But in this formulation the distance between the center points χ_{pb0d} (the center point is a special "vertex" with $v = 0$) has to be at least the half of the length of the components for the particular dimension d. E.g. in the case of two components with length $L^{\mathtt{B}}_{p_1b_1ox} = 2$ and $L^{\mathtt{B}}_{p_2b_2ox} = 1$, then for dimension $d = x$ the distance between their centers has to be ≥ 1.5. Since the distance is an absolute value the constraint is split into two parts (constraints 3.20 and 3.21). This relation is linked by constraint 3.22 to sustain the non-overlapping requisition by arranging two components side-by-side for at least one dimension. The two last constraints 3.23 and 3.24 are introduced to tighten the linear relaxation. These can only be used in the special case of packing a single container (see [Fas13]). The author also proposes some further reformulations of the model to get a better linear relaxation. [Pad00] even proposes a complete reformulation of the very first model formulation (see [Fas99]) to tighten the relaxation and reduce the amount of variables and constraints, but the conclusion states that a branch-and-cut algorithm would be necessary, which "[...] should push the limits for computability far beyond those attainable by branch-and-bound" (see [Pad00], p. 14).

Non-overlapping constraints:

$$\begin{aligned} &\forall d \in \mathcal{D}, p_1 \in \mathcal{P}, p_2 \in \mathcal{P}, p_1 < p_2, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2} : \\ &\chi_{p_1b_10d} - \chi_{p_2b_20d} \geq \frac{1}{2} \sum_{o \in \mathcal{O}} \left(L^{\mathtt{B}}_{p_1b_1od} \cdot \vartheta_{p_1o} + L^{\mathtt{B}}_{p_2b_2od} \cdot \vartheta_{p_2o} \right) - L^{\mathtt{C}}_d \cdot \left(1 - \sigma^+_{p_1p_2b_1b_2d} \right) \end{aligned} \tag{3.20}$$

$$\begin{aligned} &\forall d \in \mathcal{D}, p_1 \in \mathcal{P}, p_2 \in \mathcal{P}, p_1 < p_2, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2} : \\ &\chi_{p_2b_20d} - \chi_{p_1b_10d} \geq \frac{1}{2} \sum_{o \in \mathcal{O}} \left(L^{\mathtt{B}}_{p_1b_1od} \cdot \vartheta_{p_1o} + L^{\mathtt{B}}_{p_2b_2od} \cdot \vartheta_{p_2o} \right) - L^{\mathtt{C}}_d \cdot \left(1 - \sigma^-_{p_1p_2b_1b_2d} \right) \end{aligned} \tag{3.21}$$

$$\forall p_1 \in \mathcal{P}, p_2 \in \mathcal{P}_2, p_1 < p_2, \forall b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2} : \sum_{d \in \mathcal{D}} \left(\sigma^{+}_{p_1 p_2 b_1 b_2 d} + \sigma^{-}_{p_1 p_2 b_1 b_2 d}\right) \geq \psi_{p_1} + \psi_{p_2} - 1 \tag{3.22}$$

$$\forall p_1 \in \mathcal{P}, p_2 \in \mathcal{P}, p_1 < p_2, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2} : \sum_{d \in \mathcal{D}} \left(\sigma^{+}_{p_1 p_2 b_1 b_2 d} + \sigma^{-}_{p_1 p_2 b_1 b_2 d}\right) \leq \psi_{p_1} \tag{3.23}$$

$$\forall p_1 \in \mathcal{P}, p_2 \in \mathcal{P}, p_1 < p_2, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2} : \sum_{d \in \mathcal{D}} \left(\sigma^{+}_{p_1 p_2 b_1 b_2 d} + \sigma^{-}_{p_1 p_2 b_1 b_2 d}\right) \leq \psi_{p_2} \tag{3.24}$$

The main optimization goal of this approach is to maximize the packed volume, as seen in Equation 3.25. Alternatively V_p^{P} can be substituted to model any virtual costs assigned to a piece.

$$\max z = \sum_{p \in \mathcal{P}} \psi_p \cdot V_p^{\mathsf{P}} \tag{3.25}$$

The proposed technique provides some possibilities of approximating more complex forms by these "Tetris"-like objects, but on the downside this also directly increases the complexity of the formulation, regarding the number of variables and constraints. This MIP is the basic foundation for the model proposed in section 4.3.

3.2 Heuristics

As already discussed in the literature and stated here before the considered types of problems are hard to solve in a limited time-horizon when dealing with reasonably sized instances. Hence, many different heuristics were proposed as approximative solution methods. This was especially important when exact algorithms were still in their early stages of development, because they were not yet efficient enough. For the particular 3DBPP and related problems this is still the case for problem instances of practical size (see [CLS95], [WLGdS10], [JMY12b], [JMY12a]). Thus, [GR80] define one of the first heuristic algorithms for a three-dimensional packing problem. The proposed method relies on the concept of filling a container by layers and each of these layers by strips, so-called wall building. So the first piece determines the depth of the new layer it introduces. After allocating this piece the layer is filled with strips consisting of pieces. These strips are filled by exactly solving a one dimensional knapsack-problem. The contribution was extended in many publications over the following years (see [Pis02], [PAVTO08], [PAVOT10], [ABK11]).
Another early heuristic emphasizes on packing every new piece at a "bottom-left"-stable position, so that it cannot move downwards or to the left (see [BCR80]). Although this approach is proposed for two-dimensional bin packing, the general idea is the base for many subsequent publications, which box the pieces iteratively beginning

at a corner point of the container (see [BHI+07], [WGC+10], [ABK11], [ZZOL12]). Unfortunately, most two-dimensional approaches cannot be adapted directly to overcome three-dimensional problems or, at least, are not quite efficient for these (see [CPT09]). On the other side, as stated before, many of the basic concepts for two-dimensional packing are reused for the three-dimensional problems.
[Pis02] explicitly handles the knapsack container loading problem. The goal of the approach is to fill as many pieces into a container as possible. The authors rely on the wall-building technique introduced by [GR80] and discussed above. As an extension a tree-search heuristic is used to search for the "[...] set of layer depths and strip widths which results in the best overall filling" (see [Pis02], p. 385).
[FPZ03] propose a Guided Local Search (GLS) metaheuristic, which "[...] sits on top of other heuristic methods with the aim to improve their efficiency or robustness" (see [VTA10], p. 322). The employed local search algorithm alters a solution by two principles. On the one hand a piece can be moved from one box to another, on the other hand a piece can be moved along one of the axes. The authors state that by this definition the optimal solution can be reached when exploring the search space. Also any possible packing pattern can be reached from any other one. This also implies that the non-overlapping requirement has to be relaxed. Therefore, the authors sum the total overlapping volume of all pairs of pieces and aim to minimize it. Thus, a feasible solution has a value of 0.
Integer linear programming techniques combined with greedy heuristics get employed by [HNW14] to solve the 3D-SBSBPP. At first, pieces are selected to be packed into a container during a "selection phase" and then secondly get fixed at valid positions during the "positioning phase". This is repeated for every bin until all items are packed.
The work by [Ege09] emphasizes on packing irregular shapes into a single container. It is an extension on the previously discussed work by [FPZ03]. Their focus lies on the balancing and inertia moment. The authors adapt the GLS procedure to handle the packing of irregular forms.
[EP09] use the Simulated Annealing (SA) metaheuristic with a "sequence triple" representation of the packings. Such a sequence defines the semi-ordering of the pieces inside a container. Taking all three sequences defining such a triple into consideration, a complete packing can be generated. These sequences are used as solution representations in the SA approach.
[PAVTO08] and [PAVOT10] proposed a Greedy Randomized Adaptive Search Procedure (GRASP) which in the constructive phase uses a "maximal space" representation of the free space to allocate pieces to it. In the improvement phase the procedure removes a subset of pieces from the current best solution and fills them again with the algorithm of the constructive phase. This relies on a technique first introduced by [PR00]. The procedure is extended in [AVPT13] by integrating successful construction heuristics into the GRASP framework, as well as combining the approach with a Path Relinking procedure to obtain higher quality solutions by building elite solution sets. Here the authors alongside with the solution quality use some distance measures to obtain an elite set (ES) of solutions which are of high quality and also diverse.

The bottom left depth (BLD) method for packing containers in the air cargo application is employed by [YMS08]. This method uses insertion points similar to the extreme point concept, but without the projection of the corner points along the axes, thus every new allocated piece introduces three new points instead of six. The basic algorithm then allocates the pieces in a greedy manner depending on a predefined order of the pieces. This order is improved by an iterative local search method to achieve higher volume utilization. The approach also allows a small overhang of pieces outside the container's domain.
When concerning bounds, for the given 3DBPP a continuous lower bound on the objective value is given by Equation 3.26 with $V_p^{\mathtt{P}}$ as the volume of piece p and $V_c^{\mathtt{C}}$ the volume of container c. This means that the number of containers required to pack the defined pieces has to be greater than the given ratio. [MPV00] state that "The asymptotic worst-case performance ratio of [...]" this lower bound "[...] is $\frac{1}{8}$ even if rotation of the items (by any angle) is allowed" (see [MPV00], p. 257). The containers considered in this case all must have the same size. They also present further bounds for certain problem specifications as well as a branch-and-bound based approach for solving the problem. In the branching-tree of this method pieces are assigned to bins while the exact position inside the container is calculated in each node.

$$\left\lceil \frac{\sum_{p \in \mathcal{P}} V_p^{\mathtt{P}}}{V_c^{\mathtt{C}}} \right\rceil \tag{3.26}$$

3.2.1 Extreme points

The first main concept of the literature this work relies on is the concept of Extreme Points (EP). These EPs are introduced by [CPT08] as an extension of the "corner point" concept introduced by [MPV00]. In a simple constructive heuristic, which fills containers in a greedy manner, pieces are inserted with their front-left-bottom vertex at these points. The basic concept of such points is called "insertion points" during this work, since some extensions are made here. EPs in particular refer to insertion points, which are generated in a specific way. The generation of such EPs is done for every cuboid piece and is depicted in Figure 3.1. Every vertex $(1, 2, 3)$ of the piece is projected along two axes until the projection hits another piece or the container's wall. In more detail this means:

1. Point $(x_p + L_{px}^{\mathtt{P}}, y_p, z_p)$ is projected along the Y- and Z-axes, resulting in the new EPs e_{11} and e_{12}.

2. Point $(x_p, y_p + L_{py}^{\mathtt{P}}, z_p)$ is projected along the X- and Z-axes, resulting in the new EPs e_{21} and e_{22}.

3. Point $(x_p, y_p, z_p + L_{pz}^{\mathtt{P}})$ is projected along the X- and Y-axes, resulting in the new EPs e_{31} and e_{32}.

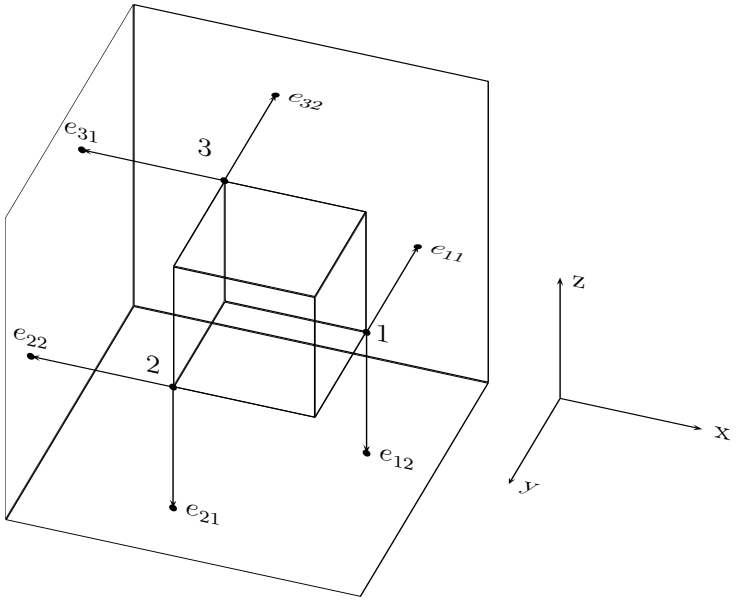


Figure 3.1: Extreme Point generation concept

At the beginning of the proposed construction heuristic every container is initialized with a first EP at $(0, 0, 0)$. So the first piece is allocated to the front-left-bottom corner of the container. Naturally, since all EPs are projected towards the origin along the axes, the packing "grows" from this corner to the boundaries of the container. [CPT08] introduce two simple heuristic methods exploiting the EPs. Both heuristics allocate pieces to the container at a previously generated EP while verifying that no overlaps emerge from the allocation and subsequently generate new EPs depending on the just added piece. If no valid EP is available any more a new container is added with the default EP of $(0, 0, 0)$. This is repeated until no further pieces are left. The first of the two methods, called Extreme Points First Fit Decreasing (EP-FFD), allocates the piece to the first valid EP that is available. The second method, called Extreme Points Best Fit Decreasing EP-BFD, uses different merit-functions defining the merit value of the valid EPs available. The piece is then loaded at the EP with the highest merit value. Also the order in which the pieces are inserted is very important to the efficiency of the approach. Therefore, different metrics for sorting the pieces are introduced, mostly depending on the piece's volume, height or base area.
Another concept used by this work is based on the one proposed by [ZZOL12]. The authors also exploit the idea of Extreme Points, but extend the construction heuristic with Space Defragmentation concepts. The resulting approach is called "Bin Shuffling using Extreme Point insertion with Space Defragmentation" (BS-EPSD). Bin shuffling describes an improvement routine relying on the re-ordering of the bins. After a first solution is generated, bins which are considered not "well-packed" are emptied, re-ordered and filled in the new order. Later in this work the particular improvement technique is substituted with the approach proposed by [BPT12]. However, the construction heuristic used by the authors was also investigated in this work. The

technique extends the simple EP-FFD by [CPT08]. Likewise EPs are used as insertion points for the pieces, but are deleted after each accommodation and calculated again. This is required since the method exploits a "push-out" concept. This means that for an EP the pieces around that point can be pushed away from it to make room for a new piece which is accommodated at that particular position. An invariant is shown which can be used to calculate the distances by which a piece can be moved along an axes safely before another piece is hit. These distances are used to move the pieces already inside the container away, then the new piece is inserted at the EP and at last the container is normalized. I.e. all pieces are pushed towards the origin of the container as far as possible until no piece can be pushed any further. This idea is also used to "inflate" already accommodated pieces and then substitute them with another, bigger piece. If non of the techniques are able to insert the piece into a container a new container is added with the default EP at $(0, 0, 0)$ like in [CPT08]. Both of the two heuristic methods proposed above are not able to handle any more complex shapes than cuboids or rotations by any angle.

3.2.2 Greedy Adaptive Search Procedure

The last heuristic concept this work relies on is the approach introduced by [PCT11] and [BPT12]. The idea of the work is to use a Greedy Adaptive Search Procedure (GASP) to control the underlying EP-based greedy constructive heuristic. Since the performance of such a heuristic strongly depends on the order with which the pieces are inserted, this particular order is updated by the GASP-metaheuristic and then used and evaluated with the greedy heuristic. Figure 3.2 depicts the basic procedure (see [BPT12]). At first an initial solution is generated with the constructive heuristic and a first score is generated depending on the insertion order of the pieces of the initial solution. The piece inserted first gets the highest score. After each iteration this score gets updated. During an update the pieces of the first half of containers are considered well-packed and their score is decreased by a dynamic value, while the score of the other pieces gets increased. If no improvement on the objective value can be done after a specific number of iterations the score is reinitialized. The procedure terminates if a stopping criterion is met. The dynamic values influence the scores corresponding to the number of reinitializations and improvements done to the best solution so far.

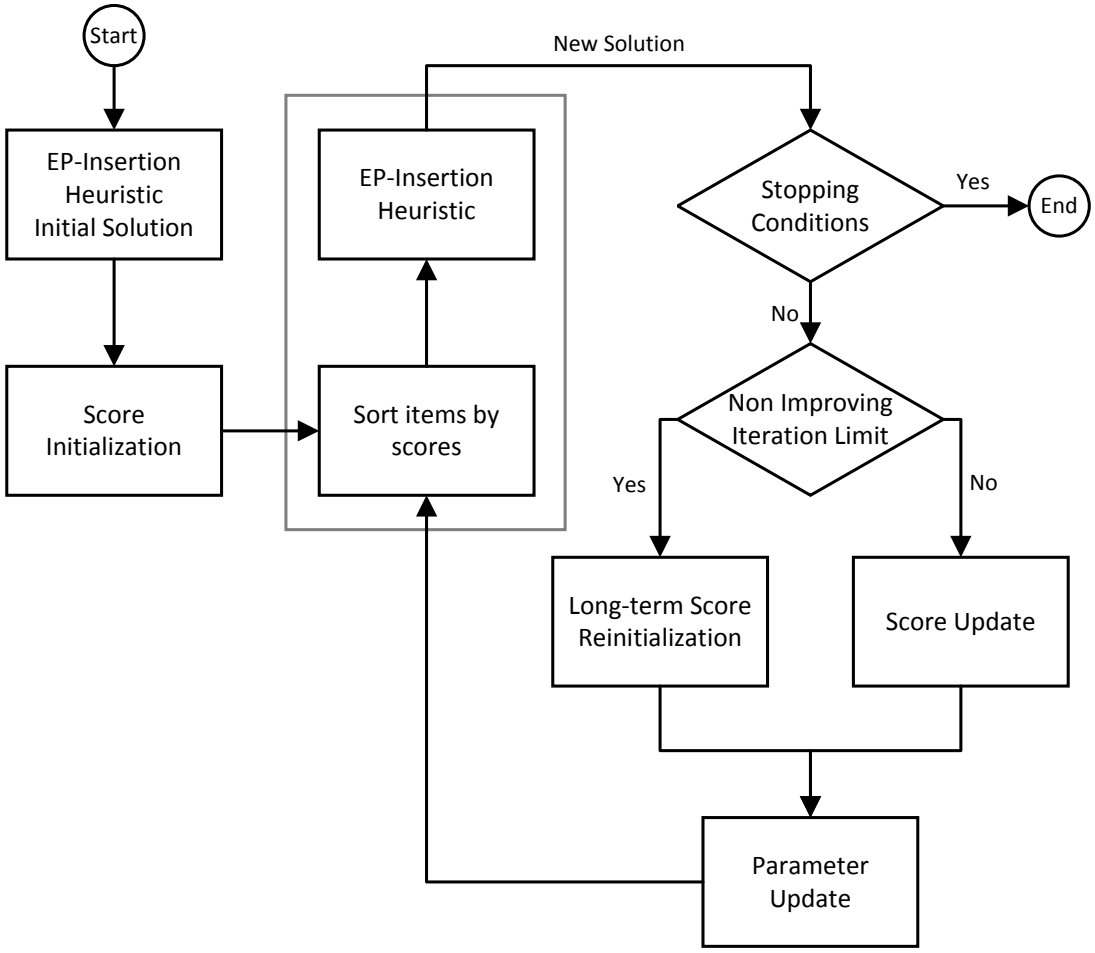


Figure 3.2: GASP execution procedure

3.3 Overview of problem characteristics

This section gives an overview about the requirements already considered in the literature and in all conscience the ones not yet handled. For this purpose the previously discussed requirements are investigated for the publications best matching the application of this thesis. The investigated issues are more complex shapes, multiple containers (MC), different sized containers (DC), the possibility of rotating pieces by at least 90° (R), the stability of the packing (LS), the load bearing (LB), handling of forbidden orientations (FO) and the compatibility (C). Additionally the basic method type is exposed to identify the necessary research for models as well as heuristics. The model formulations used in the literature often have some practical constraints already integrated, despite the fact that the field is still quite young (see [PSL12], p. 1). On the other hand, the heuristic methods rarely apply practical constraints, but are almost always designed to handle only basic three-dimensional packing problems efficiently.
As seen in Table 3.1 many of the requirements are still not considered in the literature.

MIP formulations of the problem already integrate subsets of these, while most of the heuristic methods only concentrate on solving the basic three-dimensional packing problem. The challenge of handling incompatibilities as well as forbidden orientations almost always is not respected by the current literature. Moreover, the influence of handling complex shapes is not considered by most of the publications. The majority of these only handles cuboid shapes. Furthermore, while most studies allow multiple containers none of them allows these to have different sizes or proportions, though this aspect seems to be critical, especially in the field of air cargo. At last, the load stability and load bearing requirements are handled by some MIP formulations, but are not exactly respected by one of the heuristic techniques.

Table 3.1: Requirements already discussed in the literature

Publication	Shape	MC	DC	R	LS	LB	FO	C	Technique
[PSL12]	Cuboid	✓	✗	✓	✓	✓	✓	✗	MIP
[JMY12a]	Cuboid	✗	✗	✓	✓	✓	✗	✗	MIP
[Fas13]	"Tetris"	✗	✗	✓	✗	✗	✗	✗	MIP
[Pis02]	Cuboid	✗	✗	✓	✗	✗	✗	✗	Heuristic
[CPT08]	Cuboid	✓	✗	✗	✗	✗	✗	✗	Heuristic
[Ege09]	Irregular	✗	✗	✗	✗	✗	✗	✗	Heuristic
[EP09]	Cuboid	✗	✗	✓	✗	✗	✗	✗	Heuristic (& MIP)
[PAVOT10]	Cuboid	✓	✗	✗	✗	✗	✗	✗	Heuristic
[ZZOL12]	Cuboid	✓	✗	✗	✗	✗	✗	✗	Heuristic
[BPT12]	Cuboid	✓	✗	✓	✗	✗	✗	✗	Heuristic (& MIP)
[HNW14]	Cuboid	✓	✗	✗	✗	✗	✗	✗	Heuristic

4 Model formulations

The following section introduces the model formulations of this work in detail. Namely the two discussed formulations by [PSL12] and [Fas13] get extended and implemented in order to investigate their capability of handling the underlying problem. Before going into detail some preliminaries are discussed in section 4.1. These information give a better understanding of the way the formulations work. The concepts are also used for the heuristics later discussed in chapter 5. Afterwards the cuboid-based formulation is introduced, employing the work by [PSL12]. At last the "Tetris"-based formulation is discussed, extending the work of [Fas13].

4.1 Preliminaries

In this section the basic concepts underlying the different formulations are introduced. These are also relevant to the heuristics described in chapter 5. Therefore, at first the vertex information of the pieces is discussed in more detail, alongside with some remarks about the different possible orientations of them. After that the concept of approximating complex shapes with a set of cuboids is shown. Finally, the notation of sets and parameters of this work and the objective functions are introduced. The notation is kept consistent throughout all methods. So, these get introduced only once in the following.

4.1.1 Approximation of complex shapes

As pieces to be packaged are not necessary of cuboid shape this fact has to be handled somehow. In most cases a so-called bounding-box is used to approximate the form of the piece. This bounding-box is the smallest cuboid in which the piece completely fits. Obviously a lot of space is lost with such a simple approximation approach, but on the bright side the most of the efficient state of the art techniques are immediately usable. However, in this work some investigation is done on the effect of using more complex approximation techniques. The approach discussed here uses a set of cuboids to approximate the form of a more complex shape. Thus for a perfect three-dimensional "L"-shape only two cuboids are necessary to represent the original piece's form equivalently. This idea is based on the work by [Fas13]. A more complex example of such an approximation is depicted in Figure 4.1. As seen in the example the presence of non-orthogonal sides may greatly increase the number of cuboids necessary to approximate the piece. Also the degree of precision in which the form is approximated influences the number of necessary boxes greatly. It is conceivable that

the approach will fall back to the bounding-box technique, if no precision is desired. An exact procedure of how such an approximation would work will not be discussed in here in detail. The approach leads to shapes, which are classified as "Tetris"-like pieces or "Tetris"-shapes in this work. This way any possible three-dimensional object can be approximated and transformed in such a "Tetris"-shape. Even theoretical pieces consisting of two components with their positions fixed in relation to each other are possible. Another interesting example are donut-like objects with holes in their shape.

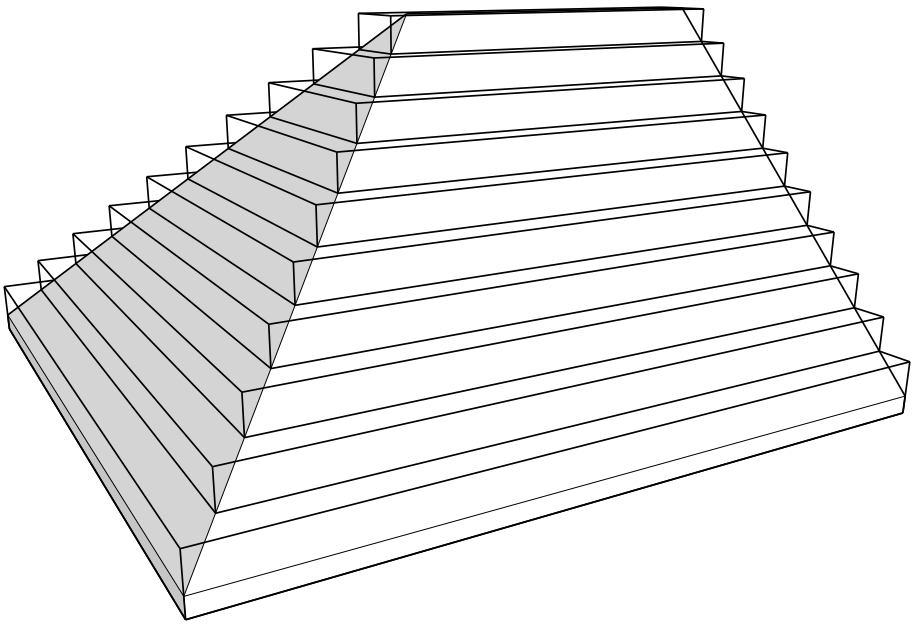


Figure 4.1: Approximation of a complex shape by using Tetris-shapes

One of the main goals by this technique is to exploit all of the available space, not only what is left when building the bounding-boxes of all pieces, but by doing this some problems arise. At first the complexity of a model using this technique greatly increases by means of the number of variables and constraints. This is due to the fact that for every additional component the same non-overlapping constraints apply. Furthermore, a practical requirement arises for the quality of the data of specific problem instances. In other words only the given data can be optimized and in some cases such detailed data about the shape of a piece may not be available. This depends on the technical methods by which the form of a piece is recognized. Also the load-bearing of the approximated areas has to be considered, but obviously the same problem also arises when dealing with bounding-boxes. However, in fields of application like air cargo, where goods are quite heterogeneous in their form and the containers are of limited size, this approach may improve the utilization of these containers.
To test the proposed model formulations and algorithms already "Tetris"-shaped synthetic pieces are used.

4.1.2 Vertices and orientations

This section defines the vertices used to describe the position and also the implicit orientation of a piece in three-dimensional space. Every cuboid in space can be explicitly represented by the positions of its vertices. Certainly a cuboid has got 8 vertices defining its corner points. In this work an additional virtual "vertex" is defined as the geometrical center of the cuboid. Every of these vertices is assigned an ID $v \in \mathcal{V}^*$ to reference it. The virtual center vertex is assigned the ID 0. The other vertices ID's are assigned like depicted by Figure 4.2. Note that vertex $v = 1$ corresponds to the FLB corner point, while vertex $v = 8$ corresponds to the RRT corner point. This is especially important in the formulation proposed in section 4.2. The overall enumeration of the vertex-IDs depends on the lexicographic order of the three axes x, y, z. These vertices and their indices are used in all model formulations and methods of this work.

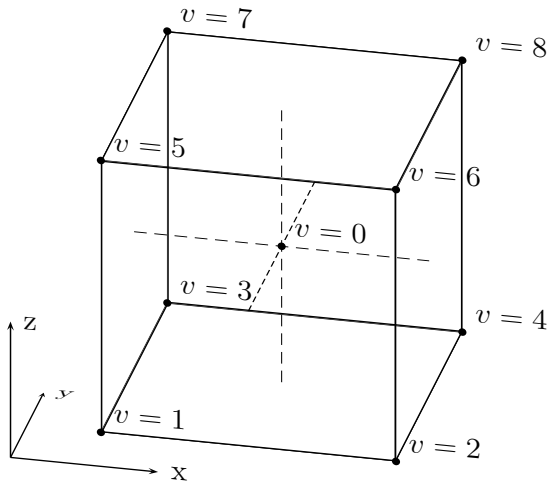


Figure 4.2: Definition of the vertex enumeration

The positions of the vertices in relation to a local reference frame, a point by which the relative positions of the vertices are transformed to absolute positions in the space, depend directly on the orientation $o \in \mathcal{O}$ of a piece. If no orientations are allowed, like in many publications, the relative positions can be assumed to be static. When allowing rotations the position of each vertex is defined by the current active orientation of its piece. The ID of each vertex stays the same in all orientations, e.g. vertex $v = 8$ is always the RRT point of a cuboid. Since in this work only 90° rotations are considered it is sufficient to model 6 orientations when handling cuboid pieces, because only 6 of them are distinct. These can simply be generated by rotating a cuboid lying on it's bottom, front and left face once by 90° degrees each time, resulting in 6 orientations. For a more complex piece like a piece with a "Tetris"-like shape 24 orientations are necessary to model all possible but distinct ones. The indices of these are defined as: $\mathcal{O} = \{0, \ldots, 23\}$. The generation of these works similar to the one for cuboids, but now the piece has to be rotated on every of

its 6 faces three times, resulting in the 24 orientations. Figure 4.3 depicts the different orientations for a "Tetris"-like piece (red / top rows). Also the same 24 orientations of a cuboid piece (blue / bottom rows) can be seen. Obviously only 6 of them are distinct and therefore sufficient.

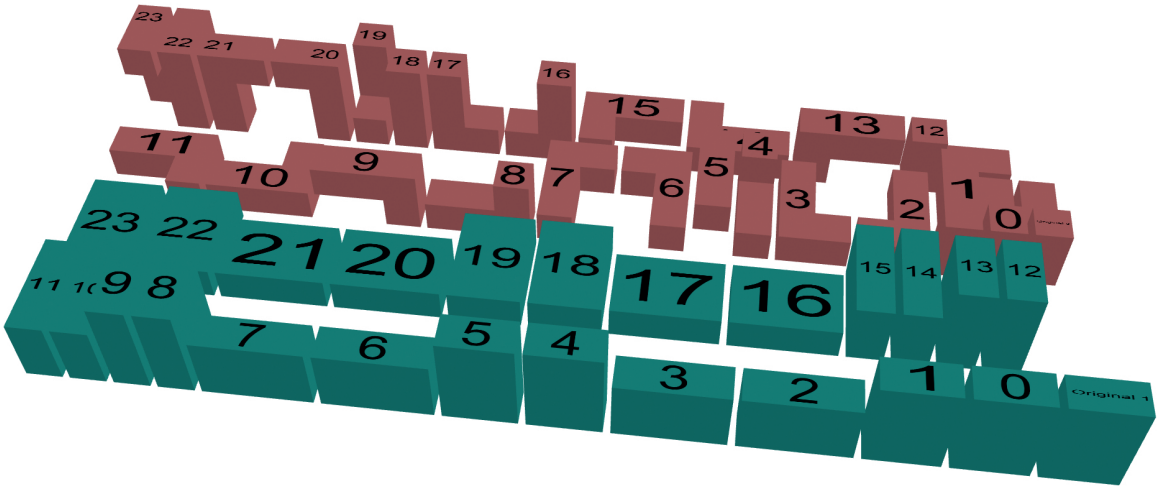


Figure 4.3: Comparison of orientations for a "Tetris"-piece and a cuboid (Screenshot of the prototype)

4.1.3 Sets and parameters

This section contains the information about all sets and parameters that are used in the model formulations and methods introduced in this work. As mentioned before all notations were kept as consistent as possible throughout the different model and algorithm descriptions. Thus, some of the parameters or sets below are not used by specific approaches.

$\mathcal{D} := \{x, y, z\}$	Set of dimension indices. This set always consists of the three markers x, y and z for the three respective dimensions.
$\mathcal{C}$	The set of containers, which is another basic input of the problem definition.
$\mathcal{P}$	The set of pieces, which is a fundamental part of the problem definition.
$\mathcal{P}_2 := \mathcal{P}_{=2}(\mathcal{P})$	All combinations of pieces with cardinality 2. Note that $\mathcal{P}_{=2}()$ at this point denotes the subset of the power-set with elements of cardinality equal two.
$\mathcal{B}_p \subseteq \mathcal{B}$	Set of components b the piece p consists of. These are used to describe the characteristics of the so-called "Tetris"-pieces.

$\mathcal{O} := \{0, \ldots, 23\}$	Set of all possible orientations-IDs o. The specific concept is explained in more detail in subsection 4.1.2.
$\mathcal{V}^* := \{0, \ldots, 8\}$	Set of all vertex-IDs v, including the ID for the geometrical center 0. (see subsection 4.1.2)
$\mathcal{V} := \mathcal{V}^* \setminus \{0\}$	Set of all vertex-IDs v, excluding the ID for the geometrical center 0. (see subsection 4.1.2)
$\mathcal{R} := \{1, 2, 3\}$	Set of all rotation-indices. These denote the indices of the rotation-matrix.
$L^{\mathtt{P}}_{pod}$	Side-length of piece p regarding dimension d in orientation o, i.e. Length × Width × Height of piece $p \in \mathcal{P}$ is equal to $L^{\mathtt{P}}_{pox} \times L^{\mathtt{P}}_{poy} \times L^{\mathtt{P}}_{poz}$ for the respective orientation. Note that this represents the bounding box of p. Furthermore, note that the bounding box of p equals $b \in \mathcal{B}_p$ if $\lvert\mathcal{B}_p\rvert = 1$. This can be accepted with no loss of generality, since only parallelepiped-forms are allowed.
$L^{\mathtt{B}}_{bod}$	Side-length of component $b \in \mathcal{B}_p$ in orientation o regarding dimension d. These lengths are used when handling "Tetris"-pieces instead of the bounding box' lengths $L^{\mathtt{P}}_{pod}$.
$L^{\mathtt{C}}_{cd}$	Side-length of container $c \in \mathcal{C}$ in dimension d.
$V^{\mathtt{P}}_p$	Volume of piece p. Note: $V^{\mathtt{P}}_p = \sum_{b \in B_p} V^{\mathtt{B}}_b$ in case of handling "Tetris"-like pieces, otherwise it is the volume of the bounding box $V^{\mathtt{P}}_p = \sum_{d \in \mathcal{D}} L^{\mathtt{P}}_{p0d}$. It is assumed with no loss of generality that the volume of a piece or component is the same for every possible orientation $o \in \mathcal{O}$.
$V^{\mathtt{B}}_b$	Volume of component b. Note: $V^{\mathtt{B}}_b = \sum_{d \in \mathcal{D}} L^{\mathtt{B}}_{b0d}$.
$V^{\mathtt{C}}_c$	Volume of container c. Note: $V^{\mathtt{C}}_c = \sum_{d \in \mathcal{D}} L^{\mathtt{C}}_{cd}$.
$P^{\mathtt{P}}_{povd}$	Position of piece p's bounding box's vertex v according to dimension d relative to its local reference frame origin.
$P^{\mathtt{B}}_{bovd}$	Position of component b's vertex v according to dimension d relative to its local reference frame origin.
$P^{\mathtt{C}}_{cvd}$	Position of container c's vertex v according to dimension d relative to its local reference frame origin.
M	big-M - A sufficiently large value capable of relaxing certain constraints when multiplied with a binary variable indicating the activity of the particular constraints. Due to technical issues this value also should be as small as possible, because it may influence the solution time due to numerical aspects. For the following model formulations M can be set to $\max_{c \in \mathcal{C}} (\max_{d \in \mathcal{D}} (L^{\mathtt{C}}_{cd}))$ without loss of generality.

E_p	Fractional part of the contact-face's side-length which might protrude another piece when piece p is lying on it
$G_p \in \mathcal{G}$	Goods-class of piece p. This defines special handling materials, such as flammables, which cannot be packed with specific other materials or only in special containers.
$\mathcal{G}_p^f \subset \mathcal{G}$	The set of materials incompatible to p.
$\mathcal{G}_2 := \mathcal{P}_{=2}(\mathcal{G})$	All combinations of materials with cardinality 2. Note that $\mathcal{P}_{=2}()$ at this point denotes the subset of the power-set with elements of cardinality equal two.
$\mathcal{G}^f \subset \mathcal{G}_2$	The set of material-combinations which can not be assigned to the same container together.
$\mathcal{P}^s \subseteq \mathcal{P}$	The set of pieces which are not stackable. This means that no other piece may be packed on one of those pieces.
$\mathcal{O}_p^f \subset \mathcal{O}$	The set of forbidden orientations for piece p.

4.1.4 Objective

When packing pieces into a container there are many different objectives possible. According to the classic knapsack problem one could maximize the utilization of the available space. Here it would also be possible to assign a value to each piece representing for example the importance of packing the piece. Alternatively considering the classic bin packing problem a goal may be to minimize the number of used containers with all pieces assigned. Beyond that any other possible goal is imaginable, as well as hybridizations of them. Both of the proposed models are capable of handling the classic goals of knapsack and bin packing. However, only the knapsack objective of maximizing the volume utilization is used in the evaluation due to the comparability with the proposed methods of this work.
The first objective function of minimizing the container count is defined by Equation 4.1. In this term the variable indicating an active container η_c is summed over all containers and minimized. This goal requires a specific allocation constraint defined in the two models in Equation 4.4 and Equation 4.35.

$$\min z = \sum_{c \in \mathcal{C}} \eta_c \tag{4.1}$$

The second objective maximizes the occupied space along all given containers (see Equation 4.2). With this goal another specific allocation constraint has to be used, which makes the allocation of a piece to any container to be optional (see Equation 4.5 and Equation 4.36). The term of this objective function sums the volume $V_p^{\mathtt{P}}$ of the allocated pieces ψ_{pc} and maximizes it. This goal is quite versatile since the value of

the volume could by substituted by any constant depicting further information like the priority of the piece.

$$\max z = \sum_{p \in \mathcal{P}} \left(V_p^{\mathsf{P}} \cdot \sum_{c \in \mathcal{C}} \psi_{pc} \right) \tag{4.2}$$

4.2 Cuboid-based approach

This chapter introduces the cuboid model formulation based on the work of [PSL12]. At first the variables are introduced and then the different constraints defining the solution space. All sets and parameters used here were defined in subsection 4.1.3.
The basic concept of this formulation relies on only explicitly modeling the FLB and RRT corner points of each cuboid piece. Like depicted by Figure 4.4 only the positioning variables with index $v = 1$ and $v = 8$ are used. This is a sufficient representation for pieces in the cuboid formulation to model all further requirements. The size of the piece are given by the parameter L_{p1d}^{P} for each dimension $d \in \mathcal{D}$ in the default orientation $o = 0$. As opposed to the "Tetris"-formulation introduced in section 4.3 the lengths for the other possible orientations of a piece are dynamically set by the rotation-constraints 4.7 up to 4.11 depending on the explicitly modeled rotation-matrix.

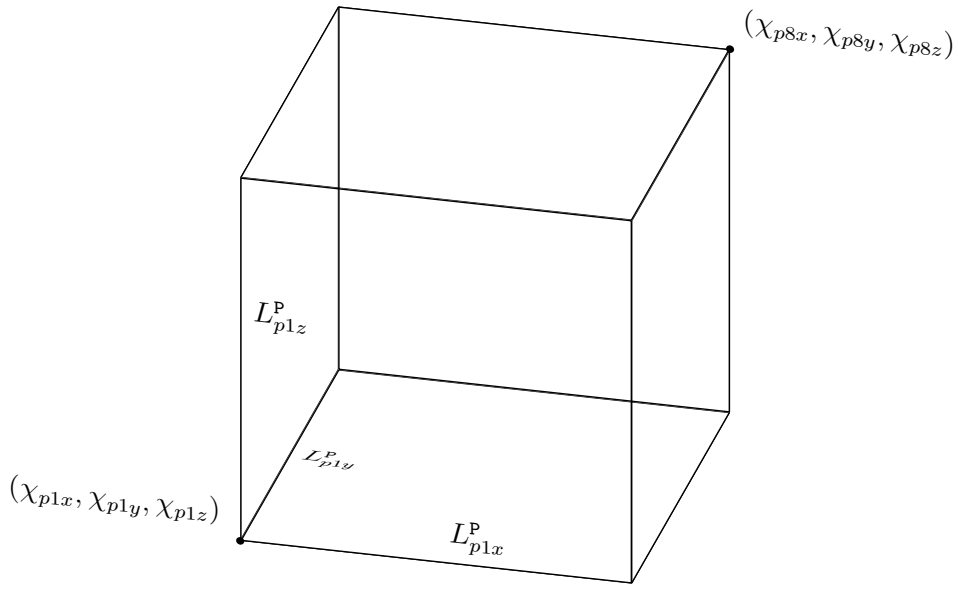


Figure 4.4: Representation of a piece in the cuboid-model. To emphasize the dimension contained in d the vertex-variables are disassembled.

4.2.1 Variables

The following decision variables are used for the model formulation. All of these are binary except for the piece's position in space which is continuous. Hence, the problem can be classified as a mixed binary program MBP.

$\chi_{pvd} \in \mathbb{R}_0^+$ Position of piece p's vertex v regarding domain d. Thus, a position is always defined by three values $(x, y, z) \in \mathbb{R}_0^+ \times \mathbb{R}_0^+ \times \mathbb{R}_0^+$. For the cuboid formulation the vertex v is limited to $\{1, 8\}$, because only the FLB and RRT corner points are modeled. This is a continuous non-negative variable.

$$\eta_c = \begin{cases} 1 & \text{if container } c \text{ is used,} \\ 0 & \text{else.} \end{cases}$$

$$\psi_{pc} = \begin{cases} 1 & \text{if piece } p \text{ is contained in container } c, \\ 0 & \text{else.} \end{cases}$$

$$\tau^b_{dp_1p_2} = \begin{cases} 1 & \text{if piece } p_1 \text{ is completely below } p_2 \text{ regarding} \\ & \text{dimension } d \Rightarrow \chi_{p_1 8d} \leq \chi_{p_2 1d}, \\ 0 & \text{else.} \end{cases}$$

$$\tau^a_{dp_1p_2} = \begin{cases} 1 & \text{if piece } p_1 \text{ is completely above } p_2 \text{ regarding} \\ & \text{dimension } d \Rightarrow \chi_{p_1 8d} \geq \chi_{p_2 1d}, \\ 0 & \text{else.} \end{cases}$$

$$\phi_{cp_1p_2} = \begin{cases} 1 & \text{if piece } p_1 \text{ and } p_2 \text{ are both contained in container } c, \\ 0 & \text{else.} \end{cases}$$

$$\omega_{p_1p_2} = \begin{cases} 1 & \text{if piece } p_1 \text{ is located on piece } p_2, \\ 0 & \text{else.} \end{cases}$$

$$\omega^g_p = \begin{cases} 1 & \text{if piece } p \text{ is located on the ground,} \\ 0 & \text{else.} \end{cases}$$

$$\rho_{pr_1r_2} = \begin{cases} 1 & \text{if cell in rotation matrix at the given index is 1,} \\ 0 & \text{else.} \end{cases}$$

4.2.2 Basic constraints

In the following the different constraints of the cuboid approach are described. At first the basic three-dimensional packing requirements like non-overlapping and rotatability are considered, followed by the more special requirements regarding stability, compatibility, forbidden orientations and load bearing.
The basic allocation of pieces to containers is handled by the constraints 4.3 up to 4.5. At first it has to be ensured that no piece can be assigned to an inactive container. Thus, constraint 4.3 states that every piece p assigned to a container c denoted by ψ_{pc} implies that the corresponding container has to be active (η_c). Furthermore, for the

bin packing approach, every piece has to be assigned to exactly one container depicted by constraint 4.4. Analogously defined, every piece can at most be assigned once in the case of the knapsack approach, stated in constraint 4.5.

$$\psi_{pc} \leq \eta_c \qquad \forall p \in \mathcal{P}, c \in \mathcal{C} \tag{4.3}$$

$$\sum_{c \in \mathcal{C}} \psi_{pc} = 1 \qquad \forall p \in \mathcal{P} \text{ (for bin packing)} \tag{4.4}$$

$$\sum_{c \in \mathcal{C}} \psi_{pc} \leq 1 \qquad \forall p \in \mathcal{P} \text{ (for knapsack)} \tag{4.5}$$

Next, the requirement of the pieces to stay within the container's domain has to be taken into account. Since the variable χ_{p1d} depicting the position of the FLB corner point of piece p is non-negative, it can not overlap with the container's front, left or bottom side. Constraint 4.6 then ensures that the piece does not extend the container's rear, right or top side. This is done by limiting the RRT corner position χ_{p8d} of the piece, regarding all dimensions $d \in \mathcal{D}$, to the respective container length $L^{\mathtt{C}}_{cd}$. As an extension to the basic formulation by [PSL12] the side-lengths of the containers can be different, as opposed to the assumption of uniform sized containers of the original formulation.

$$\chi_{p8d} \leq L^{\mathtt{C}}_{cd} \qquad \forall p \in \mathcal{P}, d \in \mathcal{D} \tag{4.6}$$

As discussed before rotations about 90° are possible in this model formulation. This is achieved by explicitly modeling a rotation matrix as already described in subsection 3.1.1. For that the constraints 4.7, 4.8 and 4.9 link the distances between the FLB (χ_{p1d}) and RRT (χ_{p8d}) corner point to the original lengths of the piece according to the current state of the rotation matrix denoted by the binary variable $\rho_{pr_1r_2}$. In order to sustain a correct rotation matrix for this approach constraints 4.10 and 4.11 set the row- and column-sums to be exactly 1, implying only one active rotation variable $\rho_{pr_1r_2}$ per row and column. As stated before this is sufficient to model the orientations arising from the possible 90° rotations of a cuboid piece. Thus, every piece can obtain every of its 6 distinct orientations, as long as they are not forbidden by further requirements. When handling more complex forms or continuous rotations of the pieces this approach would have to be extended.

$$\chi_{p8x} - \chi_{p1x} = \rho_{p11} \cdot L^{\mathtt{P}}_{p0x} + \rho_{p12} \cdot L^{\mathtt{P}}_{p0y} + \rho_{p13} \cdot L^{\mathtt{P}}_{p0z} \qquad \forall p \in \mathcal{P} \tag{4.7}$$

$$\chi_{p8y} - \chi_{p1y} = \rho_{p21} \cdot L^{\mathtt{P}}_{p0x} + \rho_{p22} \cdot L^{\mathtt{P}}_{p0y} + \rho_{p23} \cdot L^{\mathtt{P}}_{p0z} \qquad \forall p \in \mathcal{P} \tag{4.8}$$

$$\chi_{p8z} - \chi_{p1z} = \rho_{p31} \cdot L^{\mathtt{P}}_{p0x} + \rho_{p32} \cdot L^{\mathtt{P}}_{p0y} + \rho_{p33} \cdot L^{\mathtt{P}}_{p0z} \qquad \forall p \in \mathcal{P} \tag{4.9}$$

$$\sum_{r_1 \in \mathcal{R}} \rho_{pr_1r_2} = 1 \qquad \forall p \in \mathcal{P}, r_2 \in \mathcal{R} \tag{4.10}$$

$$\sum_{r_2 \in \mathcal{R}} \rho_{pr_1r_2} = 1 \qquad \forall p \in \mathcal{P}, r_1 \in \mathcal{R} \tag{4.11}$$

To avoid overlapping the following constraints are introduced, similar to the ones used by the basic model. Again the relative position markers $\tau^b_{dp_2p_1}$ and $\tau^a_{dp_2p_1}$, both binary variables, are used to indicate whether a piece p_1 is completely located before respectively after another piece p_2 regarding a specific dimension d. Thus, if a relative position marker indicates that a first piece is located behind a second piece, then the difference between the position of the FLB corner point of the first piece and the position of the RRT corner point of the second piece needs to be positive. The same applies analogously for the markers indicating whether a piece is in front of another one. This results in 6 relative position markers. At least one of them has to be true respectively equal to 1 so that the two considered pieces do not overlap. This is ensured by constraint 4.14. In the case that two pieces are not assigned to the same container ($\psi_{p_1c} + \psi_{p_2c} \neq 2$) for an arbitrary but fixed container c, this requirement has to be loosened. This is done by the third part of both constraints 4.12 and 4.13 using a big-M approach.

$$\begin{aligned} &\chi_{p_1 1d} - \chi_{p_2 8d} \\ &+ \left(1 - \tau^b_{dp_2p_1}\right) \cdot M \\ &+ (2 - (\psi_{p_1c} + \psi_{p_2c})) \cdot M \geq 0 \qquad \forall p_1, p_2 \in \mathcal{P}, c \in \mathcal{C}, d \in \mathcal{D} \end{aligned} \tag{4.12}$$

$$\begin{aligned} &\chi_{p_2 1d} - \chi_{p_1 8d} \\ &+ \left(1 - \tau^a_{dp_2p_1}\right) \cdot M \\ &+ (2 - (\psi_{p_1c} + \psi_{p_2c})) \cdot M \geq 0 \qquad \forall p_1, p_2 \in \mathcal{P}, c \in \mathcal{C}, d \in \mathcal{D} \end{aligned} \tag{4.13}$$

$$\sum_{d \in \mathcal{D}} \left(\tau^b_{dp_2p_1} + \tau^a_{dp_2p_1}\right) \geq 1 \qquad \forall p_1, p_2 \in \mathcal{P} \tag{4.14}$$

To provide a simple bound on the objective a redundant volume limitation constraint is introduced. It is obvious that for every container the summed volume of the pieces allocated to it cannot exceed the volume of the container's domain. Therefore constraint 4.15 explicitly formulates this relationship.

$$\sum_{p \in \mathcal{P}} \left(\psi_{pc} \cdot V^{\mathtt{P}}_p\right) \leq V^{\mathtt{C}}_c \qquad \forall c \in \mathcal{C} \tag{4.15}$$

4.2.3 Specific constraints

Having these constraints defined, the basic model formulation is ready to handle three-dimensional packing problems. The next constraints will extend this formulation to handle more complex requirements.
At first an idea of basically handling the stability requirement is given, which describes the requisition of the final packing not to collapse when it is realized like proposed by a solution to the model. This is not trivial and many different influences need to be considered. At this point the goods to be packed are assumed to have uniformly distributed weight. With the assumption of cuboid shaped goods it is obvious that every piece's bottom face has to be supported either by another piece's top face or

the ground. In addition this formulation shall require the top piece's base area not to protrude the top face of it's supporting piece by too much. To overcome this requirement binary variables are used indicating whether a piece p_1 is supported by another piece p_2 ($\omega_{p_1p_2} = 1$) or the ground ($\omega^g_{p_1} = 1$). Constraint 4.17 ensures that at least one of the two supporting types is true. In this way either the ground supports the particular piece or another one $p_2 \in \{p \in \mathcal{P} \mid p \neq p_1\}$. The case of a piece lying directly on the ground can be modeled simply by enforcing the FLB corner position regarding the third dimension $d = z$ to 0. This is formulated by constraint 4.16. If no support by the ground is necessary this is relaxed by M.
For the support of a piece by positioning it on another piece more complex constraints are required. Therefore, at first the top piece p_1's FLB corner position has to match the supporting piece p_2's RRT position regarding the $d = z$ dimension. This is expressed by constraints 4.18 and 4.19 in the case of an active support indicated by $\omega_{p_1p_2} = 1$. Thus, the height at which the supported piece p_1 is positioned has to match the one at which p_2's top face is positioned. Additionally it has to be ensured that the top piece is located directly above the supporting one. Therefore, constraints 4.20 up to 4.23 enforce the FLB position of the top piece to be greater than the one of the supporting piece as well as the RRT position to be smaller than the one of the supporting piece regarding the two horizontal dimensions $d = x$ and $d = y$. This ensures that the bottom face does not protrude the supporting piece's top face. Since the approach is very strict in reality an addition is made to relax the requirement. If the parameter E_{p_1} is greater than 0, the restriction is loosened a bit so that the top piece may protrude by a fraction (E_{p_1}) of its bottom face length. At last the possibility of two pieces in such a supporting relation, but located in two different containers has to be suppressed. So, constraint 4.24 links the allocation information to the binary variable $\phi_{cp_1p_2}$, thus indicating that both pieces p_1 and p_2 are allocated to the same container c. Constraint 4.25 then ensures that if one piece is supported by another, both are residing in the same container.

$$\chi_{p1z} \leq (1 - \omega^g_p) \cdot M \qquad \forall p \in \mathcal{P} \tag{4.16}$$

$$\omega^g_{p_1} + \sum_{\substack{p_2 \in \mathcal{P} \\ p_1 \neq p_2}} \omega_{p_1p_2} = 1 \qquad \forall p_1 \in \mathcal{P} \tag{4.17}$$

$$\chi_{p_1 1z} \leq \chi_{p_2 8z} + (1 - \omega_{p_1p_2}) \cdot M \qquad \forall p_1, p_2 \in \mathcal{P} \text{ with } p_1 \neq p_2 \tag{4.18}$$

$$\chi_{p_1 1z} + (1 - \omega_{p_1p_2}) \cdot M \geq \chi_{p_2 8z} \qquad \forall p_1, p_2 \in \mathcal{P} \text{ with } p_1 \neq p_2 \tag{4.19}$$

$$\begin{aligned} \chi_{p_1 8x} \leq \chi_{p_2 8x} + E_{p_1} \cdot (\chi_{p_1 8x} - \chi_{p_1 1x}) \\ + M \cdot (1 - \omega_{p_1p_2}) \end{aligned} \qquad \forall p_1, p_2 \in \mathcal{P} \text{ with } p_1 \neq p_2 \tag{4.20}$$

$$\begin{aligned} \chi_{p_1 1x} \geq \chi_{p_2 1x} + E_{p_1} \cdot (\chi_{p_1 8x} - \chi_{p_1 1x}) \\ + M \cdot (1 - \omega_{p_1p_2}) \end{aligned} \qquad \forall p_1, p_2 \in \mathcal{P} \text{ with } p_1 \neq p_2 \tag{4.21}$$

$$\begin{aligned} \chi_{p_1 8y} \leq \chi_{p_2 8y} + E_{p_1} \cdot (\chi_{p_1 8y} - \chi_{p_1 1y}) \\ + M \cdot (1 - \omega_{p_1p_2}) \end{aligned} \qquad \forall p_1, p_2 \in \mathcal{P} \text{ with } p_1 \neq p_2 \tag{4.22}$$

$$\chi_{p_1 1y} \geq \chi_{p_2 1y} + E_{p_1} \cdot (\chi_{p_1 8y} - \chi_{p_1 1y}) + M \cdot (1 - \omega_{p_1 p_2}) \qquad \forall p_1, p_2 \in \mathcal{P} \text{ with } p_1 \neq p_2 \tag{4.23}$$

$$\psi_{p_1 c} + \psi_{p_2 c} \geq 2 \cdot \phi_{c p_1 p_2} \qquad \forall c \in \mathcal{C}, p_1, p_2 \in \mathcal{P} \text{ with } p_1 \neq p_2 \tag{4.24}$$

$$\sum_{c \in \mathcal{C}} \phi_{c p_1 p_2} \geq \omega_{p_1 p_2} \qquad p_1, p_2 \in \mathcal{P} \text{ with } p_1 \neq p_2 \tag{4.25}$$

Another requirement is the compatibility of packed pieces. To respect all existing regulations some pieces cannot be packed into the same container. Constraint 4.26 ensures this by limiting the allocation ψ_{pc} of two particular pieces with incompatible material G_{p_1} and G_{p_2} to one. Thus, only one of the two pieces may be allocated to container c.

$$\psi_{p_1 c} + \psi_{p_2 c} \leq 1 \qquad \forall \{p_1, p_2\} \in \mathcal{P}_2 \text{ with } \{G_{p_1}, G_{p_2}\} \in \mathcal{G}^f \tag{4.26}$$

To ensure basic load bearing constraint 4.27 prevents fragile pieces of becoming supporting pieces. To achieve this it is sufficient to set the variable indicating that another piece p_2 is supported by the fragile piece p_1 to zero for all other pieces. This is a very strict consideration of the fragility, but since this implies no other piece can get stacked on a fragile piece this leads to viable packing plans. Another approach could be to sum the weight which is packed on a fragile piece and limit this to a predefined bound. On the downside this would require a more complex formulation.

$$\omega_{p_2 p_1} = 0 \qquad \forall p_1 \in \mathcal{P}^s, p_2 \in \mathcal{P} \text{ with } p_1 \neq p_2 \tag{4.27}$$

Since some pieces may be marked with a "this side up"-sign or similar packing advises, it is necessary to explicitly forbid the corresponding orientations. This is done by forbidding the specific states of the orientation matrix. Constraints 4.28 until 4.33 limit the sum of the respective rotation matrix elements $\rho_{r_1 r_2}$ corresponding to the orientations forbidden by their ID $o \in \mathcal{O}_p^f$. These three respective elements of the implicit rotation matrix together mark one distinctive orientation of a piece p, if all of them are active. I.e., the sum is equal to 3. The restraint of one particular orientation is then simply done by limiting the sum to 2. This notation derives from the harmonization of the two models of this paper, thus using the same orientation-IDs. In this cuboid-formulation only a fixed subset needs to be considered at all times, e.g. $\mathcal{O}' = \{0, 2, 8, 10, 16, 18\} \subset \mathcal{O}$.

$$\rho_{p11} + \rho_{p22} + \rho_{p33} \leq 2 \qquad \forall p \in \mathcal{P} \text{ with } \mathcal{O}_p^f \cap \{0, 1, 4, 5\} \neq \emptyset \tag{4.28}$$

$$\rho_{p11} + \rho_{p23} + \rho_{p32} \leq 2 \qquad \forall p \in \mathcal{P} \text{ with } \mathcal{O}_p^f \cap \{2, 3, 6, 7\} \neq \emptyset \tag{4.29}$$

$$\rho_{p12} + \rho_{p21} + \rho_{p33} \leq 2 \qquad \forall p \in \mathcal{P} \text{ with } \mathcal{O}_p^f \cap \{8, 9, 12, 13\} \neq \emptyset \tag{4.30}$$

$$\rho_{p13} + \rho_{p21} + \rho_{p32} \leq 2 \qquad \forall p \in \mathcal{P} \text{ with } \mathcal{O}_p^f \cap \{10, 11, 14, 15\} \neq \emptyset \tag{4.31}$$

$$\rho_{p12} + \rho_{p23} + \rho_{p31} \leq 2 \qquad \forall p \in \mathcal{P} \text{ with } \mathcal{O}_p^f \cap \{16, 17, 20, 21\} \neq \emptyset \tag{4.32}$$

$$\rho_{p13} + \rho_{p22} + \rho_{p31} \leq 2 \qquad \forall p \in \mathcal{P} \text{ with } \mathcal{O}_p^f \cap \{18, 19, 22, 23\} \neq \emptyset \tag{4.33}$$

4.3 "Tetris"-based approach

This section introduces the "Tetris"-based formulation. The basic model formulation thereby extends the work proposed by [Fas13]. Additional to the handling of "Tetris"-like pieces multiple containers of different size are considered, as opposed to the basic formulation by [Fas13] which only handles a single container. Further on, the model considers some basic practical requirements like the stability, compatibility, load bearing and forbidden orientations of the pieces. Like in the case of the cuboid-formulation the used sets and parameters are defined in subsection 4.1.3.
As opposed to the cuboid-formulation all vertices of a piece are explicitly modeled with the position variable χ_{bvd}. This variable is defined for every "real" vertex $v \in \{1..8\}$ plus the virtual vertex $v = 0$ (see subsection 4.1.2) of all components $b \in \mathcal{B}_p$ of a piece p. Every piece then has a local reference frame χ^o_{pd} to which the positions of all components' vertices are linked. The constraints defining this relation are introduced in this section.

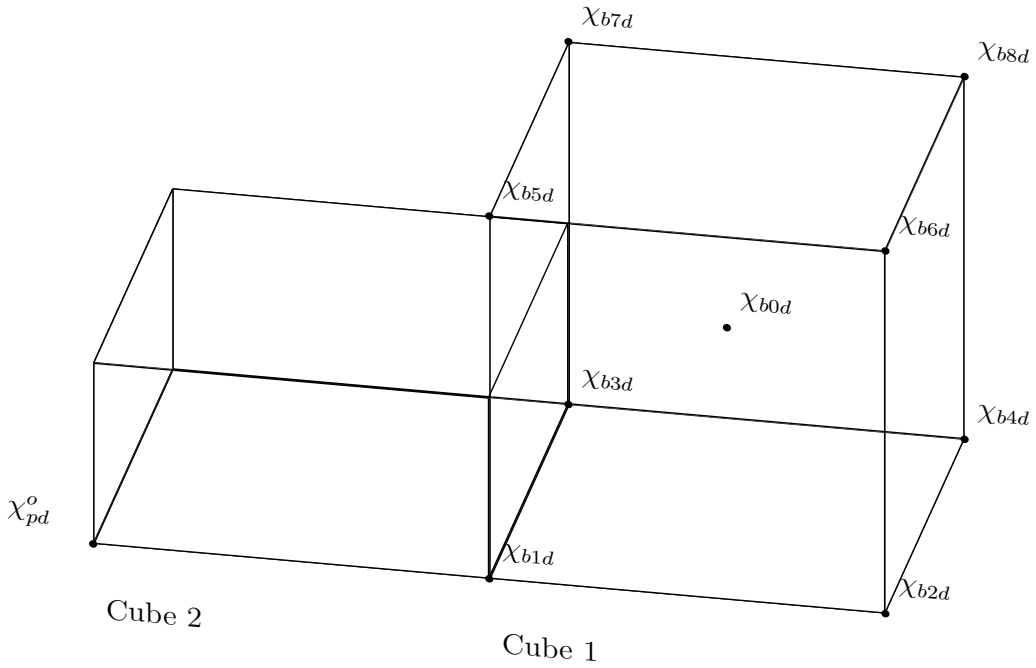


Figure 4.5: Representation of a piece in the Tetris-model. For reasons of clarity the variables are only depicted for the first cuboid of the piece.

4.3.1 Variables

The following decision variables are used for the model formulation. Most of these are binary like in the cuboid-formulation with the exception of the two continuous position variables χ^o_{pd} and χ_{bvd} and the relative variable for the protrusion prevention λ_{cgbv}.

$\chi_{bvd} \in \mathbb{R}_0^+$ Position of box b's vertex v regarding dimension d. Thus, a position is always defined by three values $(x, y, z) \in \mathbb{R}_0^+ \times \mathbb{R}_0^+ \times \mathbb{R}_0^+$. In the "Tetris"-approach all 8 vertices plus the midpoint $v \in \mathcal{V}^*$ are formulated explicitly.

$\chi^o_{pd} \in \mathbb{R}_0^+$ Position of piece p's local reference frame regarding dimension d. Every vertex' position χ_{bvd} is set relative to this "anchor".

$\lambda_{cv'bv} \in \mathbb{R}_0^+$ Is a non-negative variable used to keep all vertices inside the container's domain. This is defined for every vertex v''s bound of container c and every box b's vertex v.

$$\eta_c = \begin{cases} 1 & \text{if container } c \text{ is used,} \\ 0 & \text{else.} \end{cases}$$

$$\psi_{pc} = \begin{cases} 1 & \text{if piece } p \text{ is contained in container } c, \\ 0 & \text{else.} \end{cases}$$

$$\vartheta_{po} = \begin{cases} 1 & \text{if piece } p \text{ is oriented in orientation } o, \\ 0 & \text{else.} \end{cases}$$

$$\phi_{cp_1p_2} = \begin{cases} 1 & \text{if piece } p_1 \text{ and } p_2 \text{ are both allocated to container } c, \\ 0 & \text{else.} \end{cases}$$

$$\omega_{b_1b_2} = \begin{cases} 1 & \text{if component } b_1 \text{ of piece } p_1 \text{ is supported by} \\ & \text{component } b_2 \text{ of piece } p_2, \\ 0 & \text{else.} \end{cases}$$

$$\omega^g_p = \begin{cases} 1 & \text{if piece } p \text{ is supported by the ground,} \\ 0 & \text{else.} \end{cases}$$

$$\sigma^+_{db_1b_2} = \begin{cases} 1 & \text{if box } b_1 \text{ is positioned completely above } b_2 \\ & \text{regarding the dimension } d, \\ 0 & \text{else.} \end{cases}$$

$$\sigma^-_{db_1b_2} = \begin{cases} 1 & \text{if box } b_1 \text{ is positioned completely below } b_2 \\ & \text{regarding the dimension } d, \\ 0 & \text{else.} \end{cases}$$

4.3.2 Basic constraints

The following introduces the different constraints of the "Tetris"-based formulation. The same structure is used as for the cuboid approach: at first the basic constraints like non-overlapping and piece allocation are proposed, then more special constraints like stability, compatibility, load bearing and forbidden orientations of the pieces.
Similar to the cuboid-formulation at first the allocation of the pieces to a set of different containers needs to be handled. This is different to the basic formulation by [Fas13] since multiple containers of different size are considered here. At first the

variable allocating a piece p to a container c (ψ_{pc}) is bound by the activation variable η_c of the respective container. So, constraint 4.34 implies that a piece can only be accommodated to an active container. Furthermore, the allocation of pieces has to be limited when dealing with the knapsack objective function, hence constraint 4.36 limits the assignment of one specific piece for all containers to one. While handling the bin packing goal all pieces have to be allocated to the containers, expressed by constraint 4.35 fixing the allocation of a piece to one. These constraints extend the basic model since they are necessary to handle multiple containers.

$$\psi_{pc} \leq \eta_c \qquad \forall p \in \mathcal{P}, c \in \mathcal{C} \tag{4.34}$$

$$\sum_{c \in \mathcal{C}} \psi_{pc} = 1 \qquad \forall p \in \mathcal{P} \text{ (for bin packing)} \tag{4.35}$$

$$\sum_{c \in \mathcal{C}} \psi_{pc} \leq 1 \qquad \forall p \in \mathcal{P} \text{ (for knapsack)} \tag{4.36}$$

The handling of the rotatability of the pieces as well as the linkage between the local reference frame of a piece and its vertices is done by the following group of constraints. The orientation of piece p is depicted by the binary variable ϑ_{po}. Constraint 4.37 then states that, if a piece is allocated to any container indicated by the right-hand side, one orientation has to be used. The identifier for the specific orientation here again is denoted by $o \in \mathcal{O}$. The sum across all containers is now required, because of the handling of multiple containers. Next the position of the local reference frame χ^o_{pd} is linked to all the vertices v of all components $b \in \mathcal{B}_p$ of a piece $p \in \mathcal{P}$. The position of a vertex χ_{bvd} in space then arises from the one of the local reference frame and the vertex' relative position P^{B}_{bovd} according to the currently used orientation o of the piece. This can be seen in constraint 4.38. To avoid any protrusion of a piece over the containers domain 4.39 defines a positive continuous variable $\lambda_{cv'bv}$ for every component and any of the container's vertices. Note the double sum which is again necessary due to the multiple containers. Since the variable needs to be positive at all times the position of the vertex will stay in between all vertices of the container. Constraint 4.40 then links the variable to the allocation of a piece. Thus, if a piece is allocated to a container, every vertex of every component of the piece is limited to the container's domain and cannot protrude it. The set of constraints is similar to the ones used in the original model, but these are additionally capable of handling multiple different sized containers instead of only one single container. This allows to use the model also for more complex problems, but on the downside the model also gets more complex.

$$\sum_{o\in\mathcal{O}} \vartheta_{po} = \sum_{c\in\mathcal{C}} \psi_{pc} \qquad \forall p \in \mathcal{P} \tag{4.37}$$

$$\chi_{bvd} = \chi^{o}_{pd} + \sum_{o\in\mathcal{O}} \left(P^{\mathtt{B}}_{bovd} \cdot \vartheta_{po}\right) \qquad \forall d \in \mathcal{D}, p \in \mathcal{P}, b \in \mathcal{B}_p, v \in \mathcal{V}^* \tag{4.38}$$

$$\chi_{bvd} = \sum_{c\in\mathcal{C}} \sum_{v'\in\mathcal{V}} \left(P^{\mathtt{C}}_{cv'd} \cdot \lambda_{cv'bv}\right) \qquad \forall d \in \mathcal{D}, p \in \mathcal{P}, b \in \mathcal{B}_p, v \in \mathcal{V} \tag{4.39}$$

$$\sum_{v'\in\mathcal{V}} \lambda_{cv'bv} = \psi_{pc} \qquad \forall p \in \mathcal{P}, b \in \mathcal{B}_p, v \in \mathcal{V}, c \in \mathcal{C} \tag{4.40}$$

Another basic requirement is the non-overlapping of the different pieces. This is in the case of "Tetris"-like objects more complex, since all components of every piece need to be considered. Similar to the basic model proposed by [Fas13] the distance between the position of the virtual center vertex of a component χ_{b0d} and every other component's midpoint needs to be at least the sum of half the diameter of both components in that particular orientation. Therefore, constraint 4.41 states that the distance between component b_1 and b_2 ($\chi_{b_10d} - \chi_{b_20d}$) has to be greater than half the sum of the lengths of the components regarding the current orientation o their pieces are set to. The minimal distance then is satisfied for the positive case, if the variable $\sigma^{+}_{db_1b_2}$ is set to 1. The same is analogously defined in constraint 4.42 for the negative case to ensure an absolute value of the distance. Non-overlapping is then given, if at least one of the defined distances is big enough. Constraint 4.43 secures this requirement by inducing at least one of the relative position markers $\sigma^{+}_{db_1b_2}$ or $\sigma^{-}_{db_1b_2}$ to be set to 1 in the case that both of the corresponding pieces are assigned to the same container denoted by ψ_{pc}. Constraint 4.44 and 4.45 refer to the redundant constraints described in the base model (see subsection 3.1.2). Since the model was extended to handle multiple different sized containers these constraints can only be applied in the case of handling one container. Otherwise the constraint is not applicable anymore. The overlapping especially determines the complexity of the specific model instance in terms of the constraint and variable count, since six constraints and six variables are required for every component pair. Thus, the complexity not only mainly depends on the number of pieces, but also on the degree of detail of the "Tetris"-shapes.

$$\chi_{b_10d} - \chi_{b_20d} \geq \frac{1}{2} \cdot \sum_{o\in\mathcal{O}} \left(L^{\mathtt{B}}_{b_1od} \cdot \vartheta_{p_1o} + L^{\mathtt{B}}_{b_2od} \cdot \vartheta_{p_2o}\right) - L^{\mathtt{C}}_{cd} \cdot \left(1 - \sigma^{+}_{db_1b_2}\right)$$
$$\forall d \in \mathcal{D}, \{p_1, p_2\} \in \mathcal{P}_2, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2} \tag{4.41}$$

$$\chi_{b_20d} - \chi_{b_10d} \geq \frac{1}{2} \cdot \sum_{o\in\mathcal{O}} \left(L^{\mathtt{B}}_{b_1od} \cdot \vartheta_{p_1o} + L^{\mathtt{B}}_{b_2od} \cdot \vartheta_{p_2o}\right) - L^{\mathtt{C}}_{cd} \cdot \left(1 - \sigma^{-}_{db_1b_2}\right)$$
$$\forall d \in \mathcal{D}, \{p_1, p_2\} \in \mathcal{P}_2, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2} \tag{4.42}$$

$$\sum_{d\in\mathcal{D}} \left(\sigma^{+}_{db_1b_2} + \sigma^{-}_{db_1b_2}\right) \geq \psi_{p_1c} + \psi_{p_2c} - 1$$
$$\forall c \in \mathcal{C}, \{p_1, p_2\} \in \mathcal{P}_2, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2} \tag{4.43}$$
$$\sum_{d\in\mathcal{D}} \left(\sigma^{+}_{db_1b_2} + \sigma^{-}_{db_1b_2}\right) \leq \psi_{p_1c}$$
$$\forall c \in \mathcal{C}, \{p_1, p_2\} \in \mathcal{P}_2, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2} \text{ only if } |\mathcal{C}| = 1 \tag{4.44}$$
$$\sum_{d\in\mathcal{D}} \left(\sigma^{+}_{db_1b_2} + \sigma^{-}_{db_1b_2}\right) \leq \psi_{p_2c}$$
$$\forall c \in \mathcal{C}, \{p_1, p_2\} \in \mathcal{P}_2, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2} \text{ only if } |\mathcal{C}| = 1 \tag{4.45}$$

As in the cuboid-formulation a redundant volume limitation constraint is introduced (see constraint 4.46), but as opposed to the one stated before in this case the volume of the pieces is exactly the sum of the volume of its components. This is already comprised by the parameter $V_p^{\mathtt{P}}$.

$$\sum_{p\in\mathcal{P}} \left(\psi_{pc} \cdot V_p^{\mathtt{P}}\right) \leq V_c^{\mathtt{C}} \qquad \forall c \in \mathcal{C} \tag{4.46}$$

4.3.3 Specific constraints

At this stage the model can already be used to generate valid solution to the 3DBPP with "Tetris"-shapes. But still some more requirements have to be added to prevent pieces from floating in mid-air or to keep fragile ones from breaking. This is, similar to the cuboid-approach, done with the help of the following constraints.
The first special requirement concerned is the stability of the packing. Like stated before the main goal here is to avoid that the packing collapses. Despite different possible approaches to handle this requirement, the idea of the one for the cuboid approach is adapted to handle "Tetris"like pieces. But in this case a component b of a piece p needs to be supported by another component. This is done to supply a basic stability of the packing while in certain cases this formulation would not suffice to guarantee a valid packing plan. However, constraints 4.47 and 4.48 are used to match the height of the lower face of piece p_1's component b_1 with the upper face of piece p_2's component b_2, if p_2 supports p_1 indicated by $\omega_{b_1b_2} = 1$. Otherwise M is used to relax this restriction. Constraint 4.56 then ensures that every piece p_1 has at least one component b_1 which is supplied by another component b_2 of another piece or the ground ($\omega^g_{p_1} = 1$). If the ground supplies the respective piece, the height of its lower face is set to zero by constraint 4.55. In the case of one component supporting another their position regarding the x and y dimensions have also to be matched. Therefore, the constraints 4.49 to 4.52 ensure that the component's center of gravity given by its mid point χ_{b_10d} stays inside the supporting area given by component b_2's upper face. So, this position has to be smaller than the upper edge's position depicted by χ_{b_28d}, respectively greater than the lower edge's position depicted by χ_{b_21d} for $d \in \{x, y\}$.

At last it has to be ensured that both pieces are allocated to the same container, if one of them supports the other. Therefore, constraint 4.53 introduces the variable $\phi_{cp_1p_2}$ which indicates that both pieces are allocated to the same container c, if the variable is active. Then, constraint 4.54 ensures, if at least one combination of two components support each other the respective pieces have to be allocated to the same container.
This is in view of the reality a loose formulation, since special pieces may exist which are not stable, if only supported at one component. An example may be a clockwise rotated "L"-shape. This shape can be integrated into the model by two components sticked together. On supporting only one of them in the mentioned orientation the piece will fall over. Such specialties give rise to more complex formulations which are not part of this thesis. For that the area of support given by the supporting pieces needs to be calculated leading to non-linearities to overcome. The piece's center of gravity then needs to stay inside the supporting area.

$$\chi_{b_1 1z} \leq \chi_{b_2 8z} + (1 - \omega_{b_1 b_2}) \cdot M \quad \forall p_1, p_2 \in \mathcal{P}, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2}, p_1 \neq p_2 \tag{4.47}$$
$$\chi_{b_1 1z} + (1 - \omega_{b_1 b_2}) \cdot M \geq \chi_{b_2 8z} \quad \forall p_1, p_2 \in \mathcal{P}, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2}, p_1 \neq p_2 \tag{4.48}$$
$$\chi_{b_1 0x} \leq \chi_{b_2 8x} + M \cdot (1 - \omega_{b_1 b_2}) \quad \forall p_1, p_2 \in \mathcal{P}, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2}, p_1 \neq p_2 \tag{4.49}$$
$$\chi_{b_1 0x} \geq \chi_{b_2 1x} + M \cdot (1 - \omega_{b_1 b_2}) \quad \forall p_1, p_2 \in \mathcal{P}, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2}, p_1 \neq p_2 \tag{4.50}$$
$$\chi_{b_1 0y} \leq \chi_{b_2 8y} + M \cdot (1 - \omega_{b_1 b_2}) \quad \forall p_1, p_2 \in \mathcal{P}, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2}, p_1 \neq p_2 \tag{4.51}$$
$$\chi_{b_1 0y} \geq \chi_{b_2 1y} + M \cdot (1 - \omega_{b_1 b_2}) \quad \forall p_1, p_2 \in \mathcal{P}, b_1 \in \mathcal{B}_{p_1}, b_2 \in \mathcal{B}_{p_2}, p_1 \neq p_2 \tag{4.52}$$

$$\psi_{p_1 c} + \psi_{p_2 c} \geq 2 \cdot \phi_{cp_1p_2} \quad \forall c \in \mathcal{C}, p_1, p_2 \in \mathcal{P}, p_1 \neq p_2 \tag{4.53}$$
$$\sum_{c \in \mathcal{C}} \phi_{cp_1p_2} \geq \sum_{b_1 \in \mathcal{B}_{p_1}} \sum_{b_2 \in \mathcal{B}_{p_2}} \omega_{b_1 b_2} \quad \forall p_1, p_2 \in \mathcal{P}, p_1 \neq p_2 \tag{4.54}$$
$$\chi^o_{pz} \leq (1 - \omega^g_p) \cdot M \quad \forall p \in \mathcal{P} \tag{4.55}$$
$$\omega^g_{p_1} + \sum_{b_1 \in \mathcal{B}_{p_1}} \sum_{\substack{p_2 \in \mathcal{P} \\ p_1 \neq p_2}} \sum_{b_2 \in \mathcal{B}_{p_2}} \omega_{b_1 b_2} = 1 \quad \forall p_1 \in \mathcal{P} \tag{4.56}$$

Next the compatibility requirement of the packed pieces shall be considered. To respect all applying regulations some pieces cannot be packed into the same container. This is ensured with constraint 4.57 by limiting the allocation ψ_{pc} of two respective pieces with incompatible material G_{p_1} and G_{p_2} to one. Thus, only one of these may be allocated to container c at a time.

$$\psi_{p_1 c} + \psi_{p_2 c} \leq 1 \qquad \forall \{p_1, p_2\} \in \mathcal{P}_2 \text{ with } \{G_{p_1}, G_{p_2}\} \in \mathcal{G}^f \tag{4.57}$$

The requirement of load bearing likewise needs to be considered for viable packing plans. Similar to the cuboid-formulation this is achieved by preventing fragile pieces from becoming supporting pieces. This is done by setting the variable indicating the

support of piece p_2 by piece p_1 to zero ($\omega_{p_2p_1}$) for all fragile pieces in $p_1 \in \mathcal{P}^s$ and every other piece $p_2 \in \mathcal{P}$. Constraint 4.58 ensures this issue.

$$\omega_{p_2p_1} = 0 \qquad \forall p_1 \in \mathcal{P}^s, p_2 \in \mathcal{P} \text{ with } p_1 \neq p_2 \tag{4.58}$$

The prohibition of forbidden orientations is straightforward in the "Tetris"-formulation. For every orientation that is marked as forbidden $o \in \mathcal{O}_p^f$ for piece p the corresponding binary variable depicting the orientation in use ϑ_{po} has to be set to zero. This requirement is stated in constraint 4.59.

$$\vartheta_{po} = 0 \qquad \forall p \in \mathcal{P}, o \in \mathcal{O}_p^f \tag{4.59}$$

4.4 Overview of problem characteristics

Similar to the matching done in section 3.3 this section will discuss the requirements covered by the proposed model formulations. Therefore, Table 4.4 depicts a matching of the two models against the requirements issued in chapter 2. The matched issues are more complex shapes, multiple containers (MC), different sized containers (DC), the possibility of rotating pieces by at least 90° (R), the stability of the packing (LS), the load bearing (LB), handling of forbidden orientations (FO) and the compatibility (C).
The cuboid-formulation as well as the "Tetris"-formulation are both capable of handling multiple different sized containers with possible notches. These can be considered by fixing pieces representing the space to exclude from the container at the specific position. For the "Tetris"-formulation this can be approximated to model the irregular shaped container as accurately as necessary. The rotation by 90° is also possible in both models, just as prohibiting certain orientations. The load stability and load bearing requirements are especially tricky to handle. The cuboid-formulation ensures the stability and load bearing, but is very strict. This way some packings feasible in reality might get cut off. The "Tetris"-formulation handles the stability in a more relaxed manner to allow a more thorough exploitation of the space by using "Tetris"-shapes. On the downside the horizontal stability as defined by [PSL12] cannot be guaranteed at all times. At last the compatibility of the different pieces is respected by both formulations.

Table 4.4: Matching the requirements considered by the models

Model	S	MC	DC	R	LS	LB	FO	C
Cuboid-formulation	Cuboid	✓	✓	✓	✓	✓	✓	✓
"Tetris"-formulation	"Tetris"	✓	✓	✓	(✓)	(✓)	✓	✓

5 Heuristic algorithms

This chapter introduces the different heuristic methods of this work in detail. The three primal constructive procedures are explained first. All of them exploit different insertion point approaches. An insertion point (IP) is a more general term for positions inside a container like the EPs at which a piece may or may not be inserted. The Extreme Point Insertion (see section 5.2) as well as the Space Defragmentation technique (see section 5.3) both rely on the EP-concept, while the Push Insertion technique (see section 5.4) uses another new concept. After the introduction of the constructive heuristics an improvement technique in terms of a simple metaheuristic is explained. For this a Greedy Adaptive Search Procedure (GASP) (see section 5.5) is exploited and extended. But at the beginning some preliminary matters are introduced.

5.1 Preliminaries

The following section introduces preliminary information about the implementation background used to embed all different approaches of this thesis (see subsection 5.1.1). Because all constructive algorithms use the same input and output data, except for their configuration, subsection 5.1.2 consolidates the description of that. Furthermore, additional parameters and variables used by the heuristics are shown (see subsection 5.1.3). These extend the mathematical parameters and sets described above (see subsection 4.1.3). In subsection 5.1.4 the different initial sorting metrics of the pieces for the container insertion are explained.

5.1.1 Object-model

This section describes the basic architecture of the implementation which is fundamental for both, the heuristic methods and the model transformation.
Figure 5.1 depicts the structure of a problem instance. The class diagram consists of an `Instance` object and all of its parts necessary to describe one specific instance. The `Instance` itself only has an identifying name, but is linked to all containers, pieces and generated solutions. Thus, one instance may have a set of containers assigned to it which are available for piece allocation. Such a `Container` has an identifying ID and the Mesh depicting its boundaries. All pieces which have to be inserted are represented by the `Piece` class. Each of them is identified by the ID and contains information about the fragility, forbidden orientations and the material. The material field abstractly describes the content of the piece and is used to identify incompatibilities with other pieces. Additionally, the volume field is used to measure

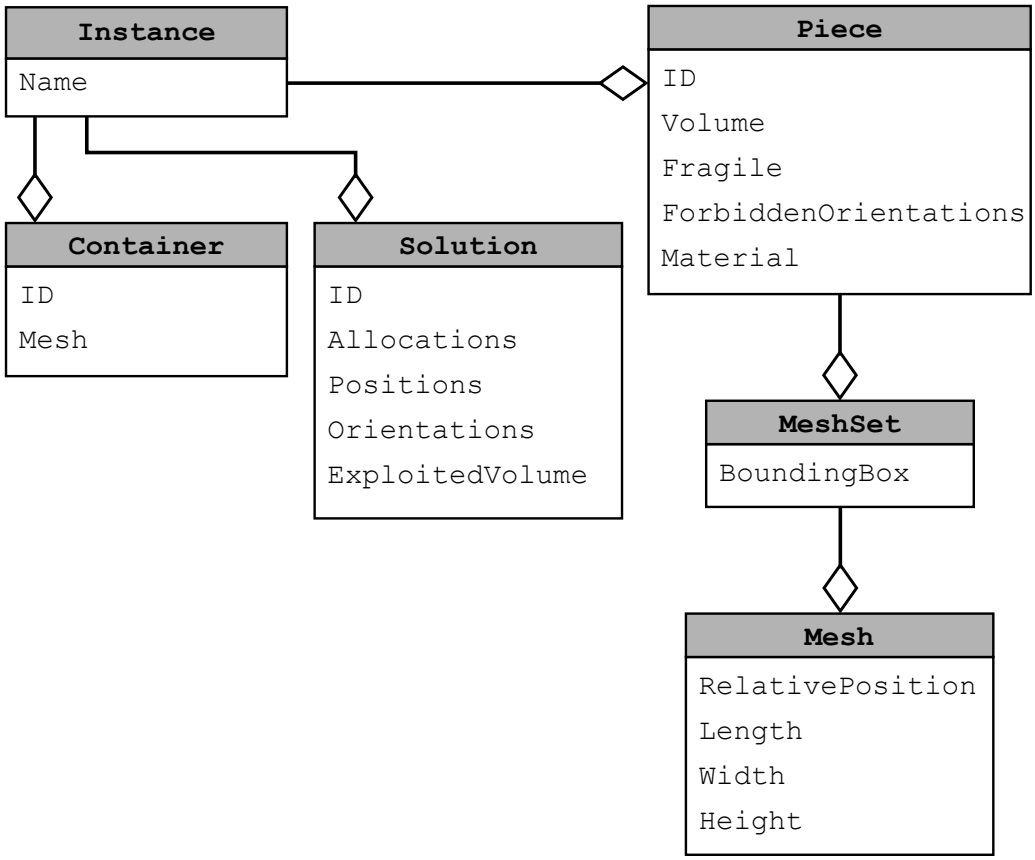


Figure 5.1: Class diagram of the object model

the value of packing the piece into a container. In this thesis the field will always contain the real volume of the piece measured in the same dimension as the containers' volumes. If other goals like priorities or similar information need to be considered, simply modifying this field would be possible to consider the information. I.e., a piece of higher priority might be assigned a multiple of the intrinsic value. Due to the handling of "Tetris"-pieces every piece consists of a set of meshes. Like described by the `Mesh` class each of these have a position relative to the piece's local reference frame $(0, 0, 0)$ and information about the size. The `MeshSet` class then describes the bounding box of those components by the smallest rectangular hull. Methods which are not capable of handling "Tetris"-pieces use this information instead of the more accurate component information to identify overlaps or other aspects. The last important class is the `Solution` describing one solution to the problem instance. Besides an identifier it contains all information about the pieces' positions inside the containers, the orientations in which they were packed and the container they were allocated to. The exploited volume is the main evaluation value defining the quality of a solution given by the sum of volumes of the packed pieces.
The second part of the architecture is given by Figure 5.2. The main element here is the `Method` interface. Every method regardless of the used technique is given an

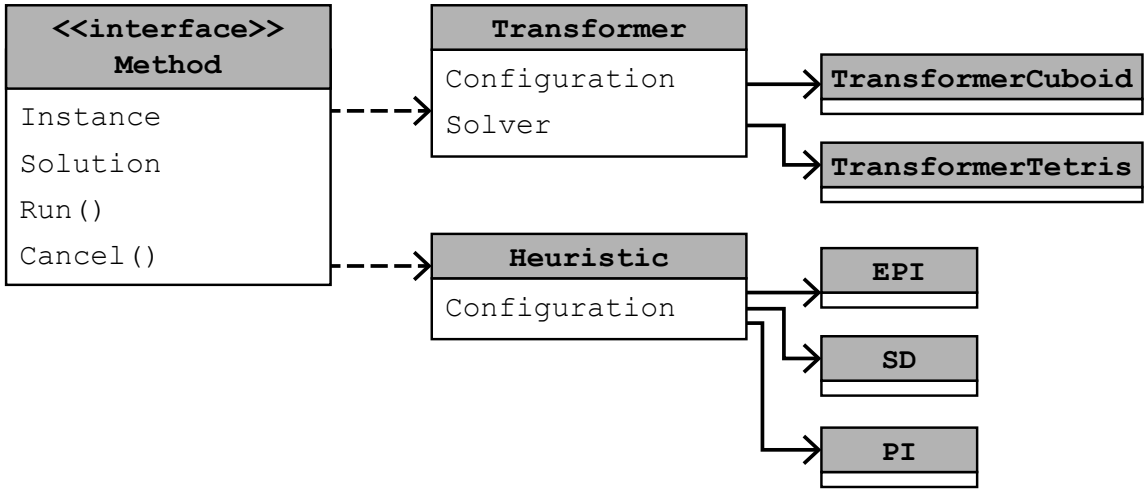


Figure 5.2: Class diagram of the method architecture

instance which is solved during the `Run()` procedure resulting in a solution. If the process needs to be canceled before the termination criterion is met, `Cancel()` is called and the best solution so far is returned. Both base classes `Transformer` and `Heuristic` implement this interface consolidating the model transformation of the cuboid-formulation and the "Tetris"-formulation as well as the three constructive heuristics EPI, SD and PI. Additional to the configuration of the particular approach the transformer references a solver used to optimize the model instance. In this work the commercial solvers CPLEX 12.5 (see [IBM13]) and Gurobi 5.6 (see [Gur13]) were used. Note that the three heuristics are wrapped by the metaheuristic shown in section 5.5 in each case.

5.1.2 Algorithm input / output

The input data of all constructive algorithms EPI, SD and PI are three ordered sets. The first one ($\mathcal{C}$) contains all containers available for piece allocation. These need to be of cuboid shape but may have various side-lengths. The second set ($\mathcal{P}$) contains all pieces, which have to be inserted into the containers in a predefined order. This order greatly influences the result obtained by the constructive heuristic and can be based on different principles explained in subsection 5.1.4. This also applies to some extent for the third set, the orientations in a specific order per piece. This is due to the fact that the algorithm tries to insert the pieces in the same order they are submitted. Thus, if a piece gets inserted at a specific position the algorithm will never change that position anymore. This also applies for the orientation order which defines the order of the orientation-IDs $o \in \mathcal{O}_p$ per piece p, if the best fit strategy is not used. Then the piece gets inserted in the first orientation fitting the container, thus influencing the quality of the solution by the predefined orientation order. The result of the algorithm then is a solution to the problem instance. This solution

consists of three parts. The set of pieces allocated to a container denoted by $\mathcal{S}_c^C$ for all containers $c \in \mathcal{C}$, as well as the position S_p^P and orientation S_p^O of each piece $p \in \mathcal{P}$.

5.1.3 Parameters and variables

The following section consolidates the notation used by the different algorithm descriptions. This can be seen as an extension to the notation used to describe the models, which is necessary to introduce the different used parameters and additional runtime-variables. The description is given in Table 5.1. The first column introduces the variable or parameter, the second column indicates which heuristics make use of the respective symbol and additionally a short description is given. To support the clarity of the table the algorithms are shortened as follows: EPI (1), SD (2), PI (3) and GASP (4).

Table 5.1: Special symbols of the heuristics

$\mathcal{S}_c^C$	1,2,3,4	The set of pieces contained in a specific container c. This is the first part of a solution to a problem instance.
S_p^P	1,2,3,4	The position $(S_p^P.x, S_p^P.y, S_p^P.z)$ as compared to $(\chi_{px}^o, \chi_{py}^o, \chi_{pz}^o)$ (see section 4.3) of piece p. The same local reference frame is used to describe the position of a piece inside a container. This is the second part of a solution to a problem instance.
S_p^O	1,2,3,4	The orientation o of piece p in the solution. This is the last part necessary to describe a complete solution to a problem instance.
S^U	4	The volume of the packed containers in the solution. This value is used for fast access and is equal to $\sum_{c \in \mathcal{C}} \sum_{p \in \mathcal{S}_c^C} V_p^{\mathtt{P}}$.
S^N	4	The number of containers used in the solution to pack the pieces. This is also a value used for fast access and is equal to $\left\lvert \left\{ c \in \mathcal{C} \mid \mathcal{S}_c^C \neq \emptyset \right\} \right\rvert$.
$\mathcal{W}$	4	A set of workers $w \in \mathcal{W}$ able to execute tasks in parallel. Every worker then is executed in its own thread.
i	4	The current iteration number. This value is increased by one with each completed iteration.
i^L	4	The iteration number of the last improvement made to the best solution so far. This number is updated each time an improvement is done and therefore can be used to measure the distance in iterations since this event.
i^{SD}	4	The stagnation distance defining the number of iterations done without an improvement before the progress terminates.

i^R	4	The iteration number of the last score reinitialization. This is updated and handled similar to the last improvement marker i^L.
i^{RD}	4	The number of iterations without an improvement of the solution until a long-term reinitialization is conducted.
s^P	4	The number of possible swaps, which influences the score modification. As s^P increases the effect of the modification on the score-value increases.
s^{max}	4	The maximal number of possible swaps, which is used to bound s^P.
m^I	4	Initial percentage of modification when updating the score. This is the second influence of the modification defining a fractional change depending on the current value of the score.
m^{max}	4	The maximum percentage of modification when updating the score. This value is increased by one at each long-term reinitialization.
r^S	4	A randomly applied "salt"-value which is used to further permutate the order of the pieces.

5.1.4 Piece insertion orders

As previously mentioned the order in which the pieces are inserted into the containers has a high influence on the solution quality. Therefore, some metrics are used to supply a good initial sorting of the pieces. This sorting is then used to generate the first solution. Most of the implemented metrics derive from the work by [CPT08]. These principles are very intuitive heuristics, like beginning with the largest piece.

Volume Sorts pieces by their volume ($V_p^{\mathtt{P}}$) in descending order.

Volume-Height Sorts pieces by their volume ($V_p^{\mathtt{P}}$) in descending order using a height tie-breaker, which orders pieces by their height ($L_{poz}^{\mathtt{P}}$) in descending order.

Height-Volume Sorts pieces by their height ($L_{poz}^{\mathtt{P}}$) in descending order using a volume tie-breaker, which orders pieces by their volume ($V_p^{\mathtt{P}}$) in descending order.

Height-Width-Length Sorts pieces by their height ($L_{poz}^{\mathtt{P}}$), then by their width ($L_{poy}^{\mathtt{P}}$) and then by their length ($L_{pox}^{\mathtt{P}}$). This concept is especially useful when using the push-insertion heuristic, since the default normalization-order in this method is (x, y, z). By that the highest pieces are inserted first giving place to pieces pushed in afterwards, because pieces may block each other when pushed in by only one dimension.

Area-Height Sorts pieces by the size of their base-area ($L_{pox}^{\mathtt{P}} \cdot L_{poy}^{\mathtt{P}}$), then by their height ($L_{poz}^{\mathtt{P}}$).

Height-Area Sorts pieces by their height (L^{P}_{poz}), then by the size of their base-area ($L^{\mathsf{P}}_{pox} \cdot L^{\mathsf{P}}_{poy}$).

5.2 Extreme Point Insertion

The first constructive approach is the Extreme Point Insertion (EPI) technique. This method is based on the EP-FFD respectively EP-BFD heuristics proposed by [CPT08]. Thus, Extreme Points (EPs) are used as IPs at which new pieces are allocated into the container. These EPs are calculated in the same way as described in subsection 3.2.1. The extensions done by this work to the respective approach are first of all the possibility of rotating pieces by 90° while also restraining forbidden orientations. Furthermore, the compatibility of two pieces is considered and an insertion of a piece incompatible with one already assigned to the container is prevented. At last the handling of "Tetris"-like pieces is possible. This last adjustment requires the most changes to the basic algorithm, since many parts need to be reconsidered.

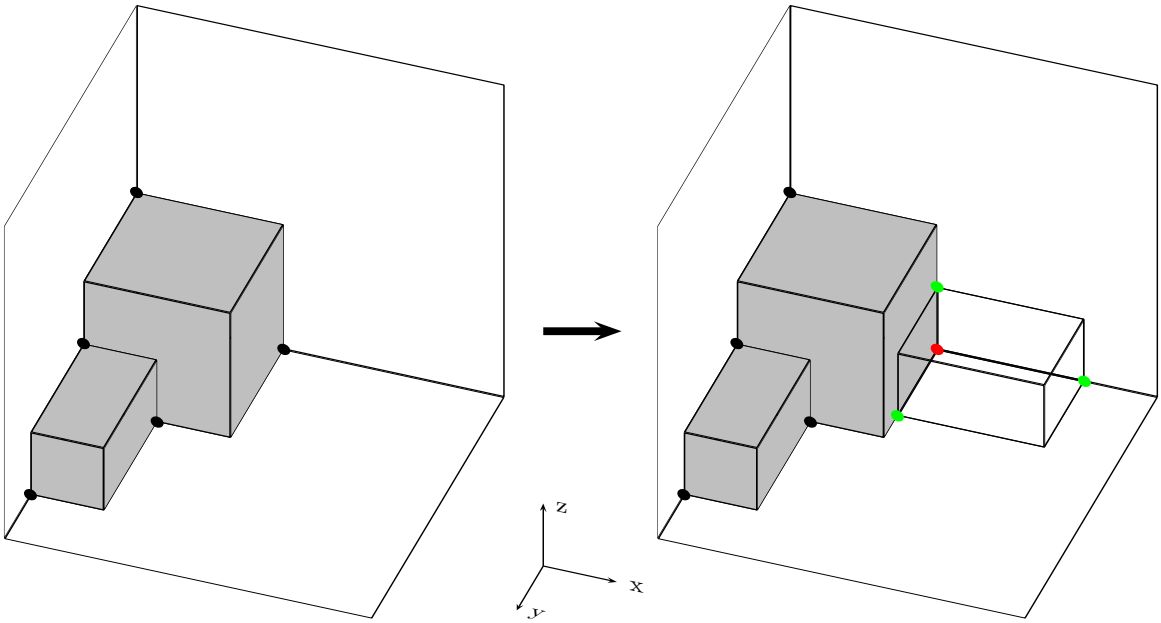


Figure 5.3: Extreme Point Insertion example

Algorithm 5.1 shows the pseudo-code of the basic EPI procedure. A representation is used, which fits the problem representation used by the model formulations with a few extensions to match an algorithm description. Table 5.1 summarizes these extensional symbols. Alongside with the algorithm description an example of a typical piece insertion is given by Figure 5.3. In the graphic the available insertion points are marked black while the used one is marked red and the newly generated ones are green. The small wire-frame depicts the inserted piece.
When initializing the algorithm at first a default IP is defined for every container at the position $(0, 0, 0)$ and then inserted into the set of available IPs per container $\mathcal{E}_c$. These

sets are the main component of this approach. For every piece the insertion points contained in $\mathcal{E}_c$ are used to identify positions at which the piece may be allocated to the container. So the outer-loop iterates all pieces in the predefined order, trying to insert them consecutively. The next loop rotates the piece in all given rotations denoted by $\mathcal{O}_p$, skipping the forbidden ones $\mathcal{O}_p^f$ (see line 7). By this no piece can be allocated in a forbidden orientation. Next the containers available for insertion are iterated in the fixed order (see line 8). At this point only containers not including any incompatible materials are available for insertion of the particular piece p. Hence, no two incompatible goods will ever be packed into the same container. E.g.: If the first container has loaded some flammable goods, this one will be skipped when loading an explosive piece.
Before the investigation of possible packing positions is done, the volume is quickly analyzed to skip containers, which do not have enough space available to load piece p at all. In the investigation of a container all available insertion points $e \in \mathcal{E}_c$ are examined (see line 10). These are also ordered ascending by their position values $(e.x, e.y, e.z)$. With the height $e.z$ having highest precedence and breaking ties with $e.x$ and $e.y$. This leads to a packing, which "grows" from the containers origin $(0, 0, 0)$ to its extents. For every insertion point it is then checked, if the piece can be loaded relative to any vertex $v \in \mathcal{V}$ of every component $b \in \mathcal{B}_p$ it consists of (see lines 11 & 12). This means that the piece is loaded with the respective vertex fixed at the position given by the insertion point e. The positions of all other vertices of the piece are then set relative to this "insertion-vertex" v of component b. This introduces certain insertion possibilities when handling "Tetris"-pieces, for example enabling insertions relative to an inner edge. This way a "L"-shape may be packed "around" a box, even if the side-lengths of them do not match. If "*BestFit*" is not enabled the search for a fitting insertion position ends immediately after a valid one is found (see line 13). Instead, if "*BestFit*" is enabled all possible combinations of inserting a piece are investigated and a score is calculated for each one. This score can be based on different merit-functions defined in section B.1. After investigating all possible insertions the best one is performed. In both cases such a successful insertion of a piece is identified by the `InsertionCheck` routine explained below.
On allocating a piece to the container the EPs for this piece are generated and added to the set of available insertion points ($\mathcal{E}_c$). The generation of the EPs is done like explained in section A.1, with the extension to the basic EP concept by [CPT08] that EPs are generated for every component b of piece p. Also equivalent insertion points get proned in each iteration, since equal points would otherwise result in an unnecessary double-check of the same conditions.
The main adaption which has to be done when handling "Tetris"-pieces not only concerns the position at which the item is inserted, but also the check for non-overlapping of the piece with other pieces or the containers domain. The procedure `InsertionCheck` (see Algorithm 5.2) is used here to determine whether the allocation to a position is valid, or not. To achieve this at first the absolute insertion point is calculated with respect to the given insertion point e in combination with the vertex v's position of component b. If no component b is supplied, the local reference frame

Algorithm 5.1: Extreme Point Insertion (with "Tetris"-pieces)

Input: $\mathcal{C}, \mathcal{P}, \mathcal{O}_p$
Output: $\mathcal{S}_c^C, S_p^O, S_p^P$

1 **foreach** $c \in \mathcal{C}$ **do** $\mathcal{E}_c \leftarrow \{(0,0,0)\}$
2 $p_{best} \leftarrow$ nil, $o_{best} \leftarrow 0$, $c_{best} \leftarrow$ nil, $e_{best} \leftarrow$ nil, $v_{best} \leftarrow 0$, $b_{best} \leftarrow$ nil
3 **foreach** $p \in \mathcal{P}$ **do**
4 placed $\leftarrow$ false
5 $score_{best} \leftarrow \infty$
6 **if** *TimeUp* **then return**
7 **foreach** $o \in \mathcal{O}_p \setminus \mathcal{O}_p^f$ **do**
8 **foreach** $c \in \{c \in \mathcal{C} \mid \nexists p' \in \mathcal{S}_c^C : G_{p'} \in \mathcal{G}_p^f\}$ **do**
9 **if** $\sum_{p' \in \mathcal{S}_c^C} V_{p'}^{\mathtt{P}} + V_p^{\mathtt{P}} \leq V_c^{\mathtt{C}}$ **then**
10 **foreach** $e \in \mathcal{E}_c$ **do**
11 **foreach** $v \in \mathcal{V}$ **do**
12 **foreach** $b \in \mathcal{B}_p$ **do**
13 **if** *placed* $\wedge \neg$*BestFit* **then** Break search
14 **if** `InsertionCheck`$(\mathcal{S}_c^C, S_p^O, S_p^P, c, e, p, b, v, o)$ **then**
15 **if** *BestFit* **then**
16 **if** `Score`$(c, p, o, e) < score_{best}$ **then**
17 placed $\leftarrow$ true
18 $score_{best} \leftarrow$ `Score`(c, p, o, e)
19 $p_{best} \leftarrow$ p, $o_{best} \leftarrow$ o, $c_{best} \leftarrow$ c, $e_{best} \leftarrow$ e, $v_{best} \leftarrow$ v, $b_{best} \leftarrow$ b
20 **else**
21 placed $\leftarrow$ true
22 $p_{best} \leftarrow$ p, $o_{best} \leftarrow$ o, $c_{best} \leftarrow$ c, $e_{best} \leftarrow$ e, $v_{best} \leftarrow$ v, $b_{best} \leftarrow$ b
23 **if** *placed* **then**
24 $\mathcal{S}_{c_{best}}^C \leftarrow \mathcal{S}_{c_{best}}^C \cup \{p_{best}\}, S_{p_{best}}^O \leftarrow o_{best}, S_{p_{best}}^P \leftarrow e_{best}$
25 $\mathcal{E}_{c_{best}} \leftarrow \mathcal{E}_{c_{best}} \cup$ GenerateEPs$(e_{best}, p_{best}, c_{best})$

is used. For EPI a such component will always be supplied, but the routine is shared with the PI heuristic where a relative insertion component is not useful. Then it is checked whether the piece would extent the containers domain (see line 5). For this only the bounding box needs to be considered. Additionally the potential overlapping of every two components of the piece and every other piece is checked (see line 10). In the case of only one overlap the procedure signals an invalid packing and returns *false*, otherwise *true* is returned, indicating a valid packing.

Algorithm 5.2: Insertion check (with Tetris-pieces)

Input: $\mathcal{S}_c^C, S_p^O, S_p^P, c, e, p, b, v, o$
Output: true / false

if $b \neq nil$ **then**
  $e \leftarrow (e.x - P^{\mathtt{B}}_{bovx}, e.y - P^{\mathtt{B}}_{bovy}, e.z - P^{\mathtt{B}}_{bovz})$
else
  $e \leftarrow (e.x - P^{\mathtt{P}}_{povx}, e.y - P^{\mathtt{P}}_{povy}, e.z - P^{\mathtt{P}}_{povz})$
if $e.x < 0 \vee e.y < 0 \vee e.z < 0 \vee L^{\mathtt{C}}_{cx} < e.x + L^{\mathtt{P}}_{pox} \vee L^{\mathtt{C}}_{cy} < e.y + L^{\mathtt{P}}_{poy} \vee L^{\mathtt{C}}_{cz} < e.z + L^{\mathtt{P}}_{poz}$ **then**
  return *false*
foreach $b \in \mathcal{B}_p$ **do**
  foreach $p' \in \mathcal{S}_c^C$ **do**
    foreach $b' \in \mathcal{B}_{p'}$ **do**
      if $S^P_{p'}.x + P^{\mathtt{B}}_{b'S^O_{p'}1x} \geq e.x + P^{\mathtt{B}}_{bo8x} \vee$
      $S^P_{p'}.x + P^{\mathtt{B}}_{b'S^O_{p'}8x} \leq e.x + P^{\mathtt{B}}_{bo1x} \vee$
      $S^P_{p'}.y + P^{\mathtt{B}}_{b'S^O_{p'}1y} \geq e.y + P^{\mathtt{B}}_{bo8y} \vee$
      $S^P_{p'}.y + P^{\mathtt{B}}_{b'S^O_{p'}8y} \leq e.y + P^{\mathtt{B}}_{bo1y} \vee$
      $S^P_{p'}.z + P^{\mathtt{B}}_{b'S^O_{p'}1z} \geq e.z + P^{\mathtt{B}}_{bo8z} \vee$
      $S^P_{p'}.z + P^{\mathtt{B}}_{b'S^O_{p'}8z} \leq e.z + P^{\mathtt{B}}_{bo1z}$ **then**
        continue
      else
        return *false*
return *true*

The proposed algorithm is capable of inserting "Tetris"-pieces into a set of multiple different-sized containers, while respecting the previously discussed requirements. An exception here is the stability, which is somewhat respected by packing from the bottom up, but cannot be guaranteed. Alongside the load bearing cannot be guaranteed since it directly depends on the stability issue. Also it is possible to fall back to a cuboid-based approach by limiting the vertices and components relative to which the pieces are inserted. Therefore, the method capable of handling "Tetris"-pieces is called EPI-Tetris, while the cuboid based method is called EPI.

5.3 Space Defragmentation

The next method of this work is the Space Defragmentation (SD) technique, which extends the work initially proposed by [ZZOL12], which itself mainly bases on the EP-principle introduced by [CPT08]. The extensions done here consider different orientations $o \in \mathcal{O}_p$ of the pieces alongside the forbidden ones $o \in \mathcal{O}_p^f$ as well as the compatibility of two pieces. As opposed to the EPI algorithm no "Tetris"-like pieces are considered. This is due to the push-out approach used by the SD technique, which calculates the distance pieces can be pushed into a specific direction without the risk of overlapping with other pieces. Pushing thereby is always conducted away from the origin along one of the three axes $d \in \mathcal{D}$. But since the bounding boxes of "Tetris"-pieces may overlap, while the pieces themselves do not overlap, the algorithm used by the approach cannot be used to calculate safe pushing distances. Additionally the authors proposed a bin-shuffling algorithm wrapping the constructive space defragmentation technique called BS-EPSD resulting in an improvement heuristic (see [ZZOL12]). The work uses a similar algorithm, which "shuffles" the order of the pieces instead of the order of the bins. This algorithm is introduced in section 5.5.
The SD algorithm proposed here is shown in pseudo-code in Algorithm 5.3 with the help of additional symbols defined in Table 5.1. Furthermore, an example of an insertion with push-out is given in Figure 5.4. In this graphic the already allocated pieces are grey while the newly inserted piece if depicted by a wire-frame. The respective sets of available insertion points before and after the insertion are drawn as black dots. Like the EPI algorithm this method initializes the default insertion points for all containers to $(0, 0, 0)$. The set of available insertion points is also denoted by $\mathcal{E}_c$. The outer-loop then iterates all pieces in the given order, trying to insert them at the available positions. This is aborted in the case of an exceeded time-limit. The main loop now splits into three stages:

1. Insertion at an IP with push-out (see line 5)
2. Insertion by inflating and replacing an already allocated piece (see line 16)
3. Inserting at a default position of a new container (see line 27)

The algorithm conducts the insertion by trying all possibilities of one stage and then moving to the next stage. If no stage can find a valid insertion the piece will not be inserted. For each of these stages the algorithm, similar to EPI, iterates all not forbidden orientations and all containers with compatible pieces. If the piece can be allocated to any position the search will immediately abort and move on to the next piece.
The first stage not only tries to insert the piece p at one of the available insertion points e, but also pushes all pieces away to make space for the insertion (see line 9). The push is done along the three axes consecutively. For every axis the pieces with an RRT position (χ_{p8d}) lying to the right of the insertion point are pushed. This bases on a comparability graph proposed by [ZZOL12]. If the push-out produces sufficient

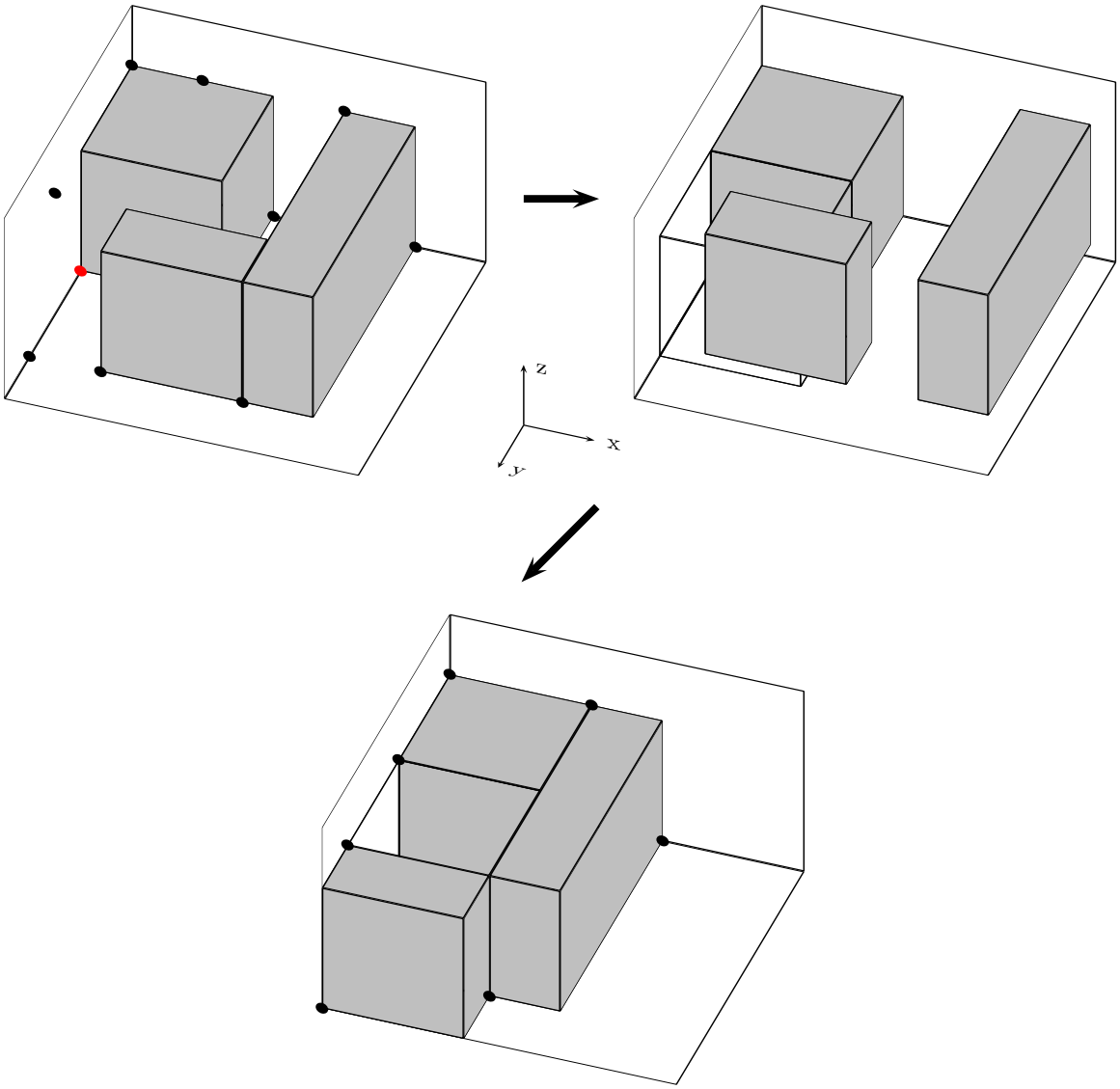


Figure 5.4: Space Defragmentation's Insertion with PushOut example

space the piece is allocated to the respective position and the pushed pieces remain in their new positions. However, if the push-out does not produce sufficient space the push is revoked. In the successful case the now pushed pieces need to be normalized again (see line 11). This means that all pieces are pushed towards the containers origin along all three axes as much as possible. If no further pushing is possible the normalization terminates. The exact mechanism of normalization is described in section A.2. Also the set of EPs is updated. In SD previously generated insertion points need to be forgotten since the push-out and normalization procedures render them useless. So, every time a piece is successfully inserted all insertion points of the container $\mathcal{E}_c$ are replaced by the newly generated ones for every piece $p \in \mathcal{S}_c^C$ (see line 12). At last the positional information for future push-out operations has to be

refreshed. The exact procedure and used invariant is described by [ZZOL12] in detail. In the second stage the algorithm employs this push-out technique to inflate already allocated pieces and replacing them afterwards (see line 20). This is attempted only for pieces that are bigger than an allocated one. In this case the push-out operation is conducted on the FLB position (χ_{p8d}) of the piece p' to be replaced. Thus, pushing out all pieces around the considered one. If this operation produces enough space the "inflated" piece p' is removed and replaced by the new and bigger one p. Like before the container is normalized and new EPs and push-out information is computed (see line 22 to line 24). An example of this technique depicted in two dimensions is shown in Figure 5.5.

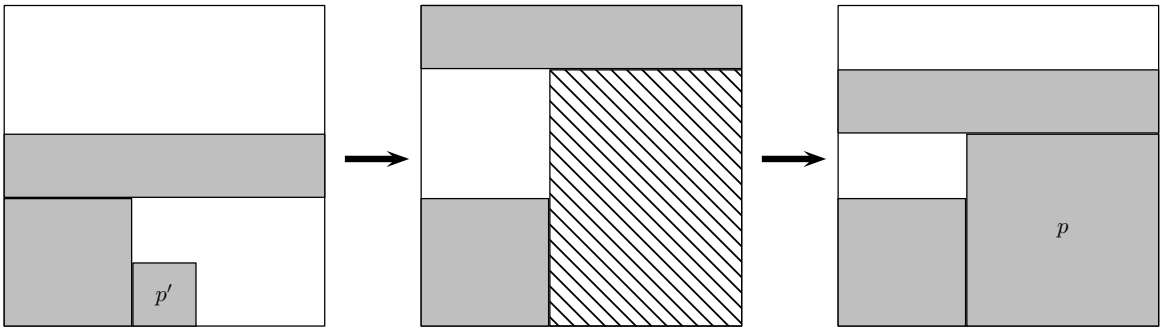


Figure 5.5: Space Defragmentation's Inflate and Replace example

In the third or last stage the piece is simply allocated to the default position of a new container, if it fits in it. After a successful insertion also all information is updated accordingly. A normalization is not required in this case, because the piece already is positioned at the origin of the container.

The described algorithm is capable of inserting cuboid-pieces into a set of multiple different-sized containers, while respecting most of the previously discussed requirements. The exception are the stability and the load bearing, which cannot be guaranteed. However, due to the fact that the pieces are constantly pushed towards the container's origin the resulting packing will always be somehow dense and placed on the ground, thus pieces will supply each other to a certain degree. But a complete confirmation of the packing's stability cannot be given.

Algorithm 5.3: Space Defragmentation

Input: $\mathcal{C}, \mathcal{P}, \mathcal{O}_p$
Output: $\mathcal{S}_c^C, S_p^O, S_p^P$

1 **foreach** $c \in \mathcal{C}$ **do** $\mathcal{E}_c \leftarrow \{(0,0,0)\}$
2 **foreach** $p \in \mathcal{P}$ **do**
3 placed $\leftarrow$ false
4 **if** *TimeUp* **then return**
5 **foreach** $o \in \mathcal{O}_p \setminus \mathcal{O}_p^f$ **do**
6 **foreach** $c \in \{c \in \mathcal{C} \mid \nexists p' \in \mathcal{S}_c^C : G_{p'} \in \mathcal{G}_p^f, \mathcal{S}_c^C \neq \emptyset\}$ **do**
7 **if** $\sum_{p' \in \mathcal{S}_c^C} V_{p'}^{\mathtt{P}} + V_p^{\mathtt{P}} \leq V_c^{\mathtt{C}}$ **then**
8 **foreach** $e \in \mathcal{E}_c$ **do**
9 **if** `InsertionCheckWithPushOut`$(\mathcal{S}_c^C, S_p^O, S_p^P, c, e, p, o)$ **then**
10 $\mathcal{S}_c^C \leftarrow \mathcal{S}_c^C \cup \{p\}, S_p^O \leftarrow o, S_p^P \leftarrow e$
11 `Normalize`$(c, (x, y, z))$
12 $\mathcal{E}_c \leftarrow$ `GenerateEPs`(c)
13 `UpdatePushOut`(c)
14 placed $\leftarrow$ true, Break search
15 **if** $\neg$*placed* **then**
16 **foreach** $o \in \mathcal{O}_p \setminus \mathcal{O}_p^f$ **do**
17 **foreach** $c \in \{c \in \mathcal{C} \mid \nexists p' \in \mathcal{S}_c^C : G_{p'} \in \mathcal{G}_p^f, \mathcal{S}_c^C \neq \emptyset\}$ **do**
18 **if** $\sum_{p' \in \mathcal{S}_c^C} V_{p'}^{\mathtt{P}} + V_p^{\mathtt{P}} \leq V_c^{\mathtt{C}}$ **then**
19 **foreach** $p' \in \mathcal{S}_c^C$ **do**
20 **if** `InflateAndReplace`$(\mathcal{S}_c^C, S_p^O, S_p^P, c, p, p', o)$ **then**
21 $\mathcal{S}_c^C \leftarrow \mathcal{S}_c^C \cup \{p\} \setminus \{p'\}, S_p^O \leftarrow o, S_p^P \leftarrow e$
22 `Normalize`$(c, (x, y, z))$
23 $\mathcal{E}_c \leftarrow$ `GenerateEPs`(c)
24 `UpdatePushOut`(c)
25 placed $\leftarrow$ true, Break search
26 **if** $\neg$*placed* **then**
27 **foreach** $o \in \mathcal{O}_p \setminus \mathcal{O}_p^f$ **do**
28 **foreach** $c \in \{c \in \mathcal{C} \mid \mathcal{S}_c^C = \emptyset\}$ **do**
29 **if** $\sum_{p' \in \mathcal{S}_c^C} V_{p'}^{\mathtt{P}} + V_p^{\mathtt{P}} \leq V_c^{\mathtt{C}}$ **then**
30 **if** `InsertionCheck`$(\mathcal{S}_c^C, S_p^O, S_p^P, c, e, p, o)$ **then**
31 $\mathcal{S}_c^C \leftarrow \mathcal{S}_c^C \cup \{p\}, S_p^O \leftarrow o, S_p^P \leftarrow e$
32 $\mathcal{E}_c \leftarrow$ `GenerateEPs`$(c, \mathcal{S})$, `UpdatePushOut`$(c, \mathcal{S})$
33 placed $\leftarrow$ true, Break search

5.4 Push Insertion

The third constructive heuristic proposed here is the Push Insertion (PI) technique. This method is a very simple new approach of solving the 3DBPP. The basic concept relies on the normalization procedure discussed before. PI is thereby capable of handling "Tetris"-like pieces and rotating them. Again 90° rotations around all axes are possible, resulting in the 24 distinct orientations $o \in \mathcal{O}$ discussed previously. The possible rotations are again limited to respect the forbidden orientations requirement. The compatibility of two pieces is also taken into account by forbidding the allocation of a piece to a container, if an incompatible one is already packed into it. In the following the details will be introduced.
The procedure is described in Algorithm 5.4 using the same extended notation of the model formulation as used by the pseudo-code before (see Table 5.1). In order to supply a more vivid depiction of a push insertion move Figure 5.6 illustrates an example of this. The path on which the inserted piece is moved is depicted by the red arrows.

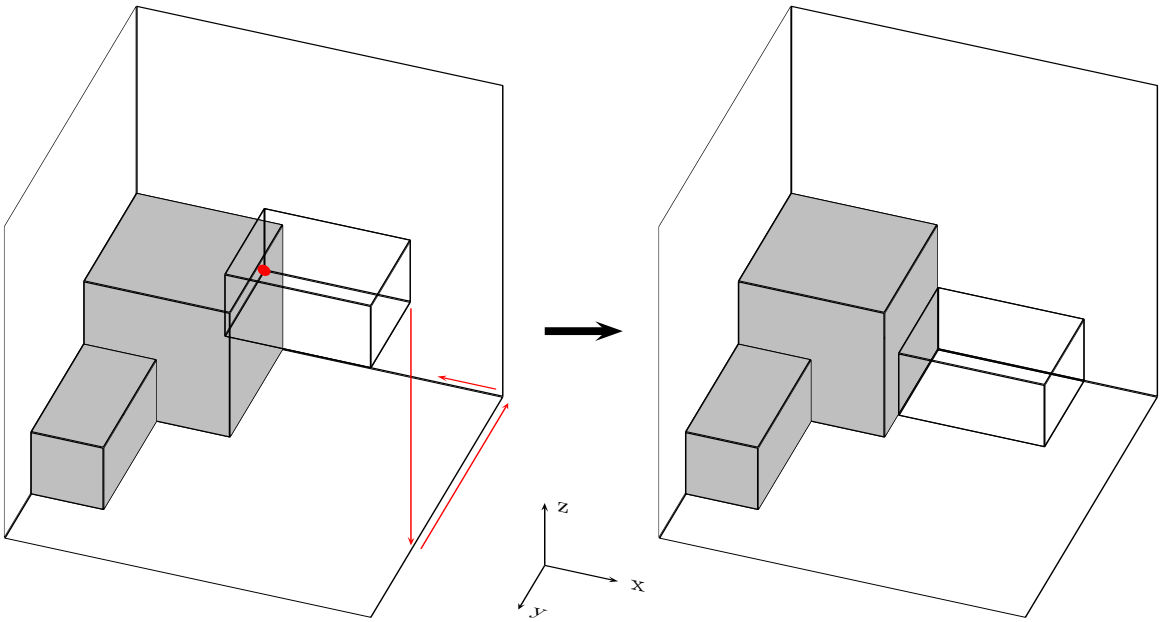


Figure 5.6: Push Insertion example

Like the other constructive heuristics proposed in this work, PI's outer-loop iterates the given pieces $p \in \mathcal{P}$ in the given order trying to insert each one of them. As opposed to the previously shown approaches, no set of available insertion points is managed, but each one is generated on-the-fly. For each piece all valid orientations $o \in \mathcal{O}_p \setminus \mathcal{O}_p^f$ and compatible containers $c \in \{c \in \mathcal{C} \mid \nexists p' \in \mathcal{S}_c^C : G_{p'} \in \mathcal{G}_p^f, \mathcal{S}_c^C \neq \emptyset\}$ are investigated for valid insertion positions (see line 5 & 6). All compatible containers must not contain a piece p', which is incompatible to p. Also every vertex $v \in \mathcal{V}^I$ of the container c is used as a relative insertion anchor. This means that the vertex v' of the piece's bounding box matches the vertex v of the container, with $v = v'$. In other

words the piece is inserted at a corner of the container. For example inserting a piece at vertex $v = 8$ of the container results in an insertion point, which is always relative to $v = 1$ of the piece, resulting in $e = (L^{\mathtt{C}}_{cx} - L^{\mathtt{P}}_{pox}, L^{\mathtt{C}}_{cy} - L^{\mathtt{P}}_{poy}, L^{\mathtt{C}}_{cz} - L^{\mathtt{P}}_{poz})$. While inserting a piece relative to $v = 1$ results in $e = (0, 0, 0)$. If one of this insertions is successful the next step is the normalization of the container. Hence, all pieces are pushed repeatedly towards the origin of the container along all axes until no piece can be pushed any further, as it is indicated by Figure 5.6 (see line 15). The detailed technique is described in detail in section A.2. Some extensions obviously need to be done to detect the collisions when pushing "Tetris"-pieces instead of cuboid-pieces. For that reason, collisions are detected on a component-level of the pieces. This algorithm also has the option of falling back to a technique only handling the bounding boxes of the pieces.

Algorithm 5.4: Push Insertion

Input: $\mathcal{C}, \mathcal{P}, \mathcal{O}_p$
Output: $\mathcal{S}^C_c, S^O_p, S^P_p$

```
1  foreach p ∈ P do
2      placed ← false
3      if TimeUp then
4          return
5      foreach o ∈ O_p \ O_p^f do
6          foreach c ∈ {c ∈ C | ∄p' ∈ S_c^C : G_p' ∈ G_p^f} do
7              foreach v ∈ V^I do
8                  if Σ_{p'∈S_c^C} V_p'^P + V_p^P ≤ V_c^C then
9                      e ← (L_cx^C − L_pox^P, L_cy^C − L_poy^P, L_cz^C − L_poz^P)
10                     if v ∈ {1,3,5,7} then e.x ← 0
11                     if v ∈ {1,2,5,6} then e.y ← 0
12                     if v ∈ {1,2,3,4} then e.z ← 0
13                     if InsertionCheck(S_c^C, S_p^O, S_p^P, c, e, p, o) then
14                         S_c^C ← S_c^C ∪ {p}, S_p^O ← o, S_p^P ← e
15                         Normalize(c, (x, y, z))
16                         placed ← true, Break search
```

The PI algorithm, like the EPI algorithm, is capable of inserting "Tetris"-pieces into a set of multiple different-sized containers, while respecting the previously discussed requirements. Here, too, an exception is the stability and alongside the load bearing, which bases on the stability. Both requirements cannot be guaranteed to hold. Again it is possible to fall back to a cuboid-based approach. Therefore, the method capable of handling "Tetris"-pieces is called PI-Tetris, while the cuboid based method is called PI.

5.5 Greedy Adaptive Search Procedure

This section describes the Greedy Adaptive Search Procedure (GASP) adapted and extended from the work by [BPT12]. The basic idea is to build a simple metaheuristic capable of improving the solutions built by the previously discussed constructive heuristics. To achieve this the latter are employed by the GASP procedure as generators of new solutions in each iteration. This relies on the fact that the order in which the pieces are inserted into the given containers greatly influences the solution quality (see [BPT12]). For every permutation of the piece-order a different solution is likely to be generated when using the discussed greedy constructive heuristics. When extending the approaches to handle 90° rotations, the same applies for the order of the orientations per piece. As a quick approximation of the order of magnitude the number of permutations of the order of pieces as well as the order of orientations per piece has to be taken into account. This would result in $|\mathcal{P}|! \cdot \prod_{p \in \mathcal{P}} |\mathcal{O}_p|$ possible solutions to evaluate. Therefore the GASP algorithm (see Algorithm 5.5) utilizes the concept of reordering this sets to improve the initial solution. The sequence used to generate this initial solution is simultaneously the first sequence to initialize GASP with. The order can be based on different heuristic concepts like sorting the pieces in descending order corresponding to their volume, thus beginning the insertion with the biggest pieces. The different concepts for this are introduced in subsection 5.1.4. Every piece is then assigned a score S_p^{Score} based on the initial order. After each iteration the score is modified and resorted to generate different solutions. If no improvement can be done during a given number of iterations the score is reset to the sequence of the best solution so far. This concept is extended by also shuffling the order of the orientations per piece and investigating them in parallel. So the parallelism of this approach will not result in a speed-up in terms of iterations, but can somewhat increase the diversification during the search.
The input data of the algorithm consists of an initial solution described by the allocation information $\mathcal{S}_c^C$, the orientations per piece S_p^O and the positions per piece S_p^P. Additionally the used volume S^U of this solution as well as the number of used containers S^N is given. The initial order of the pieces and orientations is given by $\mathcal{P}$ and $\mathcal{O}_p$. These are modified along the search process, while the given order of containers $\mathcal{C}$ stays fixed. $\mathcal{W}$ defines a set of indices used to mark the different worker-threads available for parallelization. Thus, $|\mathcal{W}|$ is the number of parallel workers available. At last the constructive heuristic is supplied, which is used in each step to generate the solutions to the different piece and orientation orders. The output of the algorithm is the best solution found during the search process, again defined by the allocation information, the used orientations and the positions of the pieces.
When executing the algorithm at first some iteration trackers need to be set. These are used during the search process to identify stagnation. Next the first score-initialization is done, assigning the score based on the order in which the pieces were initially inserted into the containers. The first piece is assigned $|\mathcal{P}'|$, a number based on the inserted pieces and for every following piece this number is decreased by one. Consequentially the main while-loop iterates until the time-limit is reached or the search

stagnates for too many iterations, i.e. $i - i^L \geq i^{SD}$ with the current iteration number i, the iteration number of the last improvement on the incumbent value i^L and the maximal number of iterations without an improvement i^{SD}. Next additional termination criteria are checked, which use the current container utilization to recognize no possible further improvement.
The main-loop begins with reordering the pieces by their current score values in descending order (see line 5). Additionally for each available working thread the orientations are shuffled, so that for one single piece order different solutions according to the orientation orders are evaluated (see line 8). This solution generation and evaluation is then done in parallel by the set of worker-threads (see line 9). After every worker finished its task the algorithm is synchronized again. Now all solutions generated are investigated for improvements on the best solution so far. If a solution of equal quality is found then it is used as the new best solution so far (see line 15). This is done to avoid getting stuck on one specific solution, by restarting the search with a different one which is only equal regarding its quality. If an improvement on the objective could be done, the markers for long distance reinitialization i^R and last improvement i^L are reset. Also the number of possible swaps s^P is increased by one, if it does not yet exceed the maximal number of swaps s^{max}. These swap values along with the modification amount denoted by m^I and m^{max} represent the main components influencing the score order (see Equation 5.1). At last the current best objective value S^U and the number of used containers S^N are updated.
The next step in the main-loop defines the update of the scores, which are used for ordering the pieces. If no score reinitialization is needed ($i - i^R < i^{RD}$), this update is done. At first the set of "well-packed" containers is build. Since constructive heuristics for Bin Packing are assumed to generate good packings for the first bins (see [PCT11]), the respective containers are simply the first half of the used containers. In other words all containers $c \in \mathcal{C}'$ with an index smaller than $\left\lfloor \frac{|\{c \in \mathcal{C} | \mathcal{S}_c^C \neq \emptyset\}|}{2} \right\rfloor + 1$ are considered "well-packed". By that empty containers $\{c \in \mathcal{C} \mid \mathcal{S}_c^C = \emptyset\}$ are excluded. For each piece contained in one of these containers the score S_p^{Score} is decreased by a factor based on the current values of the possible swaps s^P, the maximal swaps s^{max} and the initial m^I and maximum m^{max} percentage of modification (see line 19). Analogously this is increased for a piece not packed inside one of these containers depending on the very same values (see line 21). This results in the following score-update-rule (see Equation 5.1) proposed by [PCT11]. The values of m^I and m^{max} thereby define a fractional change of the score depending on its current value. Since m^{max} increases with every long-term reinitialization this term lessens the effect of the score modification over time. Hence, a greater focus lies on exploration, if the search stagnates often. The second influence is given by s^P and s^{max} depicting the possible

swaps respectively maximal swaps. At every update of the best solution so far s^P is increased by one, hence increasing the change done to the score-value.

$$S_p^{Score} = \begin{cases} S_p^{Score} \cdot \left(1 - \left(\frac{m^I}{m^{max}} \cdot \left(s^{max} - s^P\right)\right)\right) & , \text{ if } p \in \bigcup\limits_{c \in \mathcal{C}'} \mathcal{S}_c^C \\ S_p^{Score} \cdot \left(1 + \left(\frac{m^I}{m^{max}} \cdot \left(s^{max} - s^P\right)\right)\right) & , \text{ otherwise} \end{cases} \tag{5.1}$$

Additionally the score is modified based on a value called "salt" (r^S) (see line 24 & 26). The idea behind this value is to introduce a flexible randomization influence. With a "salt"-value of $r^S = 0$ no randomization of the order is conducted, thus the order of the pieces is completely subject to the score-update-rule. On the other hand, with a $r^S > 0$, the order also underlies some random influence. With increasing r^S a position change of a piece is more likely. This modulation of the score is applied as an increase or decrease according to a "coin-flip".
If no improvement on the incumbent objective value could be done for i^{RD} iterations the score is reset. This means that for all allocated pieces $\mathcal{P}'$ of the best solution so far the scores are reinitialized according to the order in which they were inserted into the containers (see line 29). This matches the first initialization of the score previously discussed. In conclusion the GASP procedure initially proposed by [PCT11] was extended to handle all three constructive algorithms of this work, namely EPI, SD and PI. The goal of this adaption is to supply a basic improvement metaheuristic capable of exploiting the different techniques described before. The first extension made to the algorithm itself is a "salt-value", which was applied to give control of a simple random influence on the search. This is considered to be helpful for small problem instances with only one container for which no "well-packed" classification can be determined, respectively only one container is "well-packed". Furthermore different orders of the orientations are considered when constructing a solution by generating a random order per worker-thread. Hence, parallelization is applied to give some sort of diversification of the search through the orientation order.

Algorithm 5.5: Greedy Adaptive Search Procedure

Input: $\mathcal{S}_c^C, S_p^O, S_p^P, S^U, S^N, \mathcal{C}, \mathcal{P}, \mathcal{O}_p, \mathcal{W}$, `ConstructionHeuristic`
Output: $\mathcal{S}_c^C, S_p^O, S_p^P$

$i^R \leftarrow 0, i^L \leftarrow 0, i \leftarrow 0, \mathcal{P}' \leftarrow \bigcup_{c \in \mathcal{C}} \mathcal{S}_c^C, n \leftarrow |\mathcal{P}'|$
foreach $p \in \mathcal{P}'$ **do**
  $S_p^{Score} \leftarrow s, n \leftarrow n - 1$
while $i - i^L < i^{SD} \wedge \neg$ *TimeUp* $\wedge$ *FurtherImprovementPossible* **do**
  $\mathcal{P} \leftarrow \mathcal{P}$.`OrderBy`$(S_p^{Score})$
  foreach $w \in \mathcal{W}$ **parallel do**
    foreach $p \in \mathcal{P}$ **do**
      $\mathcal{O}_{wp} \leftarrow$ `Shuffle`$(\mathcal{O}_p)$
    $(\mathcal{S}_{wc}^C, S_{wp}^O, S_{wp}^P, S_w^U, S_w^N) \leftarrow$ `ConstructionHeuristic`$(\mathcal{C}, \mathcal{P}, \mathcal{O}_{wp})$
  foreach $w \in \mathcal{W}$ **do**
    if $S^U \leq S_w^U \vee S^U = S_w^U \wedge S_w^N < S^N$ **then**
      if $S^U \neq S_w^U \vee S^N \neq S_w^N$ **then**
        $i^L \leftarrow i, i^R \leftarrow i, s^P \leftarrow \min(s^P + 1, s^{max})$
        $S^U \leftarrow S_w^U, S^N \leftarrow S_w^N$
      $(\mathcal{S}_c^C, S_p^O, S_p^P, S^U, S^N) \leftarrow (\mathcal{S}_{wc}^C, S_{wp}^O, S_{wp}^P, S_w^U, S_w^N)$
  if $i - i^R < i^{RD}$ **then**
    $\mathcal{C}' \leftarrow \left\{ c \in \mathcal{C} \mid \mathcal{S}_c^C \neq \emptyset, c \in \left\{ 1.. \left\lfloor \frac{|\{c \in \mathcal{C} | \mathcal{S}_c^C \neq \emptyset\}|}{2} \right\rfloor \right\} \right\}$
    foreach $p \in \bigcup_{c \in \mathcal{C}'} \mathcal{S}_c^C$ **do**
      $S_p^{Score} = S_p^{Score} \cdot \left(1 - \left(\frac{m^I}{m^{max}} \cdot \left(s^{max} - s^P\right)\right)\right)$
    foreach $p \in \bigcup_{c \in \mathcal{C} \setminus \mathcal{C}'} \mathcal{S}_c^C$ **do**
      $S_p^{Score} = S_p^{Score} \cdot \left(1 + \left(\frac{m^I}{m^{max}} \cdot \left(s^{max} - s^P\right)\right)\right)$
    foreach $p \in \mathcal{P}$ **do**
      if `Random()` ≤ 0.5 **then**
        $S_p^{Score} = S_p^{Score} - r^S$
      else
        $S_p^{Score} = S_p^{Score} + r^S$
  else
    $\mathcal{P}' \leftarrow \bigcup_{c \in \mathcal{C}} \mathcal{S}_c^C, n \leftarrow |\mathcal{P}'|$
    foreach $p \in \mathcal{P}'$ **do** $S_p^{Score} \leftarrow s, n \leftarrow n - 1$
    $s^P \leftarrow 1, m^{max} = m^{max} + 1, i^R \leftarrow i$
  $i \leftarrow i + 1$

5.6 Overview of problem characteristics

This section compares the proposed methods for their capability of handling the requirements issued in chapter 2. The GASP metaheuristic is not discussed here, because the requirements handled by this particular procedure emerge from the used constructive heuristic. So, only the constructive heuristics EPI, SD and PI are investigated here.
As seen in Table 5.2 all of the heuristics handle the same set of requirements, namely multiple containers (MC), different sized containers (DC), 90° rotations (R), forbidden orientations (FO) and the compatibility of pieces (C). The not considered requirements are the load stability (LS) and the load bearing (LB). The first one is in a way considered, because all methods build packings from the bottom-up. But a stable solution cannot be guaranteed by that. So, those two aspects are not part of this work. On the bright side an integration of "Tetris"-pieces was successfully done for the EPI and PI heuristics. Only the SD technique is limited to cuboid pieces, because an integration of "Tetris"-pieces is not compatible with the push-out procedure.

Table 5.2: Matching the requirements considered by the heuristics

Method	Shape	MC	DC	R	LS	LB	FO	C
EPI(-Tetris)	"Tetris"	✓	✓	✓	✗	✗	✓	✓
SD	Cuboid	✓	✓	✓	✗	✗	✓	✓
PI(-Tetris)	"Tetris"	✓	✓	✓	✗	✗	✓	✓

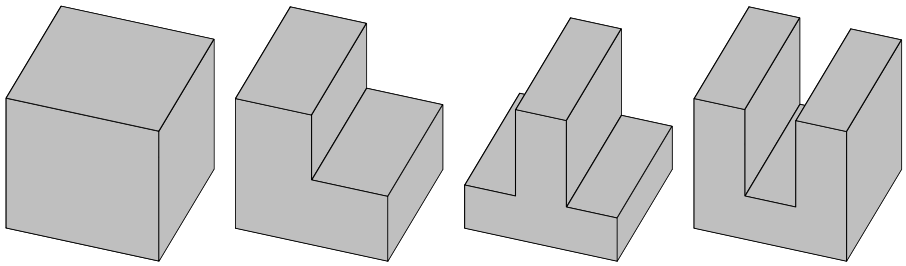
6 Computational results

This chapter discusses the computational results of this thesis. At first the origins of the test instance sets and the related instance generator is introduced (see section 6.1). Besides that, basic parameter tuning is conducted for the heuristic approaches used in this work. This is done for the different constructive algorithms embedded in the proposed metaheuristic technique (see section 6.2). After specifying these preliminaries the main evaluation is done by comparing the different methods in section 6.3. Additionally two more investigations are done on the impact of the stability and load bearing requirements and the effect of the "Tetris" extensions (see section 6.4 and section 6.5).
The implementation is done with C# in the Microsoft .Net Framework (see [Mic13b]). All methods are executed with a time-limit of 5 minutes. This is the main termination criteria, if no other previously discussed criterion is met first. The tests of the heuristic methods are done with 7 repetitions to lower the influence of randomness. The execution environment is composed by Intel Core 2 Quad 2.83 GHz Processors with 8 GB main memory running Microsoft Windows 7 Professional (see [Mic13a]). The solvers used to evaluate the model formulations are Gurobi 5.6.0 and CPLEX 12.5 (see [Gur13] and [IBM13]).

6.1 Instance generator

This section introduces the instance generator used to supply the set of instances for the evaluation. Since no benchmark set of the literature contains "Tetris"-pieces a generator capable of building these and also simple cuboid instances is built.
The generator is subdivided in two parts: the generation of containers and the one of pieces. For both the number to generate as well as the minimal and maximal side length must be supplied. In the case of generating "Tetris"-pieces, the side length is defined as the one of the resulting bounding box of the piece. The actual side length of a piece regarding a dimension d is then randomly chosen from a uniform distribution between these lower and upper bounds. Note that the value of this length is not set to an actual unit of length. Although, the transfer into realistic units is possible. The shape types supported by the generator are shown in Figure 6.1. One of these is chosen randomly corresponding to given weights when generating a single piece. When building cuboid instances this choice is limited to the first type. Additionally a minimal and maximal count of equal pieces is given by the parameters. This is done to emulate the fact that in reality often a number of equal pieces are shipped. For that purpose, every piece is cloned several times, where the actual number is

chosen randomly from an uniform distribution across an integral domain between the minimal and maximal count of equal pieces. At last a rounding of the side length may optionally be applied to get more homogeneous lengths for an instance.



(a) Cuboid (b) “L”-shape (c) “T”-shape (d) “U”-shape

Figure 6.1: The basic shapes of the instance generator

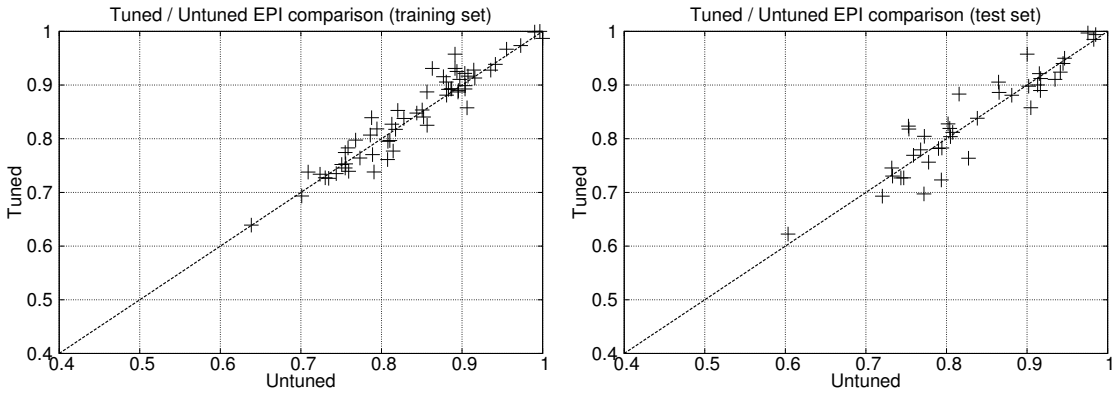
Using this generator two sets of instances are produced which get used in the evaluation. Both sets contain 100 instances of different sizes regarding piece and container count. An overview of some characteristics of the instance sets is given by Table 6.1. All values are averages across all instances of the particular set. For every instance the volume of the pieces is greater than the one of the containers. Hence, this corresponds to the classic knapsack problem discussed before. The cuboid instance set is limited to pieces of the first shape shown in Figure 6.1. Besides that, the instances of this set generally contain more pieces and containers. The “Tetris” instance set contains pieces of all four shapes, but in different proportions per instance. Only a few instances of this set contain cuboid pieces only. The materials characteristic seen in the overview depicts the number of distinct materials per instance. These give rise to incompatibilities that have to be avoided when generating feasible solutions.

Table 6.1: Characteristics of the instance set (average values)

	Characteristic	Cuboid set	“Tetris” set
Container	Volume	8901.30	2720.42
	Count	8.92	3.04
	Side length	10.14	9.74
Piece	Volume	10243.98	3209.01
	Count	218.00	52.92
	Side length	3.57	4.26
	Materials	2.96	2.54

6.2 Parameter-tuning

Due to the presence of certain parameters in the different heuristic approaches a parameter tuning is conducted. For that, a tuning algorithm is used to solve the algorithm configuration problem. The system used is the "Sequential Model-based Algorithm Configuration" (SMAC) proposed by [HHLB11]. This system utilizes random forests, a standard machine learning technique for regression and classification. As a result of the process a tuned algorithm configuration is obtained.
The parameter-tuning is conducted for the EPI, EPI-Tetris, SD, PI and PI-Tetris methods. While the basic algorithms are tuned on the cuboid instance set the "Tetris"-variants are tuned on the tetris set. For the set of parameters being subject to the tuning by SMAC see section C.1. This is done to avoid "guessing" or brute-forcing a suitable configuration for the different methods. Overall, the tuning of the parameters did not obtain significantly improved configurations. As depicted by the graph in Figure 6.2 for the EPI algorithm, the tuning had a slight positive and also negative impact depending on the particular instances. This can be seen by comparing the result of the not tuned configuration (first guess) depicted by the x-axis with the tuned configuration depicted by the y-axis. The respective axes identify the obtained evaluation values (relative volume utilization) by the two configurations. For both the training set (a) consisting of 60 instances and the test set (b) consisting of 40 instances only a very slight positive trend is seen. Thus, the trend is to small to verify a positive effect of the tuning. This might be due to the very heterogeneous instances requiring different algorithm settings. Hence, in future it may be useful to integrate instance features to find better configurations depending on those. The result would be an automatic algorithm configuration routine before solving the respective instance exploiting the results of the tuning. However, the tuned configurations are used for the rest of the evaluation done here. Additional results of the other algorithms are shown in section C.2.



(a) Training (b) Test

Figure 6.2: Tuning results for EPI

6.3 Method comparison

This section introduces the main results of this work. A comparison of all implemented techniques is done across the previously mentioned instance sets (cuboid and tetris). The main evaluation value is given by the relative volume utilization, i.e. the volume of all packed pieces divided by the maximal available volume of all containers. Hence, the maximal possible utilization is 100 %. By this the the instances are more comparable even though the information about the absolute utilization and is lost. In Table 6.2 an overview of these results is given. The heuristic methods (EPI, EPI-Tetris, SD, PI and PI-Tetris) are shown alongside with the cuboid-based and "Tetris"-based model formulations. The commercial solvers Gurobi 5.6.0 and CPLEX 12.5 represent the exact algorithms used to solve the models (see [Gur13] and [IBM13]). The relative volume utilization (V) is given by the arithmetic mean across all 100 instances per set. Additionally, the overall value is outlined. The standard deviation of V is also given by column SD(V). This results from the 7 repetitions conducted to reduce the noise for the randomized heuristics. In case of the solvers no repetitions are done, because these are deterministic, resulting in the standard deviation of 0 %. The detailed results of the comparison are supplied in section D.1 and section D.2.

Table 6.2: Overview of the computational results (after five minutes)

Method	Cuboid set		Tetris set		Overall	
	V	SD(V)	V	SD(V)	V	SD(V)
EPI	84.17%	0.18%	68.55%	0.80%	76.36%	0.49%
EPI-Tetris	-	-	73.19%	0.72%	78.98%	0.55%
SD	88.49%	0.58%	72.77%	0.49%	80.63%	0.54%
PI	81.67%	0.51%	65.22%	1.35%	71.39%	0.93%
PI-Tetris	-	-	71.53%	1.03%	76.60%	0.74%
Cuboid (CPLEX)	23.35%	0.00%	53.71%	0.00%	38.53%	0.00%
Cuboid (Gurobi)	35.63%	0.00%	65.80%	0.00%	50.72%	0.00%
Tetris (CPLEX)	12.50%	0.00%	28.25%	0.00%	20.37%	0.00%
Tetris (Gurobi)	19.14%	0.00%	31.87%	0.00%	25.50%	0.00%

As indicated by the overview, Gurobi proved to be more successful on the given instances, and therefore is used as the main competitor for the heuristics in further comparisons. Apart from that both solvers tend to have poor results overall. Table 6.2 also features a comparison of the two variants, tetris and cuboid, of the two heuristics EPI and PI on the tetris set. This is discussed in more detail in section 6.5. When considering the average relative volume utilization the results indicate the best approach for the cuboid instances is the SD technique, while the best one for the tetris

instances is the EPI-Tetris technique. Even though SD does not consider "Tetris"-pieces, it is still able to provide solutions of high quality. Thus, a future integration of "Tetris"-pieces into this approach might be interesting, but as mentioned before this is not trivially done.

6.3.1 Cuboid set

The first detailed comparison is done on the cuboid set. For that only the EPI, SD and PI heuristics as well as both model-formulations are considered. An overview of the performance of the different techniques across all instances of the set is given by the performance profile in Figure 6.3. For the sake of clarity, the x-axis depicts the relative volume utilization while the y-axis indicates the number of solved instances with a volume greater than x. This way comparable graphs are preserved which are monotonically nondecreasing. Thus, for a given relative volume utilization x each graph depicts the number of solved instances with a volume / evaluation value greater than x. Hence, the area below each graph represents the sum of the achieved evaluation values across all instances when summing the corresponding x-value over $y \in [1, \ldots, 100]$.

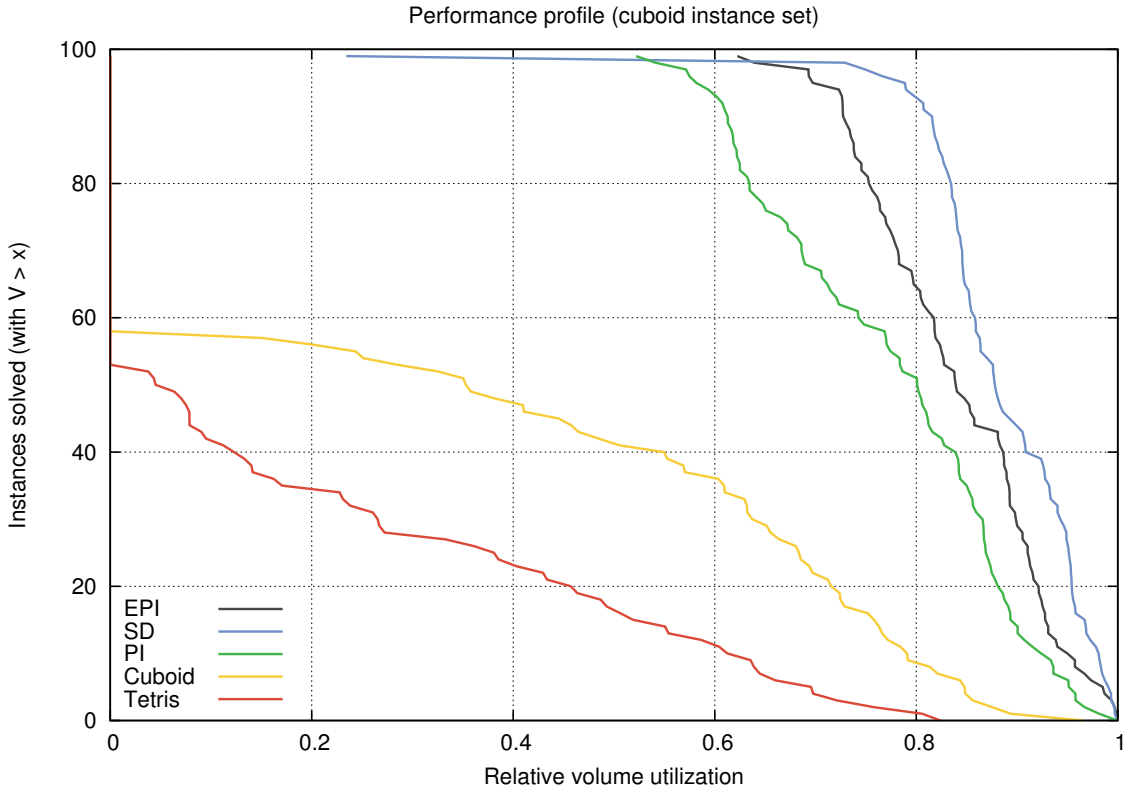


Figure 6.3: Comparison of the different methods on cuboid-instances

The most obvious result of this comparison is the strong dominance of the heuristic methods across almost all instances. The solver (Gurobi) is confronted with massive problems when solving some of the instances so that no solution is returned at all. In such a case the empty solution with V = 0 is assumed, because it is trivially given by not allocating any pieces. This is mostly due to the size of the particular instances resulting in too complex and also too large model representations which do not fit the memory of the machine (8 GB). However, the cuboid model formulation is capable of solving some of the instances with acceptable quality, the "Tetris"-formulation instead is not. This is probably due to the higher count of constraints and variables of the formulation.
On the other hand the heuristics perform quite well on the instances and are capable of achieving high quality solutions even for the more difficult ones on the bottom of the graph. These tend to be larger and more heterogeneous. The SD approach at that is the most successful variant capable of attaining a relative volume utilization of more than 80 % for most instances. This is followed closely by the EPI approach. The PI method attains the third rank, also with reasonable quality.

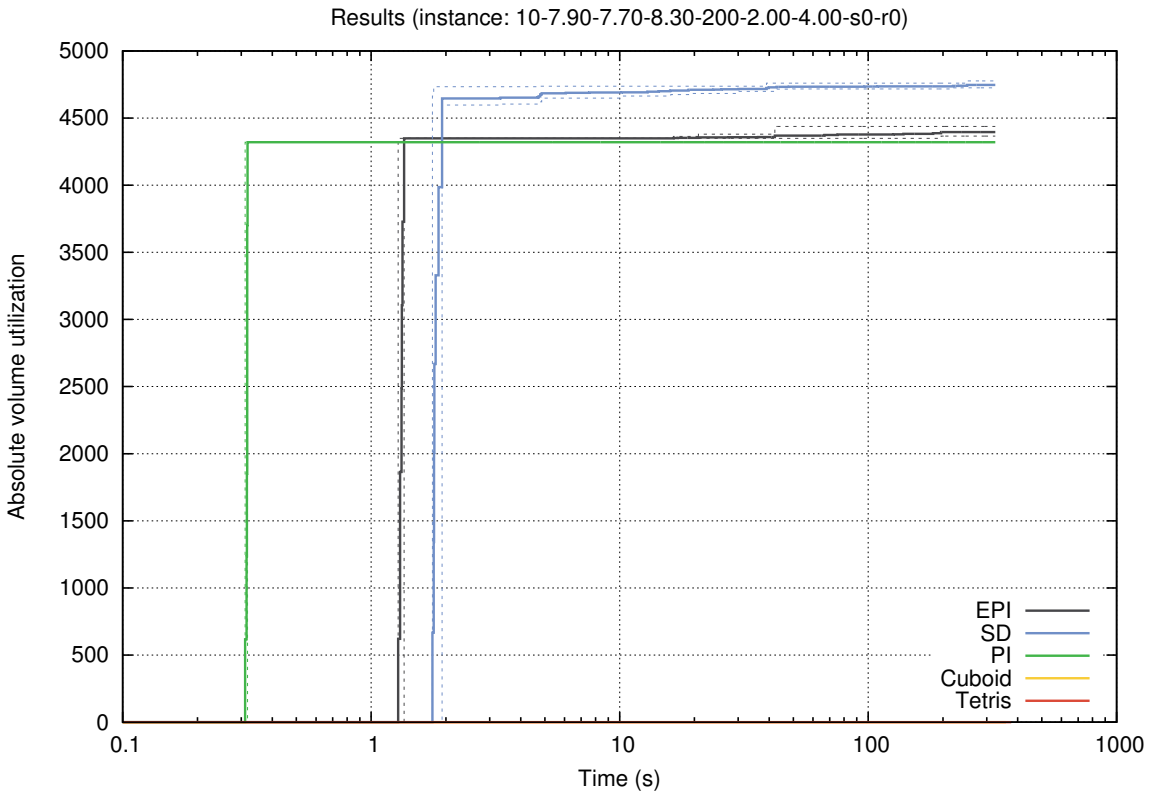


Figure 6.4: Comparison of the different methods for one cuboid-instance

At last a closer look at the development of the evaluation value over time is done for one instance exemplarily. In Figure 6.4 the solution development is depicted for a medium sized instance with 10 containers and 200 pieces with side lengths from 2 to

4. Once again the five approaches from above are compared. The diagram depicts the solution quality at the y-axis over the time shown by the x-axis in logarithmic scaling. In this case the average absolute volume utilization is drawn. The dashed lines around the heuristic graphs depict the spread given by the minimal and maximal value achieved at the respective time across all repetitions.
Both model formulations cannot be solved fast enough by the solver to supply a first solution different from the trivial one. Thus, these two curves constantly lie on the x-axis. In contrast, the heuristics almost instantly reach high solution quality and are able to slightly improve it. This is a typical result across the most instances. The SD heuristic attains the best solution for this instance, followed by very similar results for EPI and PI.

6.3.2 Tetris set

The next set of instances investigated is the tetris set. In this case only the "Tetris" variants of EPI and PI are investigated. The rest of the methods stays the same. Similar to the former case, an overview is given by a simplified performance profile (see Figure 6.5). The x-axis at this point also depicts the relative volume utilization while the number of solved instances with a utilization higher than x is indicated by the y-axis. Note that in this case the volume is given by the summed volumes of each pieces' components.

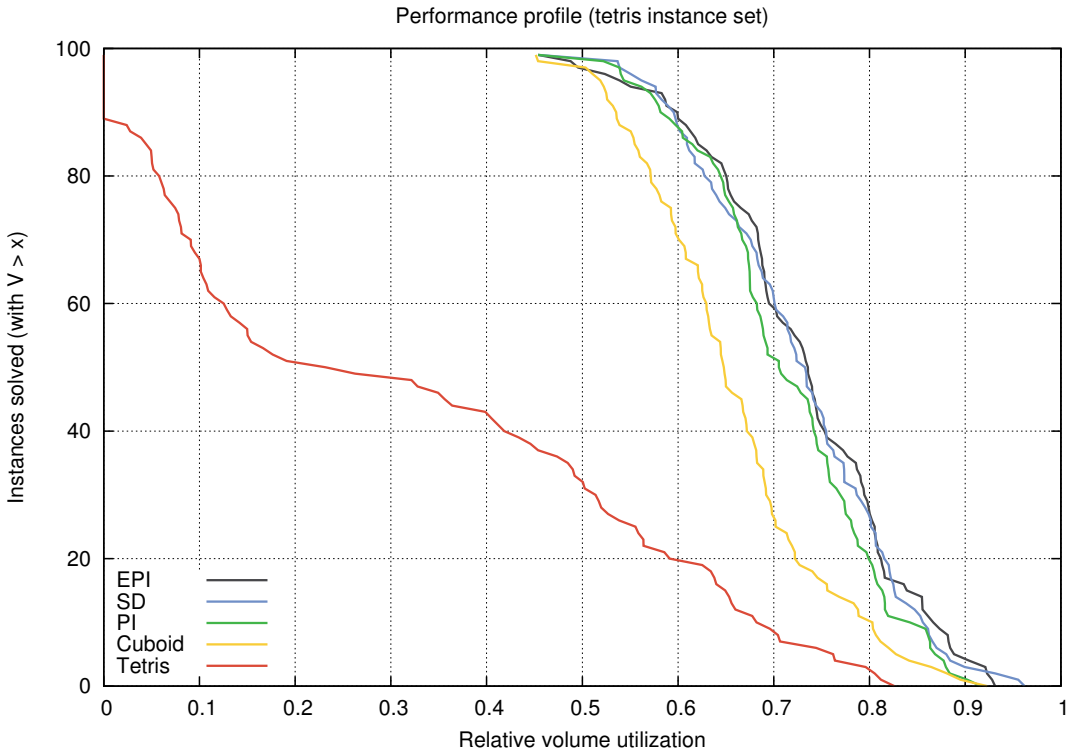


Figure 6.5: Comparison of the different methods on "Tetris"-instances

The first result depicted here is the reasonable performance of the solver with the cuboid formulation. This is probably due to the fact that all instances of the tetris set are significantly smaller regarding container and piece count. Since, the model formulation does not consider "Tetris"-pieces these instances yield in small cuboid instances. The SD technique also performs quite well on the set, because not all of the instances contain many "Tetris"-pieces or even none at all. Also, the waste of volume due to notches is not necessarily high and not all "Tetris"-pieces fit each other in a space-saving way. However, all heuristic methods show a reasonable performance.

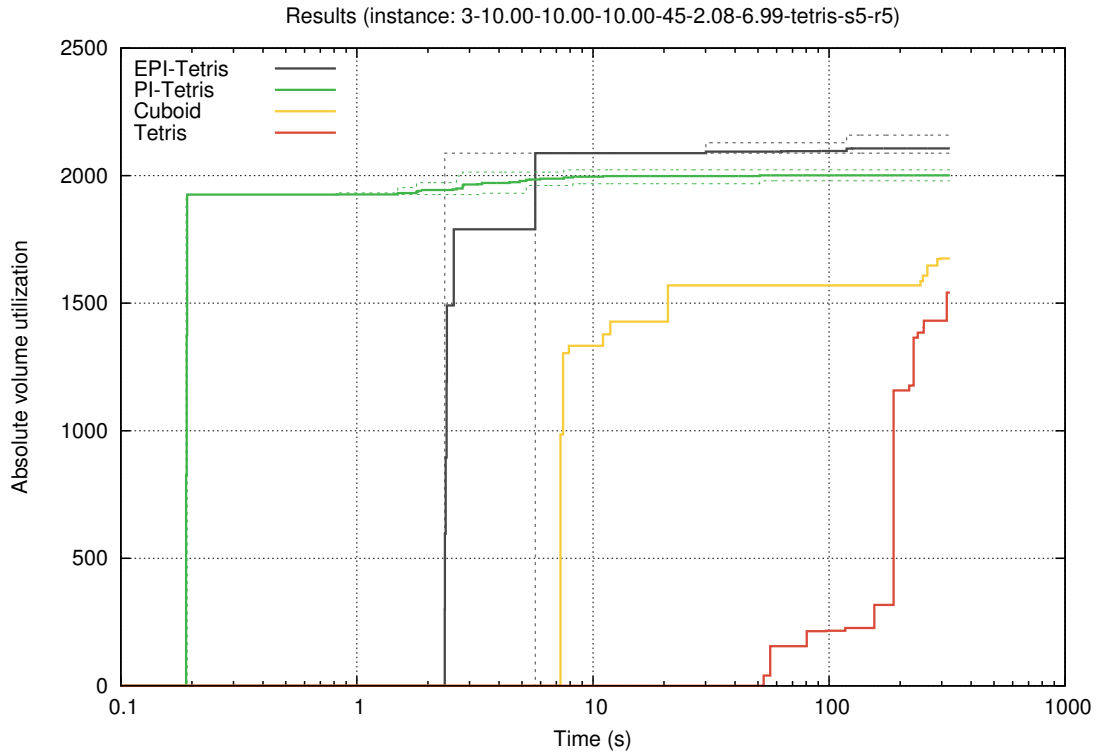
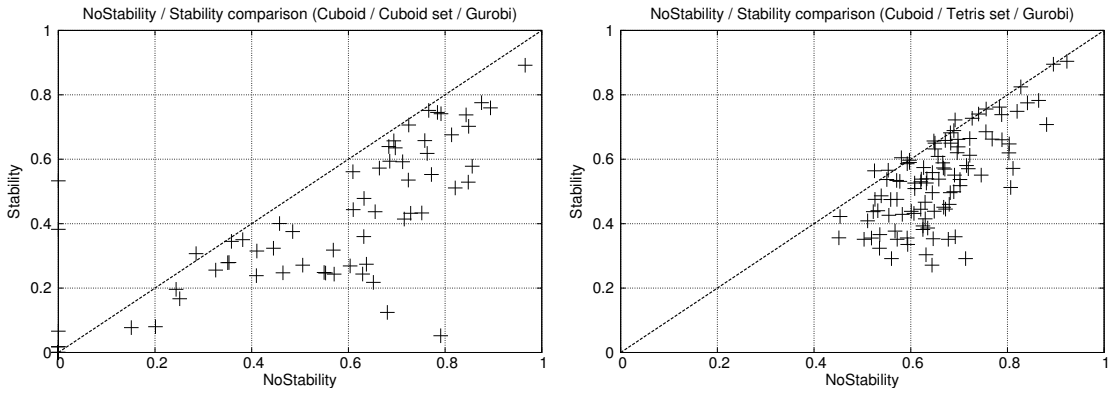


Figure 6.6: Comparison of the different methods for one "Tetris"-instance

In Figure 6.6 the development of the solution for one instance is shown. Similar to the results of the cuboid set the solver is dominated by the heuristic methods, but the cuboid formulation approach is now capable of achieving solutions in reasonable time. This is due to the quite smaller instance size of 3 containers with only 45 pieces to insert. The "Tetris"-formulation starts off much later but terminates at the time-limit with a solution almost as good as the one of the cuboid approach. Again, the two "Tetris"-approaches achieve a solution of high quality very fast and are able to improve it slightly afterwards. The PI-Tetris technique is the fastest method in reaching its first high quality solution, probably because of the simplicity of the algorithm. On the downside the algorithm terminates with a solution of less quality when compared to EPI.

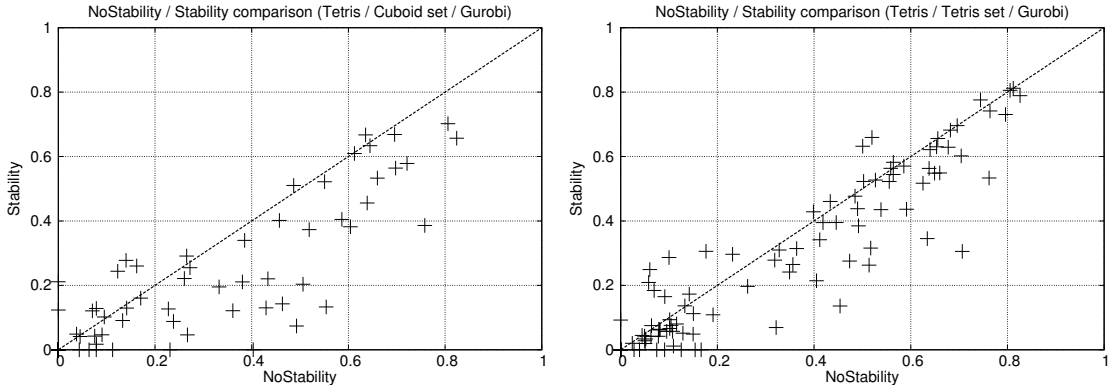
6.4 Stability influence

This section investigates the influence of stability and load bearing requirements on the performance of the solvers. In contrast to the tests conducted before, the particular constraints are applied for all instances of both sets. The results are compared to the ones of the formulations without the respect of the stability and load bearing requirements for both models. Graphs similar to the ones of section 6.5 are used to summarize the results. Thereby, the x-axis depicts the relative volume utilization for the model with deactivated constraints (NoStability) while the y-axis depicts the one with activated constraints (Stability). This comparison is investigated using Gurobi 5.6.0 as the solver (see [Gur13]).
Figure 6.7 shows the results for the cuboid instance set on the left (a) and the tetris set on the right (b). For both sets the performance of the solver with activated constraints is critically worse. This especially applies for the cuboid set, because the count of constraints increases even more, which is also assumed to be the cause of the solver performing bad on larger instances. The instances not solved by both variants are accumulated at the origin of the coordinate system. Interestingly, for certain instances only the version with activated requirements is able to generate a solution at all. This might base on the fact that the additional cuts by the constraints lead to earlier integer solutions. Note that the tetris set is quiet smaller in terms of container and piece counts, thus, the deviation in the performance is generally lower. The given condition that the stability requirement does not fit the "Tetris"-pieces in the cuboid formulation, since only their bounding box is considered, is ignored here. Similar results emerge from the comparison conducted on the "Tetris"-formulation (see Figure 6.8). Despite the fact that the formulation performs worse overall, the influence due to considering the additional constraints is negative. Only some instances represent exceptions to this result.
In conclusion the performance of the solvers is significantly weakened when respecting stability and load bearing requirements. For more realistic instance sizes the current approach would be impractical. On the other hand most of the packings generated by the heuristic approaches already stabilize themselves by their density. Of course, this needs to be extended to guarantee a feasible solution in terms of both requirements. However, for certain small instances feasible packings plans of reasonable quality can be obtained.



(a) Cuboid set (b) Tetris set

Figure 6.7: Stability vs. NoStability for the cuboid formulation



(a) Cuboid set (b) Tetris set

Figure 6.8: Stability vs. NoStability for the tetris formulation

6.5 "Tetris"-shape influence

This section compares the direct impact of handling "Tetris"-pieces on the achieved volume utilization for the two heuristics EPI and PI, which are capable of it. The goal of the investigation is to analyze whether the additional handling has a positive effect when solving different heterogeneous instances at all. EPI and PI are selected for a direct comparison to investigate this effect, because they are more independent from influences given by different techniques. Hence, both methods only differ from their "Tetris" versions by small adjustments. For each algorithm the evaluation is done on the complete tetris instance set. Note that some of these instances only contain few "Tetris"-pieces while others have multiple different shaped ones assigned to them. As in all cases, each run of the heuristics is done with 7 repetitions to reduce the influence of random noise.

The results of the first comparison for the EPI-technique are summarized in Figure 6.9. The x-axis of the graph depicts the relative volume utilization for the cuboid

version while the y-axis depicts the one for the "Tetris"-variant. Every instance is represented by a point in this graph. Thus, all points above the diagonal curve represents an improvement due to the consideration of "Tetris"-shapes. The distance to this diagonal additionally represents the magnitude of the improvement. As a result the usage of the specialized algorithm proves to be successful for the mixed set of "Tetris" instances. Only 7 instances show slightly worse results and just one is much worse. These results may be explained by the fact that the "Tetris"-variant needs more time when investigating possible insertion points and also bad insertion choices at the beginning may lead to possible overlaps when inserting the last pieces. In the particular case of the one worst instance exclusively cuboid pieces which are very large compared to the containers are given. Therefore, bad choices immediately result in a bad packing in terms of volume utilization. Note that this is also one of the limited instances the solvers perform quite well on.

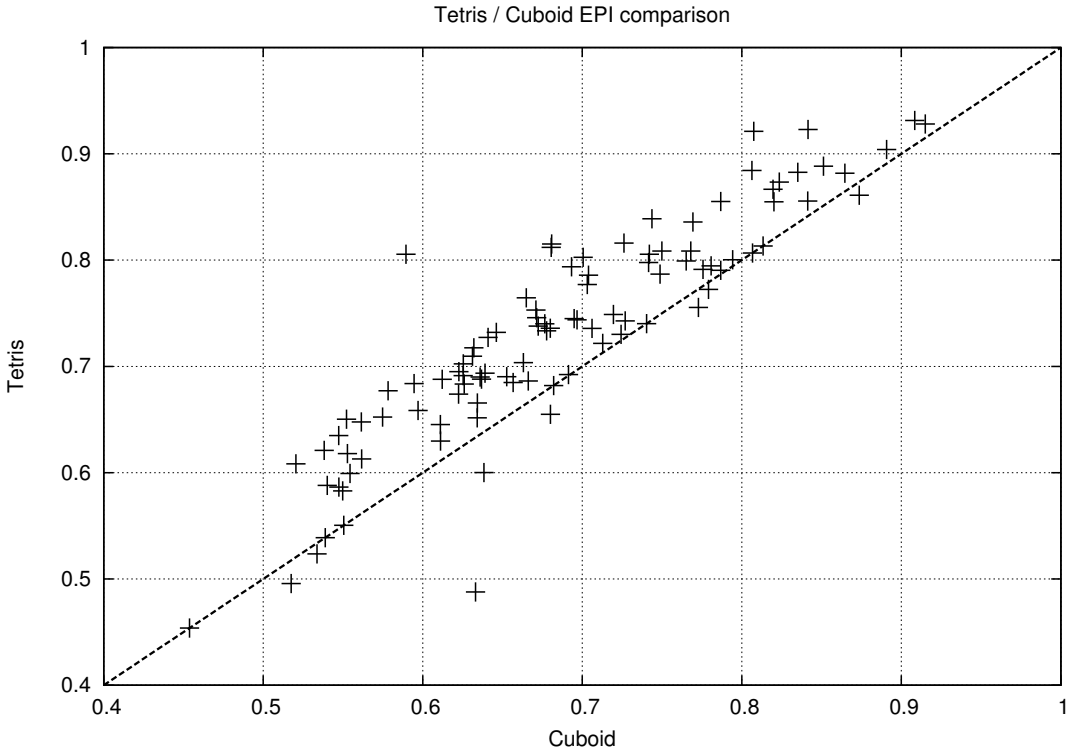


Figure 6.9: "Tetris" vs. cuboid for the EPI heuristic

Next the same effect is investigated for the PI technique with the graph shown in Figure 6.10 summarizing the results. Similar to the one for the EPI method, the graph depicts the relative volume utilization of each instance for the PI variant on the x-axis and the PI-Tetris one on the y-axis. Despite the fact that the overall results of this technique are worse than the ones of the EPI variants, the same improvement due to the exploitation of the "Tetris"-shapes can be seen. This time the "Tetris"-variant performed worse on 9 instances of the set, but on the other hand it is able to improve even more for some instances.

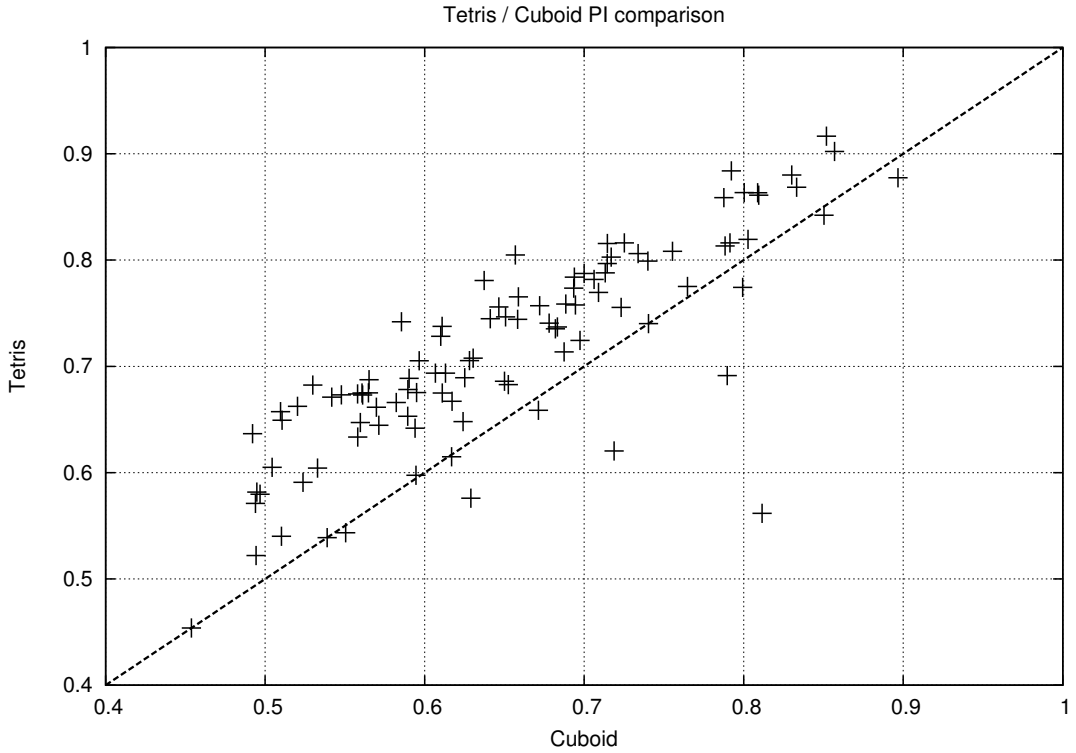


Figure 6.10: "Tetris" vs. cuboid for the PI heuristic

Overall the effect of exploiting the "Tetris"-shapes proves to be useful in terms of volume utilization. On average, both methods are able to improve their results by that. Only a few exceptions of this observation are ascertained.

7 Conclusion and outlook

This chapter concludes the work done in this thesis. After a short introduction an insight of the problem and the field of air cargo was given. Evidently, the discussed planning problem is part of a crucial process in its application and might become a bottle-neck in certain situations. Thus, an efficient decision support system (DSS) capable of generating feasible packing plans is needed. Since the basic problem can be reduced to three-dimensional bin packing or even bin packing and the knapsack problem in its basic forms, a summary of the state-of-the-art literature in the particular field was done and missing requirements were revealed. To overcome these issue, this thesis introduced two extended model formulations (Cuboid and Tetris) integrating the discussed requirements. Both are extensions of the most recent work found in the literature. Furthermore, variations of three heuristic algorithms (EPI, SD and GASP) were done to handle most of the requirements. Along with these, a new constructive technique (PI) is also proposed. All implemented methods are evaluated on a cuboid and a tetris instance set with 100 instances each. As a result both solvers (Gurobi and CPLEX) performed weak compared to the heuristics. Only for very small instances a reasonable solution quality is obtained by them. As opposed to this, the heuristic approaches are able to generate high quality solutions in a very short time-horizon even for very large instances. The evaluation also states that it is advantageous to exploit the exact form of a piece when dealing with "Tetris"-shapes, instead of only respecting the bounding box. Using the respective extended algorithms improved the solution quality even more. On the other hand, respecting the requirements of stability and load bearing cannot easily be integrated within the heuristic approaches. When integrating these into the model formulations the generation of high quality solutions becomes even harder for the solvers. The solution-time and also quality is completely impracticable for real applications and instance sizes. At this point a future integration of these requirements within the heuristics becomes more interesting. Overall, a successful adaption of the model formulations and heuristic methods was done. The resulting prototype of the optimization system can be seen in Appendix E.

In future research a more exhaustive investigation on helpful primal influences when inserting "Tetris"-pieces should be done. Many issues like the order in which pieces are inserted as well as the order of the orientations may be investigated further. E.g., very similar pieces like two "L-shapes" may be inserted consecutively to push their bounding boxes into each others. Hence, a very dense packing of these two would be the result. This may also be done during a preprocessing phase in order to achieve "meta-pieces" like the ones depicted in Figure 7.1 and Figure 7.2. Such a preprocessing phase might be designed as a learning system recognizing similar

shapes and attaching them to fixed sets of pieces like shown by the figures. Very efficient packings then could be stored for future fixations. Despite the risk of cutting off high quality solutions, the overall search will potentially speed-up due to a lesser complexity. During this preprocessing all tuples of "Tetris"-pieces could also be taken into consideration in order to minimize the bounding box of them. Thereby, very dense combinations could be identified which then would be inserted tied to each other. Preprocessing techniques would potentially reduce the complexity for both the model formulations and the heuristics. For the metaheuristic an integration of the orientation order with the score-based piece order may help to better steer the search process. Both components jointly define the solutions generated by the constructive algorithms. Thus, an integration of the components directly encodes one solution and consequently its evaluation value. This might also be used as a genome of a genetic algorithm.
At last an evaluation of the proposed techniques on real problem instances arising in the field of air cargo needs to be done. Unfortunately, no such instances were available for an evaluation during this work.

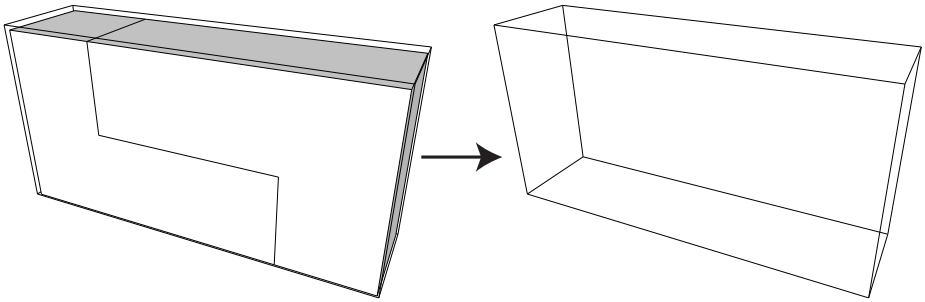


Figure 7.1: Example of preprocessing opportunities with "L"-shapes

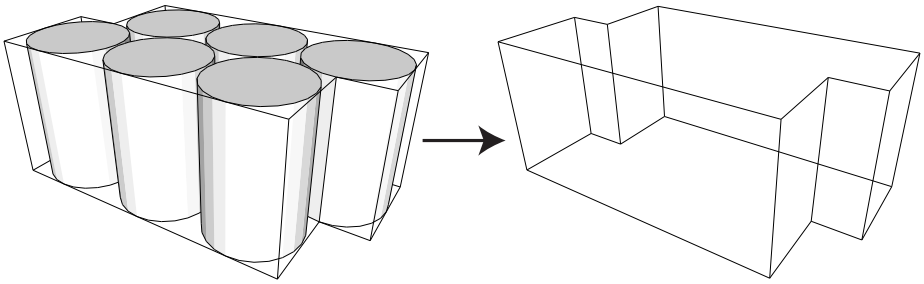


Figure 7.2: Example of preprocessing opportunities with barrels

Bibliography

[ABK11] S. D. Allen, E. K. Burke, and G. Kendall. A hybrid placement strategy for the three-dimensional strip packing problem. *European Journal of Operational Research*, 209(3):219–227, 2011.

[ABM12] S. D. Allen, E. K. Burke, and J. Mareček. A space-indexed formulation of packing boxes into a larger box. *Operations Research Letters*, 40(1):20–24, 2012.

[AIK+08] D. Arnold, H. Isermann, A. Kuhn, H. Tempelmeier, and K. Furmans. *Handbuch Logistik*. VDI-Buch. Springer Berlin Heidelberg, 2008.

[All11] S. D. Allen. *Algorithms and data structures for three-dimensional packing*. PhD thesis, University of Nottingham, 2011.

[AVPT13] R. Alvarez-Valdes, F. Parreño, and J.M. Tamarit. A grasp/path relinking algorithm for two- and three-dimensional multiple bin-size bin packing problems. *Computers & Operations Research*, 40(12):3081–3090, 2013.

[AW13] T. Achterberg and R. Wunderling. Mixed integer programming: Analyzing 12 years of progress. In M. Jünger and G. Reinelt, editors, *Facets of Combinatorial Optimization*, pages 449–481. Springer Berlin Heidelberg, 2013.

[BCR80] B. Baker, E. Coffman, Jr., and R. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980.

[BHI+07] N. Bansal, X. Han, K. Iwama, M. Sviridenko, and G. Zhang. Harmonic algorithm for 3-dimensional strip packing problem. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1197–1206, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[BPT12] M. M. Baldi, G. Perboli, and R. Tadei. The three-dimensional knapsack problem with balancing constraints. *Applied Mathematics and Computation*, 218(19):9802–9818, 2012.

[CLS95] C. S. Chen, S. M. Lee, and Q. S. Shen. An analytical model for the container loading problem. *European Journal of Operational Research*, 80(1):68–76, 1995.

[CPT08] T. G. Crainic, G. Perboli, and R. Tadei. Extreme point-based heuristics for three-dimensional bin packing. *INFORMS Journal on Computing*, 20(3):368–384, 2008.

[CPT09] T. G. Crainic, G. Perboli, and R. Tadei. TS2PACK: A two-level tabu search for the three-dimensional bin packing problem. *European Journal of Operational Research*, 195(3):744–760, 2009.

[Ege09] J. Egeblad. Placement of two- and three-dimensional irregular shapes for inertia moment and balance. *International Transactions in Operational Research*, 16(6):789–807, 2009.

[EP09] J. Egeblad and D. Pisinger. Heuristic approaches for the two- and three-dimensional knapsack packing problem. *Computers & Operations Research*, 36(4):1026–1049, 2009.

[Fas99] G. Fasano. Cargo analytical integration in space engineering: A three-dimensional packing model. In *Operational Research in Industry*, Ichor Business Bks, page 232. Ichor Business Books, 1999.

[Fas04] G. Fasano. A MIP approach for some practical packing problems: Balancing constraints and tetris-like items. *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 2(2):161–174, 2004.

[Fas08] G. Fasano. MIP-based heuristic for non-standard 3D-packing problems. *4OR*, 6(3):291–310, 2008.

[Fas13] G. Fasano. A global optimization point of view to handle non-standard object packing problems. *Journal of Global Optimization*, 55(2):279–299, 2013.

[FPZ03] O. Faroe, D. Pisinger, and M. Zachariasen. Guided local search for the three-dimensional bin-packing problem. *INFORMS Journal on Computing*, 15(3):267–283, 2003.

[GG65] P. C. Gilmore and R. K. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13(1):94–120, 1965.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman New York, 1979.

[GJ00] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 22. print. edition, 2000.

[Gom10] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs and an algorithm for the mixed integer problem. In

M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 77–103. Springer Berlin Heidelberg, 2010.

[GR80] J. A. George and D. F. Robinson. A heuristic for packing boxes into a container. *Computers & Operations Research*, 7(3):147–156, 1980.

[Gur13] Gurobi Optimization. Gurobi Optimizer 5.6.0. Gurobi: `http://www.gurobi.com/`, 2013. last retrieved 11.03.2014.

[HHLB11] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of LION-5*, pages 507–523, 2011.

[HNW14] M. Hifi, S. Negre, and L. Wu. Hybrid greedy heuristics based on linear programming for the three-dimensional single bin-size bin packing problem. *International Transactions in Operational Research*, 21(1):59–79, 2014.

[IAT10] IATA. *Dangerous Goods Regulations*. International Air Transport Association, 2010.

[IBM13] IBM Corporation. IBM ILOG CPLEX Optimizer 12.5. IBM: `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`, 2013. last retrieved 11.03.2014.

[ICA12] ICAO. Annual report of the council. Technical report, International Civil Aviation Organization, 2012.

[JMY12a] L. Junqueira, R. Morabito, and D. S. Yamashita. MIP-based approaches for the container loading problem with multi-drop constraints. *Annals of Operations Research*, 199(1):51–75, 2012.

[JMY12b] L. Junqueira, R. Morabito, and D. S. Yamashita. Three-dimensional container loading models with cargo stability and load bearing constraints. *Computers & Operations Research*, 39(1):74–85, 2012. Special Issue on Knapsack Problems and Applications.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

[Law76] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover Books on Mathematics Series. Dover Publications, 1976.

[LSL11] S. Limbourg, M. Schyns, and G. Laporte. Automatic aircraft cargo load planning. *Journal of the Operational Research Society*, 63(9):1271–1283, 2011.

[Mic13a] Microsoft Corporation. Microsoft Windows 7. Microsoft: `http://windows.microsoft.com/en-US/windows7/products/home/`, 2013. last retrieved 11.03.2014.

[Mic13b] Microsoft Corporation. .Net Framework. Microsoft: `http://www.microsoft.com/net/`, 2013. last retrieved 11.03.2014.

[MPV00] S. Martello, D. Pisinger, and D. Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, 2000.

[Pad00] M. Padberg. Packing small boxes into a big box. *Mathematical Methods of Operations Research*, 52(1):1–21, 2000.

[PAVOT10] F. Parreño, R. Alvarez-Valdes, J. F. Oliveira, and J. M. Tamarit. A hybrid GRASP / VND algorithm for two- and three-dimensional bin packing. *Annals of Operations Research*, 179(1):203–220, 2010.

[PAVTO08] F. Parreño, R. Alvarez-Valdes, J. M. Tamarit, and J. F. Oliveira. A maximal-space algorithm for the container loading problem. *INFORMS Journal on Computing*, 20(3):412–422, 2008.

[PCT11] G. Perboli, T. G. Crainic, and R. Tadei. An efficient metaheuristic for multi-dimensional multi-container packing. In *Automation Science and Engineering (CASE), 2011 IEEE Conference on*, pages 563–568, 2011.

[Pis02] D. Pisinger. Heuristics for the container loading problem. *European Journal of Operational Research*, 141(2):382–392, 2002.

[PR00] M. Prais and C. C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12(3):164–176, 2000.

[PSL12] C. Paquay, M. Schyns, and S. Limbourg. Three dimensional bin packing problem applied to air cargo. In *ILS 2012 Proceedings*, 2012.

[Sal13] M. Sales. *The Air Logistics Handbook: Air Freight and the Global Supply Chain*. Routledge, 2013.

[SM13] L. Suhl and T. Mellouli. Software zur Lösung und Modellierung. In *Optimierungssysteme*, Springer-Lehrbuch, pages 77–93. Springer Berlin Heidelberg, 3. edition, 2013.

[TZL08] C. Tian, H. Zhang, and F. Li. Air cargo load planning. In *Service Operations and Logistics, and Informatics, 2008. IEEE/SOLI 2008. IEEE International Conference on*, volume 1, pages 1349–1354, 2008.

[VTA10] C. Voudouris, E. P.K. Tsang, and A. Alsheddy. Guided local search. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 321–361. Springer US, 2010.

[WGC+10] L. Wang, S. Guo, S. Chen, W. Zhu, and A. Lim. Two natural heuristics for 3d packing with practical loading constraints. In B.-T. Zhang and M. A. Orgun, editors, *PRICAI 2010: Trends in Artificial Intelligence*, volume 6230 of *Lecture Notes in Computer Science*, pages 256–267. Springer Berlin Heidelberg, 2010.

[WHS07] G. Wäscher, H. Haußner, and H. Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, 2007.

[WLGdS10] Y. Wu, W. Li, M. Goh, and R. de Souza. Three-dimensional bin packing problem with variable bin height. *European Journal of Operational Research*, 202(2):347–355, 2010.

[WN99] L. A. Wolsey and G. L. Nemhauser. *Integer and Combinatorial Optimization*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1999.

[Woe03] G. J. Woeginger. Exact algorithms for NP-hard problems: A survey. In M. Jünger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial Optimization - Eureka, You Shrink!*, volume 2570 of *Lecture Notes in Computer Science*, pages 185–207. Springer Berlin Heidelberg, 2003.

[YMS08] N. Yano, T. Morinaga, and T. Saito. Packing optimization for cargo containers. In *SICE Annual Conference, 2008*, pages 3479–3482, Aug 2008.

[Zha03] A. Zhang. Analysis of an international air-cargo hub: the case of Hong Kong. *Journal of Air Transport Management*, 9(2):123–138, 2003.

[ZZOL12] W. Zhu, Z. Zhang, W.-C. Oon, and A. Lim. Space defragmentation for packing problems. *European Journal of Operational Research*, 222(3):452–463, 2012.

Glossary

3D-SBSBPP	The three-dimensional single bin-size bin packing problem.
3DBPP	The three-dimensional bin packing problem.
ATV	An Automated Transfer Vehicle is an unmanned spacecraft used to transport supplies to space. The vehicle was developed by the European Space Agency (ESA).
AWB	The air waybill, which covers the whole transportation in air cargo.
BS-EPSD	Bin Shuffling using Extreme Point insertion with Space Defragmentation heuristic.
DGR	Dangerous goods regulations as defined by the IATA.
DSS	Decision Support System.
EP	Extreme Point. Used as a special insertion point, when generating solutions with heuristics.
EP-BFD	Extreme Points Best Fit Decreasing heuristic.
EP-FFD	Extreme Points First Fit Decreasing heuristic.
EPI	Extreme Point Insertion constructive heuristic, which unifies and extends the EP-FFD and EP-BFD technique.
EPI-Tetris	Extreme Point Insertion constructive heuristic, which unifies and extends the EP-FFD and EP-BFD technique and can also handle "Tetris"-like pieces.
ESA	The European Space Agency (ESA).
FLB	Refers to the front-left-bottom corner point of a piece.

GASP	Greedy Adaptive Search Procedure metaheuristic.
GLS	The Guided Local Search metaheuristic.
GRASP	Greedy Randomized Adaptive Search Procedure metaheuristic.
IATA	The International Air Transport Association.
IP	Insertion Point. A more general point at which pieces are inserted into a container by constructive heuristics.
MBP	A mixed binary program consisting of continuous and binary variables.
MIP	A mixed integer program consisting of both continuous and integral variables.
PI	Push Insertion constructive heuristic.
PI-Tetris	Push Insertion constructive heuristic with the extended capability of handling "Tetris"-like pieces.
RFS	Road Feeder Service used to serve airports not handled by air or short distances.
RRT	Refers to the rear-right-top corner point of a piece.
SA	The Simulated Annealing metaheuristic.
SD	Space Defragmentation constructive heuristic, which extends the EPSD technique.
SMAC	The Sequential Model-based Algorithm Configuration.
Tetris	A game developed in the eighties known for its typical shapes.
ULD	A Unit Load Device, which consolidates containers and palettes specifically shaped to fit into an airplane.

Appendix A

Algorithms

A.1 Extreme point generation with tetris-pieces

This section introduces the generation of EPs when handling "Tetris"-pieces in more detail. The concept is an extension of the technique proposed by [CPT08]. The major distinction is the generation of EPs for every cuboid component of a piece instead of only for the piece itself. The procedure is depicted in Algorithm A.1.
The algorithm starts with a given solution and generates all EPs for a particular piece p at its insertion point e in a container c. The result is a set of EPs $\mathcal{E}'$ which are added to the list of available insertion points $\mathcal{E}$. For every component $b \in \mathcal{B}_p$ of the piece the six default Extreme Points are generated (see lines 6 through 11). These match the ones introduced by [CPT08]. In lines 12 through 23 these points are projected along the respective axes towards the containers origin until a blocking component is hit. The following term (see Equation A.1) defines this particular set of components with a position smaller than the one of e_1 regarding axis d blocking the projection of point e_2 along the same axis d. This set is always defined within the context of solution $\mathcal{S}$.

$$\mathcal{B}^{\texttt{block}}_{de_1e_2} := \{(b' \in \mathcal{B}_{p'}, p' \in \mathcal{S}^C_c) \mid S^P_{p'}.d + P^{\texttt{B}}_{b'S^O_{p'}8d} \leq e_1.d \wedge \\ \forall d' \in \mathcal{D} \setminus \{d\} : S^P_{p'}.d' + P^{\texttt{B}}_{b'S^O_{p'}1d'} \leq e_2.d' \wedge e_2.d' \leq S^P_{p'}.d' + P^{\texttt{B}}_{b'S^O_{p'}8d'}\} \quad \text{(A.1)}$$

The resulting EPs are then added to the set of available insertion points $\mathcal{E}$ (see 24 through 26). If a projection of one point results in two equal points (e.g. $e_{11} = e_{22}$), only one of them is added. Additionally a second set of points is generated in the inner loop (see line 3). For every vertex $v \in \{1, 4, 6, 7, 8\}$ an additional insertion point is generated at its position, if the vertex defines an inner corner of the complete piece. I.e. the position of the vertex v of the respective component does not define a vertex (corner point) of the piece, but only a one of the particular component. Thus, the insertion of pieces within the recess of an "L"-shape is possible.
The additional insertion points enable more possible positions at which "Tetris"-pieces may be inserted into a container. On the downside the technique results in the generation of quite a large number of points which leads to higher runtimes, because more points may be examined during the constructive heuristic.

Algorithm A.1: Extreme Point generation (tetris)

Input: $\mathcal{S}_c^C, S_p^O, S_p^P, e, p, c$
Output: $\mathcal{E}$

$\mathcal{E}' \leftarrow \emptyset$
foreach $b \in \mathcal{B}_p$ **do**
  foreach $v \in \{1, 4, 6, 7, 8\}$ **do**
    if $\exists d \in \mathcal{D} : S_p^P.d < S_p^P.d + P^{\mathtt{B}}_{bS_p^O vd} \wedge S_p^P.d + P^{\mathtt{B}}_{bS_p^O vd} < S_p^P + L^{\mathtt{P}}_{pod}$ **then**
      $\mathcal{E}' \leftarrow \mathcal{E}' \cup (S_p^P.x + P^{\mathtt{B}}_{bS_p^O vx}, S_p^P.y + P^{\mathtt{B}}_{bS_p^O vy}, S_p^P.z + P^{\mathtt{B}}_{bS_p^O vz})$
  $e_{11} \leftarrow (S_p^P.x + P^{\mathtt{B}}_{bS_p^O 8x}, 0, S_p^P.z + P^{\mathtt{B}}_{bS_p^O 1z})$
  $e_{12} \leftarrow (S_p^P.x + P^{\mathtt{B}}_{bS_p^O 8x}, S_p^P.y + P^{\mathtt{B}}_{bS_p^O 1y}, 0)$
  $e_{21} \leftarrow (0, S_p^P.y + P^{\mathtt{B}}_{bS_p^O 8y}, S_p^P.z + P^{\mathtt{B}}_{bS_p^O 1z})$
  $e_{22} \leftarrow (S_p^P.x + P^{\mathtt{B}}_{bS_p^O 1x}, S_p^P.y + P^{\mathtt{B}}_{bS_p^O 8y}, 0)$
  $e_{31} \leftarrow (0, S_p^P.y + P^{\mathtt{B}}_{bS_p^O 1y}, S_p^P.z + P^{\mathtt{B}}_{bS_p^O 8z})$
  $e_{32} \leftarrow (S_p^P.x + P^{\mathtt{B}}_{bS_p^O 1x}, 0, S_p^P.z + P^{\mathtt{B}}_{bS_p^O 8z})$
  $(b_{11}, p_{11}) \leftarrow \underset{(b',p') \in \mathcal{B}^{\mathtt{block}}_{yee11}}{\operatorname{argmax}} S_{p'}^P.y + P^{\mathtt{B}}_{b' S_{p'}^O 8y}$
  if $(b_{11}, p_{11}) \neq$ **nil then** $e_{11}.y \leftarrow S_{p_{11}}^P.y + P^{\mathtt{B}}_{b_{11} S_{p_{11}}^O 8y}$
  $(b_{12}, p_{12}) \leftarrow \underset{(b',p') \in \mathcal{B}^{\mathtt{block}}_{zee11}}{\operatorname{argmax}} S_{p'}^P.z + P^{\mathtt{B}}_{b' S_{p'}^O 8z}$
  if $(b_{12}, p_{12}) \neq$ **nil then** $e_{12}.y \leftarrow S_{p_{12}}^P.y + P^{\mathtt{B}}_{b_{12} S_{p_{12}}^O 8y}$
  $(b_{21}, p_{21}) \leftarrow \underset{(b',p') \in \mathcal{B}^{\mathtt{block}}_{xee11}}{\operatorname{argmax}} S_{p'}^P.x + P^{\mathtt{B}}_{b' S_{p'}^O 8x}$
  if $(b_{21}, p_{21}) \neq$ **nil then** $e_{21}.y \leftarrow S_{p_{21}}^P.y + P^{\mathtt{B}}_{b_{21} S_{p_{21}}^O 8y}$
  $(b_{22}, p_{22}) \leftarrow \underset{(b',p') \in \mathcal{B}^{\mathtt{block}}_{zee11}}{\operatorname{argmax}} S_{p'}^P.z + P^{\mathtt{B}}_{b' S_{p'}^O 8z}$
  if $(b_{22}, p_{22}) \neq$ **nil then** $e_{22}.y \leftarrow S_{p_{22}}^P.y + P^{\mathtt{B}}_{b_{22} S_{p_{22}}^O 8y}$
  $(b_{31}, p_{31}) \leftarrow \underset{(b',p') \in \mathcal{B}^{\mathtt{block}}_{xee11}}{\operatorname{argmax}} S_{p'}^P.x + P^{\mathtt{B}}_{b' S_{p'}^O 8x}$
  if $(b_{31}, p_{31}) \neq$ **nil then** $e_{31}.y \leftarrow S_{p_{31}}^P.y + P^{\mathtt{B}}_{b_{31} S_{p_{31}}^O 8y}$
  $(b_{32}, p_{32}) \leftarrow \underset{(b',p') \in \mathcal{B}^{\mathtt{block}}_{yee11}}{\operatorname{argmax}} S_{p'}^P.y + P^{\mathtt{B}}_{b' S_{p'}^O 8y}$
  if $(b_{32}, p_{32}) \neq$ **nil then** $e_{32}.y \leftarrow S_{p_{32}}^P.y + P^{\mathtt{B}}_{b_{32} S_{p_{32}}^O 8y}$
  $\mathcal{E}' \leftarrow \mathcal{E}' \cup e_{11}$ **if** $e_{11} \neq e_{12}$ **then** $\mathcal{E}' \leftarrow \mathcal{E}' \cup e_{12}$
  $\mathcal{E}' \leftarrow \mathcal{E}' \cup e_{21}$ **if** $e_{21} \neq e_{22}$ **then** $\mathcal{E}' \leftarrow \mathcal{E}' \cup e_{22}$
  $\mathcal{E}' \leftarrow \mathcal{E}' \cup e_{31}$ **if** $e_{31} \neq e_{32}$ **then** $\mathcal{E}' \leftarrow \mathcal{E}' \cup e_{32}$
return $\mathcal{E} \cup \mathcal{E}'$

A.2 Container normalization

In the following the container normalization technique initially proposed by [ZZOL12] is explained in more detail. The method is used by the SD technique to increase the density of the packing after a push-out operation is conducted. Additionally the PI heuristic uses the method as a fast insertion technique. In both cases the input is defined by a given solution and a container c, which has to be normalized. The algorithm then basically pushes all pieces towards the origin of the container along the axes defined by $\mathcal{D}'$ until no further move of a piece is possible. The order of the list $\mathcal{D}'$ can be defined as a parameter, but should contain all axes $d \in \mathcal{D}$. Thus, a normalization is not only done along two axes for example. The push operation for an axis is then conducted for every piece consecutively beginning with the one closest to the container's inner side wall, i.e. the one with the smallest value of $S_p^P.d$ (see lines 8, 12 and 16). The `ShiftIn` operation pushes each of the pieces inwards until it is blocked by another one or the container's side wall. In the case of handling "Tetris"-pieces the collision detection is done on a component-level. This means that while pushing for every component b of the respective piece p and every other component b' of another piece p' a possible collision is investigated. If in one main iteration another change by pushing is detected, the complete process restarts.

Algorithm A.2: Normalization procedure

Input: $\mathcal{S}_c^C, S_p^O, S_p^P, c, \mathcal{D}$
Output: S_p^O, S_p^P

```
1  change ← true
2  while change do
3      change ← false
4      foreach d ∈ D' do
5          switch d do
6              case d = x
7                  foreach p ∈ S_c^C.OrderBy(S_p^P.x + P_{pS_p^O 8x}) do
8                      S_p'^P ← ShiftIn(x, c, p)
9                      if S_p'^P ≠ S_p^P then S_p^P ← S_p'^P, change ← true
10             case d = y
11                 foreach p ∈ S_c^C.OrderBy(S_p^P.y + P_{pS_p^O 8y}) do
12                     S_p'^P ← ShiftIn(y, c, p)
13                     if S_p'^P ≠ S_p^P then S_p^P ← S_p'^P, change ← true
14             case d = z
15                 foreach p ∈ S_c^C.OrderBy(S_p^P.z + P_{pS_p^O 8z}) do
16                     S_p'^P ← ShiftIn(z, c, p)
17                     if S_p'^P ≠ S_p^P then S_p^P ← S_p'^P, change ← true
18 return (S_p^O, S_p^P)
```

Appendix B

Metrics

B.1 Merit-types

This section introduces the different implemented merit functions used to select the best insertion point in a best-fit approach. This means every valid insertion point is assessed with a score calculated by the functions below, and then inserted at the one with the best score. The best score here is the lowest one. Most of this functions are already proposed by [CPT08].

- Minimize the residual volume of the container after accommodation.

  $$f(c,p,o,e) = V_c^{\mathtt{C}} - \sum_{p' \in \mathcal{S}_c^C} V_{p'}^{\mathtt{P}} - V_p^{\mathtt{P}}$$

  The goal of this technique is to utilize the containers space as volume-efficient as possible, with a very simple approach. The downside of this function is that it does not differentiate between different insertion-points of a container.

- Minimize the size of the resulting packing regarding the x- and y- dimensions

  $$f(c,p,o,e) = f_x^{PS}(c,p,o,e) + f_y^{PS}(c,p,o,e)$$

  $$f_x^{PS}(c,p,o,e) = \begin{cases} e.x + L_{pox}^{\mathtt{P}} - S_{cx}^{PS} & e.x + L_{pox}^{\mathtt{P}} > S_{cx}^{PS} \\ 0 & \text{otherwise} \end{cases}$$

  $$f_y^{PS}(c,p,o,e) = \begin{cases} e.y + L_{poy}^{\mathtt{P}} - S_{cy}^{PS} & e.y + L_{poy}^{\mathtt{P}} > S_{cy}^{PS} \\ 0 & \text{otherwise} \end{cases}$$

  with

  $$S_{cd}^{PS} = \max_{p \in \mathcal{S}_c^C}(S_p^P.d + L_{pod}^{\mathtt{P}})$$

  This merit-function tries to minimize the resulting packing's base-area size. In the best case the accommodation of a new piece to the container does not increase the area the packing consumed until then.

- Level the packing's area regarding the x- and y- dimensions

$$f(c,p,o,e) = f_x^{LV}(c,p,o,e) + f_y^{LV}(c,p,o,e)$$

$$f_x^{LV}(c,p,o,e) = \begin{cases} (e.x + L_{pox}^{\mathtt{P}} - S_{cx}^{PS}) \cdot C^{LV} & e.x + L_{pox}^{\mathtt{P}} > S_{cx}^{PS} \\ S_{cx}^{PS} - e.x + L_{pox}^{\mathtt{P}} & \text{otherwise} \end{cases}$$

$$f_y^{LV}(c,p,o,e) = \begin{cases} (e.y + L_{poy}^{\mathtt{P}} - S_{cy}^{PS}) \cdot C^{LV} & e.y + L_{poy}^{\mathtt{P}} > S_{cy}^{PS} \\ S_{cy}^{PS} - e.y + L_{poy}^{\mathtt{P}} & \text{otherwise} \end{cases}$$

  This approach extends the minimization of the packing's size by also differentiating between insertions, which do not extend the base-area. To further penalize the extension of the base-area the constant parameter $C^{LV} > \max_{c \in \mathcal{C}}(\max(L_{cx}^{\mathtt{P}}, L_{cy}^{\mathtt{P}}))$ is used.

- Maximize the residual space's utilization of the resulting packing

$$f(c,p,o,e) = \sum_{d \in \mathcal{D}} S_{ed}^{RS} - L_{pod}^{\mathtt{P}}$$

  The goal of this method is to accommodate the new piece where it fits best. This is done by keeping track of the residual space S_{ed}^{RS} available at each extreme-point. The residual space is the space emerging from the projection of the three axes until the projection hits another already accommodated piece in the container. The exact procedure alongside with the update of this information is described in more detail in [CPT08].

- Minimize the euclidean distance to the origin of the container

$$f(c,p,o,e) = \sqrt{(e.x + L_{pox}^{\mathtt{P}})^2 + (e.y + L_{poy}^{\mathtt{P}})^2 + (e.z + L_{poz}^{\mathtt{P}})^2}$$

  This method simply aims to minimize the euclidean distance of the outer vertex (seen from the origin of the container) of the piece to the origin of the container, thus also increasing the density of the packing.

- Minimize the euclidean distance to the origin of the container regarding x and y

$$f(c,p,o,e) = \sqrt{(e.x + L_{pox}^{\mathtt{P}})^2 + (e.y + L_{poy}^{\mathtt{P}})^2}$$

  This merit function, as the one stated before, tries to minimize the distance of the new piece's outer vertex to the origin of the container. This time only the x and y axes are considered to minimize the base-area of the resulting packing, instead of the volume consumed.

Appendix C

Parameter tuning

C.1 Tuned parameters

This section consolidates all parameters and their domains subject to the parameter tuning with SMAC. Note that specific parameters are only available in the respective algorithms and are not considered for others, e.g. inflation and replacement of pieces for EPI. See subsection 5.1.3 for a detailed description of the available parameters and the specific algorithms using them. Other parameters are introduced in this section initially.

- *PieceOrder* is used to identify the initial order of the pieces when inserting. The different available values are given by the possible orders defined in subsection 5.1.4.

- *BestFit* is a boolean variable defining whether to use a merit-function (see section B.1) or not. If the value is false, every piece is inserted at the first valid insertion point. The domain is $\{true, false\}$.

- *MeritType* is a conditional parameter identifying the specific merit-function to use, if *BestFit* is activated. Thus, the domain is given by the different options (see section B.1).

- *InflateAndReplaceInsertion* defines whether to use or skip the inflate-and-replace strategy of the SD technique. The domain is $\{true, false\}$.

- *NormalizationOrder* defines the order by which the container is normalized. Pieces are pushed to the corresponding directions consequentially until no further push is possible. This directly influences the normalization procedure described in section A.2. The domain is $\{XYZ, ZYX, ZXY, YZX, XZY, YXZ\}$.

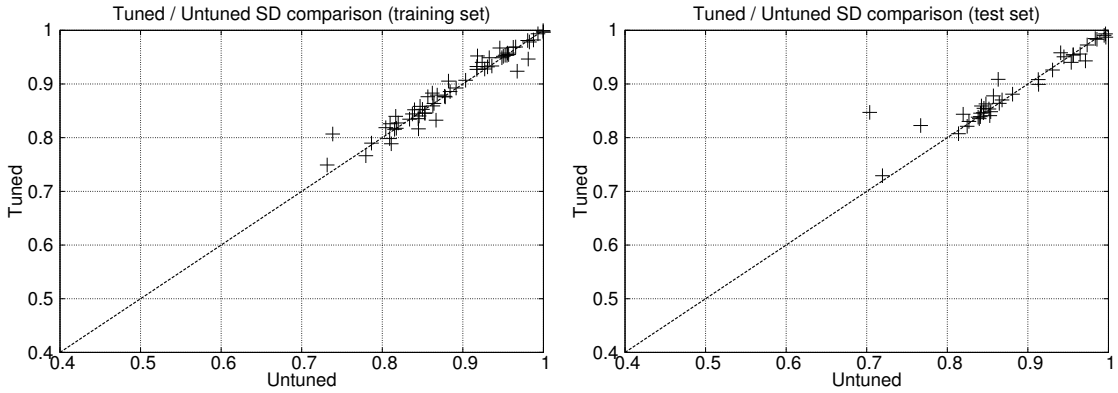
- *ScoreBasedOrder* may deactivate the score-based sorting of pieces and substitutes it with a complete random approach. The domain is $\{true, false\}$.

- i^{RD} defines the maximal distance without an improvement on the objective value before the score is reinitialized. The domain is $[50, \ldots, 2500] \subset \mathbb{Z}$.

- r^S sets the "salt"-value applied at random for every piece. The domain is $[0, \ldots, 0.9] \subset \mathbb{R}$.
- m^I sets the initial percentage of score modification. The domain is $[0.1, \ldots, 1] \subset \mathbb{R}$.
- m^{max} sets the maximum percentage of score modification. The domain is $[1, \ldots, 5] \subset \mathbb{R}$.
- s^P sets the number of possible swaps. The domain is $[1, \ldots, 4] \subset \mathbb{Z}$.
- s^{max} sets the number of maximal swaps. The domain is $[4, \ldots, 8] \subset \mathbb{Z}$.

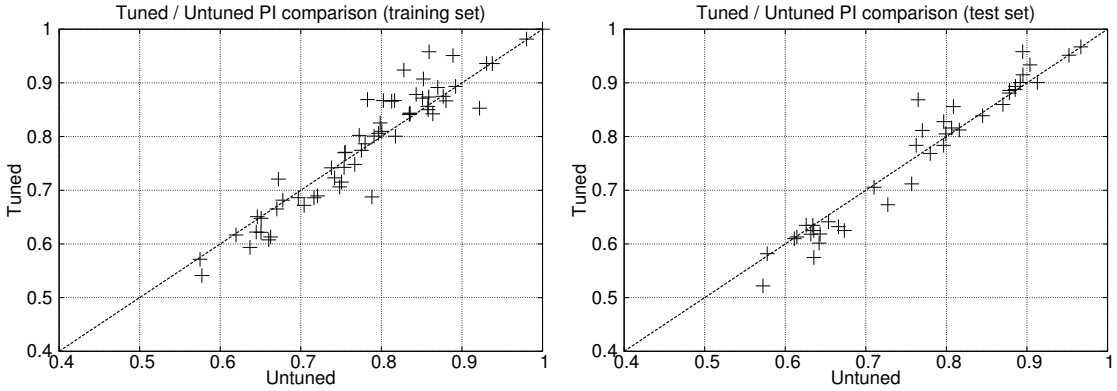
C.2 Parameter tuning results

This section contains the additional results of the parameter tuning for the EPI-Tetris (see Figure C.3), SD (see Figure C.1), PI (see Figure C.2) and PI-Tetris (see Figure C.4) approaches. The graphs use the same representation like the one described in section 6.2. Similar results like the one described for EPI before can also be drawn for the parameter tuning on the rest of the methods. This is not described here in more detail. In all graphs the relative volume utilization of the untuned setting is compared to the tuned one for the particular algorithms. Thereby, the left graph (a) depicts the results for the training set and the right one (b) for the test set. Thus, all points lying above the diagonal depict an improvement due to parameter tuning while all points beneath the diagonal represent a deterioration.



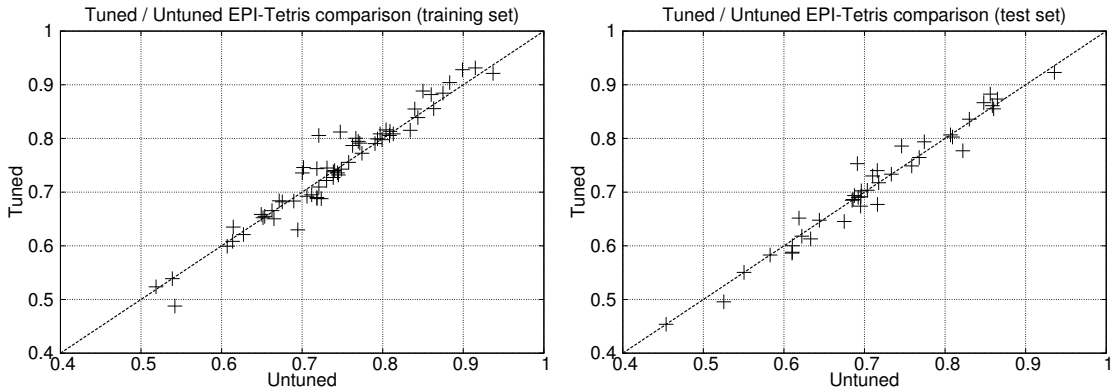
(a) Training (b) Test

Figure C.1: Tuning results for SD



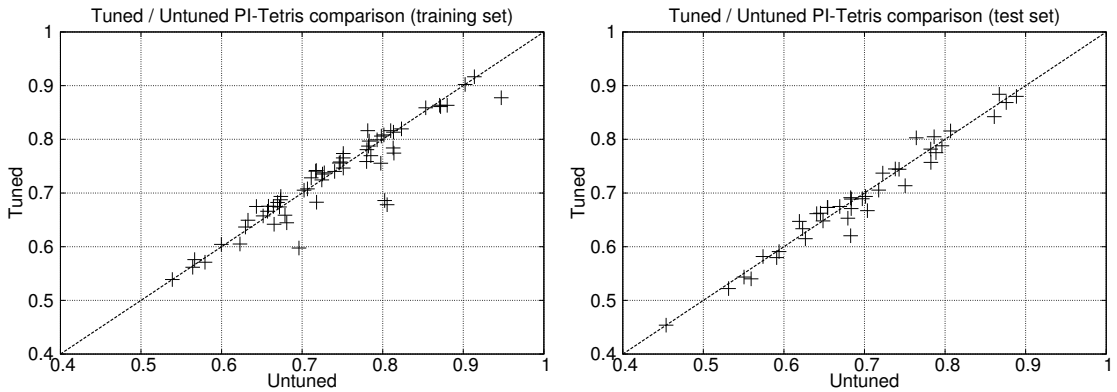
(a) Training (b) Test

Figure C.2: Tuning results for PI



(a) Training (b) Test

Figure C.3: Tuning results for EPI-Tetris



(a) Training (b) Test

Figure C.4: Tuning results for PI-Tetris

Appendix D

Detailed computational results

This section contains the detailed results of the method comparison. All model formulations were solved without respecting the stability and load bearing requirement to ensure equal conditions. The results are divided into the cuboid (see section D.1) and tetris instance set (see section D.2). For each set a plot is shown and a detailed table of the results. In the table the respective best method per instance is highlighted. The data plots shown in Figure D.1 and Figure D.2 give an overview of the relative volume utilization achieved per instance. Thus, the x-axis represents all 100 instances of the set in no particular order. The y-axis then represents the relative volume utilization or evaluation value, which is always a value between 0 and 1. So, the graph plots every method's result for each instance. Despite the overload of information, tendencies can be recognized reflecting the previously mentioned observations.

D.1 Cuboid instances

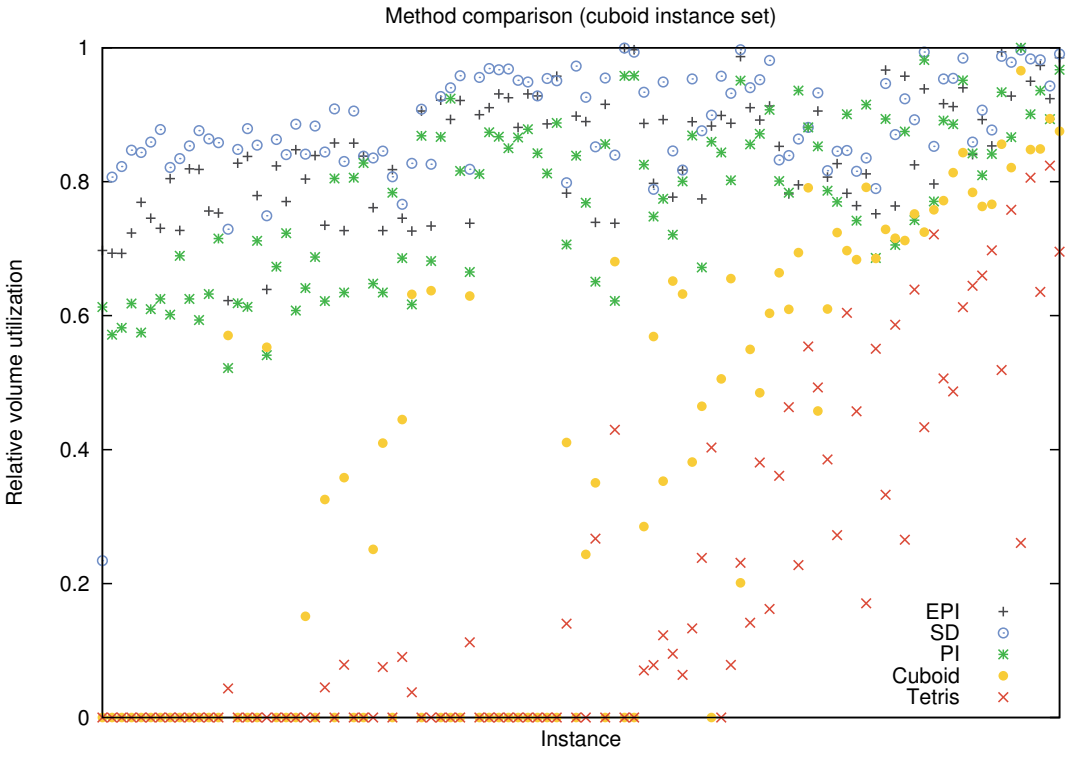


Figure D.1: Plot of the results for the cuboid instance set

Table D.1: Detailed method comparison results (cuboid instances)

Instance	EPI	SD	PI	Cuboid	Tetris
1	80.39%	84.10%	64.12%	15.10%	0.00%
2	98.69%	99.71%	95.10%	20.10%	23.10%
3	88.10%	88.10%	88.10%	79.10%	55.40%
4	73.80%	83.98%	62.21%	68.06%	42.96%
5	82.51%	89.25%	74.26%	75.18%	63.88%
6	76.37%	87.03%	70.55%	71.55%	58.65%
7	92.79%	97.83%	86.66%	82.10%	75.80%
8	95.00%	98.33%	90.07%	84.80%	80.60%
9	75.20%	78.98%	68.62%	68.55%	55.07%
10	89.26%	90.69%	80.97%	76.30%	66.00%
11	92.41%	94.30%	89.34%	89.40%	82.40%
12	79.65%	85.28%	77.06%	75.80%	72.12%
13	91.05%	94.03%	85.58%	54.97%	14.14%
14	77.70%	84.59%	72.08%	65.16%	9.51%
15	88.98%	92.60%	76.86%	24.36%	0.00%
16	76.12%	83.54%	64.77%	25.11%	0.00%
17	92.18%	92.73%	86.70%	0.00%	0.00%
18	72.33%	84.71%	61.81%	0.00%	0.00%
19	89.29%	94.03%	92.39%	0.00%	0.00%
20	69.73%	23.43%	61.29%	0.00%	0.00%
21	85.27%	83.25%	80.09%	66.41%	36.10%
22	96.68%	94.65%	89.35%	72.88%	33.25%
23	95.75%	92.38%	87.50%	71.23%	26.55%
24	80.69%	81.64%	78.65%	61.00%	38.54%
25	72.71%	83.03%	63.46%	35.81%	7.88%
26	88.71%	93.36%	82.54%	28.50%	7.04%
27	88.72%	93.20%	80.22%	65.54%	7.86%
28	90.56%	93.26%	85.28%	45.76%	49.28%
29	89.20%	95.22%	87.16%	48.48%	38.08%

Instance	EPI	SD	PI	Cuboid	Tetris
30	72.63%	82.74%	61.67%	63.18%	3.78%
31	74.55%	76.64%	68.59%	44.49%	9.04%
32	78.29%	79.84%	70.62%	41.06%	14.02%
33	72.69%	84.57%	63.46%	40.97%	7.53%
34	88.96%	95.35%	86.90%	38.14%	13.30%
35	89.27%	94.90%	77.44%	35.30%	12.28%
36	73.50%	84.43%	62.17%	32.55%	4.50%
37	82.77%	84.83%	61.85%	0.00%	0.00%
38	92.80%	92.80%	84.24%	0.00%	0.00%
39	83.82%	83.82%	82.78%	0.00%	0.00%
40	80.45%	82.12%	60.16%	0.00%	0.00%
41	81.18%	83.54%	91.52%	79.16%	17.04%
42	100.00%	99.66%	100.00%	96.57%	26.07%
43	77.42%	87.61%	67.19%	46.47%	23.82%
44	91.31%	98.10%	90.73%	60.36%	16.22%
45	82.71%	84.55%	77.00%	72.41%	27.24%
46	99.39%	98.73%	93.35%	85.61%	51.89%
47	73.93%	85.20%	65.07%	35.04%	26.71%
48	89.89%	95.76%	84.36%	50.55%	0.00%
49	69.33%	80.68%	57.15%	0.00%	0.00%
50	85.77%	90.86%	80.49%	0.00%	0.00%
51	85.76%	90.54%	80.60%	0.00%	0.00%
52	69.30%	82.27%	58.19%	0.00%	0.00%
53	92.13%	95.82%	81.60%	0.00%	0.00%
54	81.91%	85.35%	62.50%	0.00%	0.00%
55	90.01%	95.58%	81.14%	0.00%	0.00%
56	75.63%	86.40%	63.22%	0.00%	0.00%
57	85.36%	87.72%	84.14%	76.62%	69.77%
58	98.51%	99.06%	96.71%	87.55%	69.55%
59	97.35%	98.20%	93.62%	84.88%	63.56%

Instance	EPI	SD	PI	Cuboid	Tetris
60	84.04%	85.91%	84.22%	78.42%	64.47%
61	91.21%	95.43%	88.59%	81.34%	48.69%
62	76.41%	81.56%	74.18%	68.37%	45.73%
63	78.18%	83.91%	78.36%	60.94%	46.35%
64	91.65%	95.34%	89.13%	77.17%	50.62%
65	63.91%	74.92%	54.11%	55.28%	0.00%
66	73.39%	82.60%	68.16%	63.72%	0.00%
67	73.79%	81.85%	66.52%	62.94%	11.23%
68	62.24%	72.92%	52.19%	57.05%	4.33%
69	93.10%	96.71%	86.73%	0.00%	0.00%
70	81.83%	87.63%	59.35%	0.00%	0.00%
71	90.55%	90.82%	86.86%	0.00%	0.00%
72	76.92%	84.37%	57.48%	0.00%	0.00%
73	83.92%	88.31%	68.75%	0.00%	0.00%
74	95.75%	95.07%	88.76%	0.00%	0.00%
75	93.10%	94.90%	87.83%	0.00%	0.00%
76	82.36%	86.30%	67.31%	0.00%	0.00%
77	79.51%	86.38%	93.60%	69.40%	22.76%
78	73.04%	87.79%	62.49%	0.00%	0.00%
79	93.86%	99.35%	98.16%	72.48%	43.35%
80	89.79%	97.27%	83.88%	0.00%	0.00%
81	94.02%	98.45%	95.15%	84.35%	61.25%
82	88.11%	95.15%	86.59%	0.00%	0.00%
83	78.28%	84.65%	90.07%	69.70%	60.43%
84	74.56%	85.91%	60.98%	0.00%	0.00%
85	81.81%	80.74%	78.37%	0.00%	0.00%
86	88.32%	89.93%	85.97%	0.00%	40.31%
87	81.74%	81.74%	80.06%	63.23%	6.36%
88	79.75%	78.87%	74.81%	56.87%	7.84%
89	72.73%	83.45%	68.94%	0.00%	0.00%

Instance	EPI	SD	PI	Cuboid	Tetris
90	83.77%	87.93%	61.30%	0.00%	0.00%
91	88.63%	95.41%	81.23%	0.00%	0.00%
92	99.88%	99.96%	95.80%	0.00%	0.00%
93	77.04%	84.05%	72.31%	0.00%	0.00%
94	84.78%	88.60%	60.75%	0.00%	0.00%
95	91.56%	95.49%	85.63%	0.00%	0.00%
96	99.70%	99.33%	95.82%	0.00%	0.00%
97	91.06%	96.89%	87.37%	0.00%	0.00%
98	75.32%	85.83%	71.51%	0.00%	0.00%
99	77.93%	85.47%	71.18%	0.00%	0.00%
100	92.54%	96.83%	85.01%	0.00%	0.00%
Average:	84.17%	88.49%	77.57%	35.63%	19.14%

D.2 Tetris instances

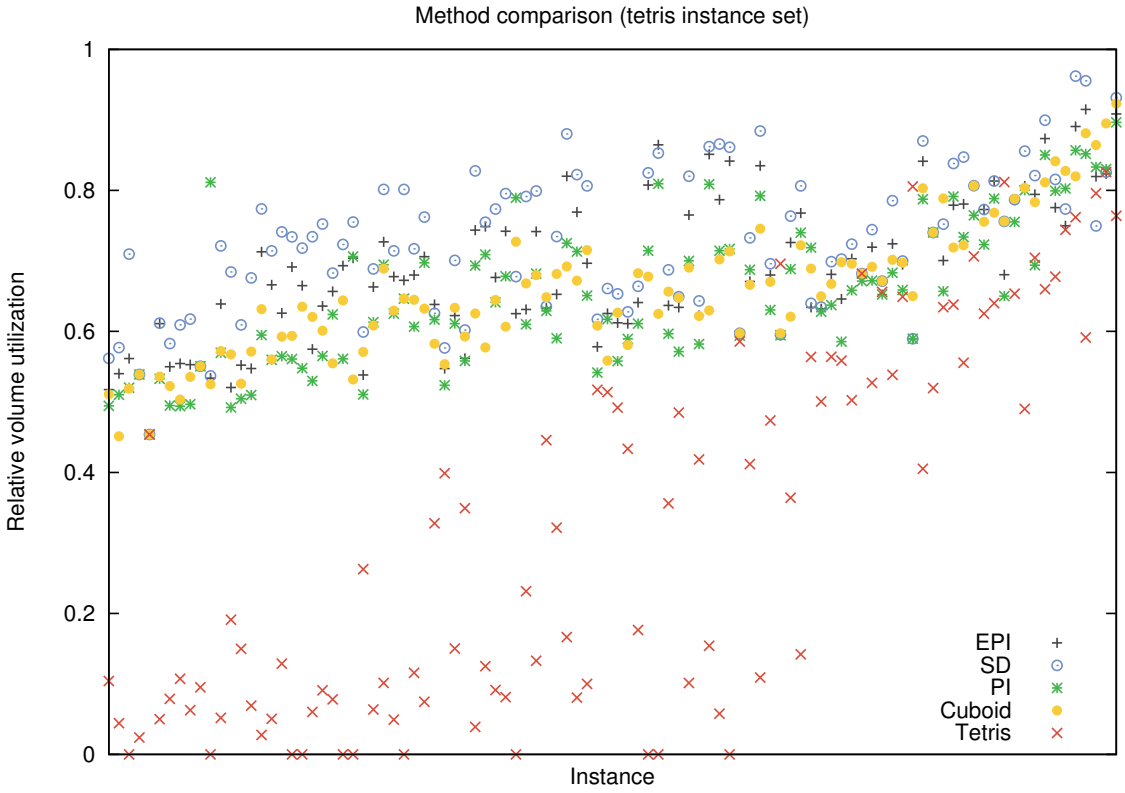


Figure D.2: Plot of the results for the tetris instance set

Table D.2: Detailed method comparison results (tetris instances)

Instance	EPI	EPI-Tetris	SD	PI	PI-Tetris	Cuboid	Tetris
1	86.46%	88.18%	85.30%	80.94%	86.11%	62.51%	0.00%
2	85.12%	88.84%	86.22%	80.87%	86.34%	62.99%	15.40%
3	70.39%	78.59%	75.51%	70.62%	78.17%	53.18%	0.00%
4	74.87%	78.69%	75.49%	70.90%	76.96%	57.73%	12.52%
5	84.15%	92.28%	86.12%	71.69%	80.28%	71.35%	0.00%
6	80.75%	92.12%	82.50%	71.45%	79.68%	67.77%	0.00%
7	66.49%	76.45%	71.81%	54.78%	67.31%	63.51%	0.00%
8	69.33%	79.38%	72.35%	56.11%	67.31%	64.40%	0.00%
9	67.14%	74.58%	67.14%	65.24%	68.28%	67.14%	65.60%
10	59.70%	65.85%	59.74%	59.40%	64.18%	59.74%	58.57%
11	68.20%	68.20%	68.20%	67.13%	65.86%	68.20%	68.20%

Instance	EPI	EPI-Tetris	SD	PI	PI-Tetris	Cuboid	Tetris
12	68.05%	81.20%	75.60%	65.00%	68.60%	75.60%	81.20%
13	63.20%	71.75%	63.46%	62.80%	70.53%	64.98%	50.05%
14	74.03%	74.03%	74.03%	74.03%	74.03%	74.03%	51.98%
15	58.95%	80.55%	58.95%	58.95%	67.82%	65.00%	80.55%
16	59.46%	68.38%	59.54%	59.46%	59.74%	59.74%	69.60%
17	63.42%	66.56%	64.92%	57.12%	64.45%	64.79%	48.48%
18	78.68%	79.04%	78.68%	75.54%	80.83%	78.80%	65.34%
19	81.96%	86.66%	74.96%	83.31%	86.86%	86.45%	79.60%
20	71.95%	74.89%	74.43%	67.20%	75.71%	69.17%	52.69%
21	68.08%	81.51%	69.89%	63.73%	78.08%	66.75%	56.40%
22	70.31%	77.70%	72.39%	65.83%	74.42%	69.60%	50.25%
23	61.22%	68.78%	65.32%	55.79%	67.41%	62.65%	49.19%
24	57.82%	67.70%	61.73%	54.17%	67.10%	60.81%	51.72%
25	67.99%	73.58%	69.60%	63.04%	70.77%	67.05%	47.36%
26	62.27%	69.51%	64.33%	58.21%	66.60%	62.18%	41.85%
27	64.61%	73.20%	70.25%	58.54%	74.20%	69.82%	55.86%
28	70.05%	80.26%	75.21%	65.68%	80.48%	78.87%	63.42%
29	69.14%	69.23%	73.41%	56.08%	67.50%	59.36%	0.00%
30	74.37%	83.89%	82.76%	69.35%	77.36%	62.54%	3.90%
31	82.02%	85.48%	88.01%	72.51%	81.61%	69.20%	16.65%
32	71.28%	72.17%	77.37%	59.49%	67.54%	63.16%	2.76%
33	54.74%	58.64%	57.65%	52.37%	59.09%	55.35%	39.87%
34	69.49%	74.49%	69.99%	65.87%	76.54%	69.76%	64.92%
35	77.28%	75.55%	77.28%	72.31%	75.55%	75.55%	62.53%
36	61.10%	64.53%	62.77%	58.94%	65.30%	58.06%	43.36%
37	82.35%	87.34%	82.59%	83.01%	88.00%	89.50%	82.60%
38	74.99%	80.86%	77.35%	80.27%	81.94%	82.77%	74.42%
39	79.43%	80.04%	82.09%	69.38%	78.40%	78.35%	70.43%
40	90.84%	93.14%	93.14%	89.67%	87.74%	92.30%	76.40%
41	66.31%	70.35%	68.88%	61.30%	69.35%	60.83%	6.38%

Instance	EPI	EPI-Tetris	SD	PI	PI-Tetris	Cuboid	Tetris
42	55.05%	55.05%	55.05%	55.05%	54.34%	55.05%	9.53%
43	54.98%	58.28%	58.28%	49.48%	58.17%	52.24%	7.87%
44	55.22%	65.04%	60.93%	50.43%	60.50%	52.60%	14.98%
45	67.76%	73.35%	71.42%	62.51%	68.93%	62.96%	4.93%
46	65.66%	68.49%	68.28%	62.41%	64.80%	55.46%	7.80%
47	55.44%	59.93%	60.93%	49.39%	57.11%	50.34%	10.73%
48	55.28%	61.80%	61.75%	49.68%	57.97%	53.54%	6.27%
49	53.89%	53.89%	53.89%	53.89%	53.89%	53.89%	2.40%
50	45.38%	45.38%	45.38%	45.38%	45.38%	45.38%	45.38%
51	80.67%	80.67%	80.67%	76.48%	77.52%	80.67%	70.65%
52	87.36%	86.10%	89.96%	85.03%	84.21%	81.15%	66.00%
53	74.20%	80.54%	79.55%	67.80%	74.07%	60.66%	8.13%
54	63.90%	69.36%	72.14%	56.97%	66.15%	57.20%	5.19%
55	74.15%	79.78%	79.90%	68.20%	73.54%	67.97%	13.28%
56	66.61%	68.63%	71.42%	55.96%	64.72%	56.01%	5.05%
57	53.82%	62.09%	59.92%	51.06%	64.92%	57.09%	26.27%
58	64.09%	72.73%	66.39%	61.10%	73.76%	68.26%	17.65%
59	56.17%	61.30%	60.20%	55.80%	63.35%	59.27%	34.93%
60	63.68%	68.80%	68.74%	59.66%	70.53%	65.64%	35.61%
61	91.50%	92.81%	95.55%	85.18%	91.66%	88.10%	59.13%
62	81.33%	81.33%	81.33%	78.83%	81.33%	76.83%	64.00%
63	80.63%	88.44%	85.58%	80.03%	86.35%	80.37%	49.00%
64	72.61%	81.60%	76.35%	68.85%	75.87%	62.08%	36.40%
65	72.42%	73.01%	78.55%	68.32%	73.70%	70.17%	53.84%
66	62.54%	70.23%	66.09%	61.71%	66.72%	55.85%	51.38%
67	78.08%	79.46%	84.72%	73.38%	80.61%	72.21%	55.55%
68	67.09%	75.30%	73.26%	68.74%	71.36%	66.62%	41.18%
69	61.11%	62.97%	61.17%	53.29%	60.43%	53.60%	5.00%
70	54.01%	58.80%	57.71%	51.02%	54.01%	45.13%	4.42%
71	89.08%	90.40%	96.22%	85.70%	90.22%	82.00%	76.20%

Instance	EPI	EPI-Tetris	SD	PI	PI-Tetris	Cuboid	Tetris
72	77.91%	77.25%	83.82%	79.14%	81.61%	71.91%	63.80%
73	62.53%	69.14%	67.77%	78.96%	69.14%	72.71%	0.00%
74	53.37%	52.36%	53.68%	81.15%	56.17%	52.50%	0.00%
75	62.24%	67.39%	70.08%	61.10%	67.49%	63.33%	15.03%
76	63.59%	69.00%	75.23%	56.52%	68.74%	60.10%	9.08%
77	76.80%	80.85%	80.63%	73.99%	79.91%	72.21%	14.19%
78	78.68%	85.51%	86.59%	71.46%	81.56%	70.20%	5.78%
79	77.56%	79.13%	81.55%	79.94%	77.42%	84.15%	67.76%
80	63.41%	65.16%	63.98%	71.87%	62.03%	68.91%	56.38%
81	65.26%	69.03%	73.45%	59.02%	68.87%	68.16%	32.15%
82	62.60%	68.34%	74.12%	56.48%	67.51%	59.26%	12.87%
83	76.52%	79.91%	82.02%	70.00%	78.74%	69.04%	10.14%
84	76.93%	83.58%	82.23%	71.32%	78.79%	67.19%	8.07%
85	69.67%	74.38%	80.64%	65.07%	74.65%	71.54%	10.00%
86	57.48%	65.23%	73.43%	52.98%	68.23%	62.06%	6.03%
87	67.65%	74.00%	77.36%	64.12%	74.49%	64.47%	9.13%
88	54.73%	63.50%	67.60%	50.96%	65.74%	57.14%	6.91%
89	83.51%	88.26%	88.43%	79.22%	88.39%	74.57%	10.91%
90	84.13%	85.55%	87.01%	78.75%	85.87%	80.31%	40.50%
91	72.69%	74.28%	80.15%	69.45%	75.78%	68.91%	10.14%
92	70.60%	73.57%	76.21%	69.73%	72.44%	63.23%	7.47%
93	68.00%	65.50%	71.72%	60.67%	69.35%	64.48%	11.57%
94	51.75%	49.57%	56.18%	49.43%	52.19%	51.08%	10.41%
95	63.83%	60.01%	62.57%	61.69%	61.49%	58.24%	32.79%
96	63.31%	48.78%	63.55%	62.90%	57.59%	64.86%	44.57%
97	52.05%	60.83%	68.42%	49.21%	63.67%	56.74%	19.10%
98	63.11%	70.95%	79.14%	61.00%	72.82%	66.81%	23.14%
99	67.24%	73.79%	80.14%	64.65%	75.59%	64.64%	0.00%
100	56.15%	64.77%	70.97%	52.03%	66.24%	51.87%	0.00%
Average:	68.55%	73.19%	72.77%	65.22%	71.53%	65.80%	31.87%

Appendix E

Prototype

This section contains some screenshots of the implemented prototype. Figure E.1 depicts an example of visualizing a solution using the prototype. The different types of pieces are marked by patterns while special handling instructions are shown by the symbols drawn onto the pieces. The numbers shown on the pieces are their respective IDs.

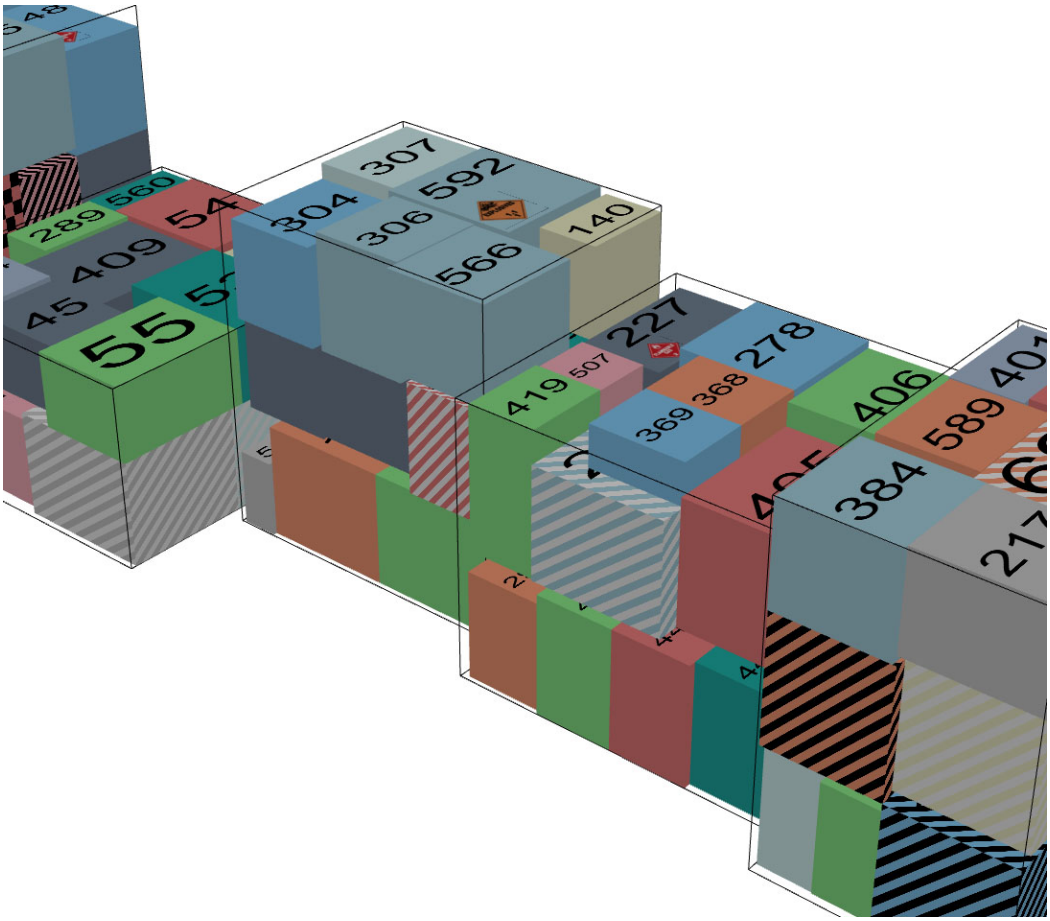


Figure E.1: Visualization from the prototype

The prototype is used to visualize the instances as well as the solutions to them. Figure E.2 shows the solution to a quite large instance. The pieces drawn in the fore-

ground, the so-called offload, could not be allocated to any container. The containers are depicted by wireframes while the pieces are solid boxes of random color.

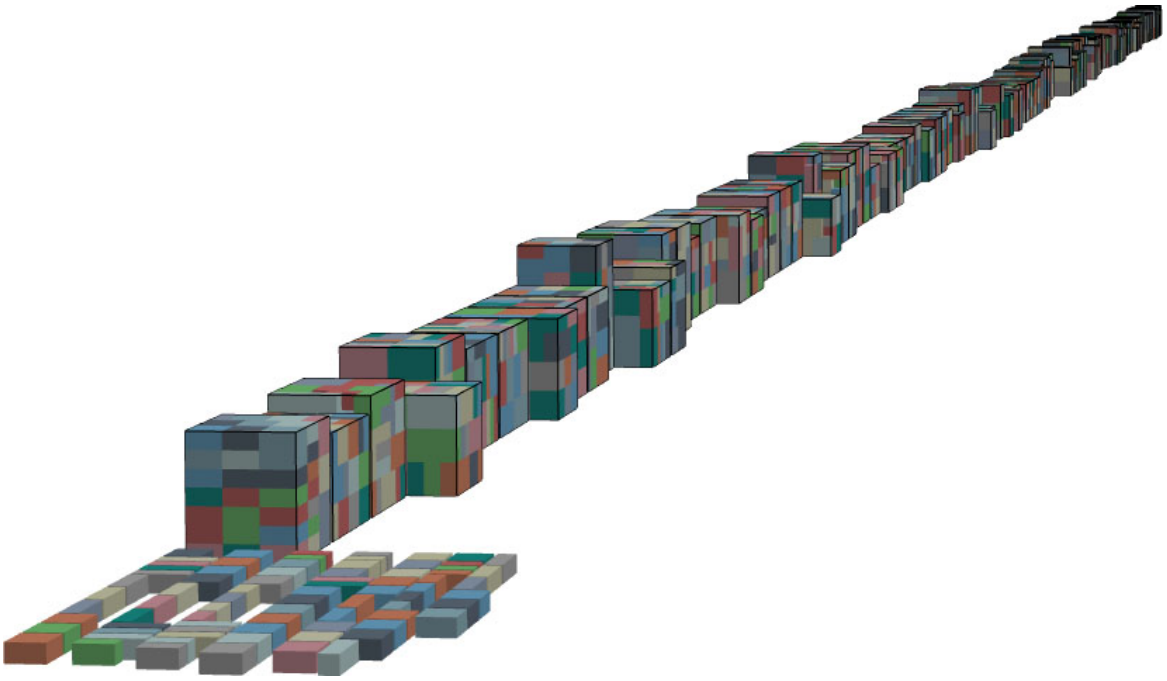
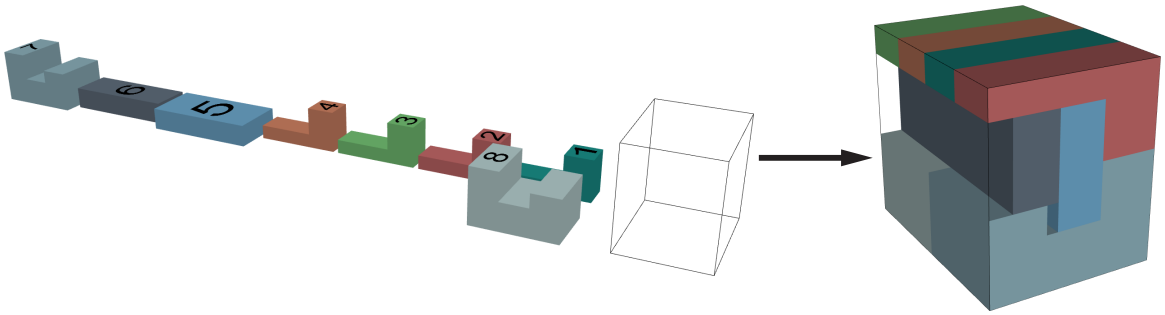
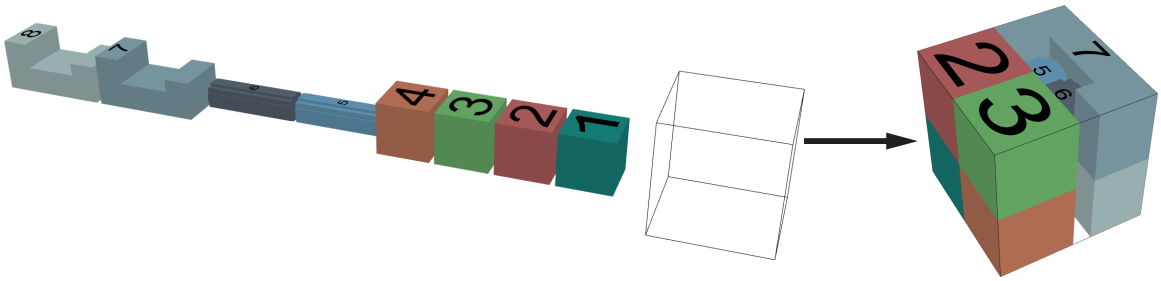


Figure E.2: Visualization of a large instance solved with the system

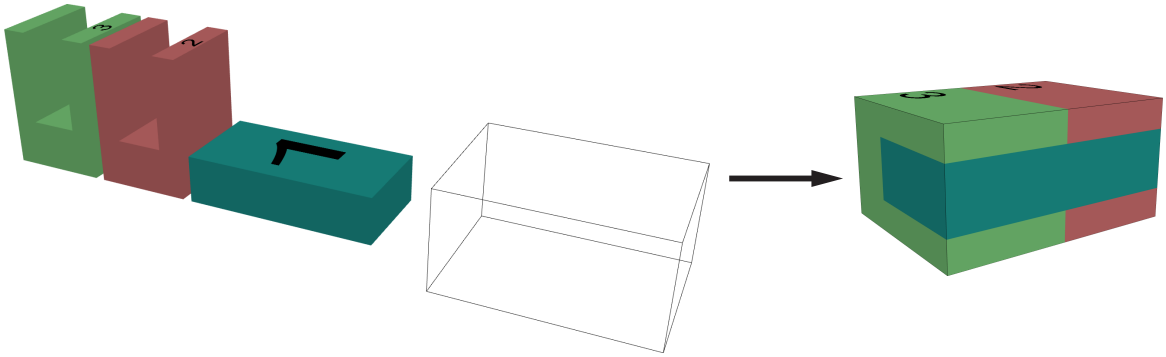
Figure E.3 contains four sample instances, and the solutions to them built by the EPI algorithm, intended to show the opportunities of handling "Tetris"-pieces. In every of the four depicted cases it would not be possible to allocate all pieces to the container, if only their bounding box is handled. E.g., for the last instance only the second piece (red) would be packed. Although the instances are synthetic they express the additional possibilities when respecting more complex shapes.



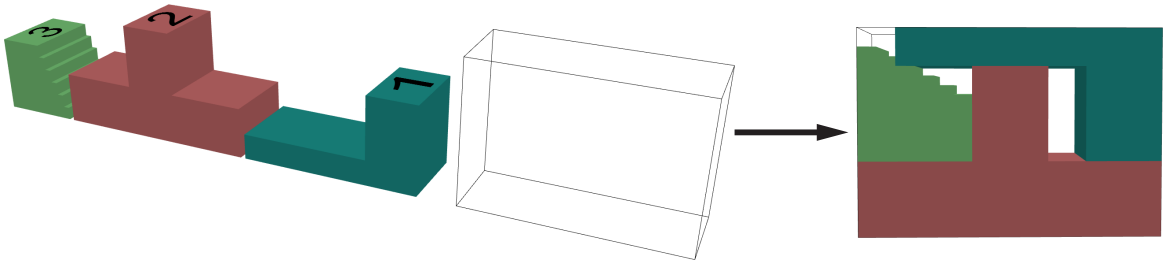
(a) Mixed instance with “L”- and “U”-shapes



(b) Mixed instance with two approximated tubes



(c) Possibility of packing one piece completely into others



(d) Showcase of an approximated slant

Figure E.3: Prototype screenshots depicting opportunities due to “Tetris”-pieces

Appendix F

Content of the disc

This section gives a summary of the content included on the disc, which is appended to this document.

Material Contains additional files like images and evaluation results

- **Evaluation** Contains all log-files underlying the computational results
- **Graphics** Contains all image files used in this work
- **Screenshots** Contains some screenshots of the prototype

Sources Contains all source files of the implementation

- **CO.Communication** The communication implementation used for a remote evaluation conducted on multiple machines
- **CO.Evaluation** The implementation of the evaluation execution routines
- **CO.EvaluationServer** Defines an executable program starting the remote evaluation server
- **CO.ExecutionHandler** Defines an executable program used by the remote evaluation
- **CO.GUI** The implementation of the GUI of the prototype
- **CO.Heuristics** The implementation of the heuristic methods
- **CO.ObjectModel** The implementation of the object model defining the main elements
- **CO.Playground** Defines an executable program used to adapt to the parameter-tuning tool SMAC
- **CO.Toolbox** Implementation of different helper-methods
- **CO.Transformer** The implementation of the procedures transforming the object model into a solver representation of the problem
- **Material** Contains some additional material like problem instances and the actual SMAC configurations
- **CargoOptimizer.sln** The main solution-file

Thesis Contains the LaTeX-sources of this thesis

- **Literature** The near-complete collection of literature in PDF-format used by this thesis
- **media** The figures of this work
- **text** The texts of this work

Prototype.zip A ZIP-file containing an executable version of the prototype

Thesis.pdf This document in PDF-format