Constraint Order Packing

Nikolaus Furian

6.Mai 2011

Doctoral Dissertation
Departement of Engineering and Business Informatics
Graz University of Technology, Austria

Declaration

I declare that this dissertation is my own work, based on my original research and expressed in my own words. Any use made within it of works of others in any form (e.g., ideas, figures, text and tables) is properly acknowledged at this point of use. I have not submitted this thesis for any other course or degree.

Graz, Mai 2010 *Nikolaus Furian*

Acknowledgments

I would like to thank all those who contributed to this dissertation by their positive support, their feedback and their constructive suggestions. First and foremost I would like to thank Prof. Vössner for his guidance and his support. I am grateful for a number of discussions with Prof. Vössner that helped to design the presented algorithms and frameworks.

I also want to thank my colleagues for making the institute a great place to work and Ramona Lichtenegger for proof reading.

I am very thankful to the company SAA Engineering and especially all the people working on the project that funded my dissertation. During my various visits in Vienna they became more than partners.

Finally, but most important, I want to thank my family, for their support during my whole life, Sabine Woschitz, for her support, patience and love over the last years and my closest friends, Philipp Glatz, Vera Jüttner, Matthias Kohlhauser, Andreas Maderbacher, David Maierhofer and Carl Suppan, for making me who I am.

Graz, Mai 2010 *Nikolaus Furian*

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit einem Problem der Produktionsplanung in der Betonfertigteil-Industrie. Betonfertigteile werden auf Metall-Paletten gefertigt die sequentiell verschiedene Fertigungsstationen durchlaufen. Dabei ist die Auslastung der Paletten, und somit Anordnung der einzelnen Teile von höchstem Interesse um den Gesamt-Output eines Werkes steigern zu können. Bereits Verbesserungen im prozentualen Bereich können zu signifikanten Umsatzsteigerungen führen.

Im Gegensatz zu bekannten Packungs- und Verschnittproblemen hat die Betonfertigteil - Industrie mit einer Vielzahl an Nebenbedingung zu kämpfen, die dem Problem eine neue Komplexität verleihen. Inhalt dieser Arbeit ist es dieses Problem mathematisch zu formulieren um bekannte Lösungsverfahren für Packungs- und Verschnittprobleme zu adaptieren und zu testen. Weiter wird ein für diese Problemklasse untypisches Verfahren auf das vorliegende Problem zugeschnitten. Ergebnisse zeigen, dass diese neue Methode weit bessere Resultate erzielt als Standardverfahren.

Um die diskutierten Algorithmen für praktische Anwendungen tauglich zu machen beschäftigt sich diese Arbeit auch mit geometrischen Fragestellungen die allen Packungs- und Verschnittproblemen zu Grunde liegen. Dabei wird eine Methode entwickelt die besonders auf die speziellen Bedürfnisse der Betonfertigteil-Industrie eingeht und in dieser Domäne Standard-Verfahren bezüglich Laufzeiten deutlich übertrifft.

Im Gegensatz zu den meisten bisher diskutierten Problemen weist das hier behandelte eine sehr große Zahl an Nebenbedingungen bezüglich der Verteilung der Teile und deren Position auf. Daher wird eine generelle Klassifizierung von möglicher Nebenbedingungen innerhalb Packungs- und Verschnittproblemen vorgenommen, die es ermöglicht alle bisherigen und auch zukünftig auftretenden Bedingungen einzuteilen. Weiter wird im Kontext spezieller Lösungsverfahren ein zusätzliches Framework zur Einteilung von Platzierungsregeln vorgestellt, welches es Entwicklern von Algorithmen ermöglicht eine bessere Erweiterbarkeit

und Modularität zu bewahren ohne große Einbußen bezüglich Laufzeiten in Kauf nehmen zu müssen.

Abstract

This thesis is concerned with a bin packing problem in production planning in the precast-concrete-part industry. Precast-concrete-parts are produced on metal-pallets which sequentially run through several production-stations. In doing so the occupancy rate of the pallets, and therefore the allocation of parts on the pallets, are of great interest to enhance the overall output of plants. Already minor improvements can have significant positive influence on the business volume of the producer.

In contrast to well known cutting and packing problems the precast-concrete-part industry has to face an enormous number of additional constraints, which lead to a whole new problem complexity. Part of this thesis is the formulation of mathematical representation and objective function of the problem to adapt popular algorithms for standard cutting and packing problems and to evaluate those. Furhtermore, a method which is usually used within other applications is redesigned to fit the precast-concrete-part industry. Results achieved with this new method clearly dominate results of standard procedures.

To enhance theoretical concepts to useful algorithms for practical applications some geometrical issues, underlying any cutting and packing problems, are discussed in this thesis. Thereby a method is developed which is responsive to the special requirements of the precast-concrete-part industry and clearly outperforms standard procedure within this domain.

In contrast to common cutting and packing tasks the proposed problem shows a big number of constraints regarding the allocation of parts and their layouts. Therefore a classification of possible constraints is made, allows the reader to classifyall previous and also future constraints. Furthermore in the domain of certain solving strategies, a second framework for the classification of layout-constraints is proposed. The latter enables designers of algorithms to maintain modularity and extendibility without increasing the computational effort significantly.

Contents

Chapter 1

Introduction

Various forms of bin packing and stock cutting problems have been introduced and discussed in literature over the last decades. Based on their origins, for example the paper industry, the clothing industry or engineering industries, the problems have many properties and boundary conditions. A common feature of all problems is that a set of small items has to be placed entirely within a set of large items so that they do not overlap. They vary though over additional restrictions to the shapes and dimensions of small and large items, cutting restrictions (Guillotine or Non-Guillotine cuts), allowed rotations and the existence of conflicts. The latter don't allow certain small items to be placed together in one large item. Furthermore, problems can be distinguished according to their optimization criteria. Either the set of small items, parts, that has to be placed is predefined and the minimal subset of large items, bins or stock materials has to be found, or the set of bins or stock materials is fixed and one tries to place the maximum subset of parts.

Constrained Order Packing (COP), the problem this thesis is based, is motivated by an application of the precast-concrete-part production. Informally speaking, it consists of a set of two dimensional parts that have to be allocated in a minimum number of identical rectangular bins. Furthermore, the packing has to satisfy a set of new constraints, consisting of order constraints, conflicts and restrictions on positions of parts in bins.

In the following section the motivation for this thesis is illustrated. In section 1.2 the industrial context of the problem is illustrated to give the reader a better understanding of where constraints and requirements come from. In section 1.3 a short overview of existing literature and classifications of bin packing and stock cutting problems is given. Section 1.5 demonstrates the complexity of COP and summarizes some thoughts on possibilities to compute exact solutions.

1.1 Motivation

During the planning step for production precast-concrete-part producers have to deal with a bin packing problem on a daily basis. The main aspects of this problem are a highly profit-sensitive behavior and special requirements arising from the production process, that give the problem a distinctive structure. Although there exist a couple of automatic planning tools, that also comprehend solving routines for the proposed problem, they lack in performance. In addition, the algorithmic background has not been published in a scientific way. Therefore the purposes of this thesis are to define a mathematical problem representing the main structures and needs of the concrete part industry and to develop an algorithm that fits these special needs.

The formulation of the problem is based on the observation that although different precast-concrete-part companies have different products, requirements and resulting bounding conditions, the main aspects and basic conditions or structures of the production process are identical in the whole industry. Therefore the aim is to build a mathematical model which comprehends all structure giving conditions while ignoring company specific regulations and which further can easily be adapted for practical applications. This enables the design, evaluation and comparison of different solving strategies from a scientific point of view.

Under the realistic assumption that plants are working to capacity and the constant availability of new commissions, improvements of the production planning process lead to a direct enhancement of the plants' total output. Considering the pricing of concrete parts compared to a plant's capacity, small changes in the planning quality can have a significant influence on the company's business volume. Therefore the utilization of the production line is from special interest for all producers. Unless most existing tools allow an automatic planning process, they often lack in the required performance. The outcome of this deficiency is that in many plants the planning is still done manually. In many cases manual planning lead to better solutions compared to results of existing automatic routines, but take quite a lot effort in terms of planning time. Hence a new automatic planning tool must consist of an algorithm that is able to outperform previously used tools and more importantly must be able to compute a planning that is at least as good as manually designed production plans. Further the time consumption has to be reduced significantly. Also from other fields it is known that human packers or planners with years of experience are able to produce astonishing good packings what makes this task quite challenging.

1.2 Industrial Context

As mentioned before COP is motivated by the production of concrete parts for prefabricated buildings. The production process of these parts can be briefly described as follows. After buildings have been planned with CAD-tools they are divided in wall and ceiling/floor elements. To produce these wall and ceiling elements they are further split in smaller parts that fit production requirements. In several steps parts are then produced on identical rectangular iron-bins in plants. Therefore bins sequently move through several workstations where different production steps are performed. These steps can roughly be summarized by:

1. Placing shutters on pallet to surround shape of parts
2. Placing iron-reinforcements corresponding to type of parts.
3. Placing mountparts, as for example windows or doors.
4. Pouring concrete in shuttered areas.
5. Harden concrete into special chambers.
6. Remove parts from pallet and staple them to stacks.

To ensure the producibility of parts during all steps the satisfaction of several constraints have to be satisfied. Examples are restrictions on combinations of parts on one single pallet, position of parts to the pallet or each other and the order in which parts are produced and therefore placed in bins.

Surrounding parts with shutters requires a certain space on the pallet where no other parts can be placed. At a glance on could assume that these shutters can be treated as additional parts that must be positioned relative to the part they are surrounding. This would yield to the usage of artificial enlarged parts during calculations that already contain the space for the required shutters. However in many cases it is often possible that the same shutter can be used on both sides to envelop the shape of two parts. In that case the space for one additional shutter is saved which results in a tighter packing.

Most structure giving constraints arise from requirements on the production sequence given by step 6. After production, parts are removed from bins and stapled to transport units, stacks, at so called staple stations. To ensure a smooth work flow on building sites the single parts of one stack should already be stapled in the order they are needed to erect the actual building. It is therefore crucial

that the parts are produced in the inverse order they are needed, so that the part of a stack that is needed first is the last one which is removed from a pallet. Since bins move sequently through different workstations in the plant parts lower in the stack must be placed on a preceding pallet than parts higher in the stack. Furthermore, only a limited number of staple stations are available according to space restrictions, which means that only a limited number of parts can be produced at a time. The stacks themselves must also be produced in a certain sequence, however, for optimality reasons this order may be violated to a certain extent. Figure 1.1 illustrates the ordering of parts.

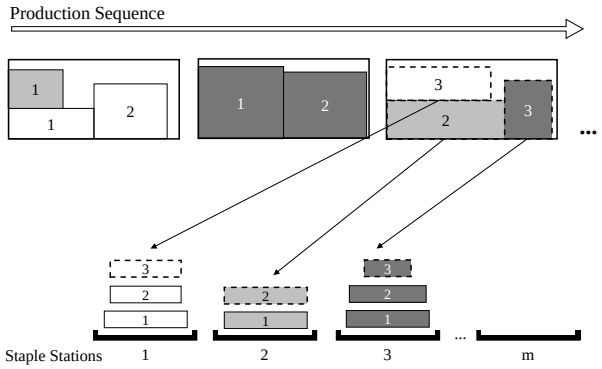

Figure 1.1: The structure of COP.

Beside these order constraints many other conditions evolve from the production process. These conditions vary over different plants and even different commissions of a plant. Some of them are explained more in detail in chapter 6.

For the design and evaluation of different solution approaches a simplified mathematical problem, based on bin packing, will be formulated in section 1.4. The porpuse of this problem is to model the most structure-giving constraints ba avoiding to deal with too many details.

1.3 Overview of different problem structures and existing work

In this section a short overview of different bin packing and stock cutting problems and relevant literature is given. Since there is a vast number of publications dealing with these topics the author is well aware that neither the overview of existing literature nor the classification of the different problem types is complete. For more general surveys and typologies of packing problems see for example

Dykhoff [26], Dykhoff and Finke [28] and Downsland and Downsland [25], or more recently Lodi, Martello and Vigo [50, 49] and Wäscher, Haußner and Schumann [66]. For a more detailed bibliography see, for example, Dyckhoff, Scheithauer and Terno [27].

1.3.1 Classifications

The first typology of cutting and packing (C&P) problems by Dykhoff [26] provides a classification based on four basic features. However, this typology has some inconsistencies, a misleading nomenclature and other drawbacks and was therefore extended and improved by Wäscher, Haußner and Schumann [66]. The latter define so called Basic and Variant C&P problems and classify them according to the following five criteria:

- dimensionality
- kind of assignment
- assortment of small items
- assortment of large items
- shape of small items

In the following the criteria are explained more in details as proposed by Wäscher, Haußner and Schumann [66]:

Dimensionality

Dimensionality describes the geometric dimension of large and small items. Basic problems can have *one*, *two* or *three* dimensions. Problems with more dimensions are classified as variant problems.

Kind of assignment

The distinction between basic optimization criteria is called the kind of assortment criteria. In case of *output maximization* problems one wants to place the subset of small items with maximum value in a given and fixed set of large items. Mostly the whole set of small items does not fit on the large items and criteria for the value of a subset may differ. These problems are selection problems for the set of small items.

Whereas in case of *input minimization* problems the entire set of small items can be placed in the set of large items and one wants to find the best selection of large items that is able to host all small items.

Assortment of small items

Assortment of small items describes the heterogeneity of small items, distinguishing three cases:

- *identical* small items, all items have the same size and orientation,
- *weakly heterogeneous assortment of small items*, small items can be split into a few groups of items with same size and orientation,
- *strongly heterogeneous of small items*, small items seldom have same size and orientation and therefore are not split into groups.

Assortment of large items

Wäscher, Haußner and Schumann [66] distinguish between problems where the set of large items consists of only one or several small items. In case of one large item one dimension may be variable. In case of several large items all dimensions have to be fixed and the set can be classified the same way as the assortment of small items.

Shape of small items

Two- or three dimensional problems can be classified among the shape of the small items. In the three-dimensional case shapes can be *regular* (boxes, balls, etc.) or *irregular*. In the two-dimensional case the classification can be extended by *rectangular items*, *circular items* and others.

Based on these features, Wäscher, Haußner and Schumann [66] define six Basic C&P problems, *Identical Item Packing Problem*, *Placement Problem*, *Knapsack Problem*, *Open Dimension Problem*, *Cutting Stock Problem* and *Bin Packing Problem*. COP is based on the two-dimensional *Bin Packing Problem*, or more precisely on the *Single Bin-Size Bin Packing Problem* according to Wäscher, Haußner and Schumann [66]. Therefore small items are referred to as parts and large items as bins from now on.

Due to rotation and packing constraints further classifications can be made. According to, for example, Lodi, Martello and Vigo [48] problems can further be classified whether parts may be rotated by 90° and whether guillotine cutting is

required. Guillotine cutting forces parts to be obtained by so called edge-to-edge cuts parallel to the edges of the bin. Based on these constraints four different classes of bin packing problems were defined by Lodi, Martello and Vigo [48]:

- $2BP|O|G$**:** the items cannot be rotated, they are oriented (O) and guillotine cutting is required
- $2BP|R|G$**:** the items may be rotated by 90° (O) and guillotine cutting is required
- $2BP|O|F$**:** the items are oriented and cutting is free
- $2BP|R|F$**:** the items may be rotated by 90° and cutting is free

For COP parts may be rotated by 90° and guillotine cuts are not required, therefore it is based on $2B|P|R|F$.

1.3.2 Relevant Algorithms

Algorithms to solve two-dimensional bin packing problems can be divided into three main classes: heuristics and metaheuristic approaches, approximation algorithms and exact methods. Exact branch and bound methods and lower bounds can be found in Martello and Vigo [53] and Fekete and Schepers [30]. Clautiaux, Jouglet and El Hayek [16], Boschetti and Mingozzi [9] and Dell'Amico et al [23] provide further lower bounds for the non-orientated case. For approximations algorithms see, for example, Carlier, Clautiaux and Moukrinh [15] or Caprara, Lodi and Monaci [12], for the special case of square bins see also Carlier, Clautiaux and Moukrinh [15], Correa [17] and Zhang[68].

There is a vast number of heuristic algorithms for Bin Packing problems and there are various types of approaches. Babu and Babu [4, 5] use fast heuristic methods to place the set of parts in the set of bins in a given sequence for both. Further, they search the space of sequences of parts and bins using a Genetic algorithm. Wu et al. [67] use the same sequence based idea but control the search over sequences with a Simulated Annealing algorithm. Rohlfshagen and Bullinaria [62] and Lima and Yakawa [43] try to overcome redundancies of sequence based algorithms and state new set based genetic approaches in which individuals are based on the bins themselves. Caprara and Pferschy [13, 14] state greedy based algorithms where one bin is filled at a time as good as possible according to the set of non placed parts. Although their concept is usually used for the one-dimensional bin packing problem, it can easily be adopted for the two-dimensional version. Lodi, Martello and Vigo [48, 47, 51] provide a Tabu

Search where parts in the weakest bins, in terms of occupancy, are tried to be allocated to other bins. Liu et al. [46] introduce a model for further objectives and constraints and uses particle swarm optimizations for solving these problems. An agent based approach for guillotine cutting problems can be found in Polyakovsky and M'Hallah [59]. Terashima-Marín et al. [64] define a virtual state cube where each point in the cube represents a progress state in the solving procedure. Furthermore, simple heuristics are assigned to the different state points in the cube. At each iteration in the solving procedure, the actual state is calculated and the nearest state point in the cube is determined. The heuristic assigned to this point is then used to perform the next step. The assignment of heuristics to points in the cube is controlled with the help of a Genetic algorithm by Terashima-Marín et al. [64] and a classifier system described in Terashima-Marín, Flores-Álvarez and Ross [65]. Hayek, Moukrin and Negre [41] introduce techniques for pretreatments of parts that reduce the complexity of an instance without losing generality.

Conflicts are often referred to as constraints that forbid certain parts to be placed together in one bin. Epstein and Levin [45] and Gendreau, Laporte and Semet [34] provide solving techniques for the one-dimensional bin packing problem based on so called conflict graphs and Epstein, Levin and van Stee [29] discuss square packing with conflicts.

1.4 Formulation of Constrained Order Packing

As mentioned above Constrained Order Packing (COP) is based on the production process of concrete parts. Their shapes are enclosed by iron shutters in the bins, concrete is poured into these shuttered areas and after further production steps parts are removed from bins and stapled to transport units.

The theoretical problem is defined as follows: We are given an infinite set of ordered identical rectangular bins $B = (b_k), k \in \mathbb{N}$ with length L and width W, a set of n stacks, $S = \{s_1 \dots s_n\}$, where each stack $s_i, i = 1, \dots, n$ consists of $m_i \geq 1$ ordered rectangular parts, $p_{i,1}, \dots, p_{i,m_i}$ of length $l_{i,j} \leq L$ and width $w_{i,j} \leq W$. Furthermore, part $p_{i,j}$ can be of two different types $t_{i,j} \in T = \{(t_1, t_2)\}$ and h different material qualities, $q_{i,j} \in Q = \{q_1, \dots q_h\}$. Further, let $P = \{p_{i,j} | 1 \leq i \leq n, 1 \leq j \leq m_i\}$ be the set of all parts, P_{t_1} the set of all parts of type t_1 and analog P_{t_2} the set of all parts of type t_2 for a given set S. Without loss of generality one can assume that all input data are integers. To formulate the problem and its constraints some further definitions are required.

Definition 1.4.1 (Allocation)**.** *For given sets B and S an allocation A denotes a function $A : P \to \mathbb{N}$ that allocates every part $p \in P$ to an integer k. An allocation*

A on $(B \times S)$ *is geometrically feasible if* $\forall k \in \mathbb{N}$ *all parts* $A^{-1}(k)$ *can be fully placed in bin* b_k*, without overlapping and further all parts* $p \in P_{t_1} \cap A^{-1}(b_k)$ *are placed on the left boarder of bin* b_k*. Let*

$$A_{k_{max}} = \max\left\{k \in \mathbb{N} : \left|A^{-1}(k)\right| \geq 1\right\} \tag{1.1}$$

be the number of active bins.

To obtain a *geometrically feasible* allocation parts may be rotated and no guillotine cuts are required. The condition that parts of type t_1 have to be placed on the left boarder of a bin represents the class of constraints that limit feasible positions of parts in bins.

Definition 1.4.2 (Open and closed stacks)**.** *For given sets* B*,* S *and an allocation* A *a stack* s_i *with* $1 \leq i \leq n$ *is called open at bin* b_k *if* $A(p_{i,1}) \leq k$ *and* $A(p_{i,m_i}) \geq k+1$ *and it is called closed at bin* b_k *if* $A(p_{i,m_i}) \leq k$*. Further let* O_k *be the set of all open stacks at bin* b_k *and* C_k *be the set of all closed stacks at bin* b_k*.*

An instance of COP is then given by $(B \times S \times Q \times T \times m \times ow)$, where m denotes the maximum number of open stacks and ow the opening window for stacks with $ow \geq m$.
Given an instance I of COP the problem is to find the smallest k_0 for which a *geometrically feasible* allocation A with $A_{k_{max}} = k_0$ exists that satisfies the following constraints:

Order within one stack

Looking at industrial applications stacks can be seen as transport units. After they have been transported to the construction sites parts of a stack are used in a certain order. To avoid reordering, they must be already stapled, and therefore also produced in that particular order. This means that if a part is allocated to a bin, all parts at a lower level in the stack have to be allocated to the same or an earlier bin in the production sequence. For all $p_{i,j}$ with $i = 1, \ldots, n$ and $j = 1, \ldots, m_i$,

$$p_{i,j} = k \Rightarrow \forall l < j, A(p_{i,l}) \leq k. \tag{1.2}$$

Order of stacks

Stacks have to be produced just in time according to their delivery date to ensure that they can be transported right away and to minimize storage costs. This

requires that they are closed in the given order. This constraint may be violated to some extent, that is a stack $s_i \geq ow + 1$ is allowed to be open at bin b_k if all stacks up to s_{i-ow} are closed at bin b_k. For all stacks s_i and bins b_k with $i = ow + 1, \ldots, n$ and $k \in \mathbb{N}$,

$$s_i \in O_k \Rightarrow \forall l \leq i - ow, s_l \in C_k. \tag{1.3}$$

Maximum number of open stacks

Stacks are stapled at so called staple-stations which are limited to m,

$$\forall k \in \mathbb{N}, |O_k| \leq m. \tag{1.4}$$

Only one material quality per bin

The production of parts requires that in one bin, only parts made of materials with same qualities are placed,

$$\forall k \in \mathbb{N}, \left|\left\{q_{i,j} | p_{i,j} \in A^{-1}(k)\right\}\right| = 1. \tag{1.5}$$

Definition 1.4.3 (Feasible Allocation)**.** *A geometrically feasible allocation that satisfies constraints (1.2), (1.3), (1.4) and (1.5) is called feasible.*

One may notice that two-dimensional bin packing is a special case of COP where each stack consists of only one part, the maximum number of open stacks and the stack window are equal to the number of stacks and parts have all the same material quality. In this case constraints (1.2), (1.3), (1.4) and (1.5) are always satisfied. Therefore COP is NP hard in the strong sense, since the two-dimensional bin packing problem is such.

1.5 Lower bounds and exact algorithms: a short overview and outlook

Martello and Vigo [53] and Fekete and Schepers [30] were the first ones who calculated lower bounds for $2B|P|O$ except the continuous bound. Further they introduced branch and bound algorithms that use outer and inner tree searches combined with lower bounds to solve $2B|P|O$ to optimality. Bounds for the non-oriented version $2B|P|R$ can be found in Dell'Amico, Martello and Vigo [23], Boschetti and Mingozzi [9] and more recently in Clautiaux, Jouglet and El Hayek [16].

To be able to use any of these bounds for COP they have to be modified. One possibility would be to partition the set of parts according to their material quality, calculate lower bounds for all disjunct subsets and sum them up to an overall lower bound. Although these bounds take constraint (1.5) into account, they neglect all other constraints (1.2), (1.3) and (1.4). For more details the reader is referred to section 3.3. In the following sections some thoughts on different methods for solving COP to optimality are summarized.

1.5.1 Branch and Bound

The idea presented in Martello and Vigo [53] can briefly be summarized as follows. Initially parts are sorted in non-increasing order of their area and a first incumbent solution is calculated with a fast heuristic. In an outer tree branching scheme, at each level j the jth part is allocated to all so called active bins and, if allowed, also to a new empty bin. Bins are closed when the continuous lower bound indicates that no further part of all unplaced parts can be placed in the bin.

The inner tree procedure checks if parts really fit into the bins they are allocated to, when a fast heuristic fails. Further on, the outer tree is searched depth first. Lower bounds are used to cut branches, check if for a given decision node optimality can already be obtained by applying a heuristic for the rest-problem and checking if a new decision node can be generated by allocating the jth part to a new empty bin.

If one wants to modify this branching scheme for COP, some additional aspects have to be considered. Before decision nodes are created by allocating a part to an active bin it has to be checked if this allocation is allowed, in the sense that the allocation doesn't makes it impossible to create feasible allocations of all further parts. Although at a first glance it seems that (1.5), (1.2), (1.3) and (1.4) might tighten the decision tree, one has to consider that for standard bin packing problems the order of bins is not crucial during the searching process described above. As parts and stacks are ordered in COP, the initially sorting method might be modified in a way, to still satisfy the order within a stack and the order of stacks themselves. But for COP it is crucial at which bin a stack is opened and at which it is closed since these decisions can affect the possibilities to allocate later parts. To ensure that no possibility is lost during the branching of the outer tree new decision nodes might be generated by allocating the part to new bins between already existing bins, if possible in the sense described above. This gives the outer tree a whole new complexity.

1.5.2 Constraint Programming

Solving COP to optimality using by Constraint Programming (CP) leads to serious difficulties. During the work for this thesis two problems for Constraint Programming solving engines were formulated. Basically the models differ in the abstraction level at which CP is used. The first model applies CP on the allocation-finding problem and uses heuristics to answer the question whether a set of parts can be placed in a single bin. The second one also considers geometrical issues and defines variables and corresponding domains for the position of parts. In addition to (1.5), (1.2), (1.3) and (1.4) constraints are that parts must not overlap, be fully placed in the bin and parts of type t_1 have to be placed on the left boarder of a bin. For both problems the objective was to find an allocation for all parts for a given number of bins, and to find the smallest number of bins that leads to a feasible solution to optimize the global objective.

Both models have been implemented and solved using the Choco [1] library for *Java* 1.6.0.17. While for the second model a feasible solution could not be found in reasonable time even for smallest instances, the solver was able to find a single feasible solution for problems with approximately 3 stacks, in total 20 parts and 4 bins using the first model within a couple of seconds. However, for bigger instance run times explode and therefore real world problems cannot be solved by applying this technique.

The main reason for the failure is that (1.2), (1.3) and (1.4) are not practicable for really restrictive or strong propagation steps during the CP solving process. Unless certain combinations of parts in a bin can be eliminated after placing a part in a bin by activating it in the CP algorithm, the remaining number of possibilities is too big. This results from the fact that at this stage no geometric information is available. Consider the following example: Placing the first part of a stack in a bin, or in other words activating a variable that represents this situation, does not necessarily give the opportunity to cut down the domain of other variables of this form. It cannot be ruled out that, for example, the last part of the stack is placed in the same bin, or that a stack which is on the outside of the opening window is opened at the same bin. It still might be possible that all parts from that stack fit in this particular bin, or in the second example that all necessary stacks can be closed in the considered bin. While (1.4) leads to the same issues, (1.5) has a more useful structure but is not restrictive enough for real world applications where for instances of 200 parts mostly not more than 3 different material qualities occur. Defining additional area constraints for each

[1] www.choco.sourcefourge.net

bin reduces this behavior but still does not allow efficient propagation steps.

1.5.3 Mixed Integer Program

While formulations of Mixed Integer Programs (MIP) are wide spread for common cutting and packing problems, an MIP for COP would be cumbersome. Even for standard problems MIP mostly fails to find optimal solutions with reasonable computational effort. For COP the situation gets even worse since especially (1.2), (1.3) and (1.4) lead to a huge number of additional bounding conditions.

Therefore, in the next chapter heuristic approaches to solve COP are adapted and discussed. Although the algorithm introduced in chapter 3 can be used to compute exact solutions using adapted lower bounds, the proper modeling of an exact algorithm and the question up to which problem sizes optimal solutions can be calculated within reasonable time is left for further research.

Chapter 2

Standard Heuristics for COP

In this chapter several heuristic methods for C&P problems from literature are revisited and modified to solve COP problems. These approaches are divided into two classes, sequence based- and set based algorithms. Sequence based algorithms try different sequences in that parts are placed in the bins by a fast and simple sub-heuristic. Whereas set based algorithms search for subsets of parts that may be placed in one or a couple of bins. They also use sub-algorithms to determine whether a set of parts can be placed into one or a few bins.

In the following sections sequence based and set based approaches that can be found in literature are described. Therefore two new models for handling sequences and constraints are introduced. Further, nesting heuristics are adapted for the special needs of these models and Simulated Annealing and Genetic Algorithms is used to search through the solutions space of different sequences. In section 2.2 a Tabu Search concept that loosely is based on a framework by Lodi et al. [48], a greedy search method based on Caprara and Pferschy [13] and a set based Genetic Algorithm based on the ideas from Iama and Yakawa [43] are introduced. In section 2.3 some benchmark instances are defined to test and evaluate the proposed approaches, in section 2.4 results are reported and some conclusions are drawn.

2.1 Sequence Based Algorithms

Sequence based algorithms are hybrid algorithms consisting of two heuristics. A global heuristic explores the space of sequences by which parts are placed in bins by using an additional, fast and simple sub-heuristic, a so called nesting heuristic. One may notice that the latter could also be used as a stand-alone algorithm by just pre-ordering the parts following a certain strategy and place them. They are

mostly used for problems with different bin sizes but can easily be adopted for problems with a fixed bin size.

Babu and Babu [4] define a sequence model that consists of two concepts, a set of bins and the order in which they are filled and a permutation of integer numbers, each representing one part. Finding a set of bins that is able to host all parts is crucial in this approach, but since for COP we assume that all bins are identical, it can be neglected for further discussion. Note that even parts with the same dimensions are represented by different integers.

Babu and Babu [4] also provide an nesting heuristic that transforms a given sequence into a actual layout. It is based on a bottom-left strategy and is extended to irregular shaped parts in Babu and Babu [5]. Wu et al. [67] provide an improved version of this nesting heuristic which is called IBH. Furthermore, Babu and Babu [4] use a Genetic Algorithm where strings are represented by sequences. Wu et al. [67] control the search through the space of different sequences by a Simulated Annealing algorithm.

Since ordering of parts is a key point in COP problems new extended models for sequences that consider at least some of the constraints (1.2), (1.3) and (1.4) are needed and defined in section 2.1.1. It follows a discussion of the IBH algorithm and the introduction of three nesting heuristics for COP, that are based on the IBH approach. Finally the Genetic Algorithm from Babu and Babu [4] and the Simulated Annealing algorithm from Wu et al. [67] are presented.

2.1.1 Sequence Models

With the sequence model proposes by [4] all possible permutations of parts are allowed. Since for COP a feasible allocation has to satisfy constraints (1.2), (1.3) and (1.4), this concept would be cumbersome for our problem. It would be useful if the allocation that results of placing each part in a single bin, according to the order given by a sequence, already satisfies some of the constraints (1.2), (1.3) and (1.4). Let's call this allocation *unit-allocation* of a sequence. Furhter, let's call a sequence *weakly ordered* if it's *unit-allocation* satisfies (1.2) and *strongly ordered* if it satisfies (1.2), (1.3) and (1.4). Finally, let's define two sequence-models, the SWSO (*Sequences with Strong Order*) model, where only *strongly ordered* sequences are allowed and second the SWWO (*Sequences with Weak Order*) model where only *weakly ordered* sequences are allowed.

Consider the example given by table 2.1, where $m = 2$, $ow = 3$ and the order of stacks and parts within stacks are given by the ordering in the table. Table 2.2 shows some examples for *weakly*-, *strongly*- and not ordered sequences.

Stacks	Parts
s_1	1,2,3
s_2	4,5,6
s_3	7,8,9
s_4	10,11,12

Table 2.1: Ordered stacks and parts.

	Sequences	Ordering
I	$(1, 4, 7, 10, 2, 3, 5, 6, 8, 11, 9, 12)$	*weakly ordered*
II	$(1, 4, 5, 6, 2, 7, 3, 10, 8, 11, 12, 9)$	*strongly ordered*
III	$(1, 2, 3, 4, 7, 9, 8, 5, 6, 10, 11, 12)$	not ordered

Table 2.2: Examples for ordered sequences.

One can see that sequence I is *weakly ordered* since its unit-allocation satisfies (1.2), but it is not *strongly ordered* for two reasons. First, looking at it's unit allocation one may notice that $|O_3| = 3$ since stack s_1, s_2 and s_3 are open at bin b_3. Second at bin b_4 stack s_4 is opened which violates constraint (1.3) since stack s_1 is not closed. Further, sequence III is not even weakly ordered since part 9 comes before part 8.

2.1.2 IBH Nesting Heuristic

The original nesting heuristic proposed by [4] fills bins one by one. Parts are placed according to a bottom-left (BL) strategy in the current bin. After placing a part new reference points are obtained corresponding to the left-top and right-bottom corners of the part. Depending on surrounding parts some points may be projected horizontally or vertically to be used. The obtained points are stored in a sorted list according to their bottom-left position, figure 2.1 illustrates this concept. The next part in the sequence is then placed with its bottom-left point on the first point in the list. If overlapping occurs or the part is not fully placed within the bin the next point from the list is considered, otherwise the point is deleted from the list and the algorithm moves on to the next part. If all points have been checked and no feasible one was found, an additional bin is selected to nest the remaining parts. Obviously, the drawback of this algorithm is that if no points are available for the next part a new bin is selected, neglecting the possibility that parts further down the sequence may fit in the current bin. Further, it does not consider rotations.

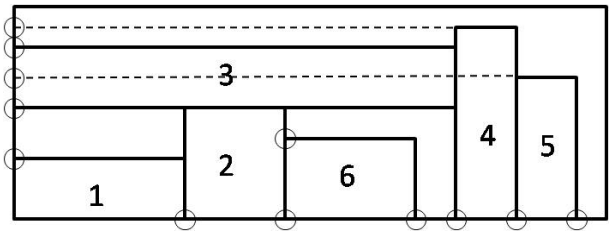

Figure 2.1: Reference points.

Wu et al. [67] propose an improved heuristic (IBH) that overcomes these drawbacks. First a preprocessing is performed where parts are ordered according to maximum area first and parts are rotated so that the longer sides are aligned to the x-axis. The list of reference points is obtained in the same way as in Babu and Babu [4]. However, if one part cannot be translated to the first point in the list without overlapping or not fully within the bin, a 90° rotation of the part is performed and it is translated again. If still no feasible placement is achieved the next part in the sequence is selected and the steps described above are repeated. If all parts have been checked and no one could be feasibly translated to the point, the latter is deleted from the list. If the list of points is empty, the next bin is selected.

In the following three nesting algorithms are proposed that perform, in different versions, on the SWSO- and SWWO models and produce feasible allocations, meaning that all requirements on the layout and constraints (1.2), (1.3), (1.4) and(1.5) are satisfied. The first is based on the IBH nesting heuristic described above, the second and third are based on first- and best fit nesting heuristics, see for example Berkey and Wang [6], combined with the placing rules from above.

2.1.3 Next Fit Nesting Heuristics

The next fit nesting heuristic (NFNH) is based on the approaches from Babu and Babu [4] and Wu et al. [67]. For the SWSO model the algorithm from Babu and Babu [4] can be used directly. However, constraint (1.5) has to be checked before a part is placed in the current bin b_k. If it is not satisfied a new bin has to be selected. To use IBH two modifications have to be done. First, no preprocessing is performed since the ordering in the sequence must not be changed. Second, if one wants to translate a part from further down the list to a reference point in the current bin, it has to be checked if a feasible allocation can be obtained by placing the part into the bin. That means that the allocation up to the current bin has

to satisfy constraints (1.2), (1.3), (1.4) and (1.5). In case that any constraint is violated the part is not considered for translation to the current reference point and the next part from the sequence is selected. Furthermore, if the current part has type t_1 only reference points on the left boarder of the bin are considered.

2.1.4 First Fit Nesting Heuristics

Several different forms of first fit nesting heuristics (FFNH) appear in literature, see for example Berkey and Wang [6]. The main idea is to place parts according to a given sequence but considering all bins used so far. Each iteration step the current part is placed in the first bin it fits or in a new bin if it does not fit in any. In the context of this work that means that a list of reference points, sorted according to the bottom-left position of reference points, for each bin used so far is kept. However, due to the special constraints of COP it may be unpossible to place the current part in the first part it fits without violating (1.2), (1.3), (1.4) or (1.5). The basic idea is to calculate a first- and a last bin to consider (FBC and LBC) for allocating the current part, neglecting geometrical constraints. It is then placed in the first bin, within this interval, it can be geometrically feasible translated to, meaning that no overlapping occurs, it is placed fully within the bin, in case the part has type t_1 it is attached to the left boarder of the bin and constraint (1.5) is not violated. If no such bin could be found a new one is selected.

SWSO:

For strongly ordered sequences the calculation of the first- and last bins to consider can be done straight forward. The last bin to consider is simply the last bin used so far, to determine the first bin to consider bins are checked backward, starting with the one that has been added last. Further, let's distinguish between parts that are first, last or middle parts in the order within their stacks. For middle parts the first considered bin is the first in which another part of the same stack is allocated. For first parts of a stack bins have to be checked whether last parts from other stacks are placed in them and the other way around. This ensures that the order of opened and closed stacks given by the strongly ordered sequence is not violated and the resulting allocation is still feasible. FBC is then

given by

$$FBC_{i,j} = \begin{cases} \max\left\{A^{-1}\left(p_{\tilde{i},m_{\tilde{i}}}\right) : 1 \leq \tilde{i} \leq n, \tilde{i} \neq i\right\} & \text{for j} = 1, \\ A^{-1}\left(p_{i,j-1}\right) & \text{for } 2 \leq \text{j} \leq \text{m}_\text{i} - 1, \\ \max\left\{\left\{A^{-1}\left(p_{\tilde{i},1}\right) : 1 \leq \tilde{i} \leq n, \tilde{i} \neq i\right\} \cup \left\{A^{-1}(p_{i,j-1})\right\}\right\} & \text{else,} \end{cases}$$

where A^{-1} is set to one for unplaced parts.

One may notice that the calculation of the first considered bin for first and last parts can be improved since the order of open and closed stacks given by the sequence can be violated and still a feasible allocation might result. However, in this approach for the SWSO model the information when stacks are opened and closed should be contained in the sequence itself. Table 2.3 shows some possible iteration steps of FFNH based on the example given by table 2.1 and sequence II from table 2.2, $(1, 4, 5, 6, 2, 7, 3, 10, 8, 11, 12, 9)$.

bins	parts allocated at step 6	parts allocated at step 9
b_1	1,4,5	1,4,5
b_2	6	6,7
b_3	2	2,3
b_4		10

Table 2.3: Iteration steps of FFNH based on the SWSO model.

At step 6 parts 1,4,5,6 and 2 have been placed in bins b_1, b_2 and b_3. The first bin to consider for the next part 7 is then bin b_2 since part 6 is the last element of stack s_2 and part 7 is the first element of stack s_3. At step 9 parts 7,3, and 10 have also been allocated, the first considered bin for part 8 is bin b_2 because placing it in bin b_1 would violate constraint (1.2).

SWWO:

Calculating first- and last bins to consider for the SWWO model is more complicated. During iterations of FFNH it might occur that constraints (1.2), (1.3) and (1.4) are violated. However, it must be secured that these errors can be corrected by placing parts from further down the sequence. Therefore it can be necessary to insert additional new bins between already activated bins and renumber all bins later than the inserted one.

Looking at sequence I in table 2.2 one may notice that after part 7 three stacks are opened, so an allocation for the first three parts must ensure that at

least one stack out of stacks s_1 and s_2 can be closed before stack s_3 is opened. In particular that means that if we assume that parts 1 and 2 are placed in bin b_1, part 7 must not be placed in bin b_1 since it might not be possible to close stack s_1 or s_2 in bin b_1. Analogously part 10 must not be placed in the same bin as part 1 because stack s_1 has to be closed before stack s_3 is opened. To ensure that errors can be corrected in later iteration steps some additional definitions and rules for calculating the first- and last bins to consider are needed. For further discussion let $(B \times S \times Q \times T \times m \times ow)$ be an instance of COP.

Definition 2.1.1 (Partial allocation)**.** *Let $\tilde{S}$ be a subset of S where each $s_i \in \tilde{S}$ consists of $\tilde{m}_i$, with $0 < \tilde{m}_i \leq m_i$, parts $p_{i,1} \dots p_{i,\tilde{m}_i}$. An allocation $\tilde{A}$ on the subproblem $(B \times \tilde{S})$ is called a partial allocation. For a partial allocation $\tilde{A}$ a stack $s_i \in \tilde{S}$ is called pseudo closed at bin b_k if $A(p_{i,\tilde{m}_i}) \leq k$ and $\tilde{m}_i < m_i$, further, let $\tilde{C}_k$ be the set of all pseudo closed stacks.*

Note that the definition for closed and open stacks stays the same for partial allocations. Furthermore, for each bin b_k a set CB_k of stacks that have to be closed before b_k is kept and updated after every iteration where a part $p_{i,1}$ with $i \in \{1, \dots, n\}$ has been placed. Given a partial allocation $\tilde{A}$ the first bin - $FBC_{i,j}$ and the last bin $LBC_{i,j}$ to consider for parts $p_{i,j}$ with $i \in \{1, \dots, n\}$ and $j \geq 2$ are given by

$$FBC_{i,j} = \tilde{A}(p_{i,j-1}) \tag{2.1}$$

and

$$LBC_{i,j} = \min\left\{\{k | s_i \in CB_k\} \cup \left\{\tilde{A}_{k_{max}} + 1\right\}\right\} - 1. \tag{2.2}$$

$LBC_{i,j}$ is the bin before the first bin where stack s_i has to be finished. Figure 2.2 illustrates FBC and LBC for a part $p_{i,j}$ with $j \geq 2$.

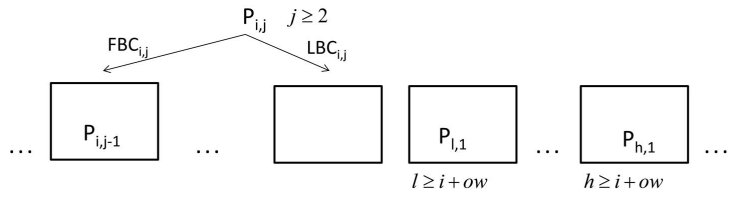

Figure 2.2: FBC and LBC for $p_{i,j}$ wiht $j \geq 2$.

To calculate $LBC_{i,1}$ for a part $p_{i,1}$ with $i \in \{1, \dots, n\}$ formula (2.2) still can be used, but (2.1) has to be changed to

$$FBC_{i,1} = \min\left\{k | \forall l \leq j - ow \wedge s_l \in \tilde{S}, s_l \in \tilde{C}_{k-1} \vee s_l \in C_k\right\}, \tag{2.3}$$

where $\tilde{C}_{-1}, C_{-1} = \{\}$. It is the first bin where all stacks, that must be finished before stack s_i, are at least *pseudo closed* at the previous bin, as also shown in figure 2.3.

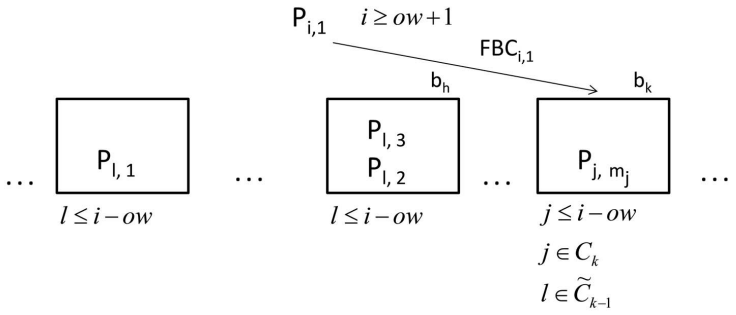

Figure 2.3: FBC for $p_{i,1}$.

One may note that $LBC_{i,1}$ might be negative or $FBC_{i,j}$ might be bigger than the number of active bins. In that case a new bin has to activated either before the first active bin or after the last active bin. If $FBC_{i,1} > LBC_{i,1}$ a new active bin has to be inserted after bin $LBC_{i,1}$. After calculating $FBC_{i,j}$ and $LBC_{i,j}$ all bins b_k with $FBC_{i,j} \leq k \leq LBC_{i,j}$ are checked on the following conditions:

1. All parts in bin b_k must have the same material quality as part $p_{i,j}$.
2. If placing a part $p_{i,1}$ then
$$|O_k| - \left|\tilde{C}_{k-1}\right| < m \text{ for } k \geq 2. \tag{2.4}$$
3. If placing a part $p_{i,1}$ and bin b_k consists of a part p_{l,m_i} then
$$\{1, \ldots, i - ow\} \subset \left\{\tilde{C}_{k-1} \cup C_k\right\}. \tag{2.5}$$
4. If placing a part p_{i,m_i} and bin b_k consists of a part $p_{l,1}$ with $l \neq i$, then
$$CB_k \subset \left\{\tilde{C}_{k-1} \cup C_k\right\}. \tag{2.6}$$
5. The part can be fully placed within the bin without overlapping and in case that it has type t_1 it can be placed on the left boarder of the bin.

Part $p_{i,j}$ is then placed in the first bin between $FBC_{i,j}$ and $LBC_{i,j}$ which satisfies these conditions. If no bin can be found a new empty bin is inserted

after $LBC_{i,}$ and the part is placed in that bin. The need of the conditions stated above is explained in the following. For all considered example let's assume that $m = 3$, all parts have the same material quality and are from the same type.

Ad 1. Part $p_{i,j}$ may only be placed in bin b_k if all parts that have been previously placed in bin b_k have the same material quality.

Ad 2. During calculations it can happen that the number of open bins exceeds m. Consider the simple weakly ordered sequence where all first parts of stacks have to be placed first, e.g.

$$(p_{1,1}, p_{2,1}, p_{3,1}, p_{4,1}, \ldots) .$$

After placing part $p_{4,1}$ there are 4 stacks open which violates constraint (1.4). Therefore (2.4) ensures that even if too many stacks are opened at bin b_k, stack s_i is only opened if enough stacks are *pseudo closed* at the bin before. Thereby enough stacks can be closed before bin b_k and a feasible overall allocation is still possible. Figure 2.4 illustrates a partial allocation that might occur during calculations, let $m_1 > 2$ and $m_2 > 2$.

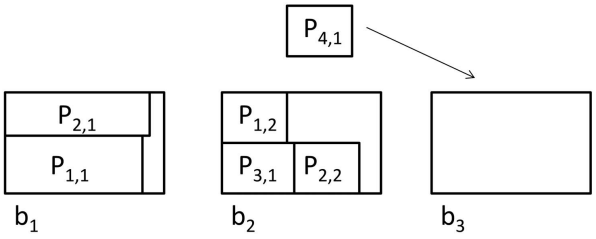

Figure 2.4: Example for (2.4).

The next part to place is $p_{4,1}$ with $FBC_{4,1} = 1$ and $LBC_{4,1} = 2$. The part does obviously not fit into bin b_1 and therefore bin b_2 has to be checked on the condition (2.4). Since $O_2 = \{s_1, s_2, s_3\}$ and $\tilde{C}_1 = \{\}$ one is not allowed to place part $p_{4,1}$ in bin b_2. None of the stacks in O_2 can be finished before bin b_2 and therefore opening s_4 would never result in a feasible allocation. A new bin has to be activated and $p_{4,1}$ is placed in latter.

Ad 3. and 4. The third and fourth conditions forbid first and last parts of stacks to be placed together in a bin unless all stacks that have to be closed before are at least *pseudo closed*. This ensures that whole stacks can be inserted between

these first and last parts, which might be necessary to satisfy (1.3). Figure 2.5 draws a simple example.

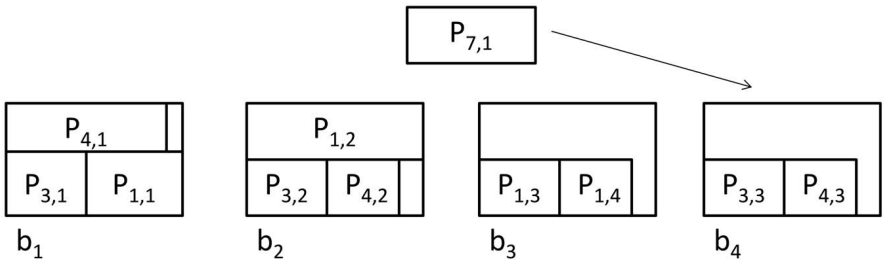

Figure 2.5: Example for (2.5).

Let's assume that $ow = 5$, $m_1 = 4$ and the next part to place is $p_{7,1}$. Then $FBC_{7,1} = 3$ and $LBC_{7,1} = 4$. Although $p_{7,1}$ fits in b_3 and s_1 is closed at b_3 one is not allowed to place $p_{7,1}$ in that bin due to the following reason: If a part does not fit in any bin between FBC and LBC it is placed in a new bin after LBC as described above. Placing $p_7, 1$ in b_3 means that $p_{2,1}$ has to be placed before that bin since s_2 has to be closed before s_7 is opened. But inserting a new bin before b_3 and placing $p_{2,1}$ in latter would result in an violation of (1.4), since 4 stacks would be open, that could not be corrected any more. The fourth condition serves the same purpose as the third the other way round.

After placing the first part of a stack s_i in bin b_k CB_k has to be updated. All stacks that must be finished before s_i according to (1.3) are inserted in CB_k. Furthermore, if the number of open stacks exceeds m, some of the *pseudo closed* stacks also have to be finished before b_k. Here the first $O_k - m$ stacks of $\tilde{C}_k$, according to the order of stacks, are chosen to be inserted in CB_k. If one places parts from table 2.1 according to sequence I from table 2.2, $(1, 4, 7, 10, 2, 3, 5, 6, 8, 11, 9, 12)$, table 2.4 shows some possible iteration steps. One may notice that stack s_1 has

b_k	step 4	CB_k	step 5	CB_k	step 7	CB_k
b_1	1		1		1	
b_2	4		4		4	
b_3	7	s_1	7,10	s_1, s_2	2,3	
b_4					7,10	s_1, s_2

Table 2.4: Iteration steps of FFNH based on the SWWO model.

to be finished before b_3 and therefore is added to CB_3. Part 10 is the next part

to place, the first considered bin is $FBC = 2$ and the last is $LBC = 3$. Placing part 10 in the third bin would result in 4 open stacks at b_3 but stack s_1 and s_2 are pseudo-closed and therefore a feasible allocation is still possible, however, s_1 and s_2 must be added to CB_3. The next part in the sequence is part 2 which is a middle part of stack s_1. The first considered bin is bin b_1 and the last is b_2 since stack s_2 is in CB_3. Let's assume that part 2 cannot be placed either in bin b_2 or b_3 due to geometrical reasons and/or different material qualities a new bin before bin b_3 has to be inserted and part 2 is placed in that bin. Let's further assume that part 3 is also placed in that bin.

2.1.5 Best Fit Nesting Heuristics

The best fit nesting heuristic (BFNH) presented works similar to the FFNH for the SWSO model from last section. It also places parts between first and last bins considered, but parts are placed in the bin they fit best. Finding the best bin for a part is a matter of evaluating all reference points in bins between FBC and LBC. Therefore a measure for the fitness of a reference point is needed. Here the aggregate length of the fraction of a part's boarder that is attached to other parts or the bin is chosen. Note that reference points can have different fitness values for the same part considered with different orientations. Figure 2.6 illustrates this fitness function. The attached fractions of part 4 are highlighted for both orientations and the same reference point.

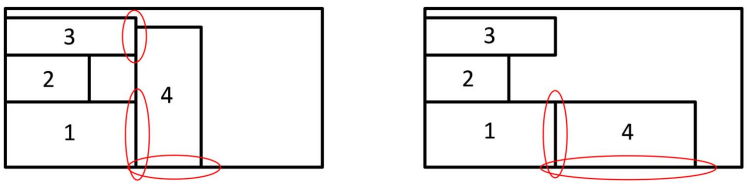

Figure 2.6: Fitness function for reference points.

2.1.6 Global Algorithms - Simulated Annealing

The Simulated Annealing algorithm was first introduced by Metropolis et al. [54] to simulate the annealing process. It is a neighborhood search meaning that at each iteration it attempts to move from the current solution to a close solution given by a defined neighborhood. Although basic neighborhood search algorithms only accept new solutions if they have better objective values than the

current solution, simulated annealing randomly allows moves to worse solutions. Therefore the difference of the objective values (ΔE) is computed. Depending on this difference and on the current temperature T the probability

$$P(\Delta E) = \exp\left(\frac{-\Delta E}{k_B T}\right) \tag{2.7}$$

for accepting the new solution is calculated, where accepting worse solutions avoids getting trapped in local optima. Temperature T starts with an high initial value T_0 and is decreasing over iterations. It stays the same for L iterations, where L is a control parameter called the Markov chain length. After L iterations it is decreased by a cooling function, which ensures that at later iterations worse solutions are more likely to be rejected. The algorithm terminates when the overall maximum number of iterations has been reached or, optionally, if for a certain number of iterations no improvement of the objective value could be obtained. Figure 2.7 shows the flow chart of a standard Simulated Annealing algorithm.

In the context of this thesis solutions are represented by the sequences and neighborhood solutions are derived from them. Therefore Wu et al. [67] propose part exchange moves, which randomly swap two parts in the sequence, as a neighborhood operator.

$$(\mathbf{1}, 4, 7, 10, 2, 3, 5, 6, \mathbf{8}, 11, 9, 12) \tag{2.8}$$

$$(\mathbf{8}, 4, 7, 10, 2, 3, 5, 6, \mathbf{1}, 11, 9, 12) \tag{2.9}$$

(2.8) shows a possible sequence for parts $1, \ldots, 10$. Part 1 and 8 have been randomly chosen and are swapped, the resulting sequence is showed by (2.9). A new solution is then obtained by placing parts using a nesting heuristic with respect to the new sequence (2.8). Furthermore, at early iterations swaps of parts that have a big distance in the sequence are preferred whereas at later iterations adjacent parts are more likely to be subjected to a part exchange move. This ensures that solutions differ more at early iterations and later when the search is more stable, fine-tuning of sequences is performed.

For SWWO and SWSO models parts cannot be randomly swapped in the sequences since that would destroy the ordering of the latter. Therefore new operators for finding neighborhood sequences are defined in the following way: Given a sequence, two parts $p_{i,j}$ and $p_{\tilde{i},\tilde{j}}$ are randomly chosen in the way that parts from the same stack cannot be chosen at the same time and first- and last parts of stack are more likely chosen for the SWSO model. Further, let $seq_{i,j}$ and

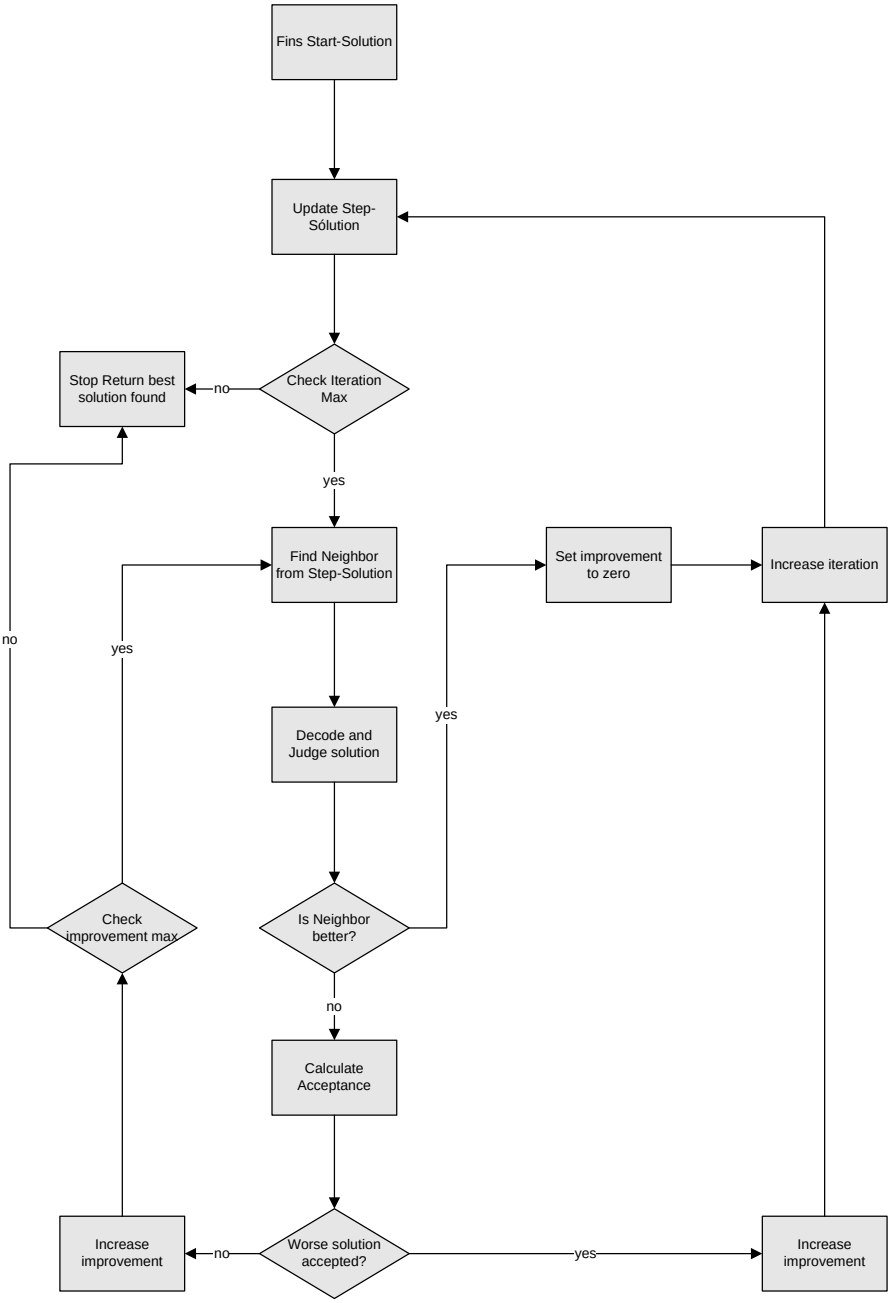

Figure 2.7: Simulated Annealing.

$seq_{\tilde{i},\tilde{j}}$ be their positions and $dist = seq_{i,j} - seq_{\tilde{i},\tilde{j}}$ the distance between them in the current sequence. Note that without loss of generality we can assume that $seq_{i,j} \leq seq_{\tilde{i},\tilde{j}}$. We iteratively try to swap part $p_{i,j}$ with the following part in sequence. If the swapping would destroy the ordering of the sequence, meaning

that its unit allocation does not satisfy the required constraints any more, or *dist* swappings have been performed, we stop. If part $p_{i,j}$ and $p_{\tilde{i},\tilde{j}}$ have been swapped we try to swap part $p_{\tilde{i},\tilde{j}}$ $dist-1$ times with the previous part, otherwise *dist* swappings are attempted. Again if swapping would destroy the ordering we stop. That means that parts $p_{i,j}$ and $p_{\tilde{i},\tilde{j}}$ are exchanged as far as possible in the sequence. For *weakly ordered* sequences parts may only be swapped if they do not belong to the same stack. For *strongly ordered* sequences also (1.3) and (1.4) have to be checked when attempting to swap the last part of stack s_i with the first part of stack s_j with $i \neq j$. In particular that means that opening stack s_j before closing stack s_i is allowed due to the order of stacks and that not more than m stacks are open. To evaluate the objective value of a sequence they are decoded with a nesting heuristic described in the previous sections.

Reconsider sequence I for the example given by table 2.1 and the SWWO model. Let's assume that part 1 and 8 have been randomly selected.

$$(\mathbf{1}, 4, 7, 10, 2, 3, 5, 6, \mathbf{8}, 11, 9, 12) \tag{2.10}$$

$$(4, 7, 10, \mathbf{1}, 2, 3, 5, 6, \mathbf{8}, 11, 9, 12) \tag{2.11}$$

$$(4, 7, \mathbf{8}, 10, \mathbf{1}, 2, 3, 5, 6, 11, 9, 12) \tag{2.12}$$

First we swap part 1 with parts 4,7 and 10 and notice that it cannot be swapped with part 2, since they belong to the same stack. In the resulting sequence (2.11) we swap part 8 with parts 6, 5, 3, 2, 1, and 10 and get the resulting neighborhood sequence (2.12). Further, also reconsider sequence II for the SWSO model and let's assume that part 6 and 10 have been selected. One may notice that part 6 can be swapped with part 2, but must not be swapped with part 7, since that would yield in three open stacks. One may also notice that part 10 cannot be swapped with part 3, since s_1 has to be finished before stack s_2 is opened. (2.14) shows the resulting sequence,

$$(1, 4, 5, \mathbf{6}, 2, 7, 3, \mathbf{10}, 8, 11, 12, 9) \tag{2.13}$$

$$(1, 4, 5, 2, \mathbf{6}, 7, 3, \mathbf{10}, 8, 11, 12, 9). \tag{2.14}$$

The same cooling procedure as Wu et al. [67] is used, starting with an initial temperature T_0 and after every L iterations multiplying the current temperature with a constant α, called the cooling rate. As the initial solution the sequence that results when one adds the parts of one stack per time to the sequence is suggested. For the example given by table 2.1 the initial solution would then be $(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)$. The objective value of a solution is simply given

by the number of its active bins.

2.1.7 Global Algorithms - Genetic Algorithms

Genetic Algorithm is a guided local search technique based on principles of genetics introduced by Goldberg [36]. A population of solutions is updated at each iteration according to natural mechanics like reproduction, crossover and mutation, to reach a global optimum. Therefore solutions have to be coded as strings and decoding algorithms, which produce actual solutions from strings, have to be defined. As in nature strings, or individuals, with good objective values are more likely to be chosen for reproduction. New strings are obtained from two chosen parent strings of the current population, by applying a crossover operator. Thereby significant features of both parent strings should be inherited to the offspring. Furthermore, strings are subject to mutations, which are small random changes of their structure. Figure 2.8 shows the flow chart of a standard Genetic Algorithm.

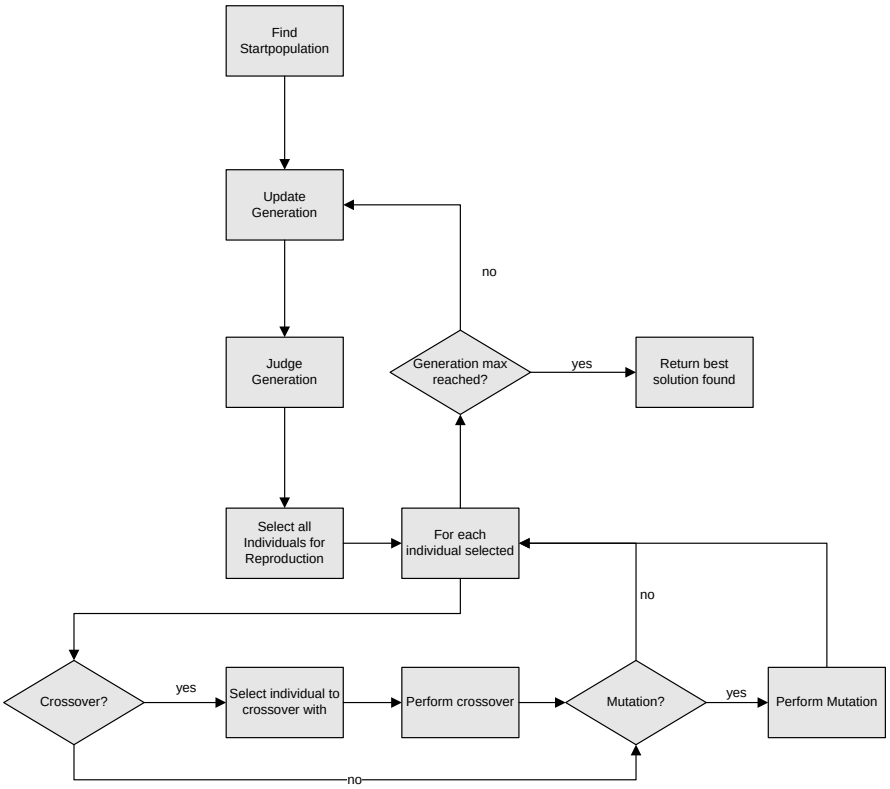

Figure 2.8: Genetic Algorithm.

Babu and Babu [4] introduce a Genetic Algorithm for bin packing problems where strings are represented by sequences of parts, same as described in the previous sections. To decode sequences to actual solutions they use the nesting algorithms described in section (2.1.2). The Genetic Algorithm proposed for COP is based on the ideas from Babu and Babu [4], although some modifications have to be performed.

Initial solution:

Sequences for the initial population are generated in the following way: For the SWWO model each sequence is created iteratively, at each iteration a stack with some parts that have not been added to the sequence yet, is chosen randomly. The first part, according to the order of parts within the stack, that has not been considered yet, is added next to the sequence. This procedure is repeated until all parts have been added to the sequence. To compute initial sequences for the SWSO model, first random *weakly ordered* sequences are creeated, decoded with the FFNH from section 2.1.4. Afterwards the obtained solution is transformed to a *strongly ordered* sequence. The fitness, or objective value, of a sequence is, as for the Simulated Annealing algorithm, the number of active bins in the decoded solution.

Selcetion:

The selection of individuals (sequences) for reproduction is done in the same way as in Babu and Babu [4], first the average and maximum objective value of all sequences (f_{avg}, f_{max}) in the current populations are calculated. The difference of the maximum value and the objective value of each sequence (f_i for the i th sequence) is then diveded by that average. The resulting value

$$p_i = \frac{f_i - f_{max}}{f_{avg}} \tag{2.15}$$

is called the probability of selecting individual i. Each sequence i with $p_i \geq 1$ is then added $\lfloor p_i \rfloor$ times to the population for reproduction. Obviously the resulting population has a smaller size than the original one and therefore the sequence with the highest decimal part of the estimated probabilities that has not been considered before, is added to the reproduction population until the size of the original population is reached.

Crossover:

The next step is to perform the crossover operator on the sequences in the reproduction population. The aim of this operator is to partially exchange information of the two parent sequences and build a new offspring based on this information. The selection of a certain crossover operator strongly depends on the specific problem. Each sequence is subjected to crossover operation with a certain probability, p_{cross}. This probabilities is usually between 0.6 and 0.9 and also dependent on the specific problem, as proposed by Goldberg [36]. Therefore a random number in the range of $[0, 1]$ for each individual in the reproduction population is generated. If this particular random number is smaller than p_{cross} the crossover operator is applied on the related sequence, otherwise it is left unchanged.

If a sequence is chosen for crossover, its mate is randomly selected from the current population. The crossover operator proposed in this work is based on the operator used by Babu and Babu [4], where the order of parts up to a randomly generated crossover site is taken from one parent sequence and the remaining parts are added in the order they appear in the other parent sequence.

Assuming that all sequences are *weakly ordered* their crossover operator can be used directly. Since the first parent sequence is *weakly ordered* the offspring sequence is weakly ordered up to the crossover site. Furthermore, since the second parent sequence is also *weakly ordered* all remaining parts are added without violating the order within stacks. Figure 2.9 shows the principle of the described operator for the SWWO model and the example from table 2.1, where underlined numbers indicate information from the first parent and bold numbers of the second parent. Further, the crossover site is selected as 5.

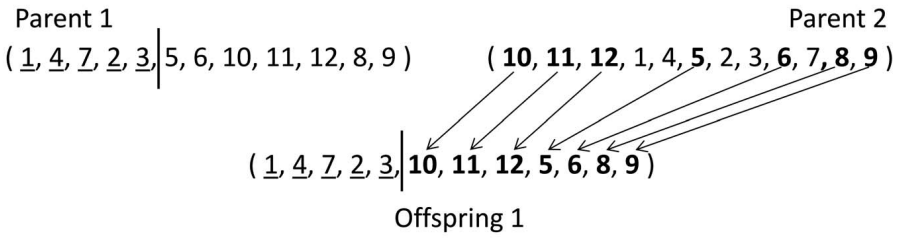

Figure 2.9: Crossover operator for SWSO.

First all parts up to part 3 from the first parent are copied to the offspring. Then all remaining parts are considered in the order given by the second parent and are added if not copied before. Therefore parts 10,11 and 12 are added next. The following parts 1 and 4 have already been copied and are neglected. The

next part to add is then part 5 and so on.

The SWSO model though requires a different crossover operator. The whole first parent sequence is copied to the offspring sequence. The parts after the crossover site are then subjected to a reordering. Iteratively all parts at positions after the crossover site, except the last, are considered for swapping with the following part. The swapping is performed if parts appear in the other order in the second parent sequence and if the offspring would maintain being *strongly ordered*. Therefore parts are not allowed to be swapped with parts from the same stack, further, (1.3) and (1.4) have to be checked. If no iteration resulted in any swapping the final offspring is found. (2.16) and (2.17) show two possible parent sequence for the example given by table 2.1 and the SWSO model. Let's assume that the crossover site was selected as 7.

$$(1, 2, 4, 5, 3, 6, \mathbf{7}, 10, 8, 11, 12, 9) \tag{2.16}$$

$$(7, 8, 9, 1, 2, 4, 3, 5, 6, 10, 11, 12) \tag{2.17}$$

The first step is to copy (2.16) to the offspring sequence. Second one may notice that part 8 comes before part 10 in (2.17) and we are allowed to swap. But part 10 cannot be swapped again since part 11 is from the same stack. Next parts to swap are 12 and 9. The offspring after the first step is shown by (2.18).

$$(1, 2, 4, 5, 3, 6, \mathbf{7}, 8, 10, 11, 9, 12) \tag{2.18}$$

In step 2 we swap part 11 with part 9 shown in (2.19). (2.20) is the final offspring sequence after swapping parts 10 and 9.

$$(1, 2, 4, 5, 3, 6, \mathbf{7}, 8, 10, 9, 11, 12) \tag{2.19}$$

$$(1, 2, 4, 5, 3, 6, \mathbf{7}, 8, 9, 10, 11, 12) \tag{2.20}$$

Mutation:

The last operator is the mutation operator. The aim of this operator is to apply minor changes to each sequence with a small probability, called the mutation probability, p_{mut}. The resulting sequence might have a better objective value or will change the sequence in a way we will benefit from in later iterations. It also overcomes that sequences in the current population become too similar and the algorithm gets trapped in a local minima. It might also occur that sequences get significantly worse and are not considered any more in the next iteration. The selection of a mutation operator and the mutation probability are problem

specific, for more detailed information see also Goldberg [36]. Babu and Babu [4] use a operator that randomly swaps two parts in a sequence and also applies 90° rotations to these parts. The mutation operator used here replaces a sequence with a neighborhood sequence obtained from the neigborhood operator described in section 2.1.6. Rotations are not applied since parts are rotated in the nesting heuristics if required.

2.2 Set Based Global Algorithms

A major drawback of sequence based approaches is that similar sequences often result in the same allocation, where set based approaches try to overcome these redundancies. They are also hybrid algorithms, where a global heuristic selects subsets of parts that are than allocated to one or more bins by a nesting algorithm, which can either be exact or heuristic. A simple, but effective example for such an approach is the SubSetSum algorithm from Caprara and Pferschy [13] which is basically a greedy algorithm for one dimensional bin packing problems. Lodi, Martello and Vigo [48] propose a tabu search that reallocates the parts placed in a sub set of bins from a current solution. Lima and Yakawa [43] try to overcome some redundancies of Genetic Algorithms based on sequences by defining crossover and mutation operators that can be applied directly to allocations.

2.2.1 SubSetSum

The SubSetSum algorithms proposed by Caprara and Pferschy [13, 14] are basically greedy methods. Bins are activated one at a time, for each bin all possible subsets of unplaced parts are computed. The problem to find all possible subsets is of course NP-hard, but, as mentioned in Caprara and Pferschy [13], still can be solved with quite low computational effort, even for large sets of parts. The best sub set that can actually be placed in one bin is then allocated to a new bin and the set of unplaced part is updated.

One may note that the number of subsets of the set of unplaced parts can be dramatically reduced for COP problems. Subsets containing parts with different material qualities don't have to be considered at all. Furthermore, only certain combinations of parts from a single stack can be allocated to a single bin. Assume that one places the first and the third, but not the second part of a stack in one single bin, then no feasible allocation can be obtained any more. Therefore one does not have to compute subsets of parts, instead ranges of unplaced parts for each stack can be combined. In addition the order of stacks must not be violated

and so parts of stack s_i with $i > ow$ have only to be considered if all stacks s_j with $j \leq i - ow$ are closed at the previous bin, or will be closed in the current bin. One may further notice that not all possible ranges must be combined. Therefore maximum ranges for each stack can be computed by adding up the areas of unplaced parts until the aggregate area exceeds the area of a bin or parts with different material qualities have been detected. If the aggregate area of parts corresponding to a certain combination exceeds the area of a bin, then no other combination that comprehends these parts has to be considered any more. This decreases the computational effort to calculate all possible subsets significantly.

Figure 2.10 shows a simple example of 5 stacks that have been allocated up to the lower lines. The upper lines indicate the maximum ranges for the current bin, for example all unplaced parts of stack s_1 would fit in the current bin. Further, the material quality and area of parts are provided in the corresponding box. Let's assume that $m = 3$, $ow = 4$ and $L \cdot W = 200$.

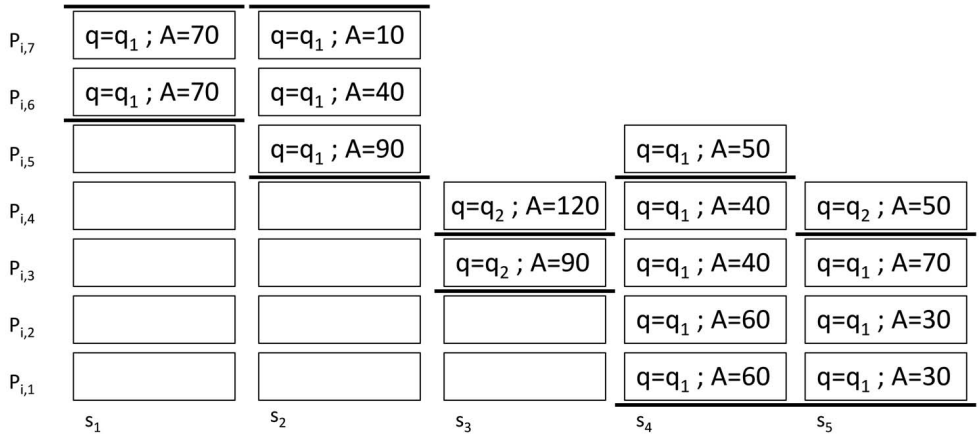

Figure 2.10: Example for computing sets of unplaced parts.

Figure 2.11 shows the search tree of computing all sets of unplaced parts for the example given by figure 2.10. Stacks are considered sequently starting with s_1. Possible ranges for stack s_1 are 0, 1 and 2. In the second step these ranges are combined with ranges of stack s_2 as there are 0, 1, 2 and 3. Note that combinations of parts from stack s_1 and s_2 can be reduced to the single set $\{p_{1,6}, p_{2,5}\}$ by checking the aggregate area of parts against the area of a bin. Furthermore, the third part of stack s_3 cannot be combined with any other part since it has a different material quality. In the next step one can see that parts of stack s_4 can only be added to combinations where either stack s_1 or s_2 has been finished, otherwise more than 3 stacks would be open. Restrictions for parts of

stack s_5 are even tighter since s_1 has to be closed before s_5 is opened.

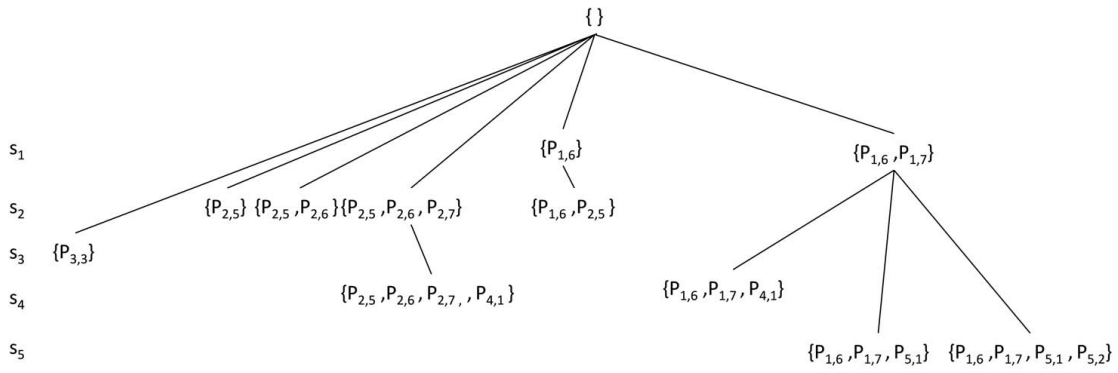

Figure 2.11: Search tree of finding possible sets of unplaced parts.

Furthermore, for two dimensional problems a nesting algorithm, that checks whether a subset of parts can be placed in a single bin, is needed. Here a sequence based heuristic that places parts on an infinite strip with the same width as the original bin is used. The goal is to find a sequence and a corresponding packing where the right most point of all parts is not further right than the length of a bin. According to the typology of Wäscher, Hauner and Schumann [66] this is a *Rectangular Strip Packing Problem*. Several publications propose sophisticated algorithms for both regular (rectangular), and irregular problems, for more details see for example Hopper and Turton [42]. Specific for this problem is that the set of parts is usually very small and computational effort is more than crucial. More general heuristics that can also deal with irregular shaped parts is introduced in chapter 4.

To place parts on the strip reference points are calculated in the same way as described for the IBH nesting heuristic in section 2.1.2. Reference points are stored in a sorted list with respect to their left-bottom position. A feasible placement of parts has been detected if all parts have been placed and the rightmost point does not exceed L. Figure 2.12 gives an example for the strip-packing method.

The fitness value of an sequence is given by the x-coordinate of the rightmost point. To guide the search through the space of different sequences with length bigger or equal to four, a 2-exchange neighborhood, as for example proposed by Gomes and Oliveira [38], and tabu lists to avoid visiting recently explored sequences are used. For general information on tabu search see e.g. Glover and Laguna [35]. For subsets consisting of less or equal than three parts we simply

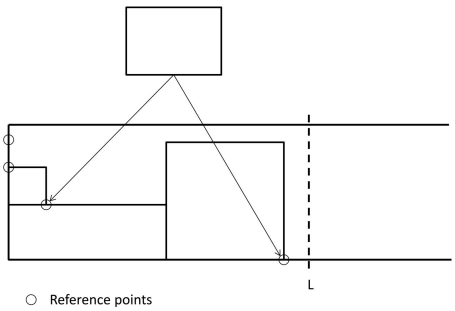

Figure 2.12: Strip packing as sub-routine.

check all possible sequences.

The SubSetSum algorithm from Caprara and Pferschy [13] is modified by Caprara and Pferschy [14]. They have observed the 'wrong' allocation of large parts cause bad solutions and even bad approximation bounds. To overcome this drawback they discuss two new versions of the SubSetSum heuristic. The *Large items first* algorithm allocates all parts that are bigger than the half of the bin capacity in separate bins. Note that in every feasible allocation these items have to be packed in separate bins anyway. In the next step all these bins are filled one by one by allocating optimal subsets of the remaining parts to them. After bins with large parts are filled, all unplaced items are packs according to the SubSetSum heuristic.

The *Largest Remaining item first* algorithm works similar as the standard SubSetSum algorithm. But instead of considering all possible subsets of unplaced parts for the current bin, only subsets containing the largest unplaced part are considered. One may notice that the *Large items first* algorithm would be cumbersome for COP problems, however, the *Largest Remaining item first* approach can be easily used for COP problems. All possible subsets of unplaced parts for the current bin are first sorted by the area of their largest part and then by their objective value, which is the degree of occupation of the bin. Subsets are then checked in that order whether their parts can actually be placed in a bin feasibly. The first feasible subset is then allocated and the next bin is activated.

2.2.2 Tabu Serach

Lodi, Martello and Vigo [47, 48, 51] propose a uniformed tabu search framework for several versions of two-dimensional bin packing problems, with or without rotations and guillotine cuts. First their framework is summarized and its dif-

ficulties when applying it to COP problems are discussed, further, a modified algorithm for COP problems, which is based on their concept is stated. A main feature of their approach is the use of a unified parametric neighborhood, whose size and structure are varied during the search. Furthermore, they use heuristic algorithms, varying over the different problem types, which actually perform neighborhood moves. These moves consist of reallocating a subset of parts from the current solution to new bins by applying a heuristic algorithm, which actually packs a subset of parts to new bins. Let H denote that heuristic, and $H(S)$ the number of bins it produces for a subset S of all parts in their approach. Moves always attempt to empty a specific target bin of the current solution. Therefore one part of the target bin and all parts of k other bins are allocated by H to new $H(S)$ bins. Note that the number of bins used for reallocating, k, defines the size and the structure of the current neighborhood. Its value automatically changes during the search and further, they keep k distinct tabu lists. The target bin is selected as the weakest bin of the current solution in terms of a filling function that considers the area occupied by parts and the number of parts in the bin. The algorithm proceeds as follows:

1. First an incumbent initial solution is computed by placing each part in a separate bin. Furthermore, a lower bound for the optimal solution is computed.

2. At each iteration a target bin is selected.

3. Given the target bin, a subroutine that considers parts in the latter one at a time is called. At each each iteration of the subroutine all possible combinations of the considered part of the target bin and the parts in k additional bins are reallocated.

4. If the number of resulting bins is less than k the move is performed immediately and the subroutine terminates.

5. If exactly k new bins result and the move is not tabu, or the weakest bin could be emptied, it is also performed immediately and the subroutine terminates.

6. In both cases k is decreased by one, the target bin is reselected and the subroutine starts again.

7. If the number of resulting new bins is equal to $k+1$ and parts in the weakest bin of all newly obtained bins and remaining parts of the current target bin

can be placed in one single bin and the move is not tabu, a penalty for that move is calculated. Otherwise the penalty for this move is set to infinity.

8. If during the whole subroutine, meaning considering all parts from the target bin and all combinations of k additional bins, no move with finite penalty has been detected, either k is increased by one or a diversity step is performed if $k = k_{max}$, otherwise the move with the smallest penalty is performed.

9. The algorithm terminates if a certain time limit has been reached or if the optimal solution has been found, which can be detected by the lower bound.

Figure 2.13 shows the flow chart of the algorithm described above, where the listed steps are also referred in the chart. Lodi, Martello and Vigo [48] propose two diversification procedures controlled by a counter d that is increased each diversification step. Instead of selecting the weakest bin for reallocating the d th weakest bin is selected. If $d = d_{max}$ or d is bigger than the number of bins of the current solution a stronger diversification step is proposed. Parts placed in the 'weaker' half of all bins are reallocated by placing each part in a separate bin and reseting all tabu lists and d. As mentioned before k_{max} tabu lists are stored, each having an own tabu tenure. For $k > 1$ the corresponding penalty values of moves are stored in the lists, for $k = 1$ the values of the filling function are stored.

Obviously selecting any k additional bins and reallocating parts in them would not work properly for COP. A way out could be to select only k neighbor bins of the weakest bin. However, using FFNH or BFNH from sections 2.1.4 and 2.1.5 to reallocate parts of them would result in too similar allocations and the search would be to narrow. Therefore each iteration the attempt to empty the current weakest bin is split in two stages:

1. In the first stage the weakest bin is determined and we select a part from the current weakest bin and try to reallocate it to any other bin using BFNH.

2. The resulting move, if not tabu, is performed and we go back to 1.

3. If all parts of the weakest bin have been considered for reallocation and there are still parts left in the weakest bin we move on to the second stage.

4. In the second stage all possible k neighbor bins of the weakest bin are considered. All parts in the neighbor bins and in the weakest bin are reallocated using the SubSetSum algorithm from section 2.2.1.

5. If the obtained solution uses less than $k+1$ bins the move is performed immediately, k is decreased and we go back to the first stage (1).

6. Same if the reallocation results in $k+1$ bins and the move is not tabu.

7. If more than $k+1$ bins result and the move is not tabu, it is still performed but k is not decreased and we go back to the first stage (1).

8. If all moves on k neighbor bins were either tabu or worse than the current bins and $k < k_{max}$, we increase k and go to the beginning of stage two again (4).

9. Else if $k = k_{max}$ and the algorithm has not reached a maximum time limit we perform a diversification step and go back to (1).

Figure 2.14 shows the flow chart from the modified tabu search algorithm.

As the initial solution also the allocation that results when placing each part in a separate bin, ordered first according to the order of stacks and second to the order within stacks is chosen. We also two different diversification steps are proposed, controlled by a counter d that is increased each time diversification is performed. For $d < d_{max}$ all parts in the d th weakest bin are placed in separate bins. If $d = d_{max}$ parts in the weaker half of the bins are placed in separate bins, furthermore, tabu lists and d are reseted.

2.2.3 Genetic Algorithm

A major drawback of the Genetic Algorithm from Babu and Babu [4], discussed in section 2.1.7, is that similar sequences often result in the same packing. These redundancies can be overcome if the crossover- and mutation operator could be applied directly on the solution, namely the bins. The latter operators for the one-dimensional bin packing problem are for example introduced by Rohlfshagen and Bullinaria [62] and Lima and Yakawa [43]. The Genetic Algorithm for COP problems presented in this section is roughly based on the ideas of Lima and Yakawa [43] and the sequence based concept from section 2.1.7. The initial population is build the same way as in section 2.1.7 by randomly generating *weakly ordered* sequences and applying the FFNH to obtain actual solutions. Also the population for reproduction is calculated in the same way as in section 2.1.7. The crossover operator for two parent allocations A_1 and A_2 works as follows. Two crossover sites, represented by stacks, are selected randomly, thereby the 'distance' of these two stacks in the order of stacks must not be smaller than the

opening window ow. Let s_i and s_j with $j - i \geq ow$ be these two stacks. Then let b_{k_i} with $k_i = A_1^{-1}(p_{i,m_i})$ be the last bin that contains a part of s_i and b_{k_j} with $k_j = A_1^{-1}(p_{j,1})$ the first bin that contains a part of s_j, in the first parent allocation. All bins up to b_{k_i} and later than b_{k_j} are copied to the offspring solution. Some parts in these bins might be from stacks between s_i and s_j. These parts are deleted from the offspring solution. All remaining and deleted parts are then placed in the order they appear in the second parent solution in bins of the offspring solution, using FFNH. Figure 2.15 shows the principle of the proposed technique.

For mutation the allocation is translated to a sequence and the mutation operator for weakly ordered sequences is applied and FFNH is used to compute a allocation from the obtained sequence.

2.3 Test data

Since, as far as the author is aware, COP is a new problem no benchmark problems exist in literature. Therefore three classes of test cases that are based on data from industrial applications are defined. Problems of class I, small problems, consists of 6 stacks and 50 parts, in particular one stack with 15 parts, 2 stacks with 10 parts and 3 stacks with 5 parts. Medium sized problems, class II problems, consists of 16 stacks and 150 parts and large problems, class III, of 28 stacks and 250 parts. Distribution of parts over stacks for class II and III problems are shown in table 2.5. We choose $m = 3, ow = 4$ for class I instance and $m = 4, ow = 8$ for class II and III. Note that the order of stacks is generated randomly. Further, distinguish

Stack sizes	Number of stacks class II	Number of stacks class III
15	3	5
10	5	7
8	3	5
7	3	5
5	2	6
Total	16	28

Table 2.5: Stacks and their sizes.

four classes of parts are distinguished. For each class lengths and widths of parts are integers and uniformly distributed over a given interval, furthermore, let's assume that $Q = \{A, B, C, D\}$. For each class the material quality of parts is uniformly distributed over a subset of Q, and parts of class PII are of type t_1 with

probability $p = 0.75$ and parts of class PIII with $p = 0.25$. The probabilities of parts being from a certain class, intervals of length and width, possible material qualities and types for the four different classes are shown in table 2.6. Each bin has length $L = 20$ and width $W = 10$. Further, 10 instances of small, medium

Classes	Probability	Length	Width	Material qualities	Types
PI	0.35	$[15, 20]$	$[1, 5]$	$\{A, B, C, D\}$	t_2
PII	0.15	$[10, 20]$	$[5, 10]$	$\{C, D\}$	t_1,t_2
PIII	0.35	$[1, 10]$	$[5, 10]$	$\{A, B, C, D\}$	t_1,t_2
PIV	0.15	$[1, 10]$	$[1, 5]$	$\{A, B\}$	t_2

Table 2.6: Classes of parts.

and large problems have been generated and all proposed algorithms are tested on them. The results and discussion are provided in the following section.

2.4 Results

To compare and discuss the different approaches introduced in this work, all heuristic algorithms are coded in Java 1.6.0_17 and implemented on a Intel(R) Core(TM)2 Duo CPU T830 @2.40 GHz 2.40Ghz PC with 2 GB RAM. Since run times of algorithms for industrial applications are very crucial a time limit of 180 seconds for each algorithm was applied. Note that, although the runtime of the SubSetSum algorithm is non controllable, all runs terminated within 180 seconds. Due to the stochastic features of the Simulated Annealing- and Genetic Algorithms 10 independent runs for each test instance have been performed and best- and average results were reported. Esed parameter sets resulted from the experience of intensive testing. The parameters chosen for the Simulated Annealing algorithms are: starting temperature, $T_0 = 75$, cooling rate, $\alpha = 0.8$ and Markov chain length, $L =$ number of parts to be placed. Furthermore, the algorithm terminates if either the time limit of 180 seconds is reached, or no improvement of the objective value appeared for 15000 iterations. Table 2.7 and 2.8 compare the best- and average bin usages for all test instances of the proposed Simulated Annealing algorithm using the SWSO and SWWO sequence models and FFNH, BFNH and NFNH heuristics.

One can see that for small instances all versions produce similar results and notable differences in performance occur only for class II and III problems. For the SWSO model we also see that NFNH performs worst in terms of average results and best results for every test instance. Further, FFNH and BFNH perform

	SWSO								
	NFNH			FFNH			BFNH		
Class×Instance	Avg.	Best	Avg. run-time(s)	Avg.	Best	Avg. run-time(s)	Avg.	Best	Avg. run-time(s)
I×1	26.2	25	3.6	27.3	27	4.1	27.2	27	7.5
I×2	23.9	23	3.6	23.7	23	38.7	21.9	21	6.7
I×3	29.5	29	4.1	29.5	29	4.1	28.6	28	6.8
I×4	27.7	27	4.0	27	27	4.4	26.1	26	6.6
I×5	25.4	23	3.3	23.4	21	4.3	23	23	7.1
I×6	26	25	4.1	25.9	25	4.1	25.9	25	6.6
I×7	26.4	25	3.6	26	26	4.0	24.1	24	6.6
I×8	23.7	23	3.9	22.2	22	4.2	21.8	21	7.3
I×9	26.3	26	4.0	26	26	4.1	26.1	26	7.1
I×10	26	25	4.1	26.1	26	4.1	25.2	25	7.4
II×1	65.3	62	12.1	63.2	60	11.7	62.6	61	24.0
II×2	70.1	67	12.7	67.6	66	12.8	67.2	66	25.2
II×3	70.1	66	12.7	65.8	64	12.0	65.7	64	24.1
II×4	73.2	71	13.0	66.7	65	12.3	66.5	64	23.2
II×5	68.8	66	13.2	63.7	60	12.1	64.3	63	24.1
II×6	72.3	70	13.5	66.5	64	12.2	67.6	63	23.9
II×7	69	66	12.8	64	62	12.4	64.7	63	24.5
II×8	71.8	70	13.4	67.1	63	12.7	66.9	64	24.7
II×9	72.4	68	13.2	69.1	67	12.2	67.6	66	23.8
II×10	67.8	65	13.4	63.8	62	12.1	63.5	62	24.7
III×1	122.9	120	23.6	112.9	110	20.4	112.1	109	38.4
III×2	119.8	114	22.5	106.5	103	20.0	106.4	105	38.3
III×3	126.7	121	24.1	116.5	111	20.7	114.5	111	40.0
III×4	114.7	112	23.0	107.7	103	20.1	106.8	103	39.4
III×5	122.3	120	22.7	111.2	107	19.7	111.6	108	37.8
III×6	123.9	121	24.5	112.7	110	20.7	112.7	110	38.1
III×7	123.3	121	24.1	115.5	112	20.2	113	109	38.5
III×8	123.1	117	22.7	112.1	109	19.9	110.9	108	37.9
III×9	119.8	117	22.2	106.6	104	19.7	107.3	104	39.4
III×10	120.8	118	22.4	108.1	102	20.3	108.7	105	39.8

Table 2.7: Average and best bin usage for Simulated Annealing using the SWSO model.

quite similar in terms of best results, but for 7 medium and 6 large instances BFNH had better average results. Also for the SWWO model FFNH and BFNH perform almost the same, but results are clearly worse than for the SWSO model. Average run times are, as expected, higher using BNFH than FFNH and NFNH. Moreover they are higher for the SWWO model.

	SWWO					
	FFNH			BFNH		
Class×Instance	Avg.	Best	Avg. run-time(s)	Avg.	Best	Avg. run-time(s)
I×1	28	28	45.2	28	28	67.2
I×2	24	24	18.3	24.9	24	96.3
I×3	29.2	29	5.4	29.8	29	15.5
I×4	27	27	5.9	27	27	16.3
I×5	22.1	22	50.9	23	23	23.1
I×6	25	25	158.6	25	25	141.2
I×7	26.1	26	31.9	24.6	24	84.4
I×8	22	22	112.7	22.1	22	138.3
I×9	26	26	67.3	26	26	52.4
I×10	26	26	5.6	25.4	25	10.4
II×1	67.8	65	65.4	67.8	66	68.5
II×2	73.8	71	83.8	69.9	67	80.3
II×3	70.1	61	55.7	68.1	66	71.1
II×4	73.6	70	78.5	74.8	71	80.3
II×5	68.7	64	40.7	66.9	65	79.5
II×6	73.9	67	64.4	72.5	68	84.2
II×7	67.3	64	47.8	67.8	66	82.9
II×8	73.1	69	74.1	73.3	72	11.1
II×9	75.4	73	66.1	73.9	69	91.7
II×10	71.6	69	134.4	71.7	67	68.7
III×1	119.6	116	156.3	120.4	114	171.9
III×2	116.3	107	146.8	117.7	114	161
III×3	133.4	125	162.3	131.3	126	177.1
III×4	116.9	112	168.1	117.4	113	171.8
III×5	121	117	165.5	119.9	113	167.7
III×6	121	116	158.2	121	117	176.8
III×7	129	126	147.7	122.4	117	161.0
III×8	120.8	115	155.7	119.4	113	157.8
III×9	117.7	111	119.7	117.8	111	157.4
III×10	116	110	158.4	118.1	113	145.9

Table 2.8: Average and best bin usage for Simulated Annealing using the SWWO model.

The parameter set for the proposed Genetic Algorithms is given by: the population size equals the number of parts to be placed, the mutation probability, $p_{mut} = 0.25$ and the crossover probability, $p_{cross} = 0.65$. The algorithm terminates if either the time limit of 180 seconds or the maximum number of iterations, 2000, has been reached. Table 2.9 summarize results for the SWSO sequence model and

table 2.10 compares the results for the SWWO model and the set based crossover operator.

	SWSO								
	NFNH			FFNH			BFNH		
Class×Instance	Avg.	Best	Avg. run-time(s)	Avg.	Best	Avg. run-time	Avg.	Best	Avg. run-time(s)
I×1	27	26	4.0	27.2	27	4.7	27.3	27	8.3
I×2	24.3	24	4.2	24	24	4.5	22.9	22	7.2
I×3	31.7	31	3.9	32.1	30	4.4	30.1	29	7.5
I×4	27.8	27	4.3	27.1	27	4.9	26	26	7.5
I×5	25.1	24	3.9	22	21	4.8	23.5	22	8.0
I×6	28.5	27	4.1	27.3	26	4.7	26.9	26	7.5
I×7	26.7	25	4.1	26.3	26	4.9	25.2	24	7.7
I×8	25.4	23	3.9	23.2	22	4.7	23	22	8.1
I×9	27.5	26	4.1	26.1	26	4.7	26.1	26	8.1
I×10	28.3	25	3.9	27.5	27	4.7	26.2	26	7.7
II×1	72.3	70	86.7	65	62	109.1	65.4	62	139.3
II×2	78.2	72	89.0	70	68	111.0	69.6	67	139.9
II×3	76.6	73	90.7	66.9	65	114.7	67.2	66	138.3
II×4	79.2	76	85.1	69.1	67	114.1	68.4	66	138.5
II×5	75.2	72	84.4	65.4	62	112.7	64.9	62	153.2
II×6	80.5	77	87.7	69.5	68	114.1	69.6	65	140.5
II×7	76.5	72	92.3	67.1	65	117.1	65.7	62	145.1
II×8	79	76	88.1	68.1	66	109.2	68	67	135.8
II×9	79.6	76	88.3	72.1	70	114.4	69.7	69	137.4
II×10	78.1	75	85.5	65.9	64	113.9	67.2	64	141.6
III×1	141.3	134	180.0	118.1	114	180.0	120.4	114	180.0
III×2	134.1	131	180.0	112.5	109	180.0	112.3	108	180.0
III×3	142.7	136	180.0	123.3	120	180.0	122.9	119	180.0
III×4	130	126	180.0	110.3	108	180.0	111.4	108	180.0
III×5	137.9	130	180.0	117.5	111	180.0	117.5	115	180.0
III×6	142.7	138	180.0	117.8	115	180.0	118.7	116	180.0
III×7	141.9	136	180.0	119.7	115	180.0	119.8	116	180.0
III×8	138.4	133	180.0	115.8	111	180.0	116	112	180.0
III×9	132.4	128	180.0	113.1	110	180.0	114.4	111	180.0
III×10	131.4	126	180.0	113.3	110	180.0	114	112	180.0

Table 2.9: Average and best bin usage for Genetic Algorithms using the SWSO model.

One can observe that for the SWSO model NFNH is also dominated, same as results for Simulated Annealing have shown. Where FFNH and BFNH work similar on medium sized instances, FFNH produces better results for large in-

	SWWO								
	FFNH			BFNH			Set based		
Class×Instance	Avg.	Best	Avg. run-time(s)	Avg.	Best	Avg. run-time(s)	Avg.	Best	Avg. run-time(s)
I×1	24.8	22	6.8	25.2	24	8.1	29	29	73.5
I×2	20.1	18	7.0	20.8	19	8.0	25.9	25	65.5
I×3	26.9	24	6.9	27.6	26	8.2	33.5	33	78.3
I×4	25.2	24	7.0	25.3	24	8.3	28	28	65.7
I×5	21.1	19	6.5	20.9	20	8.3	26.3	25	65.0
I×6	24.3	23	6.5	23.7	23	7.9	28.8	28	69.7
I×7	23.9	23	6.6	23	22	7.9	28	27	66.3
I×8	22.8	22	6.6	22.8	21	8.3	25.5	25	64.5
I×9	24.9	24	6.6	25.1	25	8.6	29.6	28	70.4
I×10	25	24	6.8	23.7	22	8.5	30.8	29	0
II×1	74.3	71	122.4	72.7	69	128.3	74.8	74	180.0
II×2	80.6	78	114.5	82	78	123.6	76.3	75	180.0
II×3	79.9	76	124.8	81.3	79	127.7	75.5	75	180.0
II×4	76.8	73	114.4	77	75	122.4	80.4	79	180.0
II×5	74.8	73	115.0	73.9	69	120.3	73.4	71	180.0
II×6	81.1	76	121.4	80.7	78	126.1	82.6	81	180.0
II×7	76	73	116.7	77.9	75	124.9	74.2	73	180.0
II×8	81.7	80	118.4	80.9	77	123.9	78.3	76	180.0
II×9	79.6	77	116.9	81.5	78	124.4	81	80	180.0
II×10	80.1	76	113.8	80.2	77	120.4	81.7	78	180.0
III×1	137.7	133	180.0	141.8	136	180.0	138.4	136	180.0
III×2	136.3	130	180.0	131	135.5	180.0	131.9	129	180.0
III×3	146.8	141	180.0	147.4	142	180.0	151.1	148	180.0
III×4	130.8	127	180.0	127	132.5	180.0	128.8	126	180.0
III×5	135.9	132	180.0	134.6	131	180.0	131.2	128	180.0
III×6	139.3	133	180.0	140.9	136	180.0	137.7	135	180.0
III×7	140.1	137	180.0	142.1	136	180.0	140.1	138	180.0
III×8	135.1	128	180.0	135.2	130	180.0	129	126	180.0
III×9	136.1	134	180.0	135.6	131	180.0	127.7	123	180.0
III×10	133.2	130	180.0	133.2	129	180.0	129.8	128	180.0

Table 2.10: Average and best bin usage for Genetic Algorithms using the SWWO model and the set based crossover operator.

stances. For eight out of ten large instances it performed same or better than BFNH in terms of average and best results. However, for the SWWO model we cannot observe that results of one heuristic are dominated by the results of the other one. Furthermore, results for the SWSO model are significantly better then for the SWWO model. Apparently crossover and mutation operators work way

better on *strongly ordered* sequences. The set based approach is in the range of the sequence based algorithm based on the SWWO model in terms of results but has notable higher run times.

Table 2.11 shows the results computed with the Tabu search-, SubSetSum- and Modified SubSetSum algorithms. The parameters for the Tabu search were chosen as follows: start value of k, $k_{start} = 23$, maximum diversification counter, $d_{max} = 10$ and the maximum number of iterations is 20000. Since these algorithms are deterministic we run them only once on each instance.

	Tabu Search		SubSetSum		Modified SubSetSum	
Class×Instance	No. of bins	run-time(s)	No. of bins	Run-time	No. of bins	run-time(s)
I×1	30	12.2	30	0.02	30	0.03
I×2	24	23.9	28	0.03	28	0.02
I×3	32	13.4	34	0.02	34	0.01
I×4	28	13.9	30	0.03	32	0.06
I×5	26	10.3	27	0.03	27	0.08
I×6	27	11.4	29	0.03	29	0.03
I×7	26	19.9	26	0.03	26	0.06
I×8	33	7.2	28	0.03	29	0.01
I×9	28	34.0	33	0.03	33	0.03
I×10	29	6.9	30	0.03	30	0.02
II×1	67	102.2	65	0.3	68	0.3
II×2	90	85.7	72	0.2	79	0.3
II×3	72	69.7	71	0.2	77	0.3
II×4	81	180.0	77	0.2	76	0.1
II×5	64	180.0	72	0.3	69	0.2
II×6	77	43.7	70	0.1	72	0.1
II×7	78	43.8	69	0.2	72	0.3
II×8	82	47.7	76	0.1	74	0.2
II×9	107	46.9	78	0.2	80	0.3
II×10	71	40.8	66	0.1	72	0.1
III×1	143	180.0	123	0.4	121	0.6
III×2	117	180.0	112	0.3	115	0.6
III×3	134	180.0	120	0.2	116	0.2
III×4	118	180.0	112	0.5	114	0.5
III×5	185	180.0	113	0.3	117	0.5
III×6	151	180.0	116	0.3	116	0.4
III×7	146	180.0	123	0.6	127	0.5
III×8	171	180.0	117	0.4	123	0.4
III×9	136	180.0	123	0.4	115	0.3
III×10	127	180.0	112	0.4	116	0.3

Table 2.11: Bin usage for Tabu Search, SubSetSum and Modified SubSetSum.

The following observations can be made: first, against intuition, the standard version of the SubSetSum algorithm performs slightly better than the modified version. One may also note that run times are very low for both versions. Further, for small and medium instances results of the Tabu search algorithm are in the range of the results of SubSetSum, for large instances though Tabu search performs far worse. Moreover the Tabu search algorithm seems to be quite unstable, meaning that it performs very badly on some particular instances.

Furthermore, one can see that sequence based algorithms perform better than set based algorithms in terms of average- and best results. For sequence based algorithms Simulated Annealing shows to be the more competitive approach. Tabu search produces very good results for a few instances and performs extraordinary poor for others. Remarkable is though that SubSetSum is indeed not that strong as Simulated Annealing but computes comparative good results with notable small run times. This shows that it uses the special problem structures to narrow the search space in an efficient way. As mentioned above the computational effort to compute all possible subsets of parts for the next bin can be reduced dramatically, compared to standard bin packing problems, by considering the additional constraints. For sequence based approaches it takes far more sophisticated techniques, as Simulated Annealing, and time to outperform SubSetSum. This lead to the design ideas for a heuristic algorithm for COP that is based on the SubSetSum algorithm but uses network-search techniques to backtrack decisions, as described in more details in the next chapter.

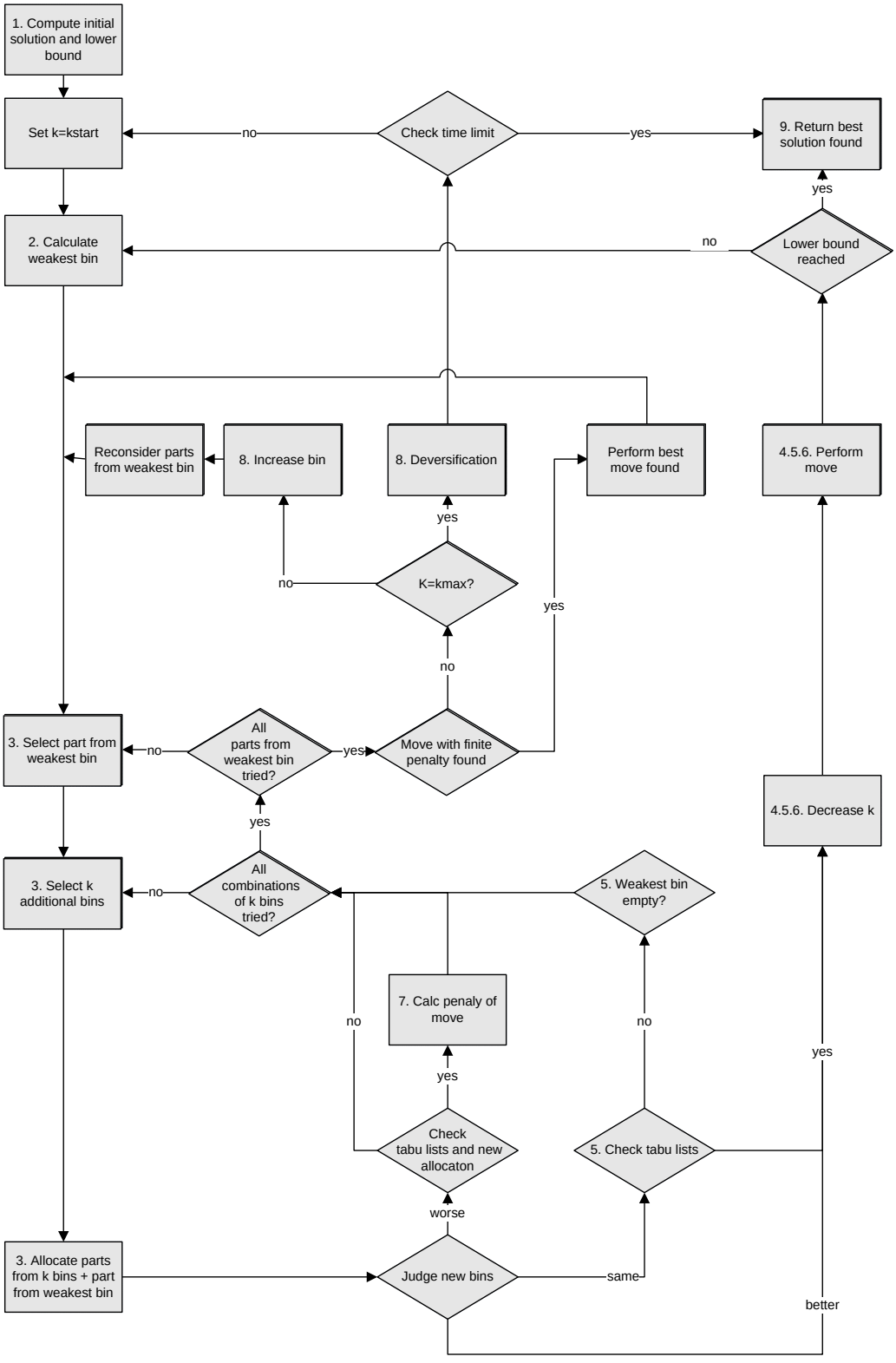

Figure 2.13: Tabu Search by Lodi et al. [48].

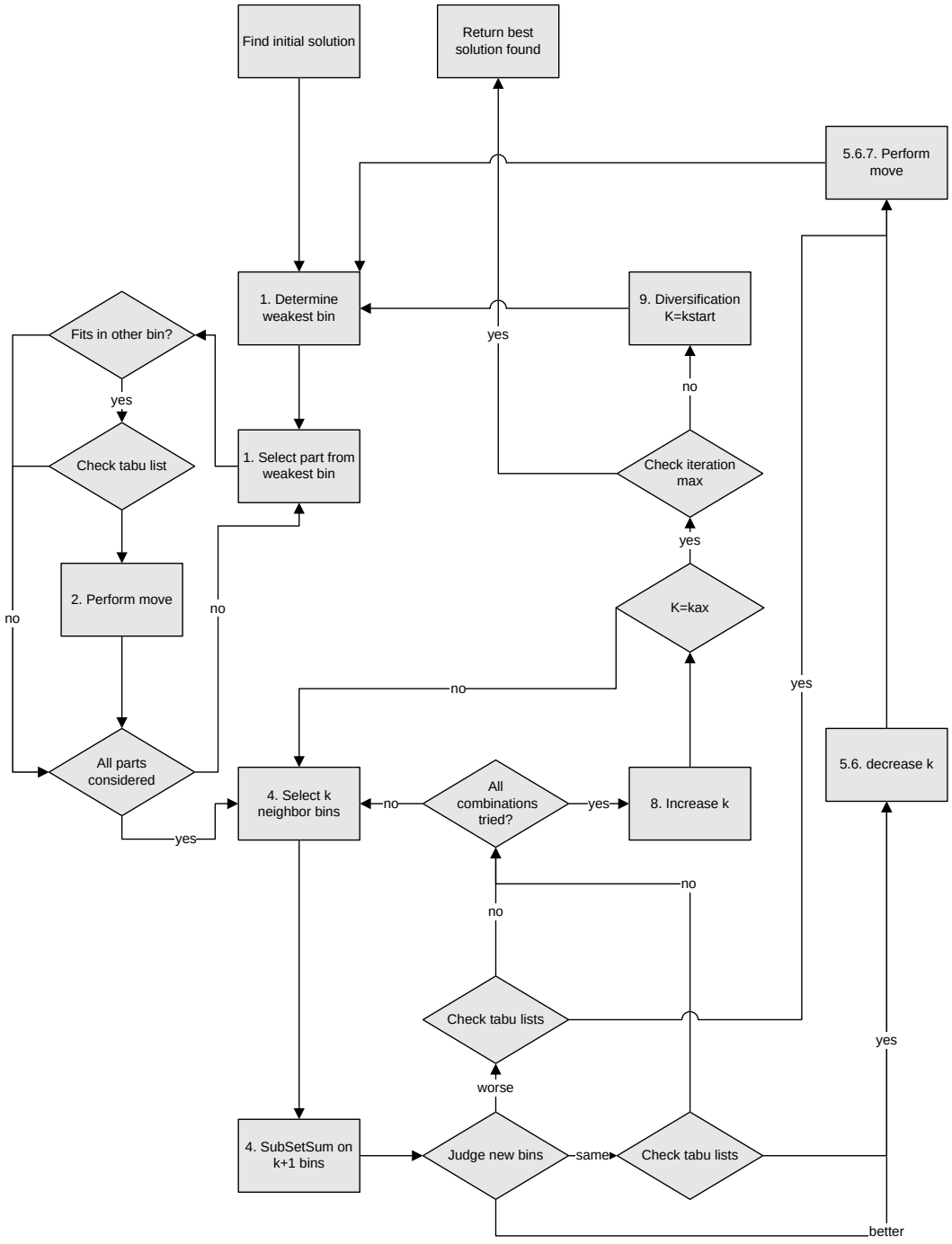

Figure 2.14: Modified tabu search.

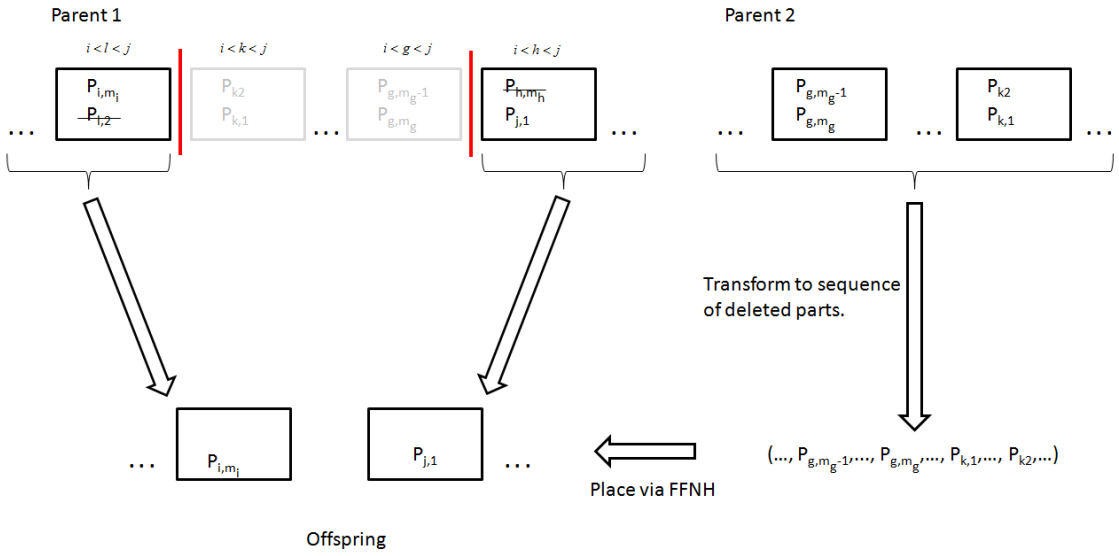

Figure 2.15: Principle of set based GA.

Chapter 3

Network Search Methods for COP

In the last chapter different approaches were classified into sequence and set based approaches and adapted to solve COP. Although Sequence based heuristics showed to be more competitive in terms of results, they were not able to benefit from the special structure of COP. Whereas SubSetSum, a simple set based greedy algorithm based on ideas from Caprara and Pferschy [13], produced comparative good results requiring small computational effort. The latter algorithm fills one bin at time by calculating all subsets of unplaced parts and allocating the best set, which can be *geometrically feasible* placed in a single bin, to an additional active bin. The advantage of this approach is that constraints (1.2), (1.3), (1.4) and (1.5) can be used to reduce the number of possible subsets of unplaced parts. In particular only combinations of ranges for each stack, instead of subsets, have to be computed when considering (1.2). Furthermore, (1.5) can be used to tighten this ranges notably and the number of ranges bigger than zero can be limited by (1.3) and (1.4).

In section 3.1 these ideas are used to construct a directed network graph where nodes represent possible allocation status of stacks during calculations. Section 3.2 provides a search algorithm on that graph, which is based on standard network search algorithms. In section 3.3 lower bounds and worst case scenarios for the proposed algorithm are discussed. In section 3.4 some techniques to speed up calculations and improve results are presented. The performance of the proposed algorithm, compared to standard heuristics, is evaluated in section 3.5.

3.1 Network Graph Representation

During each iteration of the SubSetSum algorithm all possible subsets of unplaced parts are computed. The best that can be *geometrically feasible* allocated to a single bin is then translated to a new active bin. Note that the progress of the algorithm could be measured by the set of unplaced parts, or respectively the set of already allocated parts. More precisely, at each iteration, the set of placed parts can be described by a progress vector which is defined as follows. For the following discussion let $I = (B \times S \times Q \times m \times ow)$ be an instance of COP.

Definition 3.1.1 (Progress Level)**.** *Let $\tilde{S}$ be a subset of S where each $s_i \in \tilde{S}$ consists of parts $p_{i,1} \dots p_{i,\tilde{m}_i}$, with $0 \leq \tilde{m}_i \leq m_i$. If there exists a feasible allocation $\tilde{A}$ on the sub problem $(B \times \tilde{S} \times Q \times m \times ow)$, subset $\tilde{S}$ is called a progress level of I, the vector*

$$v_{\tilde{S}} = (\tilde{m}_1, \dots, \tilde{m}_2) \tag{3.1}$$

its progress vector and

$$AP_{\tilde{S}} = \{p_{i,j} : 1 \leq i \leq n, 1 \leq j \leq \tilde{m}_i\} \tag{3.2}$$

the set of allocated parts. Further, let N be the set of all progress Levels and $\tilde{S}_1, \tilde{S}_2 \in N$. We say that $\tilde{S}_2$ dominates $\tilde{S}_1$, $\tilde{S}_1 \leq \tilde{S}_2$, if

$$v_{\tilde{S}_1,i} \leq v_{\tilde{S}_2,i} \qquad \text{for i} = 1, \dots, \text{n.} \tag{3.3}$$

Consider the directed graph $G = (N, E)$ where each node $\tilde{S} \in N$ represents one progress level of I and the set of edges, E, is given by

$$E = \left\{ \left(\tilde{S}_1, \tilde{S}_2 \right) | \tilde{S}_2 \geq \tilde{S}_1 \text{ , parts } AP_{\tilde{S}_2} \setminus AP_{\tilde{S}_1} \text{ can be placed in a single bin} \right\} \tag{3.4}$$

Note that obviously the placement of parts must satisfy all required constraints on the layout, namely that parts are placed fully within the bin, that parts do not overlap and parts of type t_1 are placed at the left boarder of the bin. Informally speaking an edge exists if and only if all parts that have been allocated at $\tilde{S}_2$ but not at $\tilde{S}_1$ can be placed in a single bin. The length of an edge e is chosen as

$$|e| = \beta - \frac{\sum_{p_{i,j} \in AP_{\tilde{S}_2} \setminus AP_{\tilde{S}_1}} l_{i,j} w_{i,j}}{LW}, \tag{3.5}$$

β minus the fraction of the unused area when placing all parts to get from $\tilde{S}_1$ to $\tilde{S}_2$ in a single bin, where β is a parameter free to choose. Obviously there is no

edge $e \in E$ with $|e| < 0$ if $\beta \geq 1$. Note that the size of G strongly depends on m, ow and most of all on the dimension of bins and parts. Further, one can see that $\tilde{S}_0 = \{\}$, the initial allocation, is the source node and $\tilde{S} = S$, the final allocation, is the sink node of G. Every feasible allocation A for I can be represented by a path from the initial allocation to the final allocation, by translating each edge to a single bin, and the other way round. The problem of finding an allocation with smallest A_{max} can then be formulated as the problem to find the shortest path from S_0 to S in G. In the next section a standard algorithm for shortest path problems is discussed and an adapted version is applied on the graph from above.

3.2 A^* Relaxation

The idea to translate Bin Packing, or more precisely stock cutting, problems to a network search problem can already be found in Albano and Sapuppo [1]. Albano and Sapuppo [1] consider the irregular Strip Packing problem where irregular shaped parts are placed on a strip with fixed width and infinite length. The goal is to find an allocation that minimizes the used length of the strip. Nodes in their network represent placed parts and their orientations. The shortest path from the initial calculation to the final allocation is obtained by the A^* algorithm. However, they conclude that the network is too large for finding optimal solutions with A^* and propose some techniques to speed up calculations. Thereby it cannot be granted that an optimal solution is found, but computational results proof that the competitive ability of the approach. Therefore next the principles of the A^* algorithm are summarized, and heuristic variants and the arising difficulties for COP are discussed. Finally some of the strategies proposed by Albano and Sapuppo [1] to speed up calculations are improved to fit COP.

3.2.1 Dijkstra's algorithm and A^*

Dijkstra's algorithm was first introduced by Dijstra [24] and computes the shortest path from a starting node v_0 to either all other nodes $v \in V$ with $v \neq v_0$ or to a destination node v_d for a directed graph $G = (V, E)$ with nonnegative length of edges $e \in E$. The two versions only differ in their termination criteria. Due to the structure of COP only the algorithm to find the shortest path between two nodes is considered. Informally speaking it consists of a set of labeled nodes LA and a set of expanded nodes O. At each iteration the node $v \in LA$ with the smallest label is selected, labels of all adjacent nodes are updated and v is put in O. The

algorithm terminates as soon as v_d is selected from LA. Let λ_v denote the label of node v, ρ_v its predecessor and $|\,(v,\tilde{v})\,|$ the length of edge $(v,\tilde{v})$. Algorithm 1 shows the pseudo code of Dijkstra's algorithm.

Algorithm 1 Dijkstra's algorithm

```
1: v = v_0;  λ_ṽ = ∞  ∀ṽ ≠ v ∈ V;  ρ_v = NULL;  LA = {v}, O = {}
2: while  v_d ∉ O do
3:    select and remove v ∈ LA with smallest λ_v and put in O.
4:    for all (v, ṽ) ∈ E do
5:       if ṽ ∉ O and λ_v + |(v, ṽ)| < λ_ṽ then
6:          λ_ṽ = λ_v + |(v, ṽ)|;  ρ_ṽ = v.
7:          if  ṽ ∉ LA then
8:             insert node ṽ in LA.
9:          end if
10:      end if
11:   end for
12: end while
```

A drawback of this algorithm is that it expands nodes regardless of the probability that they lie on the optimal path from v_0 to v_d. Consider the example where one wants to calculate the shortest path between two cities in a road network and the destination is located south of the origin. Obviously nodes north of the start node are less likely on the shortest path and therefore may not need to be expanded. Furthermore, determining the node with smallest label in L has high computational costs. Several techniques to reduce computational effort have been introduced in the past, for an general overview see for example Fu et al. [33]. Basically they can be classified whether they try to reduce the search costs for finding the node with the smallest label, by using special data structures, or if the search space of expanded nodes is tightened. As one will see, the use of special data structures, as for example buckets or threshold list, is not essential for the algorithm we propose for COP. However, since the network described in section 3.1 is huge for industrial applications, methods to cut down the number of expanded nodes are from great interest. Hart et al. [40] proposed the use of heuristic estimators $est(v, v_d)$ for the path from v to v_d. The label of a node v in algorithm 1 is substituted by $f_v = \lambda_v + est(v, v_d)$, the sum of the length of the current path from v_0 to v, the label from the algorithm 1 and the estimated value $est(v, v_d)$. This sum represents the probability of node v to be on the desired shortest path. The search algorithm based on these ideas, called A^*, selects node v with the smallest value f_v from set LA and updates its predecessors. Algorithm 2 shows the pseudo code of A^*.

Algorithm 2 A^*

1: $v = v_0;\ \lambda_{\tilde{v}} = \infty,\ f_{\tilde{v}} = \infty\ \ \forall \tilde{v} \neq v \in V;\ \ \rho_v = \text{NULL};\ \ LA = \{v\}, O = \{\}$
2: **while** $v_d \notin O$ **do**
3: select and remove $v \in LA$ with smallest f_v and put in O.
4: **for all** $(v, \tilde{v}) \in E$ **do**
5: **if** $\tilde{v} \notin O$ and $\lambda_v + |(v, \tilde{v})| + est(\tilde{v}, v_d) < f_{\tilde{v}}$ **then**
6: $\lambda_{\tilde{v}} = \lambda_v + |(v, \tilde{v})|;\ \ f_{\tilde{v}} = \lambda_v + |(v, \tilde{v})| + est(\tilde{v}, v_d);\ \ \rho_{\tilde{v}} = v.$
7: **if** $\tilde{v} \notin L$ **then**
8: insert node $\tilde{v}$ in LA
9: **end if**
10: **end if**
11: **end for**
12: **end while**

Despite to so called pruning algorithms where whole areas of the network are neglected during search, Hart et al. [40] proofed that A^* is still admissible, meaning that it always finds optimal paths, if *est* always underestimates the real lengths. In that case *est* is called conservative. For other methods to limit the search area as bi-directional search, sub goal methods or hierarchical search methods the reader is referred to Fu et al. [33].

3.2.2 An A^* Algorithm for COP

Some major problems arise when calculating a shortest path in networks describing an instance of COP. Industrial data sets often consist of more than 20 stacks and 200 parts. Obviously the corresponding networks consist of a vast number of nodes. Moreover, it would be too time consuming to check all possible edges on their existence, meaning to check if parts can be placed in a single bin. A advantage of A^* is that one does not necessarily need to know the whole network in advance. Whenever a node S_c is selected from LA, one can calculate all possibly adjacent nodes. That means that all progress steps that could be reached from S_c without violating (1.2), (1.3), (1.4) or (1.5) and the resulting edge would have length smaller than one are calculated. Let S_a be one of these possible adjacent nodes of S_s. Still there might not be n feasible allocation for parts $AP_{S_a} \setminus AP_{S_c}$ in a single bin. However, checking the existence of each edge obtained in the described way has immense computational costs and therefore is done only if required in the next steps. Furthermore, nodes are marked as checked and unchecked to avoid allocating the same set of parts more than once. For each adjacent progress level S_a the following label-update procedure is performed. If

S_a is already in O we move on to the next node. Otherwise if S_a has not been added to LA yet, we label S_a, mark it as unchecked and put it in L. For $S_a \in L$ we distinguish three cases:

- $\lambda_{S_a} > \lambda_{S_c} + |(S_c, S_a)|$
- $\lambda_{S_a} \leq \lambda_{S_c} + |(S_c, S_a)|$, with S_a unchecked
- $\lambda_{S_a} \leq \lambda_{S_c} + |(S_c, S_a)|$, with S_a checked

In the first case it is attempted to allocate parts $AP_{S_a} \setminus AP_{S_c}$ to a single bin, if possible the label and predecessor of S_a are updated and latter is marked as checked. In the second case it is attempted to allocate parts $AP_{S_a} \setminus AP_{\rho_{S_a}}$, where ρ_{S_a} denotes the current predecessor. If possible the node is marked as checked and the algorithm moves on. Else label and predecessor are updated and S_a is marked unchecked. The third case requires no action since a feasible and better path to S_a has already been found. Note that it is still an admissible search algorithm when using conservative estimators. The pseudo code of the procedure described above is provided by algorithm 3. Where ALLOC denotes an exact subroutine to allocate parts in a bin that is based on the heuristic method of section 2.2.1 but using a exact branch and bound search procedure instead of Tabu search.

The choice of a heuristic estimator *est* is crucial in the proposed approach. In section 3.3 an idea how to use lower bounds for standard Bin Packing problems for COP is provided and some results from literature are revisited. However, none of the bounds guarantee acceptable run times for real world applications. Therefore a heuristic relaxation of A^*, where run times are totally controllable, is given in the next section.

3.2.3 RA^*: A Heuristic Relaxation of A^* for COP

As mentioned above, computing optimal solutions for real world sized data sets of COP is often not possible in reasonable time. Although the use of data structures designed especially for COP reduces the computational effort significantly, the exponential complexity of the problem extinguishes these reductions for larger problems. Therefore a heuristic relaxation of A^* for COP that is based on some ideas from Albano and Sapuppo [1] is introduced. The main feature is that run times of the algorithm are controllable. The algorithm can be scaled between the exact version from last section and the SubSetSum algorithm from section 2.2.1, where obviously better solutions require more computational effort.

Algorithm 3 A^* for COP

1: $S_c = S_0;\ \lambda_{\tilde{S}} = \infty,\ f_{\tilde{S}} = \infty\ \forall \tilde{S} \neq S_c \in N;\ \rho_{S_c} = \text{NULL};\ LA = \{S_c\}, O = \{\}$
2: **while** $S \notin O$ **do**
3: set $\tilde{E} = \{\}$.
4: select $S_c \in N$ with smallest f_{S_c} and put in O.
5: calculate all S_a with $\sum_{p \in AP_{S_a} \setminus AP_{S_c}} \frac{\lambda_p w_p}{LW} \leq 1$ and put (S_c, S_a) in $\tilde{E}$.
6: **for all** $(S_c, S_a) \in \tilde{E}$ **do**
7: **if** $S_a \notin O$ **then**
8: **if** $S_a \in LA$ **then**
9: **if** $S_a > \lambda_{S_c} + |(S_c, S_a)| \ \wedge\ \text{ALLOC}(\text{AP}_{\text{S}_\text{a}} \setminus \text{AP}_{\text{S}_\text{c}})$ **then**
10: mark S_a as checked; $\lambda_{S_a} = \lambda_{S_c} + |(S_c, S_a)|$; $f_{S_a} = \lambda_{S_c} + |(S_c, S_a)| + est(S_c, S_a)$; $\rho_{S_a} = S_c$.
11: **else if** $S_a \leq \lambda_{S_c} + |(S_c, S_a)| \ \wedge\ S_a$ marked unchecked **then**
12: **if** $\text{ALLOC}\left(\text{AP}_{\text{S}_\text{a}} \setminus \text{AP}_{\rho_{\text{S}_\text{a}}}\right)$ **then**
13: mark S_a as checked.
14: **else**
15: $\lambda_{S_a} = \lambda_{S_c} + |(S_c, S_a)|$; $f_{S_a} = \lambda_{S_c} + |(S_c, S_a)| + est(S_c, S_a)$; $\rho_{S_a} = S_c$.
16: **end if**
17: **end if**
18: **else**
19: mark S_a as unchecked; $\lambda_{S_a} = \lambda_{S_c} + |(S_c, S_a)|$; $f_{S_a} = \lambda_{S_c} + |(S_c, S_a)| + est(S_c, S_a)$; $\rho_{S_a} = S_c$; insert S_a in LA.
20: **end if**
21: **end if**
22: **end for**
23: **end while**

Albano and Sapuppo [1] propose to limit LA to a fixed size, if exceeding this size nodes with worst labels are erased. Further, when the search is at the kth level only nodes with level higher than $k - t$ are expanded, where t is a given threshold and $k-t$ is called Expansion band. In the following a data structure for LA that combines and improves these ideas is defined. In addition to $\lambda_{\tilde{S}}$, $f_{\tilde{S}}$ and $\rho_{\tilde{S}}$ also the depth in the search tree, $d_{\tilde{V}}$, for each $\tilde{V}$ that is expanded during the search is stored. Instead of a simple list to store nodes in LA, t sorted lists are kept, where nodes are stored in lists according to their depth in the search tree and sorted by increasing label within a single list. Let d_{max} denote the largest depth of all nodes expanded so far, then a node $\tilde{S}$ with depth $d_{\tilde{S}} >= d_{max} - t + 1$ is stored in list $t - d_{max} + d$. Further, after all adjacent nodes of S_c have been inserted to the data-structure, lists 1 to $t-1$ are resized. Thereby the progress levels with

highest labels are removed until no more vertices than a given maximum number, α_i with $i = 1, \ldots, t-1$, are stored in list i. The maximum number of nodes is obtained by

$$\alpha_i = \frac{n_{max}}{(t-i)}, \tag{3.6}$$

where t and n_{max} are parameters free to chose. With these parameters one is able to control the total numbers of expanded nodes and thereby the runtime of the algorithm. Obviously small t, n_{max} lead to small run times and the other way around. Furthermore, the quality of obtained solutions is negative correlated to the computational effort put in. However, as already mentioned by Albano and Sapuppo [1], small changes of t have often no affect on the number of used bins. Note that since the list of nodes with the highest depth is not restricted the algorithm always terminates finding a feasible solution.

Let I be an Instance of COP. Further, let $SSS(I)$, $A^*(I)$ and $RA^*(I, t, n_{max})$ denote the number of active bins obtained by the SubSetSum-, A^*- and RA^* algorithm. Then

$$SSS(I) = RA^*(I, 1, n_{max}), \tag{3.7}$$

and

$$\lim_{t, n_{max} \to \infty} RA^*(I, t, n_{max}) = A^*(I). \tag{3.8}$$

This basically means that choosing $t = 1$ results in the SubSetSum algorithm and that for n_{max} and t chosen large enough RA^* produces optimal results. However, the non existence of approximation bounds is shown in section 3.3.2.

Since in general no optimal solution is found there is no need to use an exact method to allocate a set of parts to a single bin. Therefore the sub-routine ALLOC from the last section can be replaced by a heuristic method to save computational effort. Here the same method as in section 2.2.1 is used. A more general investigation of possible heuristic methods, that can also deal with parts of polygonal shape is given in chapter 4.

3.3 Lower Bounds and Worst Case Scenarios

As the number of expanded nodes is also dependent on the used heuristic estimator *est* and its accuracy we will revisit some lower bounds for standard Bin Packing problems and adapt them for COP. Further, the non-existence of approximation bounds for both, the SubSetSum- and RA^* algorithm is shown.

3.3.1 Lower Bounds for COP

Over the last years a couple of lower bounds for oriented and non oriented Bin Packing problems have been proposed. In general all of them could be used for COP right away. However, they do not deal with any type of conflict or ordering constraints and use only geometric information. Therefore a simple framework to tighten any bound for the non-oriented Bin Packing problem, $2PB|R|F$ according to Lodi et al. [48] is introduced by considering different material qualities of parts and the resulting conflict constraints. Let

$$P = P_{q_1} \cup \ldots P_{q_h} \quad \text{with} \quad \mathrm{P_{q_i}} \cap \mathrm{P_{q_j}} = \{\} \quad \text{for i} \neq \text{j and i, j} \in \{1, \ldots, \mathrm{h}\} \tag{3.9}$$

be a partition of parts of an Instance I of COP, where all parts of a subset P_{q_i} have the same material quality,

$$\forall\ w \in \{1, \ldots, h\}\,,\ \forall\ p_{i,j}, p_{\tilde{i},\tilde{j}} \in P_{q_w} \quad q_{i,j} = q_{\tilde{i},\tilde{j}}. \tag{3.10}$$

Further, if $LB(P)$ is a lower bound for $2PB|R|F$ then

$$LB_{\mathrm{COP}}\,(P) = \sum_{i=1}^{h} LB\,(P_{q_i}) \tag{3.11}$$

is a valid lower bound for I.

A trivial lower bound for all types of Bin Packing problems is the continuous bound that is given by

$$L_{\mathrm{C}}\,(P) = \left\lceil \frac{\sum_{p_{i,j} \in P} l_{i,j} w_{i,j}}{LW} \right\rceil. \tag{3.12}$$

Martello and Vigo [53] showed that the absolute worst-case performance of LB_{cont} is $\frac{1}{4}$ for oriented- and non-oriented standard Bin Packing problems, what in general does not hold for COP due to the additional constraints.

Martello and Vigo [53] and Fekete and Schepers [31] were the first that provided lower bounds for the oriented problem except this continuous bound. For the non-oriented problem only a few results exist in literature as for example Dell'Amico et al [23], Boschetti and Mingozzi [9] and more recently Clautiaux et al. [16]. Since dominance results of Clautiaux et al. [16] hold only for square bins results for $2PB|R|F$ are summarized and used within the framework (3.11) on instances of COP. Since some results for non-oriented problems consist of a transformation of the original instance to an instance for the oriented problem

let's first have a look on a lower bound for $2PB|O|F$ introduced by Carlier et al. [15]. Therefore let $D^R = (P \times B)$ be an instance of $2PB$, where $P = 1, \ldots, n$ is the set of parts with length l_i and width w_i and $B = L \times W$ a bin.

The lower bound L_{CCM} of Carlier et al. [15] for $2PB|O|F$

A lot of bounds for the oriented problem are obtained by applying so called *dual-feasible functions* (DFF) Johnson [44] on both dimensions of the instance and calculating the continuous lower bound for the resulting instance, see also Fekete and Schepers [32].

Definition 3.3.1. *Let f be a discrete application from $[0, X]$ to $[0, X']$ where X and X' are integers. Then f is called a discrete DFF if*

$$x_1 + \ldots + x_k \leq X \Rightarrow f(x_1) + \ldots f(x_k) \leq X'. \tag{3.13}$$

Furthermore, only DFF that are increasing and super additive are considered, i.e. if f is a given DFF, $x + y < z \Rightarrow f(x) + f(y) \leq f(z)$. Moreover Carlier et al. [15] introduce *data-dependent* DFF which have the properties of a DFF but only for a specific instance. What follows is a summary of definitions of DFF (f_0^k ,f_2^k) and DDFF (f_1^k) used by Carlier et al. [15]. Note that f_0^k is a classical family of DFF and f_2^k an improved version of latter introduced by Boschetti and Mingozzi [9]. Let C be an integer value and $k = 1, \ldots, \frac{C}{2}$ then

$$f_0^k(x) = \begin{cases} 0, & \text{if } x < k, \\ x, & \text{if } k \leq x \leq C - k, \\ C, & \text{if } C - k < x \leq C, \end{cases} \tag{3.14}$$

and

$$f_1^k(x) = \begin{cases} 0, & \text{if } x < k, \\ 1, & \text{if } k \leq x \leq \frac{C}{2}, \\ M_C(C, J) - M_C(C - x, J), & \text{if } \frac{C}{2} < x, \end{cases} \tag{3.15}$$

where f_1^k is defined for given integer values C and $c_1, \ldots, c_n$ ($I = \{1, \ldots, n\}$). Further, set J is given by $J = \left\{ i \in I : \frac{1}{2}C \geq c_i \geq k \right\}$ and $M_C(X, J)$ denotes the optimal value for the one-dimensional knapsack problem over the instance given by J and X, meaning the maximum number of items $c_i \in J$ that can be packed

in a bin of size X. Further, f_2^k is given by

$$f_2^k(x) = \begin{cases} 2\left\lfloor \frac{x}{k} \right\rfloor, & \text{if } x < \frac{C}{2}, \\ \left\lfloor \frac{C}{k} \right\rfloor, & \text{if } x = \frac{C}{2}, \\ 2\left(\left\lfloor \frac{C}{k} \right\rfloor - \left\lfloor \frac{C-x}{k} \right\rfloor\right), & \text{if } \frac{C}{2} < x, \end{cases} \tag{3.16}$$

then

$$L_{CCM}^{DDFF} = \max_{u\in\{0,1,2\},v\in\{0,1,2\}} \max_{1\le k\le W/2, 1\le l\le L/2} \left\{ \left\lceil \sum_{i\in P} \frac{f_u^k(w_i) f_v^l(l_i)}{f_u^k(W) f_v^l(L)} \right\rceil \right\}, \tag{3.17}$$

is a valid lower bound for $2PB|O|F$. L_{CCM}^{DDFF} is tightened by using the framework of Boschetti and Mingozzi [8] where four different bounds ($L_{BM}^{F,2a}$, $L_{BM}^{F,2b}$, $L_{BM}^{F,3}$ and $L_{BM}^{F,4}$) are considered. The bound first transforms the instance into a one-dimensional problem by multiplying lengths and widths of parts, the second considers *large*, *tall* and *wide* items seperatly. The third and fourth bound result from applying f_1^k and a weaker version of f_2^k on both dimensions of the problem, for more details the reader is referred to [8, 15].

The lower bound LB_{DA}^R of Dell'Amico et al. [23]

The basic idea is to transform the original instance to an instance consisting of only square parts and calculating a lower bound for latter. The square parts are obtained by an pseudo-polynomial algorithm called CUTSQ Boschetti and Mingozzi [9] that works as follows. Initially parts are considered in the orientation r where $l_i^r \ge w_i^r$. At each iteration the maximum number of squares with length w_j are cut from the current rectangle with dimensions (l_j, w_j). Afterwards the remaining rectangle is rotated by 90°. These steps are iterated as long as $w_j > 1$ of the remaining rectangle. Let M denote the set of square obtained in the way described above and λ_j the length of square $j \in M$. For a given integer $0 \le q \le \frac{1}{2}W$ M is then partitioned in the following subsets

$$\begin{aligned} S_1 &= \{j \in M : \lambda_j > L - q\} \\ S_2 &= \left\{ j \in M : L - q \ge \lambda_j > \frac{1}{2}L \right\} \\ S_3 &= \left\{ j \in M : \frac{1}{2}L \ge \lambda_j > \frac{1}{2}W \right\} \\ S_4 &= \left\{ j \in M : \frac{1}{2}W \ge \lambda_j \ge q \right\}. \end{aligned}$$

Further, let $S_{23} = \{j \in S_2 \cup S_3 : \lambda_j > W - q\}$. Then

$$LB_{DA}^{R} = \max_{0 \leq q \leq \frac{1}{2}W} \left\{ |S_1| + \tilde{L} + \hat{L} \right\}$$

is a valid lower bound for $2PB|R|F$. $\tilde{L}$ is given by

$$\tilde{L} = |S_2| + \max \left\{ \left\lceil \frac{\sum_{j \in S_3 \setminus \overline{S}_3} \lambda_j}{L} \right\rceil, \left\lceil \frac{|S_3 \setminus \overline{S}_3|}{\frac{L}{\left\lfloor \frac{W}{2}+1 \right\rfloor}} \right\rceil \right\},$$

and denotes a valid lower bound for squares in $S_2 \cup S_3$, where $\overline{S}_3$ is the set of large parts in S_3 that fit in bins where parts of S_2 have been placed. Furhter, $\hat{L}$ is given by

$$\hat{L} = \max \left\{ 0, \left\lceil \frac{\sum_{j \in S_2 \cup S_3 \cup S_4} {\lambda_j}^2 - \left(LW\tilde{L} - \sum_{j \in S_{23}} \lambda_j \left(W - q \right) \right)}{LW} \right\rceil \right\}.$$

The lower bound L_{BM}^{R} of Boschetti and Mingozzi [9]

L_{BM}^{R} is the combination of two other lower bounds $L_{BM}^{R,1}$ and $L_{BM}^{R,2}$ and is given by the maximum of these. The first, $L_{BM}^{R,1}$, reduces the problem to a oriented problem by transforming parts that might be rotated to the largest square contained. Therefore dimensions of parts are resized as follows:

$$\bar{l} = \begin{cases} \min \{l_j, w_j\}, & \text{if } l_j \leq W \text{ and } w_j \leq L \\ l_j, & \text{else} \end{cases} \tag{3.18}$$

and

$$\bar{w} = \begin{cases} \min \{l_j, w_j\}, & \text{if } l_j \leq W \text{ and } w_j \leq L \\ w_j, & \text{else} \end{cases} \tag{3.19}$$

Since, as mentioned above, orientation is not a issue for the new obtained problem, lower bounds for $2PB|O|F$ can be used to compute bounds for $2PB|R|F$. Boschetti and Mingozzi [9] propose the use of L_{BM}^{F} introduced by [8] that basically is the maximum of four different bounds. For more details the reader is referred to [8, 9]. Since L_{CCM}^{F} dominates L_{BM}^{2} let's apply the modified version of $L_{BM}^{R,1}$ where latter is used.

The second bound, $L_{BM}^{R,2}$, considers both dimension of parts and also possible rotations by 90° by them. Therefore the set of parts is divided into $P' =$

$\left\{i \in P : l_i \neq \bar{l}_i\right\}$ and $P'' = P \setminus P$. Further, let $1 \leq k \leq \frac{1}{2}W$ and $1 \leq l \leq L$, then $L_{BM}^{R,2}$ is given by

$$LB_{BM}^{R,2} = \max_{k,l} \left\{ \left\lceil \frac{\sum_{i=1}^{n} \mu^{k,l}(i)}{\left\lfloor \frac{W}{k} \right\rfloor \left\lfloor \frac{L}{l} \right\rfloor}, \right\rceil \right\} \tag{3.20}$$

where

$$\mu^{k,l}(i) = \begin{cases} \min\left\{\eta^{l,L}(l_i) \times \eta^{k,W}(w_i), \eta^{l,L}(w_i) \times \eta^{k,W}(l_i)\right\}, & \text{if } i \in P' \\ \eta^{l,L}(l_i) \times \eta^{k,W}(w_i), & \text{else,} \end{cases} \tag{3.21}$$

and

$$\eta^{k,X}(x) = \begin{cases} \left\lfloor \frac{X}{k} \right\rfloor - \left\lfloor \frac{X-x}{k} \right\rfloor, & \text{if } x > \frac{X}{2} \\ \left\lfloor \frac{x}{k} \right\rfloor, & \text{else.} \end{cases} \tag{3.22}$$

The lower bound L_{CJE}^{R} of Clautiaux et al. [16]

The idea ofClautiaux et al. [16] is to transform a given instance $I = (P \times B)$ with n parts and $B = (L \times L)$ to a new oriented problem with $2n$ parts as follows: $\hat{D}^F = (\hat{P}^F \times B)$ with $\hat{D}^F = \{1, \ldots, 2n\}$ such that

- $l_i = l_i$ and $w_i = w_i$ for $i \in \{1, \ldots, n\}$
- $l_i = w_{i-n}$ and $w_i = l_{i-n}$ for $i \in \{n+1, \ldots, 2n\}$

The main result of Clautiaux et al. [16] is that $\hat{D}^F$ needs at most twice the number of bins needed for D^R. Therefore any valid lower bound LB^F for $2PB|O|F$ can be used to obtain a valid lower bound for $2PB|R|F$ by calculating $\left\lceil LB^F(\frac{\hat{D}^F}{2}) \right\rceil$. The bound resulting from applying these scheme on L_{CCM}^{F} of Carlier et al. [15] is called L_{CJE}^{R} Clautiaux et al. [16]. However, if bins are no squares one dummy item per bin has to be added to the new instance to fill the area exceeding the largest square contained in the bin. Therefore Clautiaux et al. [16] state a lifting procedure to compute the right number of dummy items that has to be added. However, dominance results hold only for the case when bins are squares. For rectangular bins and also in the case where some parts must not be rotated L_{CJE}^{R} shows some drawbacks.

Since in general RA^* does not compute optimal solutions non-conservative bounds could also be used for calculations. To avoid time consuming calculations it is suggested in this work to use an estimator of the form

$$est = \alpha(I) L_{\mathrm{C}}, \tag{3.23}$$

where α is a function over instances of COP depending on n, m, ow and

$$\mathrm{Var}\left(\left|\mathrm{P}_{\mathrm{q_1}}\right|,\ldots,\left|\mathrm{P}_{\mathrm{q_h}}\right|\right).$$

3.3.2 Worst-Case Scenarios of RA^*

Although Caprara and Pferschy [13] showed some absolute approximation bound of the SubSetSum algorithm for the one-dimensional Bin Packing problem, for more details see Caprara and Pferschy [13], no such bounds exist neither for SubSetSum nor for RA^* for COP as will be shown in the following.

Definition 3.3.2. *Let $OPT(I)$ denote the optimal solution value for instance I of COP and $z^H(I)$ the solution value obtained by a heuristic algorithm H. The (asymptotic) worst-case ratio of H is then given by the smallest constant β such that*

$$z^H(I) \leq \beta^H \cdot OPT(I). \tag{3.24}$$

Let's assume that there exists a constant $\tilde{\beta}^{RA^*}$ satisfying (3.24) for fixed t, $n_{\max}$ and $est \equiv 0$. Consider the following instance AB of COP: $B = L \times W$, the set of stacks, $S = \{s_1, \ldots, s_{2m-1}\}$, the set of different material qualities, all parts are of type t_2, and the stack window, $ow = 2m - 1$. Let the stack sizes be given by

$$m_i = \begin{cases} m+1, & \text{if } i \leq m, \\ 4, & \text{if else.} \end{cases}$$

Further, let

$$l_{i,j} = \begin{cases} m+1, & \text{for } i \leq m \text{ and } j = 1, \\ 2, & \text{for } i \leq m \text{ and } j \geq 2, \\ 2m, & \text{for } i \geq m+1 \text{ and } j = 2, \\ 1, & \text{for } i \geq m+1 \text{ and } j \neq 2 \end{cases}$$

and

$$q_{i,j} = \begin{cases} A_j & \text{for } i \leq m, \\ A_j, & \text{for } i \geq m+1 \text{ and } j = 1, \\ B_{i,j}, & \text{else,} \end{cases}$$

and $w_{i,j} = W$ for $i \in \{1, \ldots n\}, j \in \{1, \ldots m_i\}$. Figure 3.1 illustrates dimensions and material quality of parts.

RA^* starts with placing the first part of a stack from $\{s_1, \ldots, s_m\}$ and parts $p_{m+1,1}, \ldots, p_{2m-1,1}$ together in the first bin. Afterwards the algorithm places parts

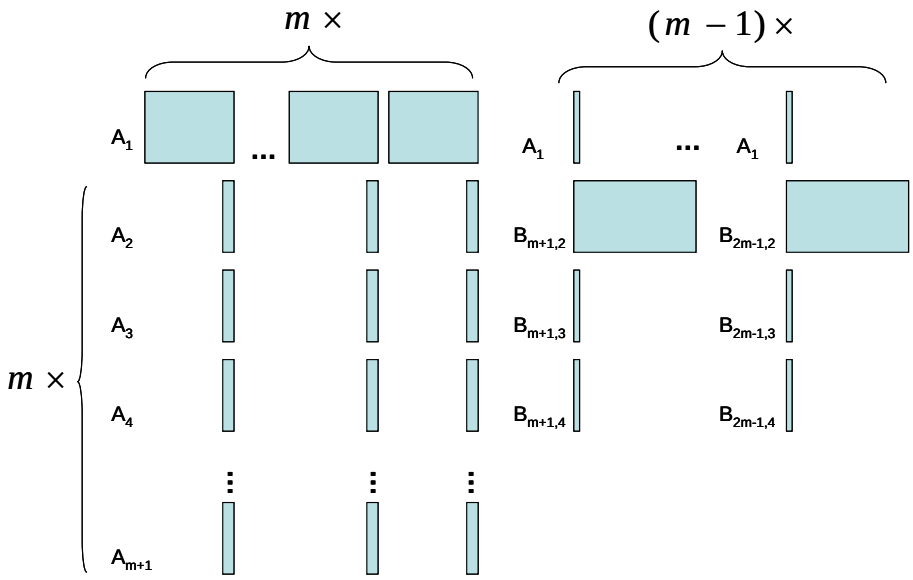

Figure 3.1: Dimensions and material qualities of parts.

$p_{m+1,2}, \ldots, p_{2m-1,2}$ in separate bins and thereby loses the ability to go back to the first node. If we chose $m \geq \max\{n_{max}, t+1\}$ all nodes S_a with $v_{C_a,i} = 4$ and depth d_{S_a} for $i > m$ are deleted from LA when a node with $d_{S_a} + 1$ is labeled. This follows from the fact that there are always $m-1$ nodes S_a with $v_{C_a,i} = 3$ explored and a new depth is always obtained by placing a part from stacks s_1 to s_m. Figure 3.2 illustrates this behavoir. Therefore RA^* places all parts $p_{i,j}$ with $i \in 1, \ldots, m$ and $j \geq 2$ in separate bins. The overall numbers of bins is given by

$$RA^*(AB) = m + m^2 + 3(m-1). \tag{3.25}$$

The optimal solution finishes stacks s_{m+1} to s_{2m-1} after the first bins an uses only $m + m - 1$ bins for the remaining parts. The overall number bins used is then given by

$$OPT(AB) = m + 3(m-1) + m - 1. \tag{3.26}$$

Therefore no approximation bound can exist since $RA^*(AB)$ is increasing quadratically with respect to m where $OPT(AB)$ shows linear behavior.

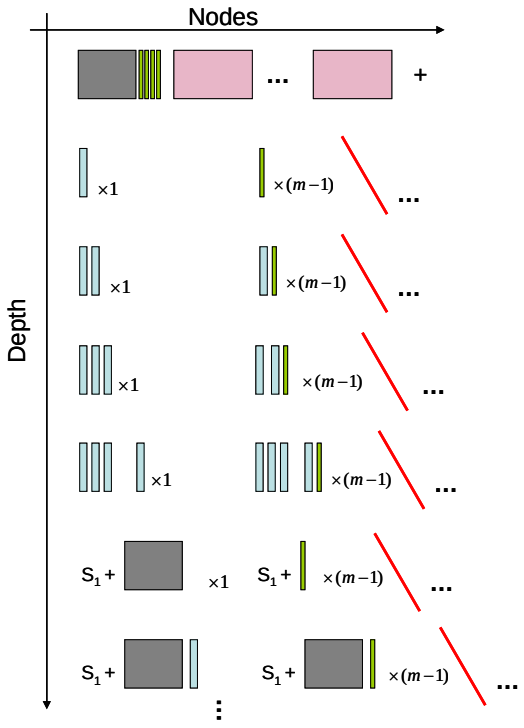

Figure 3.2: Dimensions and material qualities of parts.

3.4 Improvements for Computational Performance

Intensive testing showed that the calls of the sub-routine ALLOC and the tests if a node is in LA or O need the most computational effort within RA^*. The problem to find suitable data structures for LA and O is also a wide spread issue in the literature for shortest paths. For a more information see for example Fu et al. [33]. Therefore first an technique to reduce the number of calls of ALLOC is shown and afterwards data-structures for LA and O, which take advantage of the special structure of COP are provided.

3.4.1 Tabu Lists

Obviously the tests if a set of parts can be feasibly placed in a single bin are very time consuming. During calculations it often occurs that a set of parts that has to be allocated has already been tested before, or more general that a set of part consisting all of these parts has been allocated before. Furthermore, experience showed that especially calls of ALLOC that return a negative result, meaning that the considered parts cannot be placed in a single bin, are very time

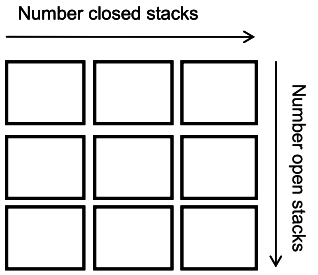

Figure 3.3: Data structure for O.

consuming. Therefore a list of sets of parts that could not be placed in a single bin is kept. ALLOC first checks whether the considered set of parts is a subset of any set stored in this list. Although the search in this list is not cheap in terms of computational effort, the overall performance of the algorithm improves significantly.

3.4.2 Improved Data Structure for O

To reduce the effort for searching O for a node S_a the list can be replaced by an array of lists. Each entry of the array stores nodes with the corresponding number of open and closed stacks. Thereby the search for S_a can be reduced to a fraction of the overall number of nodes in O. Since in most practical applications the sizes of stacks themselves do not exceed 20 parts the array is mostly growing with respect to the number of parts to be allocated. Figure 3.3 illustrates the proposed structure.

3.4.3 Improved Data Structure for LA

When expanding a possible edge (S_c, S_a) at line 8 in Algorithm 3, node S_a has to be tested on being already in LA. In general it cannot be granted that the existing node has the same depth in the search tree as the same node with updated label. Since nodes are separated to different list in LA according to their depth all list have to be searched. Especially for large instances and higher values of n_{max} and t this search procedure is very time consuming. Since the proposed relaxation of RA^* requires the structure implied by the depths of nodes a similar construct as described in the last section for O is not possible. Although keeping an additional array like in the last section and storing positions of nodes in both structures yields in a reduction of search costs, deleting nodes from LA would

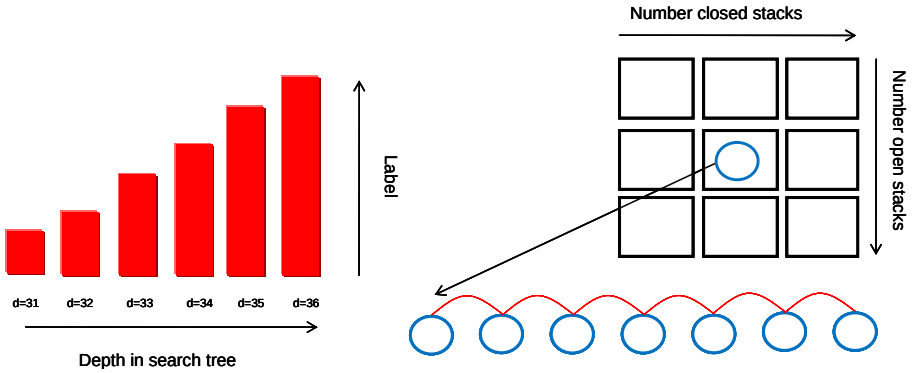

Figure 3.4: Data structure for LA.

require to change position entries of nodes and the savings of computational effort are lost. Therefore the use of chains of nodes with same number of open and closed stacks is suggested where the first nodes of these chains are stored in an array. Deleting a node from LA becomes then an issue of deleting it in the sorted lists and changing predecessor and successor of the node in the corresponding chain. The search for a node with particular coordinates is first done in the chain of nodes with the same number of open and closed stacks. If found, the search in LA can be reduced to a single list since the depth is now known. Furthermore, since the label of the node is then also known and lists in LA are sorted more sophisticated search methods can be used. Figure 3.4 shows the new structure.

3.5 Experimental Results

For evaluation of RA^* it has been tested on the instances defined in section 2.3 using the following parameter sets:

The expansion band was selected as $t = 15$. As Albano and Sapuppo [1] already stated small changes of t mostly do not lead to changes in results. The size of LA, given by n_{max}, has a much bigger influence on the quality of results. Therefore two different scenarios for n_{max}, both depending on the size of an instance were tested:

$$n^1_{max} = \begin{cases} 10000 & \text{for } n = 50, \\ 800, & \text{for } n = 150, \\ 500, & \text{for } n = 250 \end{cases}$$

and

$$n^2_{max} = \begin{cases} 20000 & \text{for } n = 50, \\ 1600, & \text{for } n = 150, \\ 1000, & \text{for } n = 250 \end{cases}$$

Furthermore, results for RA^* were obtained using the following estimator est:

$$est = (1.2 - 10\text{Var}\left(\frac{|\text{P}_{\text{Q}_\text{A}}|}{|\text{P}|}, \ldots, \frac{|\text{P}_{\text{Q}_\text{D}}|}{|\text{P}|}\right) \text{L}_\text{C}. \tag{3.27}$$

This estimator is based on the observation that lower bounds tend to be weaker for instances where the cardinalities of subsets of parts with same material qualities do not vary that much.

All parameters resulted from intensive testing. Table 3.1 shows lower bounds and results of all tested instances. Bounds were calculated using L_{COP} and different bounds for Bin Packing problems. Furthermore, Opt. denotes the optimal solution if it could be found within a time limit of 1200 seconds.

One can see that all lower bounds are quite weak, due to the fact that they do not consider order- and stack constraints. This is also a reason why optimal solutions could not be found for problems of class II and III. Comparing the results obtained under n^1_{max} with results under n^2_{max} one may note that larger n_{max} does not necessarily lead to better solutions, in most cases though results tend to be slightly better. However, runtimes are more than three times higher than for n^1_{max} what is, for industrial applications, not justified by the small improvement of used bins.

Table 3.2 shows the performance of standard heuristics from chapter 2 on the same instances. Although a time limit of 180 seconds was applied to these algorithms we can see that using n^1_{max} all runs of RA^* terminated within this limit and therefore one is able to compare results.

Figures 3.5, 3.6, 3.7, 3.8, 3.9 and 3.10 compare the results of our algorithm with the results from secion 2.4.

One can see that RA^* outperforms all other heuristics significantly, as all other algorithms are not able to profit from the structures of COP as RA^* does.

	Bounds					$RA*$					
						n^1_{max}			n^2_{max}		
	LB_C	LB^R_{DA}	L^R_{BM}	L^R_{CJE}	Opt.	Bins	Nodes	Time	Bins	Nodes	Time
I×1	15	15	18	18	27	27	8119	1.9	27	8119	1.9
I×2	12	11	14	15	21	21	7379	1.6	21	7379	1.6
I×3	15	16	20	18	28	28	7064	1.3	28	7064	1.3
I×4	16	16	19	18	25	25	8436	1.5	25	8436	1.5
I×5	14	14	15	17	21	21	8680	2.5	21	8680	2.5
I×6	14	14	15	16	25	25	13607	4.3	25	13607	4.3
I×7	13	13	16	17	24	24	9957	2.6	24	9957	2.6
I×8	13	13	14	16	21	21	9554	4.5	21	9954	4.5
I×9	15	14	17	15	26	26	12234	3.1	26	12234	3.1
I×10	15	14	16	16	25	25	6850	1.0	25	6850	1.0
II×1	37	37	42	40	∗	57	41680	48.2	56	85245	202.9
II×2	40	40	45	43	∗	60	42727	43.9	61	97768	202.0
II×3	43	43	48	44	∗	59	51408	66.3	59	97865	244.0
II×4	39	39	42	42	∗	60	46882	51.8	60	92765	196.8
II×5	39	39	44	42	∗	56	39529	44.0	56	76600	160.6
II×6	42	41	46	42	∗	61	62045	73.9	61	111920	241.0
II×7	42	43	45	44	∗	58	47541	63.5	58	94729	229.5
II×8	42	42	46	43	∗	59	44977	46.5	59	96833	197.1
II×9	44	46	49	47	∗	64	53598	65.8	63	101273	252.1
II×10	38	37	40	38	∗	59	59964	70.1	58	112178	263.9
III×1	68	68	74	70	∗	98	51055	50.8	98	101473	180.6
III×2	64	65	69	67	∗	96	55554	55.5	95	104153	189.8
III×3	67	66	75	68	∗	104	61789	63.3	101	113452	198.1
III×4	64	64	71	68	∗	93	51063	56.6	92	92262	167.7
III×5	69	68	74	70	∗	99	52907	60.3	101	113817	230.5
III×6	68	66	75	71	∗	99	57745	67.1	97	107223	239.8
III×7	67	65	73	68	∗	98	51721	56.4	101	117827	231.1
III×8	67	67	72	70	∗	101	59877	61.6	97	104818	186.7
III×9	70	70	76	71	∗	96	50151	52.1	97	109878	241.9
III×10	67	67	73	69	∗	99	53128	58.7	96	178301	178.3

Table 3.1: Experimental Results for RA^*.

	SA		GA		SSS		Tabu	
Class×Instance	Bins	Time	Bins	Time	Bins	Time	Bins	Time
I×1	27(27, 2)	7, 5	27(27, 3)	8, 3	30	< 0.1	30	12, 2
I×2	21(21, 9)	6, 7	22(22, 9)	7, 2	28	< 0.1	24	23, 9
I×3	28(28, 6)	6, 8	29(30, 1)	7, 5	34	< 0.1	32	13, 4
I×4	26(26, 1)	6, 6	26(26)	7, 5	30	< 0.1	28	13, 9
I×5	23(23)	7, 1	22(23, 5)	8, 0	27	< 0.1	26	10, 3
I×6	25(25, 9)	6, 6	26(26, 9)	7, 5	29	< 0.1	27	11, 4
I×7	24(24, 1)	6, 6	24(25, 2)	7, 7	26	< 0.1	26	19, 9
I×8	21(21, 8)	7, 3	22(23)	8, 1	28	< 0.1	33	7, 2
I×9	26(26, 1)	7, 1	26(26, 1)	8, 1	33	< 0.1	28	34, 0
I×10	25(25, 2)	7, 4	26(26, 2)	7, 7	30	< 0.1	29	6, 9
II×1	61(62.6)	24.0	62(65.4)	139.3	65	0.3	67	102.2
II×2	66(67.2)	25.2	67(69.6)	139.9	72	0.2	90	85.7
II×3	64(65.7)	24.1	66(67.2)	138.3	71	0.2	72	69.7
II×4	64(66.5)	23.2	66(68.4)	138.5	77	0.2	81	180.0
II×5	63(64.3)	24.1	62(64.9)	153.2	72	0.3	64	180.0
II×6	63(67.6)	23.9	65(69.6)	140.5	70	0.1	77	43.7
II×7	63(64.7)	24.5	62(65.7)	145.1	69	0.2	78	43.8
II×8	64(66.9)	24.7	67(68)	135.8	76	0.1	82	47.7
II×9	66(67.6)	23.8	69(69.7)	137.4	78	0.2	107	46.9
II×10	62(63.5)	24.7	64(67.2)	141.6	66	0.1	71	40.8
III×1	109(112.1)	38.4	114(120.4)	180.0	123	0.4	143	180.0
III×2	105(106.4)	38.3	108(112.3)	180.0	112	0.3	117	180.0
III×3	111(114.5)	40.0	119(122.9)	180.0	120	0.2	134	180.0
III×4	103(106.8)	39.4	108(111.4)	180.0	112	0.5	118	180.0
III×5	108(111.6)	37.8	115(117.5)	180.02	113	0.3	185	180.0
III×6	110(112.7)	38.1	116(118.7)	180.0	116	0.3	151	180.0
III×7	109(113)	38.5	116(119.8)	180.0	123	0.6	146	180.0
III×8	108(110.9)	37.9	112(116)	180.0	117	0.4	171	180.0
III×9	104(117.3)	39.4	111(114.4)	180.0	123	0.4	136	180.0
III×10	105(108.7)	39.8	112(114)	180.0	112	0.4	127	180.0

Table 3.2: Standard Heuristics.

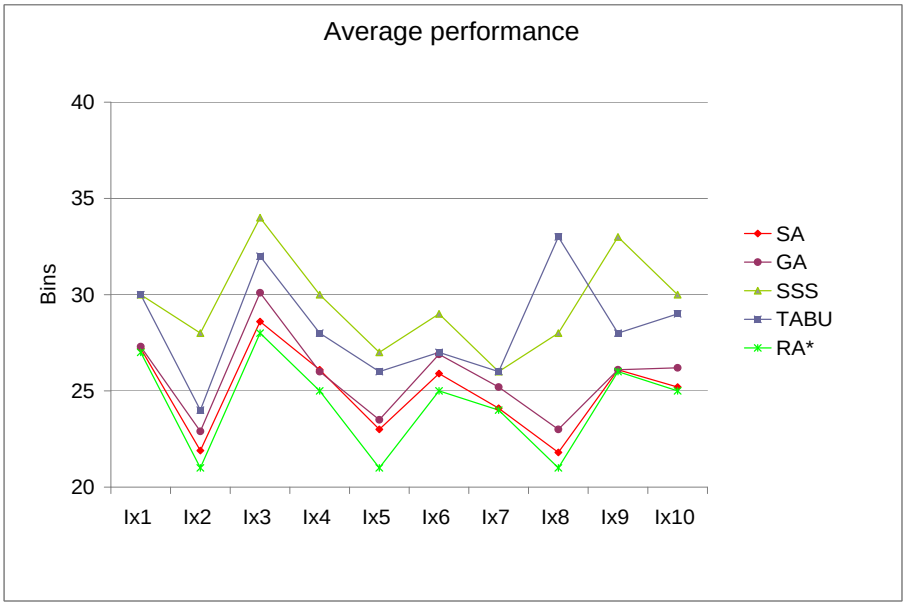

Figure 3.5: Comparisons of average performances for class I instances.

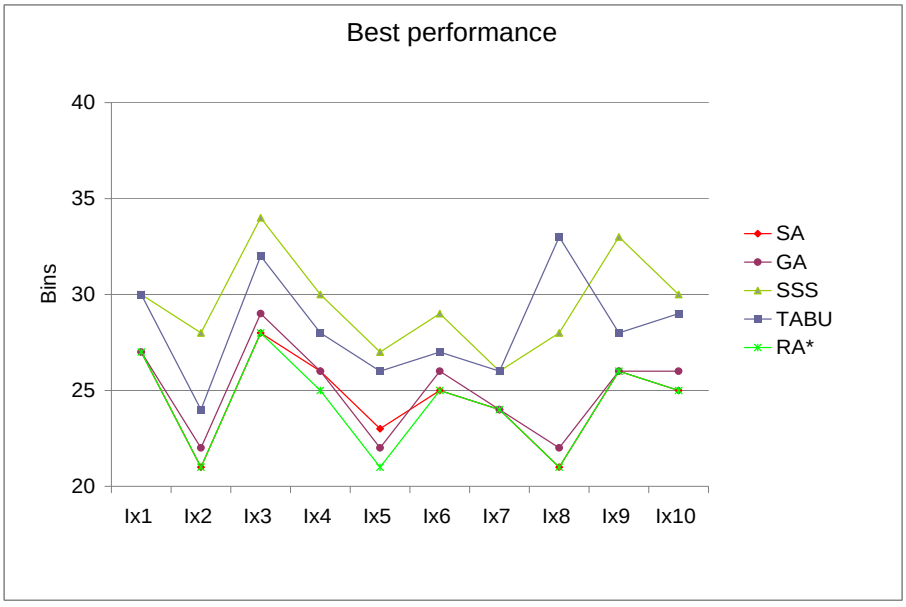

Figure 3.6: Comparisons of best performances for class I instances.

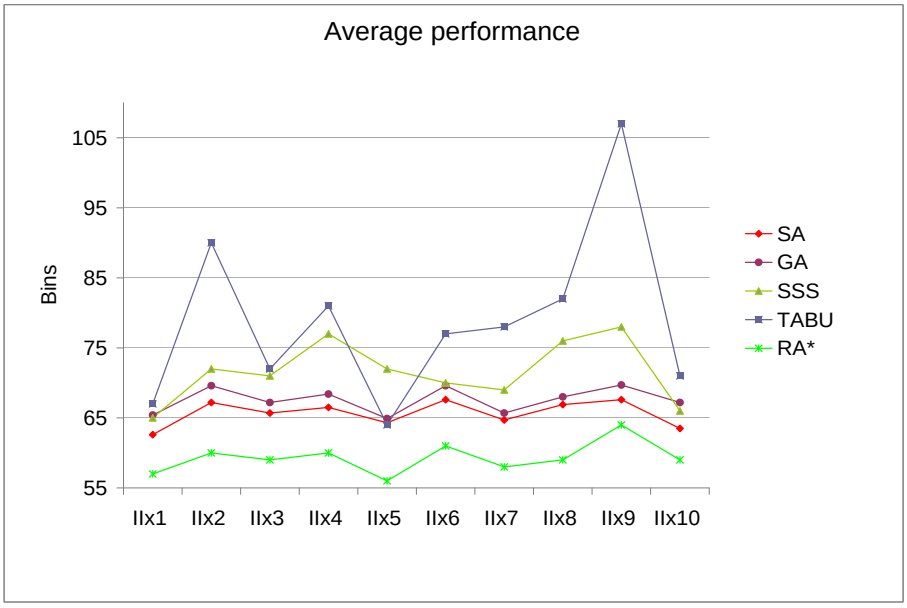

Figure 3.7: Comparisons of average performances for class II instances.

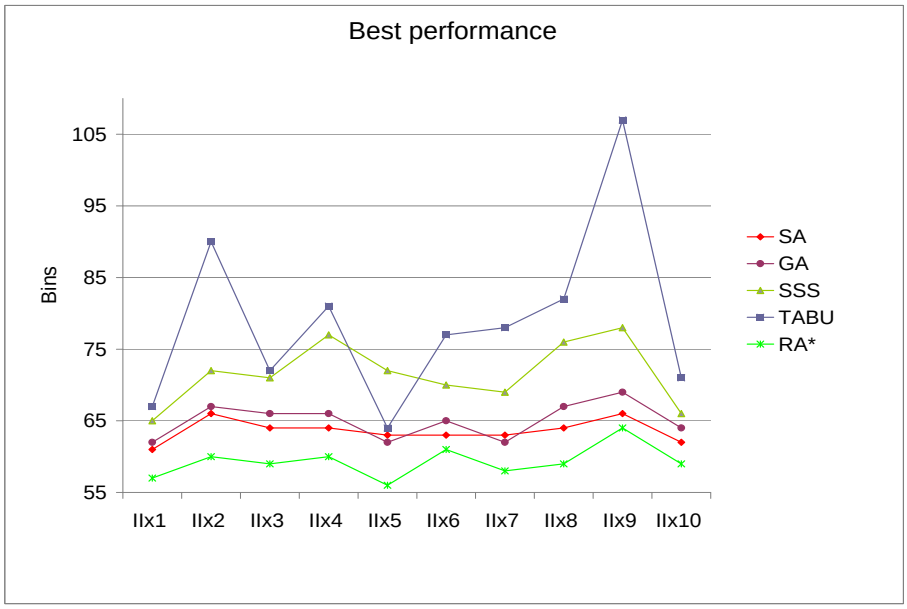

Figure 3.8: Comparisons of best performances for class II instances.

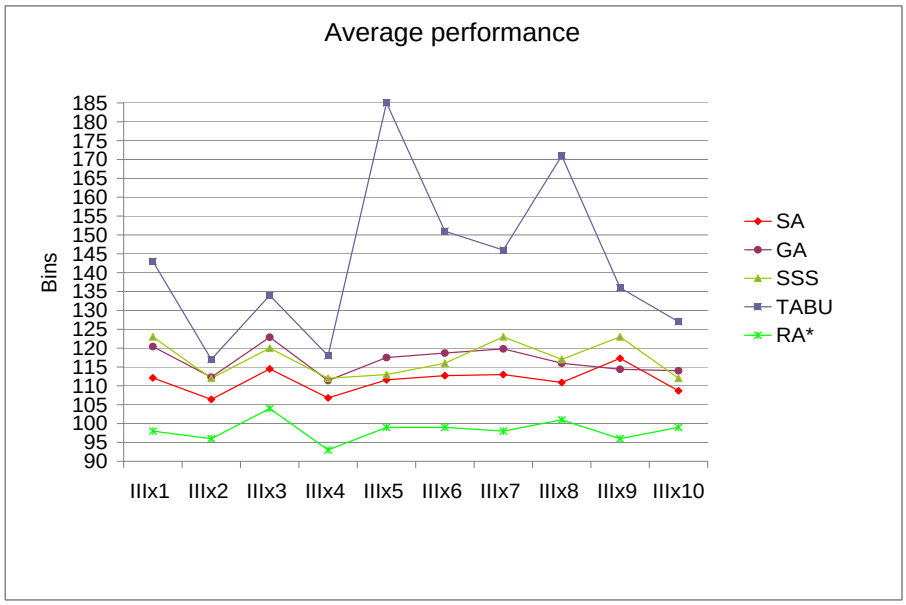

Figure 3.9: Comparisons of average performances for class III instances.

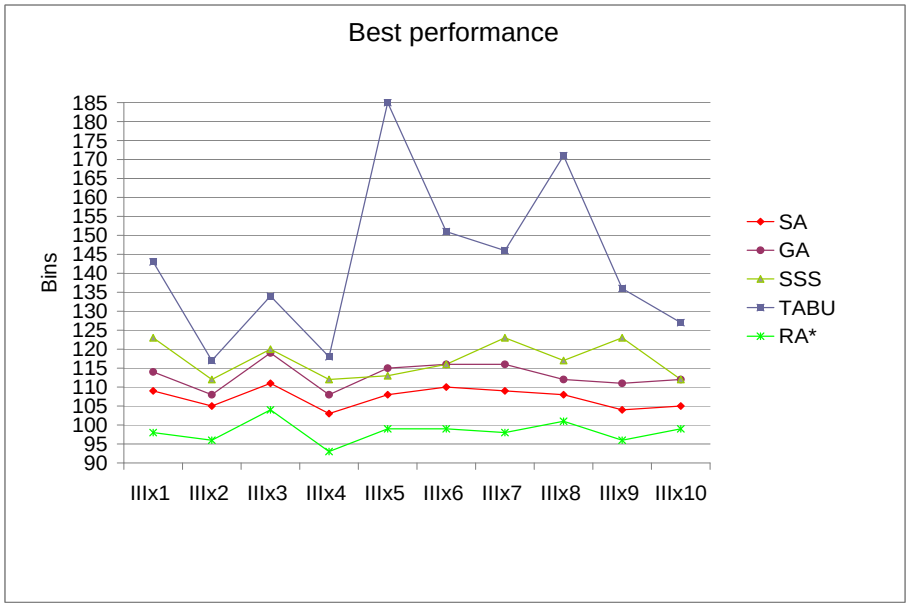

Figure 3.10: Comparisons of best performances for class III instances.

Chapter 4

Translational Containment Problem: Dealing with Geometry

This chapter serves the need of evaluation of different strategies and heuristics used for the ALLOC subroutine from last chapter. The main aim is to find suitable heuristics that are able to search the space of different sequences efficiently under consideration of special circumstances. These involve that instances placed by ALLOC are usually very small, trivial solutions have to be detected instantly and negative calls should not result in a computational overhead. Further, techniques to extend ALLOC and thereby COP to polygonal shaped parts are provided.

In the following section containment problems are introduced, in section 4.2 the concept of no-fit polygons is revisited. Further, a new method to obtain and validate placing points is introduced. In section 4.3 some heuristics for Strip Packing are adapted for the containment problem. Finally in section 4.4 some experimental results are given and conclusions are drawn.

4.1 Containment Problems

Containment problems consist of the question whether a set of n irregular shaped parts $P_1, \ldots, P_n$ can be fit fully in an enclosing shape, the container, without overlapping. They can be found in various industrial applications, as for example the clothing-, leather-, steel- and concrete parts industry, just to mention a few. Depending on their origin, problems vary over the allowance of rotations and homogeneity of parts and the shape of the container. The problem which is in the focus of this chapter is motivated by the need of an adequate ALLOC routine for RA^* and real world applications of the concrete part production. Therefore it

has some specific attributes and requirements, compared to standard problems, that affect the choice of solving procedures. It is based on the translational problem containment problem, where the container is given by a rectangle of length L and width W and parts must not be rotated. Further, more than 95 percent of parts have only orthogonal edges, only a few are irregular shaped. In addition, instances tend to be small, normally the set of parts has a cardinality between 2 and at maximum 10.

Publications dealing with the pure containment problem are very rare, Milenkovic [57, 55, 56], Milenkovic and Daniels [58], Daniels and Milenkovic [19, 21, 20] and Daniels [18] propose solving techniques for the rotational and translational containment- and also minimum enclosure problems. Grinde and Cavalier [39] focus on the special case where the set of parts consists only of two polygons.

Since the containment problem is NP hard (as shown for example in V.J Milenkovic [57]) finding exact solutions, in terms of 'yes' or 'no', has quite high computational costs. A drawback of heuristic methods is that they cannot correctly answer the question on the existence of a feasible layout with 'no' if the fail to find one. However, for many applications correctness is far less important than computational time, i. e. the containment problem is a sub problem of a heuristic algorithm on a global problem. Optimality is already lost in the global heuristic and therefore non-exact, but quick, solutions of the containment problem might be from greater interest, which is also the case for the ALLOC routine. Further, for the applications arising from the concrete part production the problem is nested in an automatic planning tool that operates both online and offline and a semi-automatic production planning tool. In all three cases solutions are required within a few Millie seconds to ensure a smooth work flow; on the other hand the correctness of a negative solution can be neglected. In addition, a lot of instances have trivial solutions, meaning that a feasible layout can be found very easily. This is especially the case for very small instances with less or equal than 4 parts. Therefore requirements on an algorithms range from detecting trivial solutions at a glance, to find more complex layouts for medium instances and also deciding that no solution can be found within very short computational times. Examples for trivial and medium instances are given in figures 4.1 and 4.2.

Looking at heuristic solving procedures, the problem can either be formulated as Single Knapsack Problem or as Open Dimension Problem, or more precisely Strip Packing Problem, according to Wäscher, Hauner and Schumann [66]. For Single Knapsack Problems a set of strongly heterogeneous parts have to be accommodated in a single knapsack. Normally not all parts can be accommodated and so the value of parts in the knapsack has to be maximized. The Strip Packing

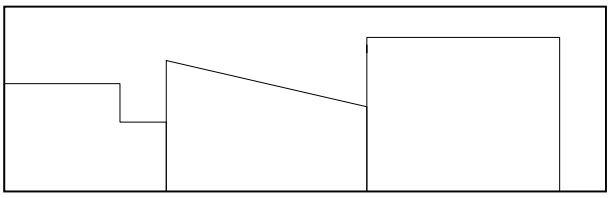

Figure 4.1: Trivial example of a containment problem with 3 parts.

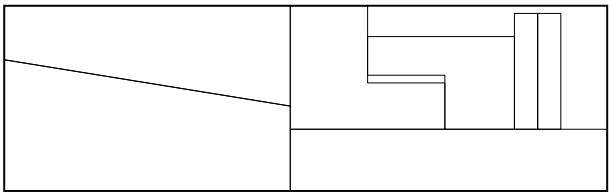

Figure 4.2: Medium instance of a containment problem with 7 parts.

Problem consists of packing strongly heterogeneous parts on a strip of fixed width and infinite length. The goal is to minimize the used length of the strip. In terms of containment a feasible solution is found as soon as a layout is obtained, where the right-most point of all parts does not exceed L, as illustrated in figure 4.3. In the following this approach is used to solve the containment problem described above.

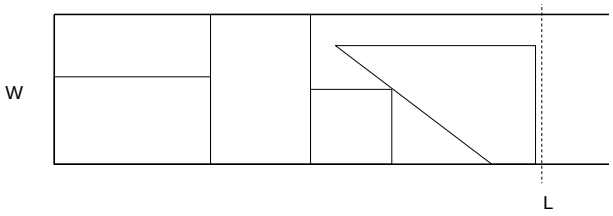

Figure 4.3: Strip packing used for containment.

Most of the algorithms on Strip Packing use no-fit polygons, for example Gomes and Oliveira [38], compaction and separation techniques, see for example Gomes and Oliveira [37], or ϕ-functions by Stoyan et al. [63] to deal with the geometry of the problem. Although especially compaction and separation and ϕ-functions are able to find very tight layouts of parts they are not cheap in terms of computational effort. When using no-fit polygons runtimes can be significantly reduced by avoiding recalculation of no-fit polygons and calculat-

ing them in advance as a preprocessing step. However, for large instances with strongly heterogeneous parts or online algorithms this preprocessing step is not possible or too time consuming. Therefore a new geometric method to obtain placing points of parts is introduced and used within some heuristics for the Strip Packing Problem.

4.2 Geometry

Any cutting and packing problem has to deal with the same geometry issues, as there are that parts must not overlap and must be fully placed within a strip bin or any other large object. In this chapter the focus is on heuristics for the strip packing problem that use nesting routines to place parts on the strip in a given sequence. Further, they also consist of a higher level heuristic that explores the solution space of different sequences. Trivial instances can thereby be solved rather quickly since mostly only one layout has to be generated, medium instances still can be solved in a reasonable amount of time and runtimes are well controllable for instances where no feasible solution can be found.

Therefore, given a sequence of parts, nesting routines must be able to quickly generate a feasible layout and should also be able to fill holes at a later stage. Assuming that shapes of parts are polygons or at least approximated by the latter, the main task for any geometric routine can then be reduced to find possible placing points for a part, considering a set of parts that has already been placed within the strip. What follows is a description of the principle of the proposed nesting routine and methods to calculate possible placing points using no-fit polygons. Further, a new approach to obtain placing points from parallel edges is introduced.

4.2.1 Nesting Routines

The proposed nesting routine is based on the procedure used by Gomes and Oliveira in [38]. Given a set of already placed parts $P_1, \ldots, P_m$ at points (x_j, y_j) with $1 \leq j \leq m$ and a set of possible placing points $(\tilde{x}_i, \tilde{y}_i)$ with $1 \leq i \leq k$ the next part is placed according to a greedy-bottom-left strategy on the strip. Meaning that it's reference point is placed on the point $(\tilde{x}_l, \tilde{y}_l)$ with $1 \leq l \leq k$ that satisfies the following conditions:

- $\tilde{x}_l \leq x_i$ for $1 \leq i \leq k$.
- If $\tilde{x}_l = \tilde{x}_i$ then $\tilde{y}_l \leq \tilde{y}_i$ for $1 \leq i \leq k$.

Due to the special requirements of the containment problem that the rightmost point of all parts must not exceed L, the proposed nesting routine stops placing parts if the current part exceeds L to improve runtimes. In that case, for a better search behavior of heuristics exploring the solution space of different sequences, the procedure returns the x-coordinate of the rightmost point plus an additional penalty of L for each unplaced part as an objective value. Whereas zero is returned, in case that a feasible layout for all parts has been found.

4.2.2 No-fit Polygon and Inner-Fit-Rectangle

The concept of the no-fit polygon was first introduced by Art [2] in 1996. Nowadays they are commonly used for intersection tests and finding possible placing points for parts. Algporithms in this work follow a constructive approach that is based on Burke et al. [11]. The no-fit polygon of part B with respect to part A, $\text{NFP}_{A,B}$ can briefly be described as follows: The polygon $\text{NFP}_{A,B}$ results of tracing a reference point of part B when orbiting part B around a fixed part A, whilst ensuring that the two parts always touch but never intersect (see also Burke et al. [11]), as illustrated in figure 4.5. As the reference point, the point resulting from combining the smallest x-coordinate with the smallest y-coordinate of all points from a part is chosen, see figure 4.4.

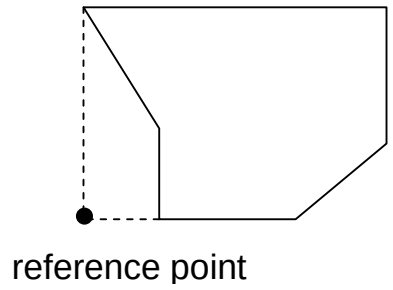

Figure 4.4: Reference point of a part.

From the definition of $\text{NFP}_{A,B}$ it immediately follows that (A.M. Gomes and J.F. Oliveira in [38]):

- If the reference point of part B is placed in the interior of $\text{NFP}_{A,B}$ then B and A intersect.
- If the reference point of part B is placed on the boundary of $\text{NFP}_{A,B}$ then B and A touch.

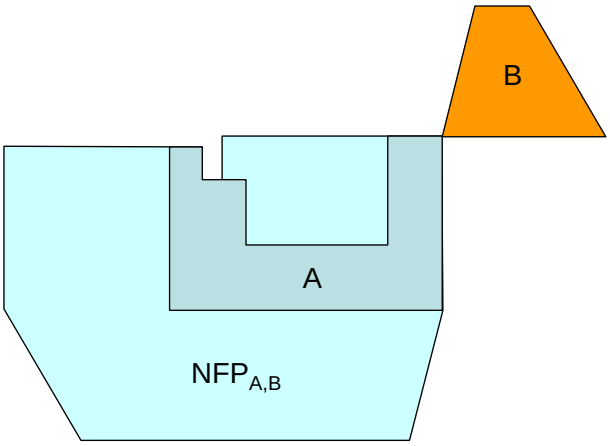

Figure 4.5: Resulting $\mathrm{NFP}_{A,B}$ when B orbits around A.

- Else both parts neither intersect nor touch.

Therefore the NFP can be used to calculate layouts where parts do not intersect, furthermore, it also reduces the detection of intersection of two polygons to a simple-point-in-polygon test. Note that $\mathrm{NFP}_{A,B}$ can be calculated independently from the position of part A. For intersection tests and placing point calculations the position of $\mathrm{NFP}_{A,B}$ has only to adjusted to the position of part A. Hence in many cases it is possible to calculate NFPs for all possible pairs of parts offline in a preprocessing step.

To ensure that all parts are fully placed within the strip a similar construct as the no-fit polygon is used the, inner-fit rectangle. The inner-fit rectangle $\mathrm{IFR}_{A,B}$ represents the feasible region for placing the reference point of a part B so that the part is fully placed within a larger rectangle A. $\mathrm{IFR}_{A,B}$ can be calculated by subtracting the width and length of the rectangle enclosure of part B from the width and length of rectangle A. From its definition it follows that (see also A.M. Gomes and J.F. Oliveira in [38]):

- If the reference point of part B is placed in the interior of $\mathrm{IFR}_{A,B}$ then B is contained by A.
- If the reference point of part B is placed on the boundary of $\mathrm{IFR}_{A,B}$ then B is contained by A and B and A touch.
- Else part B is not contained by A.

Note that the inner-fit rectangle for a strip of infinite length can also be seen as a strip with infinite length. Finding a placing point for a part P_i with respect

to a set of already placed parts $P_1, \ldots, P_j$ becomes now a matter of finding a placing point that is outside of NFP_{P_l,P_i} with $1 \leq l \leq j$ and within the interior of IFR_{S,P_i}, where S denotes the strip. For finding tight layouts we can further assume that the placing point has to be on the boundary of at least one no-fit polygon and the inner-fit rectangle, or at the boundary of at least two no-fit polygons. Thereby every part is in contact with the boundary of at least two other parts or one part and the strip.

Following the assumptions from above any feasible placing point for part P_i is:

- a vertex of NFP_{P_i,P_l},
- a vertex of IFR_{S,P_i},
- an intersection point of two edges of NFP_{P_i,P_l} and NFP_{P_i,P_k} or
- an intersection point of two edges from NFP_{P_i,P_l} and IFR_{S,P_i},

where P_l and P_k are parts that have already been placed. All points have to be checked on lying outside the interior of other no-fit polygons and within the inner-fit rectangle.

4.2.3 Placing Points from Parallel Edges

Although in most cases no-fit polygons and inner-fit rectangles can be calculated offline in a preprocessing step, there exist several industrial application where this is not possible due to runtime restrictions. Most algorithms and applications where these concepts have been used do not consist of more than approximately 50 different parts. Calculating no-fit polygons for each pair of parts, even for different rotations, can still be done in reasonable computational time. For the concrete part industry instances with more than 300 strongly heterogeneous parts and four possible rotations for each part are no exceptional case, where one would have to calculate more than $\binom{1200}{2}$ no-fit polygons. In many cases a packing of these parts in multiple containers is often required in a couple of minutes where instances change on a daily basis. Furthermore, a semi-automatic production tool where the user allocates a subset of parts to a single bin and a heuristics tries to find an actual layout of these parts would also require no-fit polygons of all pairs of parts within the loading process of the software tool. For these applications and also online algorithms it is therefore not possible to calculate all no-fit polygons in advance.

Furthermore, a main characteristic of instances from the concrete part industry is that, although parts are strongly heterogeneous and in many cases non-convex, more than 95% of parts consist only of edges with orthogonal angles. Therefore placing points can be also derived from the procedure described in the following.

Given a set of already placed parts $P_1, \ldots, P_m$ at points (x_j, y_j) with $1 \leq j \leq m$ on strip S placing the next part P_i, all pairs of edges that are parallel and might touch after placing part P_i are calculated. Let's assume that e_{d,P_i} for $d = 1, \ldots, h_{P_i}$ denote the edges of part P_i and $\alpha_{e_{d,P_i}}$ for $d = 1, \ldots, h_{P_i}$ the corresponding angles, where edges of parts are oriented anti-clockwise. Therefore possible touching and parallel edges have to satisfy

$$\left(\alpha_{e_{d,P_i}} + 180\right) \mod 360 = \alpha_{e_{\tilde{d},P_l}}. \tag{4.1}$$

In the next step for each pair of edges satisfying (4.1) a placing line for the reference point of part P_i is calculated, i.e. placing the reference point on this line results in a layout where both edges touch, as shown in figure 4.6.

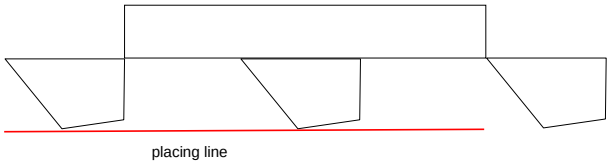

Figure 4.6: Placing line.

All placing lines resulting from edges of part P_i with edges from any part on the strip or from edges of the strip itself are stored in a list and used to obtain placing points. Therefore each pair of placing lines is checked on intersection. For any resulting placing point at least two pairs of parallel edges would touch in the resulting layout.

To compute placing points resulting from a single placing line the part is first set on the line. Iterating through each point of part P_i and parts P_j with $1 \leq j \leq m$ placing points result from intersecting a line, parallel to the considered placing line, with all edges from other parts and the strip. Figure 4.7 illustrates this behavior.

The drawback of using single placing lines to compute placing points is obviously that a lot useless points are obtained. To avoid too much computational overhead placing lines can be reduced in size by considering adjacent edges and

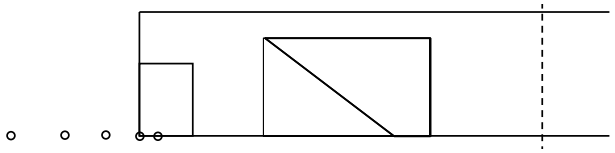

Figure 4.7: Placing points from a single placing line.

are possibly eliminated if they can never lead to feasible layouts, as shown in figure 4.8. Furthermore, placing points resulting from single placing lines are only computed if and only if at least one part of parts P_j with $1 \leq j \leq m$ and P_i has a non-orthogonal edge. In the other case intersecting all pairs of placing lines leads to all possible placing points.

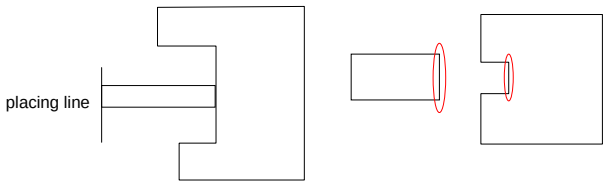

Figure 4.8: Reducing size and number of placing lines.

All resulting placing points are then ordered according to their bottom-left position. Points are checked on possible intersections and containments of parts in that order, until a feasible point has been found. In general this is quite expensive in terms of computation time, however, a lot of computational effort can be avoided by simple tests, as for example checks on bounding rectangles, special procedures if parts are rectangular shaped etc.. If the algorithm fails to find any placing points from parallel edges because no parallel edges exist, no-fit polygons are computed and placing points are obtained as described in section 4.2.2. The proposed approach is summarized by algorithm 4.

4.3 Heuristics for the Strip Packing Problem

Several heuristics for the strip packing problem have been introduced over the last decades, including approaches like Simulated Annealing, Genetic algorithms, Compaction and Separation, Tabu Search and many more. For a general overview see for example Hopper and Turton [42]. To solve the containment problem and suite the special requirements coming from the concrete part industry heuristics

Algorithm 4 Placing Points from parallel edges

1: $PL = \{\}$, $PP = \{\}$
2: **for all** parts P_l on strip S **do**
3: compute all pairs of edges of part P_l and P_i that satisfy (4.1) and put resulting placing lines in PL.
4: **end for**
5: **for all** $\left(pl, \tilde{pl}\right)$ with $pl, \tilde{pl} \in PL : pl \neq \tilde{pl}$ **do**
6: **if** pl and $\tilde{pl}$ intersect **then**
7: put intersection point in PP.
8: **end if**
9: **end for**
10: **if** parts P_j with $1 \leq j \leq m$ and P_i have non-orthogonal edges **then**
11: Compute placing points using only one placing line and put them in PP.
12: **end if**
13: **while** $P \neq \{\}$ **do**
14: find most bottom-left point p in PP.
15: translate P_i to p.
16: **if** resulting layout is non-overlapping and part is fully on strip **then**
17: **return** $\{p\}$
18: **else**
19: delete p from PP.
20: **end if**
21: **end while**
22: **if** $PL = \emptyset$ **then**
23: call no-fit polygons procedure to calculate placing points.
24: **end if**
25: **return** PP

need to be simple but effective. Therefore two methods to search the solution space of different sequences of parts were implemented during this work. They lead to good results on standard benchmark instances, but still seem to be promising for finding quick solutions for small instance of the containment problem, a description of both heuristics follows. Further, note that for instances with less or equal to 3 parts all possible sequences are tried instead of calling a heuristic method. Obviously the procedure stops as soon as layout has been found where the rightmost-point of all parts does not exceed L.

4.3.1 The 2-exchange Heuristic

The 2-exchange heuristic introduced by Gomez and Oliveira [38] is a guided local search procedure where from a current solution, represented by a sequence of parts, neighborhood solutions are obtained by exchanging parts in the sequence. It is an improvement algorithm where neighborhood solutions are only accepted if they improve the current solution. The algorithm terminates if no better solution was found within the neighborhood or a maximum number of iterations has been reached. Neighborhoods are built by exchanging pairs of parts in the current sequence where the size of the neighborhood is controlled by a parameter δ, by exchanging only parts with a smaller distance in the sequence than δ. Three different neighborhood sizes were tried, $\delta \in \{1, 2, 3\}$. Furthermore, three strategies to choose a better solution from the neighborhood to become the new current solution were proposed:

- the first better solution (first better)
- the best solution (best)
- a randomly chosen solution among all better solutions (random better).

For more details the reader is referred to by Gomez and Oliveira [38]. If a solution with smaller objective value than L is found the algorithm terminates right away.

4.3.2 A Tabu Search Approach

The approach used in this work is based on the Tabu Search combined with hill climbing method introduced by Burke et al. [10] and is only slightly different at some steps. A neighborhood of fixed size of the current solution is created by using a neighborhood operator that is chosen randomly among a set of different operators. All operators randomly exchange positions of a subset of parts in the current sequence. Operators differ over the cardinality of the subsets to be exchanged, which ranges from 1 to 4 (note that in the original algorithm also higher values than 4 have been tried) and are therefore called $1OPT$, $2OPT$,$3OPT$ and $4OPT$. Since by the definition from above, $1OPT$ would not change the sequence at all it randomly chooses one part from the sequence, removes it and inserts it at random position. Among all generated neighborhood solutions, the best is chosen and the search continues with the new current sequence. Despite to the original algorithm where a Tabu list for solutions is kept, in the approach described in this section a Tabu list is kept for operators instead for solutions. Due to the focus on small instances this avoids cycling of the search path quite effective.

Different values for the Tabu-tenure have been tried, see section 4.4. If the selected operator is tabu, a single hill-climbing step is performed where only one neighborhood-solution is generated. The algorithm terminates if no improvement of the best solution could be obtained or a maximum number of iterations has been reached. The following pseudo code shows the described algorithm (where δ is a parameter that describes the size of the neighborhood),

Algorithm 5 Tabu Search and hill climbing method

```
1: current = FindStartSolution(); best = NestingProcedure(current); neighbors = {}.
2: if best ≤ L then
3:   return true
4: end if
5: while not reached iteration maximum do
6:   operator = SelectOperator().
7:   while neighbors.size < 5 do
8:     neighbors.add(GenerateRandomNeighbor(current,operator))
9:   end while
10:  foundBetter = false
11:  for all neighbor ∈ neighbors do
12:    if NestingProcedure(neighbor)¡best then
13:      if NestingProcedure(neighbor) ≤ L then
14:        return true
15:      else
16:        best = NestingProcedure(neighbor); current = neighbor; foundBetter = true.
17:      end if
18:    end if
19:  end for
20:  if !foundBetter then
21:    return false
22:  end if
23: end while
24: return false
```

4.4 Experimental results

The proposed approaches were tested and compared on a real world data-set, consisting of 50 parts. For each $n \in \{2, \ldots, 8\}$, 1000 subsets with cardinality n have been randomly generated. All proposed algorithms have been tested on these

instances with all possible combinations of parameters, $\delta \in \{1, 2, 3\}$, first better, random better and best for the 2-exchange heuristic, Tabu tenure $l \in 1, 2, 3$ for the Tabu Search approach and maximum iteration numbers $iter_{max} \in \{2, 5, 10\}$ for both. Furthermore, all different settings have been runned using the no-fit polygon approach and the parallel-edge approach. Tables 4.1, 4.2, 4.3 and 4.4 show the results for the parallel-edge approach. Where 'feas' denotes the number of calls that resulted in a feasible layouts, 'nests' the average number of calls of the nesting procedure per instance and time the average runtime of for one instance in seconds.

parts	2-exchange-first better-parallel edges								
	$\delta = 1$			$\delta = 2$			$\delta = 3$		
	feas	nests	time	feas	nests	time	feas	nests	time
				$iter_{max} = 2$					
2	1000	1045	0.001						
3	623	3341	0.041						
4	480	2431	0.003	480	2642	0.003	490	2609	0.003
5	315	2896	0.005	315	3513	0.006	329	3446	0.005
6	141	3529	0.007	141	4213	0.008	159	4558	0.009
7	54	4060	0.009	54	4634	0.011	66	5192	0.012
8	1	4654	0.012	1	5633	0.015	1	6545	0.018
				$iter_{max} = 5$					
4	483	2642	0.010	483	11494	0.015	494	13637	0.018
5	352	3513	0.021	352	16706	0.029	347	18528	0.032
6	154	4213	0.035	154	22700	0.049	174	26193	0.056
7	69	4635	0.052	69	27001	0.071	94	29925	0.076
8	3	5633	0.066	3	30294	0.095	8	35704	0.111
				$iter_{max} = 10$					
4	483	2609	0.023	483	28853	0.040	495	40053	0.054
5	352	3446	0.053	352	47793	0.091	347	61759	0.115
6	154	4558	0.097	154	65608	0.156	177	84926	0.198
7	71	5192	0.148	71	78966	0.230	99	95461	0.273
8	3	6545	0.202	3	90327	0.328	9	108609	0.328

Table 4.1: Results for 2-exchange-first better-parallel edges.

For the different versions of the 2-exchange heuristic one may note that results tend to improve for increasing values of δ and obviously the number of iterations

parts	2-exchange-random better-parallel edges								
	$\delta = 2$			$\delta = 3$			$\delta = 1$		
	feas	nests	time	feas	nests	time	feas	nests	time
	$iter_{max} = 2$								
4	493	4623	0.006	493	8712	0.011	495	12785	0.017
5	327	7170	0.012	343	13698	0.023	342	20339	0.034
6	159	10378	0.021	174	20200	0.40	177	30104	0.060
7	70	13191	0.031	85	25938	0.059	95	38628	0.090
8	3	15967	0.043	7	31885	0.09	8	47758	0.136
	$iter_{max} = 5$								
4	495	10482	0.014	496	20378	0.027	496	30559	0.042
5	335	17164	0.030	349	33287	0.058	348	49672	0.088
6	168	25439	0.053	179	49786	0.102	182	74402	0.157
7	80	32616	0.081	92	64191	0.158	98	95550	0.238
8	5	39839	0.110	10	79471	0.233	14	118938	0.365
	$iter_{max} = 10$								
4	497	19065	0.027	496	38605	0.054	497	56972	0.079
5	344	33327	0.064	350	65098	0.200	354	97695	0.180
6	175	50142	0.112	183	98694	0.341	189	147067	0.327
7	90	64436	0.171	97	127348	0.351	102	189738	0.526
8	7	79537	0.238	10	158839	0.507	17	237127	0.781

Table 4.2: Results for 2-exchange-random better-parallel edges.

allowed. As already proposed by Gomez and Oliveira [38] *random-better* and *best* strategies outperform the *first better* strategy clearly. However, runtimes for $iter_{max}$ larger or equal to 5 reach values that might not be acceptable for applications described in previous sections. We further note that results for the Tabu-search are not competitive with the 2-exchange heuristic in terms of best results. However, taking computational effort into account quite good results can be yield by the Tabu Search approach. Furthermore, no clear conclusion on the influence of different settings of l on the obtained results can be drawn.

Results for the Tabu Search algorithm, using the no-fit polygon method, are reported in table 4.5.

The first main conclusion of the comparison of the parallel edges and the no-fit polygon approach is that, although the no-fit polygon is the more general approach, no clear dominance of one technique over the other can be observed.

parts	2-exchange-best-parallel edges								
	$\delta = 1$			$\delta = 2$			$\delta = 3$		
	feas	nests	time	feas	nests	time	feas	nests	time
	$iter_{max} = 2$								
4	492	4622	0.006	495	8699	0.012	495	12779	0.017
5	321	7181	0.013	344	13698	0.024	344	20308	0.036
6	158	10373	0.022	177	20177	0.046	177	30102	0.068
7	75	13179	0.034	89	25904	0.072	99	38545	0.108
8	5	15965	0.046	8	31872	0.102	11	47721	0.155
	$iter_{max} = 5$								
4	495	9958	0.014	497	19477	0.027	497	28933	0.040
5	335	17154	0.033	348	33162	0.063	348	49515	0.093
6	159	25498	0.101	180	49606	0.120	181	73991	0.179
7	78	32561	0.196	91	63982	0.193	102	95190	0.289
8	6	39809	0.132	12	79360	0.296	13	118819	0.456
	$iter_{max} = 10$								
4	495	18018	0.014	497	36397	0.051	497	54253	0.077
5	335	33404	0.033	348	64912	0.126	348	97245	0.188
6	159	50458	0.101	180	98026	0.241	181	145811	0.356
7	78	64677	0.100	92	126788	0.403	102	189102	0.589
8	7	79216	0.131	13	157308	0.616	13	236779	0.976

Table 4.3: Results for 2-exchange-best-parallel edges.

This is a result of situations where placing parts on worse points, in terms of bottom left positions, yields in better overall packings. Secondly one can see that runtimes are dramatically reduced by using the parallel edge approach. Even for higher values of $iter_{max}$ the average values hardly exceed a tenth second. This shows that the parallel-edge approach is able to compute competitive results with significant savings in terms of effort. Therefore it suites the described scenarios and their applications better than previously published algorithms for relative problems.

parts	Tabu Search-parallel edges								
	$l = 1$			$l = 2$			$l = 3$		
	feas	nests	time	feas	nests	time	feas	nests	time
	$iter_{max} = 2$								
4	491	5138	0.007	486	5207	0.007	488	5202	0.007
5	323	6523	0.012	316	6614	0.012	324	6464	0.011
6	152	7795	0.016	153	7874	0.017	156	7845	0.016
7	67	8599	0.022	75	8594	0.022	66	8620	0.022
8	5	8983	0.027	6	9084	0.028	5	9003	0.028
	$iter_{max} = 5$								
4	496	11461	0.016	490	10011	0.014	493	9459	0.013
5	341	14625	0.028	342	12844	0.025	336	12143	0.023
6	170	17882	0.042	169	15884	0.032	170	14741	0.034
7	85	19655	0.056	83	17378	0.049	83	16146	0.045
8	9	20800	0.073	8	18279	0.063	5	17344	0.059
	$iter_{max} = 10$								
4	497	21233	0.003	497	17705	0.025	495	14646	0.021
5	343	27856	0.054	346	22620	0.044	342	19043	0.036
6	179	34602	0.084	174	28066	0.069	175	23514	0.056
7	91	37556	0.115	98	30624	0.092	82	25565	0.074
8	8	40848	0.155	10	33160	0.124	9	27561	0.085

Table 4.4: Results for Tabu Search-parallel edges.

parts	Tabu Search-no-fit polygons								
	$l = 1$			$l = 2$			$l = 3$		
	feas	nests	time	feas	nests	time	feas	nests	time
	$iter_{max} = 2$								
4	493	4623	0.028	495	8704	0.034	495	12773	0.039
5	328	7167	0.051	343	13688	0.066	341	20337	0.081
6	158	10362	0.084	180	20120	0.113	176	30067	0.143
7	77	13113	0.124	90	25784	0.177	95	38454	0.230
8	5	15939	0.153	7	31831	0.223	8	47697	0.307
	$iter_{max} = 5$								
4	494	10425	0.036	495	20586	0.051	495	30441	0.066
5	336	17191	0.077	349	33243	0.114	347	49756	0.155
6	168	25462	0.141	184	49647	0.226	184	74387	0.314
7	84	32446	0.221	94	63938	0.382	101	95248	0.535
8	5	39819	0.266	7	79495	0.488	8	119121	0.765
	$iter_{max} = 10$								
4	497	19042	0.055	497	38061	0.079	496	57410	0.106
5	344	33439	0.124	348	65781	0.208	348	98539	0.288
6	173	50167	0.248	186	98360	0.446	186	147544	0.629
7	88	64484	0.425	98	127108	0.803	104	189174	0.954
8	6	79562	0.573	9	158787	1.086	11	237779	1.666

Table 4.5: Results for Tabu Search-no-fit polygons.

Chapter 5

Generalized Classification of Constraints

Different industrial applications of different bin packing and stock cutting problems result in various requirements and constraints. As diverse as their origins are their influences on resulting packings, cuttings or layouts. Although various forms of constraints appear in industrial applications, some basic types can be identified. Due to the diversity of the requirements, the proposed basic types are relatively general and contain a variety of sub types which are not listed completely. However, the discussed framework should enable the reader to identify, and thereby classify, the most common constraints that appeared in literature. Further, the classification should hold for all problem types within the family of cutting and packing problems, therefore a more general nomenclature is used in this chapter. In the following the proposed types are explained and some examples are given.

5.1 Combinatorial Constraints

Combinatorial constraints deal with combinations of small items in large items or more general the distributions of small items over large items or levels of large items.

5.1.1 Multi Bin/Strip/Level Sequential Constraints

Multi regional sequential constraints describe a class of restrictions concerning the allocation of small items over more than one placement region. A placement region can either be a single large item, or a level within one large item. This

region might not have fixed dimensions, as for examples levels in strips. It is important to stress that these constraints to do not concentrate on the layout of small items within regions, they rather restrict the distribution of small items over placement areas for a higher purpose. Their main characteristic is that, in general, for their validation the information about subsets of small items in more than one region is required. This is, technically speaking, what separates them from subset constraints, which are discussed in the next section.

Constraints that focus on sequential orders of small items in a single region are also often referred to as sequential constraints. But due to the ability to check feasibility of packings only on the layout of parts in a single region, they are part of another class of constraints, in the proposed framework, and discussed in section 5.2.3.

Multi regional sequential constraints play an important role for problems dealing with some real world restrictions on due dates, limited buffer space or other machining restrictions. They mainly appear where the set of small items can be divided in several subsets, mostly representing commissions or transport units. The production sequence of these subsets is in many cases not free to choose, what requires the modeling of sequential constraints.

The most simple, but perhaps also most important example are due date constraints, see Reinertsen and Vossen [60]. Due dates require commissions to be finished either before a fixed time point or at before other commissions. In both cases the order in which small items are allocated to large items, and as a consequence produced, is constrained. In times where the reduction of storage costs becomes more and more important, these constraints obviously gain significant interest and are part of many real world applications of cutting and packing.

The other most relevant example of multi regional sequential constraints are restrictions on combinations of commissions, or more general subsets of small items, that can be produced at time. This is mostly motivated by space restrictions, as for example in Rinalid and Franzl [61], or machining restriction. Single commissions are stapled and thereby subsumed in defined areas that are only available to a certain extent. These constraints are mostly referred to as sequential constraints in literature, but to the belief of the author the term should also contain due date like constraints as described above.

The COP problem, discussed in previous chapters, contains both versions, due dates represented by constraint (1.3) and combinatorial restrictions represented by (1.4). Furthermore, it provides another example for sequential constraints, where small items within one commission have to satisfy a strict ordering, as imposed by (1.2).

5.1.2 Subset Constraints

Subset constraints are restrictions on possible subsets of small items within one large item. They do not have any effect on the layout of small items within the large item and never consider more than one large item. The mainly forbid certain sets of small items to be placed together in a single bin. Although they are not exhaustively discussed in literature, some examples should be mentioned due to their relevance for industrial applications.

Conflicts, the most simple form, are often referred to as constraints that forbid pairs of small items to be placed together in a single large item, see for example Epstein and Levin [45]. More general versions of subset constraints are given by so called color and cardinality constraints. First deal with problems where a each small item has a given color and the number of occurring colors within one large item is limited by a chosen value, as for example proposed by M. Dawande et al. [22]. Cardinality constraints require the number of small items within one large item to be between given lower- and upper bounds. Marques and Arenales [52] or Babel et al. [3] state different problems containing cardinality constraints.

5.2 Layout Constraints for Small Items

Constraints on the layout restrict possible position of small items in a single large item. They range from restrictions that deal only with positions of small items to the large items, to restrictions on the positions of all small items to each other and to one large item. Therefore they can be divided in the following subclasses.

5.2.1 Position of Small Item with Respect to a Large Item (PSLI)

This class contains constraints that define relations between the position of a small item and the large item. In other words it defines conditions on the position of a small item within a large item. Besides the obvious constraint that small parts must be fully placed within the large item, examples are rotational constraints, defective areas and many more.

Rotations

Any restriction of possible rotations of small items can be seen as restrictions on possible positions of small items with respect to the large items. They occur in several forms, some forbid rotations at all and some allow only rotations of

orthogonal angles. There are also many examples where different subsets of small items are connected with different constraints, i.e. only some items are allowed to be rotated. A more complex version of rotational constraints was introduced by Birgin and Lobato [7], where parts might be rotated by any degrees of the form $\alpha + k * 90°$ for $k \in \{0, 1, 2, 3\}$ and a fixed α for all small items.

Defective Areas

Defective areas are often referred to as regions within a large item that must not overlap with any small items. In a more general way they can be limited to sets of small items, meaning that only certain small items must not overlap with a defective area.

General Position Constraints

Other examples may be requirements of the form that parts of a certain type must be placed at a given border of the large item or must be placed in a certain region.

5.2.2 Position of Small Items to Each Other (PSS)

Constraints that deal with geometric relations of small items to each other are found in this class. Important is though that it only contains constraints that do not consider positions of parts with respect to the large item. As this definition is quite general, some examples are listed, which show some common subtypes. Since the relation between the positions of two parts is stressed in almost every cutting and packing problem it is described separately.

Position of Two Small Items to Each Other

Geometric relations between two small items are part of every cutting and packing problem as it is always requested that small items do not overlap.

Cutting Constraints

Guillotine cutting is often found by two-dimensional orthogonal problem types. It requires that all small items can be derived from edge to edge cuts from the large item they are nested in.

Another examples can be found in many industries where the cutting process produces kerf, as shown in figure 5.1. This requires that a certain distance is kept between any pair of small items.

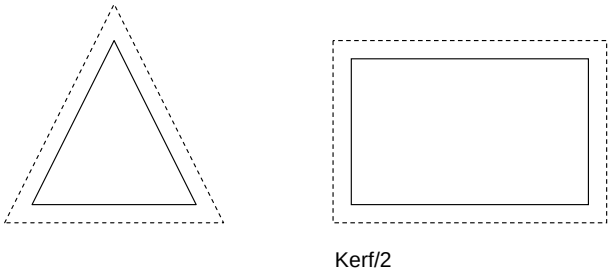

Figure 5.1: Example of items where cutting produces kerf.

Support and Loading Constraints

Support and Loading Constraints often occur for three-dimensional pallet loading problems. Support constraints ensure that for every box, boxes placed below provide enough support so that the box holds. Loading problems are the opposite, which ensure that boxes can hold the weight placed above.

General Position Constraints

For many applications it is required that certain small items are placed besides or within a certain distance. This can be found, for example, in the concrete industry where certain pairs of parts require special treatment routines and therefore need to be placed close together.

5.2.3 Positions of Small Items to Each Other and to Large Item (PSSLI)

This class is the most general one that contains all constraints that require packings or cuttings to have a certain form, meaning the whole resulting layout of small items within one large items must satisfy certain conditions with respect to the large item. Some examples are listed below.

Balance Constraints

For many problems balancing of small items is a key point, mainly because the weight of small items should be equally distributed over the large item for various

reasons.

Sequential Constraints

Sequential constraints on a single region ensure that small items are placed according to a given order within one region. They are also referred to as drop off constraints, which ensure that after removing a set of small items a given set is free admissible from one or more sides of the large item. For instance, when loading a truck it must be ensured that after removing items belonging to a commission the items of the next commission are reachable, without rearranging, from the door of the truck.

Regional Constraints

Depending on the application, large items often contain different quality regions. Unless in many cases this has only an effect on the optimization criteria it can also be demanded that, out of a given set of small items, enough items are placed within a region. Examples range from shelf packing problems to the leather industry. For shelf packing certain areas in the shelf provide a better visibility, as people tend to look at certain regions first and more often. Therefore in many problems a set of products, or at least a big enough sub set of them must be placed in certain regions. As leather stocks are obviously a natural product the quality might vary over a single leather sheet. Products of higher quality therefore have to be cut out from certain regions, what requires the formulation of regional constraints.

Level Packing

Level packing is often required for two-dimensional orthogonal problems. Basically small items are placed in levels within a strip or bin, meaning that all items in one level are placed on the same height coordinate. The next level starts then at the highest point of all items of the previous level, as illustrated by figure 5.2.

General Position Constraints

A very special constraint, but an good example for this class, can be found in the concrete part industry, where for some plants and some types of small items it is required that items are attached altering at the upper and lower border of the large item. Resulting layouts should have a form as shown in figure 5.3.

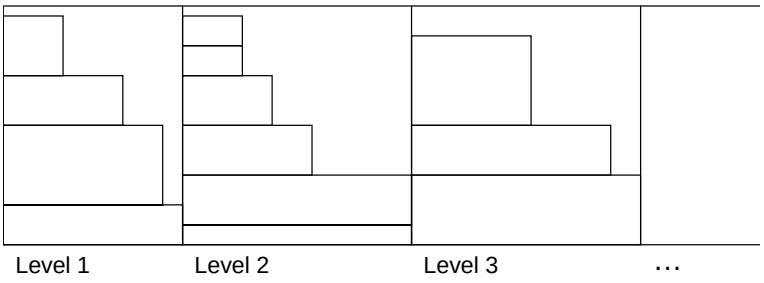

Figure 5.2: Level Packing.

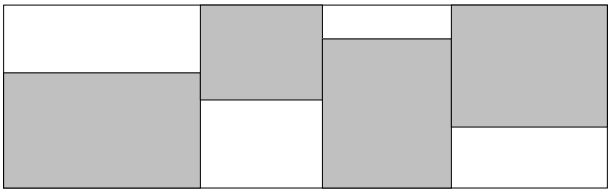

Figure 5.3: Altering attachment of small items.

5.3 Influence of Constraints

In a formal and strict way all existing constraints ensure that requirements on distributions-, sub sets- and layouts of small items are always satisfied. However, consider the following example: Several small items are cut from a single rectangular large item. After cutting small items are removed by automatic grappler. Latter is not able to reach the very left- and right side of the large item. Therefore placing thin items at the very end of the large item (in both directions) results in unproducable layouts. However it might be possible that non-reachable items can be removed by hand from a worker, what obviously takes longer, but a resulting tighter layout of small items might be from greater interest than the loss of time.

Especially for the concrete part industry scenarios like these are very common. Thus constraints that affect the optimization criteria, or in other words soft constraints, have to be modeled. In contrast to real constraints, or so called hard constraints, that actual define rules for placing small items, soft constraints extend the objective function. How to model these extensions is dependent on the underlying problem. The basic options are scaling or the definition of multi-dimensional objective functions.

Chapter 6

Constraint Handling Incorporating a Specific Problem Domain

In this chapter some general thoughts on the handling of constraints for multi-constrained problems are reported. Furthermore, they are explained using the example of a real world application, namely an automatic and semi-automatic planning tool used for production of precast-concrete-parts. This chapter serves the purpose to show the principles of handling and evaluating these constraints.

6.1 General Handling of Constraints

To formulate a general framework to handle constraints of cutting and packing problems is a very critical task. In the last chapter we have seen several classes-, types- and examples of different constraints. To find suitable techniques to model and also handle them, independently from a solving strategy for the underlying problem, is not reasonable in many cases. This is especially the case for heuristic solving methods that constructively build solutions at some stage. A classical example, that is subject of many cutting and packing problems, are guillotine cuts. As mentioned in the last chapter guillotine cutting requires that all small items can be obtained by an edge to edge cut from the remaining large item. Several constructive heuristics deal with that issue and try to find feasible lay-outs using highly sophisticated, optimized, but also very diverse strategies. Any general handling would only limit the diversity of these approaches and would therefore be counterproductive.

On the other hand a lot of constraints only differ in their origin but are quite

similar in terms of structure and purpose. For these a general structure might have some advantages, which could include, for example, generality or extensibility. Furthermore, most of multi regional sequential- and sub set constraints can be handled in the same way. To the belief of the author it is therefore worth to re-classify constraints according to their possibilities of general handling.

Note that these handlings and classifications base on the assumption that layouts are gained constructively by translating parts to placing points, as described in chapter 4. Therefore constraints on the layout have an influence on the computation, or at least evaluation of possible placing points. In particular hard constraints make some resulting placing points infeasible, where soft constraints have only an affect on the fitness value of placing points, which was in chapter 4 only defined by the bottom-left position of points. Further, note that a more general handling for non-layout constraints is omitted since this would depend too much on the underlying solving method. For all types examples coming from a real world application are given.

6.1.1 Non-Constructive Constraints

Non-constructive constraints (NCC) are conditions that do not have any effect on the computation of placing points at all. The might make some geometrically possible placing points infeasible, but do not require the computation of extra placing points. An illustrative example is the condition that parts of a certain type have to be attached to a specified border of the bin. As long as the left border is not completely covered by other attached parts most strategies to compute placing points will return at least one point that result in a layout where the item is attached to the left border.

This class mostly contains constraints of type PSLI but also PSS, for example support constrains. Evaluating a placing point becomes an issue of checking which boxes have been placed below and if they provide enough support, where in general no additional points have to be calculated.

6.1.2 Semi-Constructive Constraints

Semi-constructive constraints (SCC) are constraints that basically work the same way as NCC constraints, but require the computation of extra placing points, if all points that were returned from the standard procedure are infeasible. Therefore they can also be seen as placing point generating constraints.

Consider the example illustrated by figure 6.1, where the gray regions mark defective areas which must not overlap with any small item placed. Any point

computing routine would only return points A,B,C or D resulting from the inner-fit rectangle or parallel edges for item I. That would mean that no feasible point for item I can be found and item I would become unplacable. Therefore it is not enough to just evaluate placing points according to the defective area constraint, it is also necessary to obtain placing points from it, as for example E,F,G and H.

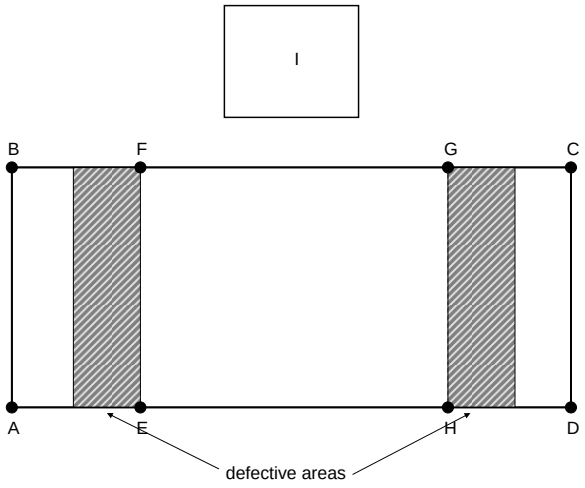

Figure 6.1: Placing points resulting from defective areas.

The main characteristic of semi-constructive constraints is that they only add areas, edges or points to the problem which the point computing algorithm has to consider. However, they never affect the way that placing points are computed.

6.1.3 Constructive Constraints

Constructive constrains (CC) have a direct influence on the strategy by which placing points are computed. They are nested within the routines that return possible placing points and therefore can neither be generalized nor evaluated in a general way.

The simplest examples are cutting procedures that produce a kerf. The constraint is not evaluated at some point but for obtaining placing points a slightly larger item is used. More complex examples will be given in later. Most examples can be found within PSS and PSSLI constraints.

6.2 Handling in an Algorithmic Way

It is very important to add at this point that NCC, SCC and CC are not an exclusive partition of constraints. Depending on the solving approach, modeling or just personal preferences, the same constraint could be either in one or the other class. When a certain balancing conditions on the layout of small items is required, one basically has the possibilities to compute the standard placing points and choose the one where the balance is optimal (NCC) or use the condition to compute the really most balancing point (CC). Other examples are guillotine cutting, level packing or even the attachment of small items to a certain border of the large item.

Especially the last example shows the tradeoff between computational effort and generality of an approach. Obviously using the constraint that a certain small item must be placed at the left hand side of the large item can reduce the costs for computing possible placing points. On the other hand working on real world applications might include a big number of constraints from this type. Therefore using all of them within the placing point generating routine can easily lead to a very complex algorithm that is not extendable and maintainable at all.

In the following the handling of several constraints that are subject of a real world problem based on COP are introduced. The problem is solved using the RA^* algorithm from chapter 3. Geometry issues are handled by a constructive approach using the 2-exchange algorithm combined with the parallel edge technique form chapter 4, thereby polygonal shaped parts can be packed. Note that, although the proposed strategy could be used for other problems as well, it is not attempted to serve for are general framework or purpose. In the following small items are referred to as parts and large items as bins.

6.2.1 A Real World Example

The discussed real world application is mainly based on the COP problem discussed in previous chapters. However, instead of rectangular parts any polygonal shaped parts might be placed and orthogonal rotations are allowed. Furthermore, as mentioned before, parts are surrounded by iron shutters and concrete is poured in these shuttered areas. At a glance these shutters could be seen similar as a kerf that results from cutting procedures and therefore makes parts only larger. But as we will see in the following discussion this is not the case in reality. Further, in addition to constraints (1.2), (1.3), (1.4) and (1.5) there exist a vast number of other constraints, on the layout and also on the combinations of parts within a single bin. These constraints have been omitted during the discussion of COP

since they are not structure giving and do not have a big influence on the performance of global algorithms. Due to the large number of them it is still worth to discuss a proper modeling and handling of the latter.

6.2.2 Handling of Combinatorial Constraints

Multi regional sequential and sub set constraints can be handled directly in the RA^* algorithm. When computing all possible adjacent vertices of the current node, possible combinations are checked on the required constraints. Note that both, hard- and soft constraints can be handled that way. Where hard constraints cut down the number of possible adjacent nodes, still neglecting geometrical feasibility, soft constraints and corresponding penalties can be added to the edge length, which becomes then

$$|e| = \beta - \frac{\sum_{p_{i,j} \in AP_{\tilde{S}_2} \setminus AP_{\tilde{S}_1}} l_{i,j} w_{i,j}}{LW} + \mathrm{p}_e, \tag{6.1}$$

where p_e is the penalty value.

6.2.3 Handling of Layout Constraints

To model SCC and NCC constraints within the algorithm proposed in chapter 4 an objective oriented data structure has been added to the problem. Based on this structure algorithm 4 was extended by evaluating functions of these constraints. As mentioned above constructive constraints cannot be handled generally and are part of placing point finding strategies.

The choice of an objective oriented approach was mainly motivated by the fact that for the discussed real world problem there exist constraints that are not active for all parts. This results from different types of parts and also varying preferences of different producers. In particular production planners are able to switch certain constraints on and off, or in case of soft constraints assign different penalty weights.

Each part holds two list of so called active constraints, constraints that are switched on, one for hard- and one for soft constraints. Each constraint is represented by an object, providing the corresponding penalty weight, a method to evaluate the position of a part and, in case of a semi-constructive constraint, a method to compute additional placing points, or if necessary additional edges or polygons to compute points. On the other hand, a placing point does not only hold the corresponding x- and y-coordinate, it also holds the overall penalty of

the part placed on it. Thereby it is possible to evaluate only relevant constraints per part and keep a good usability for production planners.

Finding the best placing point for the next part in the placing sequence becomes now a matter of computing all possible points, evaluating them and finding the point with the smallest penalty, where splits are resolved over the bottom-left criteria. The procedure is summed up by algorithm 6 and 7.

Note that constructive constraints are not modeled as objects in this framework. They are rather nested within the procedures to calculate placing points and also the procedure CheckResultingLayout(). To choose the constraints, which are modeled as constructive is a critical task and is heavily dependent on the underlying problem. As an example could be mentioned that it would be possible to model the usage of shutters as a SCC by simply generating edges within the constraint that represent the enlarged part. Tests if a placing point actually uses the enlarged parts and enough space for shutters is left, could then be done within the CheckLayout() procedure of the constraint. Obviously this is a rather cumbersome approach compared with a preprocessing step that enlarges parts in advance.

6.3 Sample Sub Set Constraints

What follows is a sample of implemented real-world sub sets constraints in addition to (1.5). Furthermore, the need and purpose of each constraint for the production procedure is explained

6.3.1 Limited Difficulty per Bin

As described above bins sequentially traverse several work stations during the production process. At each work station one or more tasks are executed. Although having bins with a very tight layout is favorable, exceeding a certain difficulty level of parts nested in one bin, can lead to considerable delays at these workstations. The gain from the tighter layout is then lost by the bottleneck produced by a very 'difficult' filled bin. The most general solution to this problem would be to ensure that bins are filled almost equally in terms of difficulty. However, this led serious problems in real world application for several reasons. Therefore a maximum level of difficulty for each bin is introduced. The difficulty of a bin is computed by adding up difficulty indicators of single parts to be nested. These parameters are calculated in a preprocessing step depending on number of edges, compared to their length and the occurrence of so called mountparts. Mountparts

Algorithm 6 Placing Points from parallel edges with constraint handling

```
1: PP = {};
2: for all rotations in {0°, 90°, 180°, 270°} do
3:    rotate part; PL = {}; EC = {} (Edges From Constraints);
4:    for all constraints CO assigned to P_i do
5:       PP = PP ∪ CO.FindPlacingPointsFromConstraint();
6:       EC = EC ∪ CO.FindEdgesFromConstraint();
7:    end for
8:    for all parts P_l on strip S do
9:       compute all pairs of edges of part P_l, P_i and EC that satisfy (4.1) and
         put resulting placing lines in PL;
10:   end for
11:   CaclulatePlacingPointsFromParEdges();
12: end for
13: for all points P in PP do
14:   for all hard constraints HC assigned to P_i do
15:      if ¬HC.CheckLayout() then
16:         delete P from PP;
17:      end if
18:   end for
19:   for all soft constraints SC assigned to P_i do
20:      P.Penalty+ = SC.CalcPenaltyOfLayout()H;
21:   end for
22: end for
23: while P ≠ {} do
24:   find point P with smallest penalty in PP; translate P_i to p;
25:   if CheckResultingLayout() then
26:      return {P}
27:   else
28:      delete P from PP;
29:   end if
30: end while
31: if PL = ∅ then
32:   return NofitProcedure();
33: end if
```

are for examples doors, windows, power outlets or connectors for facilities.

Algorithm 7 NofitProcedure

```
1: CalulatePlacingPointsFromNofitPolygons().
2: for all points P in PP do
3:    for all hard constraints HC assigned to P_i do
4:       if ¬HC.CheckLayout() then
5:          delete P from PP;
6:       end if
7:    end for
8:    for all soft constraints SC assigned to P_i do
9:       P.Penalty+ = SC.CalcPenaltyOfLayout()H;
10:   end for
11: end for
12: return {p} with smallest penalty in PP.
```

6.3.2 Preferred/Maximum Number of Parts per Bin

For similar reasons as for the limitation of the difficulty of a filled bin, some plants require the limitation of the number of nested parts in a single bin. Thereby it has to be distinct between a soft constraint that defines the preferred number of parts per bin. Further, a hard constraint simple forbids combinations of more parts than a free chosen number to be nested together in a single bin. Both, and also the difficulty constraint, are classical cardinality constraints.

6.3.3 Half Parts on One Bin

Half parts are special parts that basically split a level in a stack into two parts. However, for stapling issues it is important that they are placed in the same bin. Combinations are therefore checked on containment of none or both of every pair of half parts.

6.3.4 Conflict Constraints

Constraint (1.5) is a typical conflict constraint. If forbids parts with different material qualities to be placed together in the same bin. Although all conflict constraints could be modeled as one constraint, they are still seen as different constraints in the discussed real world problem. The main reason for this is that different plants might have different possibilities and therefore do not need all of them, therefore they need to be adjustable in a comfortable and easy way. The most important examples are:

- same concrete quality
- same part type (wall, ceiling, stone part...)
- same part height (third dimension)

6.4 Sample Layout Constraints

In the following a sample of layout constraints is presented and discussed. Thereby examples for each class of both, the general and the algorithmic, classifications are given.

6.4.1 Constructive Constraints

Shuttered Parts

Although, at a glance, shuttered parts seem to be the same as a kerf, the situation is more complex. Using separate shutters for each part only enlarges parts, however, for parallel edges the same shutter might be used on both sides depending on the circumstances. This has an affect on the calculation of placing points, and on the feasibility of layouts. In case of parallel edges standard procedures do not work anymore. While enough space between original parts has to be kept, the enlarged parts might overlap and still a feasible layout results. Feasible overlapping mainly occurs for two reasons, as shown in figure 6.2 (i): enlarged parts overlap because of the overlapping area (A) of shutters and secondly because of adjacent shutters (B). Situation B is resolved by using a shutter that only touches the parallel shutter and filling the hole with especially shaped polystyrene parts, as shown in figure 6.2 (ii). This method is also used when corners are in short distance of edges and their shutters, as illustrated by figure 6.3, where the unshuttered area is also filled with a polystyrene part.

These settings lead to a split approach, for each part the original and the enlarged shape are kept. Further, techniques to compute placing points and approaches to check resulting layouts are extended:

Calculation of placing points During the computation of placing lines resulting from parallel edges the minimal distance that must be kept between these edges, is computed. This distance results from the ability of using a single shutter and its width. According to this distance a shifted placing line is stored instead of the original one.

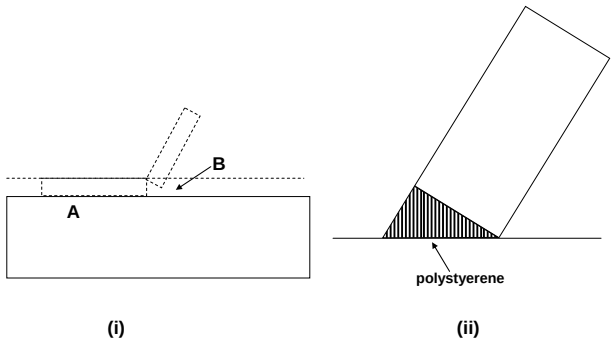

Figure 6.2: The use of shutters.

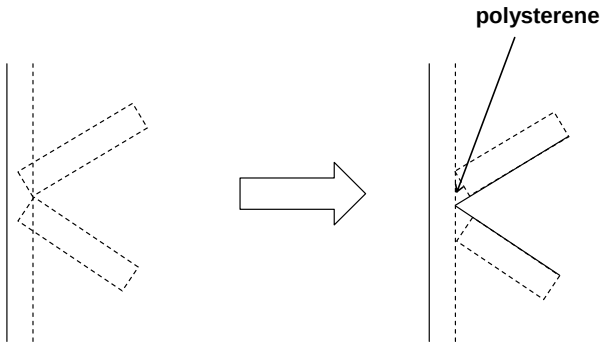

Figure 6.3: Shutters and corners.

While the calculation of placing points resulting from the intersection of two placing lines is still done in the same way, for placing points resulting from a single placing line some minor adoptions have to be made. Instead of using points and edges of original parts, the enlarged part of already placed parts and the original part to place are used to obtain points, as shown by figure 6.4.

Check on the feasibility of layouts. The check on the feasibility of the layout of an already placed part p_l and a newly placed part p_i is done in two steps. First the enlarged version of part p_l is checked on overlapping with the original part p_i. There, overlapping is strictly forbidden. In the second step parallel edges are checked separately on their distance. The latter must either be equal to the width of one shutter or bigger than the width of two shutters. The first case is only considered feasible if and only if these two edges are allowed to use one shutter on both sides.

For some applications it is also required that between edges with very acute

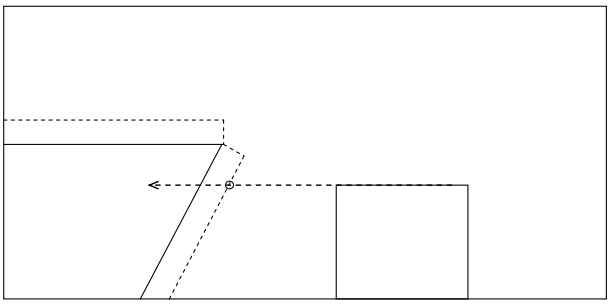

Figure 6.4: Placing points from a single parallel edge involving shutters.

angles, as shown in figure 6.5, there has to be space for an extra shutter. This avoids the need of too difficult polystyrene parts.

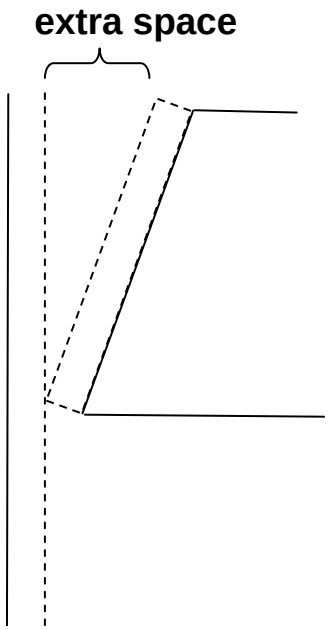

Figure 6.5: Extra space between acute edges.

Double Walls

Double walls are a very common product in the concrete industry. Basically double wall parts consist of two concrete plates that are connected with iron grids. Once brought to the building site and into position, the space between plates is filled with concrete what results in very solid walls. This requires the

production two separate plates. After producing the first it is hardened with the iron already attached, flipped it over and connected with the second, where the concrete has not hardened yet. The critical step of this procedure is obviously the flipping of the first plate. Thereby the plates are fixed on the bins and the latter is flipped. Therefore the two plates are placed with the right offset in the two different bins, as illustrated by figure 6.6.

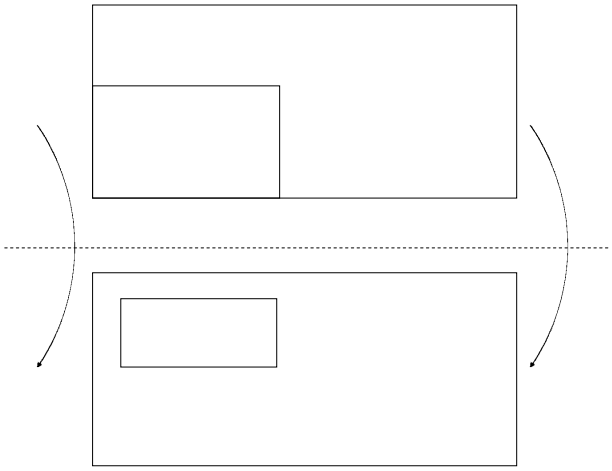

Figure 6.6: Principles of double walls.

Unfortunately in general, as one can also see from figure 6.6, the two plates do not have the same dimensions. However, for simplicity we can assume that double wall parts have only rectangular shapes. The first and trivial attempt to resolve this problem would be to find an enveloping shape of both plates and to compute layouts for latter. However, in many cases it is required that first- and second plates of different parts can be placed overlapping, see figure 6.7. This requires a more sophisticated method to gain and check the distance between two parallel edges during placing points computation and evaluation. In particular an enclosing rectangular for both plates is kept instead of the original part and in addition for each edge the distance to plate one and two is also stored. With this information it is possible to calculate a shifting length for parallel edges, which ensures that the distances between plates one and two of both parts is either equal to the width of one shutter or bigger than the width of two shutters. Figure 6.8 shows this technique. Furthermore, the check on the feasibility of a layout has also to be slightly modified.

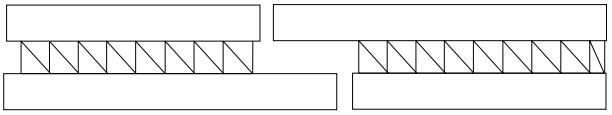

Figure 6.7: Overlapping double walls.

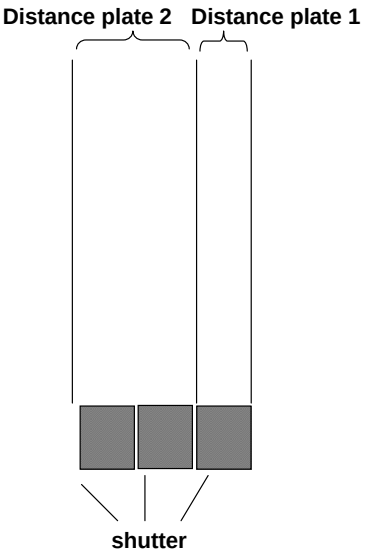

Figure 6.8: Distance between plates.

6.4.2 Semi Constructive Constraints

Mountparts and areas

As mentioned before mountparts are doors windows and etc. that are placed on a fixed position within the part. In many plants mountparts are inserted with cranes that are not able to reach all areas of a bin. Therefore mountparts must not be placed in these defective areas, what requires that the parts they are nested within are not placed in corresponding areas. These corresponding areas result from the relative position of mountparts to their 'parent'-parts. Fortunately the defective areas can be modeled as orthogonal strips, as shown in figure 6.1. To obtain extra placing points, equivalent to E,F,G and H in figure 6.1 additional edges are added, where attaching the part to these edges results in a touching layout of the mountpart and the defective area, see figure 6.9. This constraint can be seen as PSLI according to the classification of the last chapter.

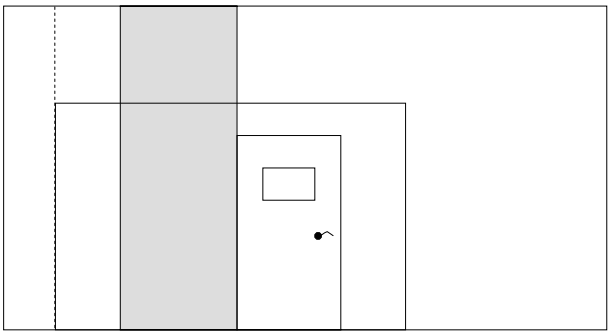

Figure 6.9: Placing points from door-mountparts.

Half parts

Half parts where already subject of a sub set constraint, namely that they have to be placed in the same bin. But placing them in the same bin is not enough to ensure producability. Moreover they have to be placed above each other, while the ordering of them does not matter. Figure 6.10 shows two different feasible layouts, where halfparts are displayed in gray. One may note that half parts are another example for PSS constraints.

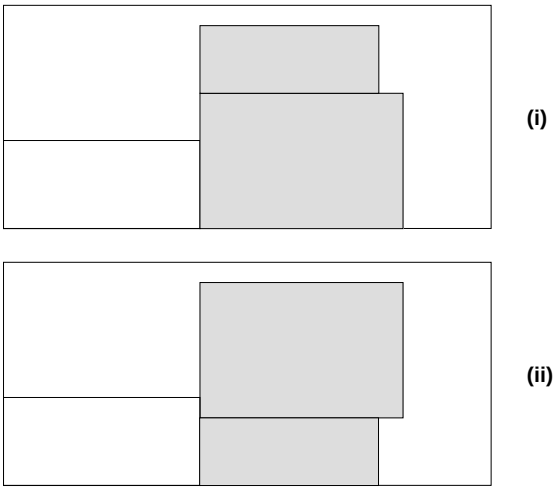

Figure 6.10: Possible positions of half parts.

Shift Parts with Iron From Side

Parts with an iron-overhang must not be attached to left and right bin sides, at least they must be shifted a little towards the center of the bin. For bins with a length of approximately ten meters the shift normally does not exceed two centimeters. During the planning step parts and their overhangs are only given in an accurate drawn plan. According to the plans, overhangs would not reach over the bin side when attaching the part to it. In the real production process inaccuracies can occur while bending the overhangs, as shown by figure 6.11, and they might reach out of the bin. To avoid this, parts have to be shifted a little bit inwards. This is realized by adding extra edges, as one can also see in figure 6.11. Since the constraints concerns only positions of parts with iron overhangs with respect to bin sides it can be classified as PSLI and SCC.

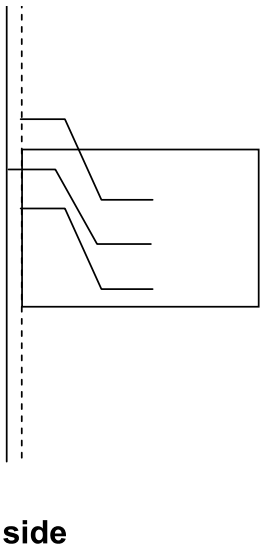

Figure 6.11: Bending errors of iron-overhangs.

6.4.3 Non Constructive Constraints

Fine Edges

Some edges of parts are marked as fine, which means that they will form an outer edge of the overall resulting building. Therefore any inaccuracies and small errors are more visible in the end product than for other edges. Since bins normally have fixed shutters of higher quality at their sides one wants fine edges to be attached to them. This constrains exist in both version, soft and hard, depending on

preferences of different plants. It can also happen that a part has more than one fine edges, in that case, the aim is at least one of them is attached to a preferred border.

This constraint is classical example for being non-constructive and PSLI, any point that does not result in the attachment of at least one fine edge of the part is either deleted or punished.

Attachment

In this context attachment constraints are referred to as constraints that force parts of certain types to be attached to defined borders of the bin. Moreover in some cases also altering layouts, as described by figure 5.3, are requested. For altering layouts placing points are evaluated over the position and sequence of previous placed parts. Possible specifications are:

- all parts parts attached to either lower or upper border
- all parts attached to lower border of bin
- all parts attached to upper border of bin
- altering, starting at lower border
- altering, starting at upper border
- altering, starting at any border

Note that these settings might vary over different types of parts and the constraint is of type PSSLI.

Balancing of Parts

Balancing of parts is an issue in many packing problems. In this particular example it can be necessary to place parts altering left and right according to the center point of the weight of already placed parts. This is mainly to avoid that parts placed on one half of the bin are not to heavy compared to parts placed in the other half. For simplicity this constraint is modeled non-constructive. Placing points are evaluated on their contribution to the balance of the resulting layout. Although this is neither a very general nor sophisticated approach, it fulfills the requirements of the underlying problem.

Maximum Number of Parts in Y-Direction

To avoid too complicated layouts some production planner do not want the number of parts aligned in y-direction of a bin to exceed a fixed value. This can easily be modeled as a non-constructive constraint that forbids placing points that would result in a higher number of parts aligned in y-direction. Consider the example given by figure 6.12, assuming that the number of parts in y-direction is limited by 4. Translating the next part to point A would result in 5 parts above each other and is therefore not feasible, whereas point B results in a valid layout.

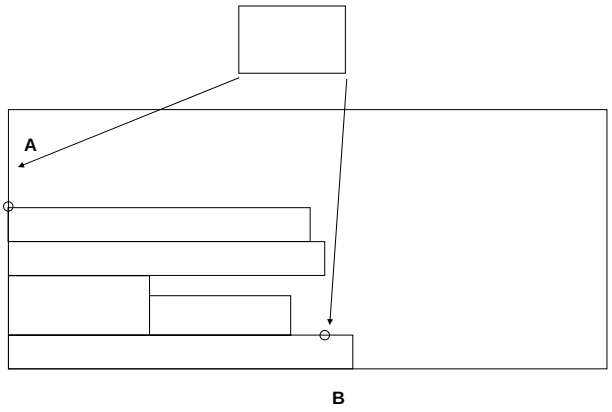

Figure 6.12: Feasibility of placing points depending on y-direction constraint.

It is simply counted how much parts would reach into the strip generated by the boundaries of the translated part and the resulting number is compared to the maximum number. The only information needed for evaluation are the positions of previous placed parts, therefore the constraint is member of the PSS class.

6.5 Discussion

In this chapter a classification of constraints with respect to algorithmic handling, under the domain of certain solving strategies, was introduced. Thereby three main types of layout constraints can be identified, namely NCC, SCC and CC. In general these classes are not exclusive; according to the developer, certain constraints can be either SCC/NCC or CC. However, given a problem with a specific set of constraints, the chosen design and corresponding distribution of constraints over the three classes, has a striking influence on the quality of the intended algorithm. In the process additional aspects have to be considered.

From a theoretical point of view SCC and NCC can slow down the point generating routines within the algorithm. This is because all points, even infeasible ones, are first calculated and then evaluated with respect to the constraints. Using the constraints during the point calculating process would lead to less points, and therefore less computational effort. Consider the example where a part has to be placed on the lower boarder of a bin, only points, and therefore edges on the lower boarder have to be taken into account when computing placing points. However, for practical applications this effect can be lost when the number of constraints exceeds a certain size. A big number of CC makes the methods cumbersome and the time savings often almost vanish. Tests have shown that in practical applications with a quite small medium number of layout constraints (2-5), the savings are negligible small.

On the other hand modeling only CC constraints makes algorithms and code hardly extendable and adaptable, as for the the precast-concrete-part industry, where each plant has different requirements and new products are launched on a regular basis. New products lead to new parts, which often have special requirements during the production process, which lead to new constraints. With a clean objective oriented modeling of NCC and SCC these constraints can easily be added without changing the core algorithm. Further, constraints can easily be 'switched' on and off to fulfill every company's special needs.

To sum up, for problems with a small set of constraints, which does not change over time the modeling of NCC and SCC can lead to small drawbacks in terms of computational effort. For medium to large sets of constraints, which might also be changing over time, these drawbacks hardly exist and the benefit of extendibility and adaptability clearly dominates.

Chapter 7

Conclusion and Further Work

This thesis dealt with a highly constrained bin packing problem in the precast-concrete-part industry. The discussion of this problem included a proper mathematical formulation as well as representation of the problem itself and the optimization objective, the adaption of standard solving procedures and the development of a more problem specific technique. During this process also lower bounds for COP were discussed. Finally, the work deals with different types of constraints that have occurred within C&P problems over the last decades. In the following the main aspects are summarized and some ideas for further research are given.

7.1 The Model

A basic mathematical model, that comprehends all main aspects of real world problems in the proposed industry, was formulated. The aim was to keep the model as simple as possible, without losing the structure which makes it differ from standard problems. Therefore stacks and allocations of parts were defined. Furthermore, four different combinatorial constraints, which are part of almost every real world application in that field, were introduced.

The first constraint is a standard conflict constraint that allows only parts made of the same material quality to be placed in the same bin. The second one ensures that parts of a single stack are placed in bins according to their levels in the stack. This avoids time consuming reordering on actual building sites and is part of almost every ceiling production as well as most wall productions. Thirdly, the number of stacks that can be produced at a time is limited to a given parameter. Due to limited space in so called stapling areas every producer has to deal with restrictions of this form. Finally, stacks themselves must be produced

in a given order, which might be violated to some extent. In all companies in which the production is at least near 'on demand' stacks have to be finished in the order of their delivery dates.

Further, one very basic representative of layout constraints was added to the model. This was mainly because local search routines should also work and be tested under circumstances where they are not allowed to place parts free in bins.

7.2 Solving Methods

After a brief discussion of possibilities for solving COP to optimality in chapter 1, standard heuristics for bin packing were discussed in chapter 2. Two classes of heuristics have been defined; sequence based and set based algorithms.

Sequence based algorithms use nesting heuristics to place parts in bins in a given sequence and a global algorithm to control the search over different sequences. Since the ordering of parts is a key aspect of COP two extended models for sequences were defined and tested, *weakly and strongly ordered sequences*. Further, well known nesting heuristics were adapted for computing solutions of COP. The first main result of this work was that *strongly ordered sequences* lead to significantly better results than *weakly ordered sequences*. Although *strongly ordered sequences* lead to a smaller possible neighborhood and therefore a more narrow search, *weakly ordered sequences* ask for more complex nesting heuristics and placing rules. This leads to cumbersome routines and therefore to worse results. The second observation was that Simulated Annealing outperforms other heuristics, as Genetic Algorithms, for searching the space of different sequences.

Set based algorithms do not represent solutions with sequences, but they define operators directly on allocation of parts. Based on examples taken from literature a Tabu Search method, a Genetic Algorithm and a Greedy Algorithm were adapted and tested for COP. To conclude one can say that sequence based algorithms outperform set based methods.

However, the promising results of the Greedy Algorithm led to the formulation of a network search method. Thereby a network of decision nodes was defined. Nodes in this network represent partial allocations of parts, or in other words progress levels of the allocation process. Edges represent decisions to place the difference of parts between two levels in a single bin. The length of edges was defined according to the utilization of the bin's capacity. In this context a solution of COP can be represented by a path through this network, where the start node is the level where no part was placed and the end node represents a full allocation of parts. One may note that the Greedy Algorithm basically

chooses the cheapest adjacent edge in every node and thereby computes a greedy path through the network. Based on this network a special version of Dijkstra's algorithm was proposed which is able to compute optimal solutions of COP. Since for practical applications the network gets too big and Dijkstra's algorithm takes too much computational effort, a heuristic relaxation RA^* from an A^* algorithm was developed. Results showed that RA^* suited the special restrictions of COP better than the discussed standard heuristics. During that discussion a framework of lower bounds for COP was introduced. Furthermore lower bounds for bin packing problems were used within that framework.

7.3 Polygonal Parts

For the purpose of designing and evaluating solving strategies for COP polygonal shaped parts could be neglected. However, in industrial applications parts can also have polygonal shape or be at least approximated by the latter. Therefore in chapter 4 methods to deal with more complex geometrical tasks were discussed under the domain of strip packing problems. The wide-spread no-fit polygon approach was compared to a more practical method, which was especially suitable for the concrete industry. Results showed that for small instances, where parts tend to have mainly orthogonal edges the parallel edge approach, introduced in this work, is far more efficient in terms of computational effort than standard no-fit procedures. Although the no-fit approach is more general, the solution qualities do not differ significantly.

7.4 Constraints in General

Constraints appear in many cutting and packing problems nested in many different industries. One aim of this thesis was to define a general framework that allows the reader to classify various types of constraints. Thereby two main classes, combinatorial- and layout constraints, were identified, whereby each class can be divided in further subclasses.

Especially for layout constraints a second classification towards the algorithmic handling was proposed. It is important to add that this classification is only valid in the domain of point-generating placing methods within nesting routines. Further it is not general and can differ over problems, solving strategies and personal preferences. However, the classification can be assisting for a developer to identify constraints during the design process and evaluate their influence on

extendibility and performance of the intended method.

7.5 Further Research

During this project some tasks, that could be from interest for further research, were left undone. First, results showed that lower bounds computed by the framework in chapter 3 tend not to be very tight. Since a lot of exact solving procedures need tight bounds to perform well, meaning to find optimal solution for medium sized problems in a reasonable amount of time, the improvement of bounds could be an interesting topic. Furthermore, the formulation of more sophisticated exact algorithms, as methods like branch and bound, column generation or other linear programming techniques, is also left undone. Although results of RA^* were quite promising, it could still be of interest to investigate the performance of other heuristics or further enhancements of RA^* on COP.

The constraint classification mentioned in chapter 5 could also be extended by a detailed bibliography of practical constraints that appear in literature, since the examples given do not claim completeness.

Bibliography

[1] A. Albano and G. Sapuppo. Optimal allocation of two-dimensional irregualr shapes using heuristic search methods. *IEEE Transactions on Systems, Man and Cybernetics*, 10(5):242–248, 1980.

[2] R.C. Art. An approach to the two dimensional irregular cutting stock problem. *IBM Cambridge Scientific Centre*, Report 36-Y08, 1966.

[3] L. Babel, B. Chen, H. Kellerer, and V. Kotov. Algorithms for on-line bin-packing problems with cardinality constraints. *Discrete Apllied Mathematics*, 143:238–251, 2004.

[4] A. Ramesh Babu and N. Ramesh Babu. Effective nesting of rectangular parts in multiple rectangular sheets using genetic and heuristic algorithms. *International Journal of Production Research*, 37(7):1625–1643, 1999.

[5] A. Ramesh Babu and N. Ramesh Babu. A generic approach for nesting of 2-d parts in 2-d sheets using genetic and heuristic algorithms. *Computer-Aided Design*, 33:879–891, 2001.

[6] J.O. Berkey and P.Y. Wang. Two dimensional finite bin packing algorithms. *Journal of the Operations Research Society*, 38:423–429, 1987.

[7] E.G. Birgin and R.D. Lobato. Orthogonal packing of rectangles within isotropic convex regions. *Submitted*.

[8] A. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. part i: New lower bounds for the oriented case. *4OR*, 1:27–42, 2003.

[9] A. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. part ii: New lower and upper bounds. *4OR*, 1:135–147, 2003.

[10] E. K. Burke, R. S. R. Hellier, G. Kendall, and G. Whitwell. A new bottom-left-fill heuristisc algorithm for the two-dimensional irregular packing problem. *Operations Research*, 54(3):587–601, 2006.

[11] E.K. Burke. Complete and robust no-fit polygon generation for the irregular stock cutting problem. *European Journal of Operational Research*, 179:27–49, 2007.

[12] A. Caprara, A. Lodi, and M. Monaci. An approximation scheme for the two-stage, two-dimensional bin packing problem. In *Proc. 9th Conf. Integer Programming and Combinatorial Optinmization (IPCO 2002), Springer Lecture Notes in Computer Science*, volume 2337, pages 320–334. Springer, Berlin, 1999.

[13] A. Caprara and U. Pferschy. Worst-case analysis of the subset sum algorithm for bin packing. *Operations Research Letters*, 32:159–166, 2004.

[14] A. Caprara and U. Pferschy. Modified subset sum heuristics for bin packing. *Information Processing Letters*, 96:18–23, 2005.

[15] J. Carlier, F. Clautiaux, and A. Moukrin. New reduction procedures and lower bounds for the two-dimensional bin packing problem with fixed orientation. *Computers and Operations Research*, 33:2223–2250, 2007.

[16] F. Clautiaux, A. Jouglet, and J. El Hayek. A new lower bound for the non-oriented two-dimensional bin packing problem. *Operations Research Letters*, 35:365–373, 2007.

[17] J. Correa. Near-optimal solutions to two-dimensional bin packing with 90 degree rotations. *Electronic Notes in Discrete Mathematics*, 18:89–95, 2004.

[18] K. Daniels. *Containment algorithms for nonconvex polygons with apllications to layout.* Dissertation, Harvard University, Cambridge, MA, 1995.

[19] K. Daniels and V. J. Milenkovic. Rotational polygon containment and minimum enclosure using only robust 2d constructions. *In: Proc., 8th Canad. Conf. Comput. Geom.*, pages 33–38, 1996.

[20] K. Daniels and V. J. Milenkovic. Multiple translational containment: Approximate and exact algorithms. *In: Proc., 6th ACM-SIAM Sympos. Discrete Algorithms*, pages 205–214, 1997.

[21] K. Daniels and V. J. Milenkovic. Multiple translational containment, part i: An approximate algorithm. *Algorithmica*, 19:148–182, 1997.

[22] M. Dawande, J. Kalagnaman, and J. Sethuraman. Variabled sized bin packing with color constraints. *Electronic Notes in Discrete Mathematics*, 7:154–157, 2001.

[23] M. Dell'Amico, S. Martello, and D. Vigo. A lower bound for the non-oriented two dimensional bin packing problem. *Discrete Applied Mathematics*, 118:13–24, 2002.

[24] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[25] K.A. Downsland and W.B. Downsland. Packing problems. *European Journal of Operational Research*, 56(1):2–14, 1992.

[26] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[27] H. Dyckhoff, G. Scheithauer, and J. Terno. Cutting and packing. In F. Maffioli M. Dell' Amico and S. Martello, editors, *Annoted Bibliographies in Combinatorial Optimization*. John Wiley & Sons, Chichester, 1997.

[28] H. Dyckhoff and U.Finke. *Cutting and packing in production and distribution: A typology and bibliography*. Physica Verlag, Heidelberg, 1992.

[29] L. Epstein, A. Levin, and R. van Stee. Two-dimensional packing with conflicts. *Acta Informatica*, 45(3):155–175, 2008.

[30] S.P. Fekete and J. Schepers. On more-dimensional packing iii: Exact algorithms. *Technical paper ZPR97-290, Mathematisches Institut, Universitt zu Kln*, 1997.

[31] S.P. Fekete and J. Schepers. On more-dimensional packing ii: Bounds. *Technical Report ZPR97-289, Mathematisches Institut, Universitt zu Kln*, 2000.

[32] S.P. Fekete and J. Schepers. A general framework for bounds for higher-dimensional orthogonal packing problems. *Math., Methods Oper. Res.*, 60:311–329, 2004.

[33] L. Fu, D. Sun, and L.R. Rilett. Heuristic stortest path algorithms for transportation applications: State of the art. *Computers and Operations Research*, 33:3324–3343, 2006.

[34] M. Gendreau, G. Laporte, , and F. Semet. Heuristics and lower bounds for the bin packing problem with conflicts. *Computers and Operations Research*, 31.

[35] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, 1997.

[36] D.E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.

[37] A. M. Gomes and J. F. Oliveira. Solving irregular strip packing problems by hybridising simulated annealing and linear programming. *European Journal of Operational Research*, 171:811–829, 2006.

[38] A. M. Gomes and J.F. Oliveira. A 2-exchange heuristic for nesting problems. *European Journal of Operations Research*, 141(2):359–370, 2002.

[39] R .B. Grinde and T .M. Cavalier. A new algorithm for the two-polygon containment problem. *Computers & Operations Research*, 24(3):231–251, 1997.

[40] P. E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heurisitic determination of minimum cost paths. *IEEE Transactions, Systems Science and Cybernetics*, 4(2):100–107, 1968.

[41] J. El Hayek, A. Moukrim, and S. Negre. New resolution algorithm and pretreatments for the two-dimensional bin-packing problem. *Computers and Operations Research*, 35.

[42] E. Hopper and B.C.H. Turton. A review of the application of meta-heuristic algorithms to 2d strip packing problems. *Artificial Intelligence Review*, 16:257–300, 2001.

[43] H. Iima and T. Yakawa. A new design of genetic algorithm for bin packing. *Congress on Evolutionary Computation*, 2:1044–1049, 2003.

[44] D.S. Johnson. *Near optimal bin packing algorithms*. Dissertation, Massachussetts Institute of Technology, Cambridge, MA, 1973.

[45] l. Epstein and a. Levin. *Approximation and Online Algorithms*, volume 4368, chapter On Bin Packing with Conflicts, pages 160–173. Springer Heidelberg.

[46] D.S. Liu, K.C. Tan, S.Y. Huang, C.K. Goh, and W.K. Ho. On solving multiobjective bin packing problems using evolutionary particle swarm optimization. *European Journal of Operational Research*, 190:357–382, 2008.

[47] A. Lodi, S. Martello, and D. Vigo. Approximation algorithms for the oriented two-dimenional bin packing problem. *European Journal of Operational Research*, 112:158–166, 1999.

[48] A. Lodi, S. Martello, and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dinemsional bin packing problems. *INFORMS Journal on Computing*, 11:345–357, 1999.

[49] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *European Journal of Operational Research*, 123:379–396, 2002.

[50] A. Lodi, S. Martello, and D. Vigo. Two dimensional bin packing problems: A survey. *European Journal of Operational Research*, 141:241–252, 2002.

[51] A. Lodi, S. Martello, and D. Vigo. Tspack: A unified tabu search code for multi-dimensional bin packing problems. *Annals of Operations Research*, 131:203.213, 2004.

[52] F. P. Marques and M. N. Arenales. The constrained compartmentalised knapsack problem. *Computers & Operations Research*, 34:2109–2129, 2007.

[53] S. Martello and D. Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3), 1998.

[54] N. Metropolis, A.W. Rosenbluth, A. Teller, and E.J. Teller. Simulated annealing. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[55] V. J. Milenkovic. Translational polygon containment and minimal enclosure using linear programming based restriction. *In: Proc. 28th Ann. ACM Sympy. Theory Comput*, 6:109–118, 1996.

[56] V. J. Milenkovic. Multiple translational containment, part ii: Exact algorithms. *Algorithmica*, 19:183–218, 1997.

[57] V. J. Milenkovic. Rotational polygon containment and minimum enclosure using only robust 2d constructions. *Computational Geometry*, 13:3–19, 1999.

[58] V. J. Milenkovic and K. Daniels. Translational polygon containment and minimal enclosure using mathematical programming. *International Transactions in Operational Research*, 6:525–554, 1999.

[59] S. Polyakovsky and R. M'Hallah. An agent-based approach to the two-dimensional guillotine bin packing problem. *European Journal of Operational Research*, 192:767–781, 2009.

[60] H. Reinertsen and T.W.M. Vossen. The one-dimensional cutting stock problem with due dates. *European Journal of Operational Research*, 201:701–711, 2010.

[61] F. Rinaldi and A. Franz. A two-dimensional strip cutting problem with sequencing constraint. *European Journal of Operational Research*, 183:1371–1384, 2007.

[62] P. Rohlfshagen and J.A. Bullinaria. A genetic algorithm with exon shuffling crossover for hard bin packing problems. *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1365–1371, 2007.

[63] Y. Stoyan, G. Scheithauer, N. Gil, and T. Romanova. ϕ-functions for complex 2d-objects. *4OR: Quarterly Journal of the Belgian French and Italian Operations Research Societies*, 2:69–84, 2004.

[64] H. Terashima-Marín, C. J. Farías Zárate, P. Ross, and M. Valenzuela-Rendón. A ga-based method to produce generalized hyper-heuristics for the 2d-regular cutting stock problem. *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 591–598, 2006.

[65] H. Terashima-Marín, E. J. Flores-Álvarez, and P. Ross. Hyper-heuristics and classifier systems for solving 2d-regular cutting stock problems. *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 637–643, 2005.

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

[67] TH. Wu, JF. Chen, C. Low, and PT. Tang. Nesting of two-dimensional parts in multiple plates using hybrid algorithm. *International Journal of Production Research*, 41(16):3883–3900, 2003.

[68] G. Zhang. An 3-approximation algorithm for two dimensional bin packing. *Operations Research Letter*, 33:121–126, 2005.