Cutting Problem and Box Problem A relational Analysis of Packing Problems

Bachelor Thesis of

Marcel Posch

At the Department of Informatics and Mathematics
Chair of Theoretical Computer Science

Reviewer:	Prof. Dr. Ignaz Rutter
Advisors:	Prof. Dr. Ignaz Rutter
	Peter Stumpf, M.Sc.

Time Period: 16th May 2019 – 12th August 2019

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Passau, August 12, 2019

Abstract

We consider the problem of packing 2-dimensional rectangles in a container of fixed, integer width, such that no two rectangles overlap, while minimizing the height of the solution. The problem is NP-hard in the strong sense, as shown in this thesis. It finds applications in cutting and packing. Furthermore, we examine a relaxation of the former problem, where we are allowed to cut placed rectangles into unit squares and move those squares along the y-axis, as long as we keep rectangles from overlapping. We examine the relation of both problems, with the main focus being the equivalence of both problems. In particular we show the equivalence for the case, where all rectangles have a width of at most 2 and the case, where all rectangle have a height of 1. Furthermore, we propose a counterexample for the general case, where all rectangle heights and widths are at most 3.

Deutsche Zusammenfassung

Wir beschäftigen uns mit dem überschneidungsfreien Packen von Rechtecken in einem Behälter von fester ganzzahliger Breite und unbegrenzter Höhe, unter Minimierung der Höhe. Das Problem ist stark NP-schwer, wie in dieser Arbeit gezeigt wird, und findet Anwendungen in Packen und Zuschnitt. Neben diesem Packungs Problem betrachten wir eine Relaxierung des Problems, die es erlaubt, platzierte Rechtecke in Einheitsquadrate zu zerschneiden und diese entlang der y-Achse zu bewegen, solange dadurch keine Überschneidungen entstehen. Wir untersuchen den Zusammenhang beider Probleme, wobei der Hauptfokus auf der Äquivalenz der Probleme liegt. Insbesondere zeigen wir die Äquivalenz für den Fall, in dem die Breiten der Rechtecke kleiner gleich 2 sind und den Fall, in dem die Höhen der Rechtecke gleich 1 sind. Neben den Äquivalenzfällen geben wir auch ein Gegenbeispiel für den allgemeinen Fall, wobei wir nur Rechtecke mit Breite und Höhe kleiner gleich 3 verwenden.

Contents

1. Introduction

We consider the problem of orthogonally packing 2-dimensional rectangles of integer height and width, into a container of fixed, integer width, with the goal of minimizing the required height. We refer to this problem as *Box Problem.* The Box Problem is known to be NP-hard in the strong sense [MMV03]. Applications of the Box Problem include, packing of containers, vehicles and pallets. Further applications include the cutting of stock material such as textile, paper, leather and metal. It is also known as *Strip-Packing Problem* [MMV03], [Ste97]. However unlike some variations of Strip Packing, we do not allow any rotation. Other problems that are related to the Box Problem include Bin Packing (cf. [dC99], [MV98], [PS05]), Cutting Stock Problem [KR00] and the Knapsack Problem [MPT00]. Dyckhoff [Dyc90] provides an overview of similarities, differences and applications of the aforementioned problems, as well as other related problems.

Furthermore, we consider a relaxation of the Box Problem. The relaxation allows us to cut rectangles into unit squares, however these squares may only have the same x-coordinates, as they would, if they were placed as a whole rectangle (cf. Figure 1.1). We can think of this problem, as a variant of the Box Problem, where overhanging rectangle parts fall downwards (cf. Figure 1.2). This reduces our solutions from a 2-dimensional (x, y) mapping for rectangles to a 1-dimensional x mapping for each rectangle. We refer to this relaxation as *Cutting Problem.*

The Cutting Problem can be considered as a resource scheduling problem, where the goal is to minimize the required resources. For this we consider the container width as the available time, the rectangles as tasks, where the rectangle width is the duration of the task, and the rectangle height is the amount of required resources. In this form the problem finds applications in electricity consumption scheduling (see [LBBR15]).

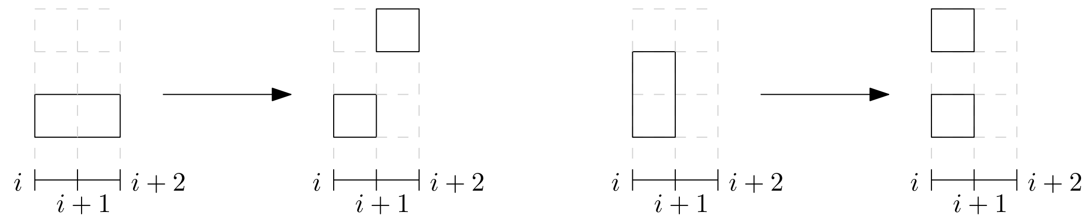

Figure 1.1: An example of the different ways we can cut rectangles. Of course we are neither limited to a single type of cut, nor limited to single spaces between the rectangles.

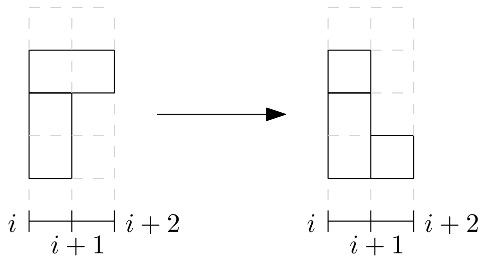

Figure 1.2: We can also imagine the cutting of rectangles as rectangle parts falling downwards.

We examine the relation of the Cutting Problem and the Box Problem, since the Cutting Problem is a relaxation of the Box Problem it will provide us with a lower bound for the Box Problem. Furthermore, the Cutting Problem is likely to be easier than the Box Problem, as solutions are only 1-dimensional as opposed to the 2-dimensional solution in the Box Problem. Therefore, the *equivalence*, in relation to the minimal height of placements, is highly desirable. As such we aim to either show such an equivalence or to find examples where the minimal height of both problems differs.

During our search for equivalences and counterexamples we examine bounding properties and geometric properties, as well as the computational complexity of both problems. Furthermore, we examine the equivalence of both problems for cases where height or width are limited. In particular, we show the equivalence for the case rectangle height at most 1 and the equivalence for the case rectangle width at most 2. For the latter case we consider two approaches, first a bounding approach and secondly a construction approach. The bounding approach seeks to show equivalence through lower and upper bounds, whereas the construction approach seeks to create an equivalent solution in the Box Problem, based on an optimal solution in the Cutting Problem. Additionally, we propose a counterexample, for the general case, where at least one rectangle dimension is 1 and heights and widths are at most 3. Lastly since we found the two problems to be not equivalent we aim to use the Cutting Problem, to approximate the Box Problem. For this, we propose an algorithm, that transforms a placement in the Cutting Problem into a feasible placement in the Box Problem, by moving overlapping rectangles upwards.

2. Preliminaries

In this chapter we introduce the problems that we examine. Furthermore, we introduce some important definitions which we will use throughout the thesis.

Definition 2.1. *We consider* rectangles $r_1, ..., r_n$ *to be abstract objects. In order to define rectangle height and width we use functions* $h : \{r_1, ...r_n\} \to \mathbb{N}$ *for the height of the rectangle* r *and* $w : \{r_1, ..., r_n\} \to \mathbb{N}$ *for the rectangle width of the rectangle* r*, where* $r \in \{r_1, .., r_n\}$*.*

For the sake of convenience and readability we use (a, b), to refer to a rectangle r, where $w(r) = a$ and $h(r) = b$.

We consider the bottom left corner of the container to be 0 in both x- and y-coordinates. Since rectangles have at least width 1, the rightmost coordinate is $B - 1$, in a container of width B. Similarly we consider a rectangle's position to be the position of its lower left corner.

Furthermore, we consider *boxes* to be axis-aligned rectangles which have been assigned x- and y-coordinates. As with rectangles we use the functions h and w for height and width respectively. In order to identify the position of a box we use the functions $x : \{r_1, .., r_n\} \to \mathbb{N}_0$ and $y : \{r_1, ..., r_n\} \to \mathbb{N}_0$ for x- and y-coordinates respectively.

The *Box Problem* is the problem of packing 2 dimensional rectangles of integer height and width into a container of fixed, integer width and virtually infinite height, such that no rectangle overlaps with another rectangle, and the height is minimal. However unlike some other variations of rectangle packing, we are not allowed to rotate rectangles. Furthermore we only allow integer dimensions for the rectangles and integer sized containers.

Definition 2.2. *Let* R *be a set of rectangles,* $B \in \mathbb{N}$*,* $h : R \to \mathbb{N}$ *and* $w : R \to \mathbb{N}$*, then* (B, R, w, h) *is a problem instance of Cutting and Box Problem.*

Definition 2.3. *Let* (B, R, w, h) *be a problem instance, then* (B, R, w, h, x) *is a* placement *in the Cutting Problem, where* $x : R \to \{0, ..., B - 1\}$*. and* (B, R, w, h, x, y) *is a* placement *in the Box Problem, where* $x : R \to \{0, ..., B - 1\}$ *and* $y : R \to \mathbb{N}_0$*.*

Definition 2.4. *A placement in the Box Problem* (B, R, w, h, x, y) *is* feasible*, if and only if, there exists a* placement *given by functions* (x, y) *such that* $x : R \to \{0, ..., B-1\}$, $y : R \to \mathbb{N}_0$ *and* $\forall i, j \in R$ *with* $i \neq j$*:*

$$\begin{aligned}&(x(i) + w(i) \leq x(j)\\ &\vee x(j) + w(j) \leq x(i)\\ &\vee y(i) + h(i) \leq y(j)\\ &\vee y(i) + h(i) \leq y(i))\\ &\wedge 0 \leq x(i) \leq x(i) + w(i) \leq B.\end{aligned}$$

The height of such a placement $H_{BOX}(B, R, w, h, x, y)$ *is* $\max\{y(a) + h(a) \mid a \in R\}$.

In other words a placement is feasible in the Box Problem, if and only if, the placement is free of overlaps and all rectangles stay within the bounds of the container.

Definition 2.5. *A placement in the Cutting Problem* (B, R, w, h, x) *is* feasible*, if and only if, there is a function* x*, such that for all* $i, \in R$*:*

$$0 \leq x(i) \leq x(i) + w(i) \leq B,$$

The height of such a placement $H_{CUT}(B, R, w, h, x)$ *is*

$$\max\{\sum_{k \in A} h(k) \mid i \in \{0, ..., B-1\}, A = \{j \in R \mid x(j) \leq i < x(j) + w(j)\}\}.$$

The Cutting Problem is essentially a variant of the Box Problem, where we are allowed to cut a box b into unit sized boxes $s_1, ..., s_{h(b)}$ with $x(b) = x(s_1), ..., x(s_{h(b)})$

The term *optimal placement* refers to a placement, which is minimal, meaning that no other placement with lower height exists.

Definition 2.6. *Let* (B, R, w, h, x) *be a placement in the Cutting Problem then*

(B, R, w, h, x) *is* optimal*, if and only if*

$$\begin{aligned}&\forall x' \in \{a \mid (B, R, w, h, a) \textit{ is feasible in the Cutting Problem}\}\\ &\qquad : H_{CUT}(B, R, w, h, x) \leq H_{CUT}(B, R, w, h, x').\end{aligned}$$

Let (B, R, w, h, x, y) *be a placement in the Box Problem then* (B, R, w, h, x, y) *is* optimal*, if and only if*

$$\begin{aligned}&\forall (x', y') \in \{(a, b) \mid (B, R, w, h, a, b) \textit{ is feasible in the Box Problem}\}\\ &\qquad : H_{BOX}(B, R, w, h, x, y) \leq H_{BOX}(B, R, w, h, x', y').\end{aligned}$$

The function $\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h)$ *maps a container width* B *and a set of rectangles* R*, with functions* h, w *for height and width respectively, to the height of the optimal solution in the Box Problem.*

The function $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h)$ *maps a container width* B *and a set of rectangles* R*, with functions* h, w *for height and width respectively, to the height of the optimal solution in the Cutting Problem.*

We consider two placements x for the Cutting Problem and (x, y) for the Box Problem, to be *equivalent* if and only if their heights are equal.

Definition 2.7. *Let* (B, R, w, h, x) *and* (B, R, w, h, x') *be placements in the Cutting Problem, then the placements are* equivalent*, if and only if*

$$H_{CUT}(B, R, w, h, x) = H_{CUT}(B, R, w, h, x').$$

Let (B, R, w, h, x, y) *and* (B, R, w, h, x', y') *be placements in the Box Problem, then the placements are* equivalent*, if and only if*

$$H_{BOX}(B, R, w, h, x, y) = H_{BOX}(B, R, w, h, x', y')$$

Let (B, R, w, h, y) *be a placement in the Box Problem and* $B(B, R, w, h, x')$ *a placement in the Cutting Problem, then the two placements are* equivalent*, if and only if,*

$$H_{BOX}(B, R, w, h, x, y) = H_{CUT}(B, R, w, h, x').$$

3. Properties of Cutting and Box Problem

In this chapter we examine properties of both Cutting and Box Problem. The chapter is further divided into section which group the properties into categories. The first section deals with properties, in relation to upper and lower bounds, while the second section deals with properties, in relation to increasing the number of rectangles or the available container width. The third section proposes that certain rectangles have even further properties. Lastly section 4 examines the computational complexity of the Cutting Problem.

3.1 Bounding Cutting Problem and Box Problem

While lower bounds hold true for any and all solutions in both Cutting and Box Problem, upper bounds do not necessarily hold true for all solutions. In particular an upper bound only has to hold true for the optimal solutions. This means we can use the height of any existing solution as an upper bound for the corresponding problem.

Proposition 3.1. *Let (B, R, w, h) be a problem instance, (B, R, w, h, x) a placement in the Cutting Problem and (B, R, w, h, x, y) a placement in the Box Problem, then x is a feasible placement in the Cutting Problem, if (x, y) is a feasible placement in the Box Problem and $H_{CUT}(B, R, w, h, x) \leq H_{BOX}(B, R, w, h, x, y)$.*

Proof. If we compare the definitions of Box Problem (Definition 2.4) and Cutting Problem (Definition 2.5), we can see that the set of requirements for feasibility in the Cutting Problem is a subset of those in the Box Problem. Furthermore, (B, R, w, h, x, y) is an orthogonal placement, therefore the height at every coordinate is at least the sum of rectangles, hence

$$\begin{aligned}
&H_{BOX}(B, R, w, h, x, y) \\
&\geq \max\{\sum_{k \in A} h(k) \mid i \in \{0, ..., B-1\}, A = \{r \in R \mid x(r) \leq i < x(r) + w(r)\}\} \\
&= H_{CUT}(B, R, w, h, x).
\end{aligned}$$

□

Corollary 3.2. *Let (B, R, w, h) be a problem instance, then*

$$\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) \leq \mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h)$$

Proof. Follows readily from Proposition 3.1. □

A basic lower bound for the Cutting Problem is given by the total surface area of all rectangles, in the given rectangle set.

Lemma 3.3. *Let (B, R, w, h) be a problem instance and $R = \{r_1, ..., r_k\}$, then:*

$$\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) \geq \lceil \sum_{i=1}^{k} \frac{w(r_i) \cdot h(r_i)}{B} \rceil$$

Proof. Let (B, R, w, h) be a problem instance and $R = \{r_1, ..., r_k\}$, then the combined surface area of all rectangles is:

$$A := \sum_{i=1}^{k} w(r_i) \cdot h(r_i).$$

Now if we distribute this surface area evenly over the width B, we end up with height A/B. Since we are dealing with integer rectangle dimensions we have to round towards the larger integer, as such we end up with height $\lceil A/B \rceil$, which is the height of the smallest bounding box in which we can pack A unit size squares. This means there will be at least one position in the Cutting Problem with height $H \geq \lceil A/B \rceil$. Thus, $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) \geq H \geq \lceil A/B \rceil = \lceil \sum_{i=1}^{k} \frac{w(r_i) \cdot h(r_i)}{B} \rceil$. □

Another basic lower bound is given by the height of the tallest rectangle.

Proposition 3.4. *Let (B, R, w, h) be a problem instance and $r \in \{a \in R \mid h(a) = \max\{h(b) \mid b \in R\}\}$, then*

$$\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) \geq h(r)$$

Proof. Consider a feasible solution in the Cutting Problem, then there is at least one x-coordinate where r is active, namely $x(r)$. Therefore, there is at least one x-coordinate which has at least height $h(r)$ and thus $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) \geq h(r)$. □

Now we can easily combine the two lower bounds, by using the maximum of both bounds as a new lower bound.

Corollary 3.5. *Let (B, R, w, h) a problem instance, then*

$$\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) \geq \max\left(\left\lceil \sum_{i=1}^{k} \frac{w(r_i) \cdot h(r_i)}{B} \right\rceil, H\right),$$

where $H = \max\{h(k) \mid k \in R\}$.

Proof. Follows readily, by combining the bounds from Lemma 3.3 and Proposition 3.4. □

We can create an upper bound for the Box Problem, by constructing a feasible placement. In particular a placement which is always possible, if the instance is not infeasible, is stacking rectangles on top of each other.

Corollary 3.6. *Let (B, R, w, h) be a problem instance and $R = \{r_1, ..., r_n\}$, then*

$$\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) \leq \sum_{k=1}^{n} h(r_k)$$

Proof. We can create such a placement, by placing rectangles on top of each other at x-coordinate 0, starting from r_1. This means the y-coordinate for the rectangle r_i is $y(r_i) = \sum_{k=1}^{i-1} h(r_k)$, where $i \in \{1, ..., n\}$. Therefore, no rectangles overlap and thus we have a feasible placement in the Box Problem, with height $\sum_{k=1}^{n} h(r_k)$. Hence, $\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) \leq \sum_{k=1}^{n} h(r_k)$. □

We can create another upper bound, by expanding every rectangle, to the largest height and width, found in the rectangle set. This leaves us with a single rectangle type, which makes creating bounds much easier.

Lemma 3.7. *Let (B, R, w, h) a problem instance, then*

$$\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) \leq h_{max} \cdot \left\lceil \frac{w_{max} \cdot |R|}{B - (B \bmod w_{max})} \right\rceil,$$

where $h_{max} = \max\{h(a) \mid a \in R\}$ and $w_{max} = \max\{w(a) \mid a \in R\}$.

Proof. Let $w'(r) = w_{max}$ and height $h'(r) = h_{max}$ for all $r \in R$, then

$$\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) \leq \mathrm{OPT}_{\mathrm{BOX}}(B, R, w', h'),$$

as all rectangles in R have at most width w_{max} and at most height h_{max}. Furthermore, $\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) \leq \mathrm{OPT}_{\mathrm{BOX}}(B - (B \bmod w_{max}), R, w, h)$ and thus

$$\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) \leq \mathrm{OPT}_{\mathrm{BOX}}(B - (B \bmod w_{max}), R, w, h).$$

Now if we have (B, R, w', h') as our problem instance, then we can construct a solution in the Box Problem, by stacking the rectangles from left to right from bottom to top. This means we end up with height $h_{max} \cdot \lceil \frac{|R|}{B'/w_{max}} \rceil = h_{max} \cdot \lceil \frac{w_{max} \cdot |R|}{B'} \rceil = h_{max} \cdot \lceil \frac{w_{max} \cdot |R|}{B - (B \bmod w_{max})} \rceil$. Now, since this is a feasible solution it follows that $\mathrm{OPT}_{\mathrm{BOX}}(B', R, w', h') \leq h_{max} \cdot \lceil \frac{w_{max} \cdot |R|}{B - (B \bmod w_{max})} \rceil$ and therefore

$$\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) \leq \mathrm{OPT}_{\mathrm{BOX}}(B', R, w', h') \leq h_{max} \cdot \lceil \frac{w_{max} \cdot |R|}{B - (B \bmod w_{max})} \rceil.$$

□

We can also construct a packing for the Box Problem, by separating the individual rectangle types and combining the placements, for the individual rectangle types, through stacking.

Lemma 3.8. *Let (B, R, w, h) a problem instance, then $\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) \leq \sum_{(a,b) \in A} b \cdot \lceil \frac{a \cdot c_{(a,b)}}{B - (B \bmod a)} \rceil$, where $A = \{(w(k), h(k)) \mid k \in R\}$ and $c_{(a,b)} = |\{k \in R \mid w(k) = a, h(k) = b\}|$.*

Proof. Consider the subset $R_{(a,b)} = \{k \in R \mid w(k) = a, h(k) = b\}$ and

$H_{(a,b)} = b \lceil \frac{a \cdot |R_{(a,b)}|}{B - (B \bmod a)} \rceil$, where $(a, b) \in A$, then from Lemma 3.7 it follows, that

$$\mathrm{OPT}_{\mathrm{BOX}}(B, R_{(a,b)}, h, w) \leq b \lceil \frac{a \cdot |R_{(a,b)}|}{B - (B \bmod a)} \rceil = H_{(a,b)}.$$

Now we can combine these placements by stacking them on top of each other. Since the rectangles sets $R_{(a,b)}$ are disjoint and cover R, we end up with a placement for the complete set R, which has a height of $\sum_{(a,b) \in A} H_{(a,b)}$ and thus $\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) \leq \sum_{(a,b) \in A} H_{(a,b)} = \sum_{(a,b) \in A} b \lceil \frac{a \cdot |R_{(a,b)}|}{B - (B \bmod a)} \rceil$. □

3.2 Further properties of Cutting and Box Problem

Aside from the bounding properties there are also properties in relation to increasing the available width, adding rectangles and rotation.

Proposition 3.9. *Let $B \in \mathbb{N}$ and $R = \{r_1, ..., r_k\}$ be a set of rectangles, then*

$$\mathrm{OPT}_{\mathrm{CUT}}(B+1, R, w, h) \leq \mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h),$$
$$\mathrm{OPT}_{\mathrm{BOX}}(B+1, R, w, h) \leq \mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h).$$

Proof. Having a larger container width gives us more space to work with, which means the optimal solution has potentially less height. An increase in height is impossible, as we can still construct the same solutions without using the extra width. □

Proposition 3.10. *Let (B, R, w, h) a problem instance and $(B, R \cup \{r\}, w', h')$ another problem instance, which is identical to the first instance in all rectangles but the rectangle r, $\forall a \in R : w(a) = w'(a), h(a) = h'(a)$. Then $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) \leq \mathrm{OPT}_{\mathrm{CUT}}(B, R \cup \{r\}, w', h')$ and $\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) \leq \mathrm{OPT}_{\mathrm{BOX}}(B, R \cup \{r\}, w', h')$.*

Proof. Placing an additional rectangle requires us to utilize more of the available space. Therefore the height of the solution is at least as tall as the height of the solution without r. □

Another property of the Box Problem is the ability to create solutions for other instances by rotating the solution.

Proposition 3.11. *Let (B, R, w, h) a problem instance and $H := \mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h)$, then $\mathrm{OPT}_{\mathrm{BOX}}(H, R, w', h') \leq B$, where $h' = w$, $w' = h$.*

Proof. Consider a bounding box r of width B and height H, then the rectangles in R can be packed within the bounding box r. Now, if we rotate the bounding box r along with the packing by 90 degrees, we obtain a packing for (H, R, w', h'), which can be packed into the rotated bounding box r'. Therefore, $\mathrm{OPT}_{\mathrm{BOX}}(H, R, w', h') \leq B$, as the height of r' is B. □

3.3 Properties of particular rectangles

There are certain rectangles, which have even further properties. In particular rectangles that are as wide, as the container, have no impact on the rest of the placement, apart from increasing the height of the solution.

Proposition 3.12. *Let (B, R, w, h) be a problem instance, then*

$$\mathrm{OPT}_{\mathrm{CUT}}(B, R \cup \{r\}, w, h) = \mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) + h',$$
$$\mathrm{OPT}_{\mathrm{BOX}}(B, R \cup \{r\}, w, h) = \mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) + h'.$$

Proof. Since r has a width of B there is only one x-coordinate, where we can place r. As such in order to obtain a bound it suffices to examine the height of the optimal solution without r and increase the resulting height by the height of r, which is h'. As such $\mathrm{OPT}_{\mathrm{CUT}}(B, R \cup \{r\}, w, h) = \mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) + h$ and $\mathrm{OPT}_{\mathrm{BOX}}(B, R \cup \{r\}, w, h) = \mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) + h$. □

Similarly, if we have a rectangle, that is as tall as our placement's height, then the rest of the placement fits into a smaller container.

Proposition 3.13. *Let (B, R, w, h) a problem instance and $(B, R \cup \{r\}, w', h')$ another problem instance such that $\forall a \in R : h(a) = h'(a), w(a) = w'(a)$. Then*

$$\mathrm{OPT}_{\mathrm{CUT}}(B, R \cup \{r\}, w', h') = h'(r) \Leftrightarrow \mathrm{OPT}_{\mathrm{CUT}}(B - w'(r), R, h, w) \le h(r),$$
$$\mathrm{OPT}_{\mathrm{BOX}}(B, R \cup \{r\}, w', h') = h'(r) \Leftrightarrow \mathrm{OPT}_{\mathrm{BOX}}(B - w'(r), R, w, h) \le h'(r).$$

Proof. First let $\mathrm{OPT}_{\mathrm{CUT}}(B, R \cup \{r\}, w', h') = h'(r)$, then the height of the optimal solution is the height of the rectangle r. Therefore there cannot be any other rectangle at an x-coordinate occupied by r. As such the remaining width for the other rectangles is $B - w'(r)$, which means we can pack the other rectangles into a container of lower width, without exceeding the height $h'(r)$, thus $\mathrm{OPT}_{\mathrm{CUT}}(B - w'(r), R, w, h) \le h'(r)$. Now let $\mathrm{OPT}_{\mathrm{CUT}}(B - w'(r), R, w, h) \le h'(r)$, then we can create a solution for the instance (B, R, w, h) by dividing the container into two smaller containers, one of width $w'(r)$ and another one of width $B - w'(r)$. We place r in the container of width $w'(r)$ and the remaining rectangles in the container of width $B - w'(r)$. The resulting height will be the maximum of both containers, as such the resulting height is $\max\{\mathrm{OPT}_{\mathrm{CUT}}(B - w'(r), R, w, h), \mathrm{OPT}_{\mathrm{CUT}}(w'(r), \{r\}, w', h')\} = \max\{\mathrm{OPT}_{\mathrm{CUT}}(B - w'(r), R, w, h), h'(r)\}$. Since $\mathrm{OPT}_{\mathrm{CUT}}(B - w'(r), R, w, h) \le h'(r)$, the resulting height is $h(r')$. Therefore $\mathrm{OPT}_{\mathrm{CUT}}(B - w'(r), R, w, h) \le h'(r)$ implies $\mathrm{OPT}_{\mathrm{CUT}}(B, R \cup \{r\}, w', h') = h'(r)$.

We can easily apply the same arguments for the Box Problem, by replacing $\mathrm{OPT}_{\mathrm{CUT}}$ with $\mathrm{OPT}_{\mathrm{BOX}}$. Thus $\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) = h('r) \Leftrightarrow \mathrm{OPT}_{\mathrm{BOX}}(B - W, R) \le h'(r)$. □

3.4 Computational Complexity of the Cutting Problem

In this section we examine the computational complexity of the Cutting Problem. We start by examining whether the Cutting Problem can be solved in polynomial time, by a non-deterministic Turing machine.

Lemma 3.14. *The Cutting Problem is in NP.*

Proof. Let $(B, \{r_1, ..., r_n\}, w, h, x)$ be a feasible placement in the Cutting Problem. Now consider sets $S_1, ..., S_B$, where S_i contains all rectangles that are *active* at the x-coordinate i (a rectangle r is active at $x(r)$ up to but not including $x(r) + w(r)$), where $i \in \{1, ..., B\}$ we will use these sets as certificate.

First we check whether the certificate contains all rectangles, this can be done by iterating over all rectangles in all sets. Since sets in a valid certificate will contain at most n rectangles, we can test the rectangles in a single set in $\mathcal{O}(n^2)$. Similarly we have at most B sets in a valid certificate, which means we can test all rectangles in the certificate in $\mathcal{O}(n^2 \cdot B)$,

Now that we know whether the union of sets is equal to the rectangle set, we check whether rectangles cover their respective width. This can be done by iterating over the sets in ascending order and tracking how many times each rectangle has been iterated over. We will use *ongoing* for rectangles, which have been iterated over less times than they are wide and *finished* for rectangles, that have been iterated over as many times as they are wide. This means if an ongoing rectangle is not contained in the next set, then the certificate is invalid. Similarly if a finished rectangle occurs in a following set, then the certificate is also invalid. If a rectangle is not finished after the last set, then the certificate is also invalid. This iteration can similarly be done $\mathcal{O}(n^2 \cdot B)$. Now we check whether the sum of rectangle heights in each set is at most H:

$$\forall S \in \{S_1, ..., S_B\} : \sum_{a \in S} h(a) \le H$$

We can check whether a specific set $S \in \{S_1, ..., S_B\}$ has at most height H in $\mathcal{O}(\log N \cdot n)$, where $N = \max\{h(r_1), ..., h(r_n)\}$, by summing the heights of the rectangles and comparing the sum to H. As such we can calculate whether it holds true for $S_1, ..., S_B$ in $\mathcal{O}(\log N \cdot n \cdot B)$.

Since we only require polynomial time for each step, we also only require polynomial time for the whole process. This means we can verify a solution to the Cutting Problem in polynomial time. Therefore the Cutting Problem is in NP. □

Now that we know that the Cutting Problem is contained within the complexity class NP, we examine its relation to other problems which are NP-hard.

Lemma 3.15. *The Cutting Problem and the Box Problem are NP-Hard in the strong sense.*

Proof. We show that the Cutting Problem is NP-Hard by reducing the 3-Partition Problem, to the Cutting Problem.

Consider an instance of the 3-Partition Problem, here we have a set of non negative integers $S = \{a_1, ..., a_n\}$. What we want to find are three disjoint subsets S_1, S_2, S_3, such that $\sum_{a \in S_1} a = \sum_{a \in S_2} a = \sum_{a \in S_3} a$. Now consider rectangles $r_1, ..., r_n$ where the height of the rectangle $h(r_i) = a_i$ and the width of a rectangle $w(r_i) = 1$, for all $i \in \{1, ..., n\}$. Now if $\text{OPT}_{\text{CUT}}(3, \{r_1, ..., r_n\}, w, h) = \sum_{a \in S} a/3$, then the sum of all rectangle heights at each x-coordinate is exactly $\text{OPT}_{\text{CUT}}(3, \{r_1, ..., r_n\}, w, h)$ which is one third of the surface area. As such we have the existence of such disjoint subsets namely the sets of rectangle heights at each coordinate. Similarly if $\text{OPT}_{\text{CUT}}(3, \{r_1, ..., r_n\}, w, h) \neq \sum_{a \in S} a/3$, then such subsets cannot exist. As the existence of such a partition gives us a positioning for the Cutting Problem, in particular we can use any ordering of the sets as x-coordinates for the rectangle placement. Now since we can construct the rectangles in polynomial time, it follows that the Cutting Problem is NP-Hard in the strong sense, as the 3-Partition Problem is strongly NP-Complete.

Since Cutting Problem and Box Problem are equivalent for rectangles of width 1 (see Theorem 4.1), the same arguments can be applied to the Box Problem. Hence the Box Problem is also NP-hard in the strong sense. □

Now we know that the Cutting Problem is contained in NP and that it is NP-hard in the strong sense. Applying the definition of NP-completeness gives use the NP-completeness of the Cutting Problem.

Theorem 3.16. *The Cutting Problem is strongly NP-complete.*

Proof. Follows readily from Lemma 3.14 and Lemma 3.15. □

4. On the equivalence of Cutting Problem and Box Problem

In this chapter we examine the equivalence of Cutting and Box Problem. We show the equivalence for several special cases and propose a counterexample, for the general case.

4.1 Rectangles of width 1

The first case we consider is the case where all rectangles have width 1.

Theorem 4.1. *Let (B, R, w, h) be a problem instance, where $\forall r \in R : w(r) = 1$, then* $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) = \mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h)$.

Proof. The solution to the Cutting Problem provides us with x-coordinates for each rectangle. We can construct a solution to the Box Problem by stacking rectangles, that map to the same x-coordinate. Since our rectangles only have width 1, the order does not matter, as the height will still be the same. This stacking gives us a feasible solution in the Box Problem that has the same height as our Cutting Problem solution, since the height at every x-coordinate is the sum of rectangle heights. Therefore, $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) = \mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h)$. $\square$

4.2 Rectangles of height 1

Another case we consider is the case where all rectangles have height 1, unlike the previous case, it is not as apparent from the intuition introduced in the introduction.

Theorem 4.2. *Let (B, R, w, h) be a problem instance, where $\forall r \in R : h(r) = 1$, then* $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) = \mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h)$.

Proof. Consider an optimal solution S in the Cutting Problem. Then this solution provides us with x-coordinates for every rectangle in R.

Now consider the partial order set $(R, \leq)$, where $r_1 \leq r_2$ if $x(r_1) + w(r_1) \leq x(r_2)$. Then, two elements r_1 and r_2 are not comparable if neither $r_1 \leq r_2$ nor $r_2 \leq r_1$, which is the case if $x(r_1) + w(r_1) > x(r_2)$ and $x(r_2) + w(c_2) > x(r_1)$. This means two elements are not comparable, if and only if they overlap on the x-axis. Furthermore, there exists an *antichain*, which is a subset of R, where no two elements are comparable. In particular there exists a largest antichain A.

- Case 1: $A = \emptyset$ then all elements in R are comparable, which means there are no overlaps in the Cutting Solution. As such we can simply use the Cutting Solution as a Box Solution, by using 0 as y-coordinate for all rectangles.
- Case 2: $A \neq \emptyset$ then since A is an antichain, we can apply Dilworth's theorem (see [Die17, p. 52-53]), which tells us that the minimal number of chains to cover R is $H := |A|$. This gives us the existence of linear ordered, disjoint subsets $X_0, ..., X_{H-1}$, which cover R.

This provides us with y-coordinates for the Box Problem, where rectangles in the subset X_i are placed at height i, for $i \in \{0, ..., H-1\}$. Such a placement is free of overlaps, as rectangles in each set are comparable and rectangles have height 1. Furthermore, the placement has height H, due to the highest y-coordinate being $H-1$ and rectangles having height 1. Now since we know that elements are not comparable, if and only if they overlap on the x-axis, we also know that S has height $|A| = H$, as this is the maximum number of overlaps at any x-coordinate in the solution S. Furthermore $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) = H$, since S is an optimal solution. Thus, $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) = H = \mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h)$.

□

4.3 Equivalence for rectangles of surface area 2

In this section we examine the equivalence of Cutting and Box Problem for rectangles of size 2, which means we only have (1,2) rectangles and (2,1) rectangles. The chapter is further subdivided into two approaches, first the bounding approach and secondly the construction approach. Unlike the bounding approach the construction approach is not limited by the height of rectangles.

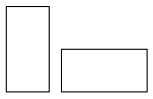

Figure 4.1: A (1,2) rectangle and a (2,1) rectangle.

4.3.1 Bounding approach

We can show the equivalence of Cutting and Box Problem, by showing equivalence of lower and upper bounds. In this section we use R to refer to which consists solely of (2,1) and (1,2) rectangles. The number n refers to the number of (1,2) rectangles, the number l refers to the number (2,1) rectangles and k refers to the total number of rectangles, $k = n + l$. All these definition are confined to this section and are valid for all proofs in the section, unless stated otherwise.

Corollary 4.3. *Let (B, R, w, h) be problem instance, then* $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) \leq \lceil \frac{2k}{B} \rceil$.

Proof. Follows directly from Lemma 3.3. □

We start out with a very simple case, the case where we limit ourselves to only (1,2) rectangles. While this case is already covered by the more general case of width 1 (see Theorem 4.1), it provides us with a lower bound which will be useful later on.

Lemma 4.4. *Let (B, R, w, h) be a problem instance, where R consists solely of (1,2) rectangles, then* $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) \geq 2\lceil k/B \rceil$.

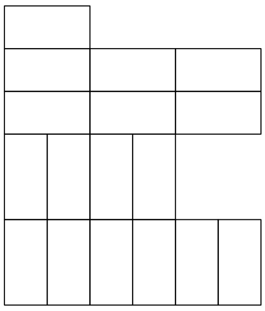

Figure 4.2: An example of a packing, where the different rectangle types are packed separately.

Proof. Since the height in the Cutting Problem is the sum of rectangles, which are active at a specific coordinate and our rectangles have only width 1, we can take a look at the problem instance of the same rectangle number, where we only have unit squares and scale the resulting solution, thus multiplying the resulting height by 2. The Lemma 3.3 gives us the minimal height for unit size squares, which is $\lceil k/B \rceil$. Now through scaling we obtain the minimal height for problem instances with only (1,2) rectangles which is $2\lceil k/B \rceil$. Hence, $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) \geq 2\lceil k/B \rceil$ □

Now we limit ourselves to an even container width. This allows us to create an upper bound for the Box Problem by introducing a way of packing, which consists of packing (1,2) rectangles and (2,1) rectangles separately. In the following B refers to an even container width, unless stated otherwise.

Lemma 4.5. *Let (B, R, w, h) be problem instance, then* $\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) \leq \lceil \frac{2l}{B} \rceil + 2\lceil \frac{n}{B} \rceil$.

Proof. We can split this problem into two smaller problems by examining the height of the solutions for each rectangle type. If we combine the heights of the solutions we get the height of a solution containing both types. First let us take a look at the (1,2) rectangles. If we place these rectangles from left to right, then we end up with $\lfloor n/B \rfloor$ fully filled rows and possibly one partially filled row resulting in $\lceil n/B \rceil$ rows, of (1,2) rectangles. Since these rectangles have height 2, we end up with a height of $2\lceil n/B \rceil$ Now let us take a look at the (2,1) rectangles. These have width 2, meaning that we can only fit $B/2$ of these rectangles in a single row. Thus, if we stack these from left to right we end up with $l/(B/2) = 2l/B$ rows, which means we end up with a height of $\lceil 2l/B \rceil$. We can create a solution that includes both types by stacking these partial solutions on top of each other, resulting in a height of $\lceil 2l/B \rceil + 2\lceil n/B \rceil$. Thus, $\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) \leq \lceil 2l/B \rceil + 2\lceil n/B \rceil$. (Figure 4.2 provides an example of such a separated packing.) □

Now that we have a basic upper bound for the Box Problem we can easily show the equivalence in the case where the container width divides the number of (1,2) rectangles.

Lemma 4.6. *Let (B, R, w, h) be a problem instance, where $n \bmod B = 0$. Then*

$$\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) = \mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h).$$

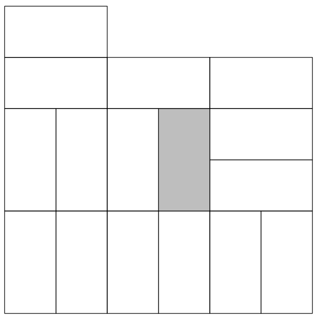

Figure 4.3: A new and improved packing strategy. The gray area is free space in the case of an uneven number of (1,2) rectangles.

Proof. From Lemma 4.5 we know that there exists a solution in the Box Problem, such that the height is $\lceil \frac{2l}{B} \rceil + 2\lceil \frac{n}{B} \rceil$. Since $n \bmod B = 0$ it follows that:

$$\begin{aligned} \lceil \frac{2l}{B} \rceil + 2\lceil \frac{n}{B} \rceil &= \lceil \frac{2l}{B} \rceil + 2\frac{n}{B} \\ &= \lceil \frac{2l}{B} + \frac{2n}{B} \rceil \\ &= \lceil \frac{2k}{B} \rceil \end{aligned}$$

Corollary 4.3 tells us that the optimal Cutting Problem solution has at least height $\lceil 2k/B \rceil$ and since every Box Problem solution is a valid solution for the Cutting Problem (see Proposition 3.1), it follows that $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) = \mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h)$. □

Now we introduce a new packing strategy (see Figure 4.3), which is inspired by the separate packing of Figure 4.3, but utilizes the space left on the last row of (1,2) rectangles.

Using this new packing strategy we obtain an upper bound for the case where the (2,1) rectangles fit into the space.

Lemma 4.7. *Let (B, R, w, h) be a problem instance, where $(n \bmod B) \neq 0$ and $l \leq a := B - (n \bmod B) - (n \bmod 2)$. Then* $\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) \leq 2\lceil \frac{n}{B} \rceil$

Proof. If we stack the (1,2) rectangles from left to right we end up with 2 $\lfloor n/B \rfloor$ completely filled rows of (1,2) rectangles and two partially filled rows, as (1,2) rectangles have height 2. These partially filled rows are filled to width $n \bmod B$, as such there is a free space of $B - n \bmod B$ width and height 2. Since this free space might not be of even width we are going to consider the free space to be of width $a := B - (n \bmod B) - (n \bmod 2)$. This means that we can fit $a/2$ (2,1) rectangles in width in this free space, however since the space has height 2 we can actually fit two such rows in there, as such we can fit $2 \cdot (a/2) = a$ (2,1) rectangles in the free space. Since $l \leq a$ we can fit all (2,1) rectangles, without increasing the height any further. This means our height is only determined by the (1,2) rectangles and thus ends up being $2\lceil n/B \rceil$. Hence, $\mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h) \leq 2\lceil n/B \rceil$. □

Now we can use the simple case we considered at the beginning of this chapter to show equivalence for another special case, namely the case where we limit ourselves to a small number of (2,1) rectangles.

Lemma 4.8. *Let (B, R, w, h) be a problem instance, where $l \leq B - (n \bmod B) - (n \bmod 2)$ and $n \bmod B \neq 0$. Then* $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) = \mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h)$.

Proof. As Lemma 4.7 tells us $\mathrm{OPT}_{\mathrm{BOX}}(B,R,w,h) \leq 2\lceil n/B \rceil$. Furthermore, Lemma 4.4 tells us that $\mathrm{OPT}_{\mathrm{CUT}}(B,R',w,h) \geq 2\lceil n/B \rceil$, where R' is the subset of R without the (2,1) rectangles, $R' = \{r \in R \mid h(r) = 2\}$. Since the height can only increase with more rectangles, it follows that $\mathrm{OPT}_{\mathrm{CUT}}(B,R,w,h) = \mathrm{OPT}_{\mathrm{BOX}}(B,R,w,h)$. □

Using the new packing strategy we can introduce a new upper bound. Since we have already shown the case where we only have a few (2,1) rectangles, we now consider the case where we have a minimum of such rectangles.

Lemma 4.9. *Let (B,R,w,h) be a problem instance, where $l > a := B - (n \bmod B) - (n \bmod 2)$ and $n \bmod B \neq 0$. Then $\mathrm{OPT}_{\mathrm{BOX}}(B,R,w,h) \leq \lceil \frac{2l-2a}{B} \rceil + 2\lceil \frac{n}{B} \rceil$.*

Proof. If we stack the (1,2) rectangles from left to right we end up with $\lfloor n/B \rfloor$ completely filled rows of (1,2) rectangles and one partially filled row. This partially filled row is filled to a width of $n \bmod B$, as such there is a free space of $B - (n \bmod B)$ width and height 2. Since this free space might not be of even width we are going to consider the free space to be of width $B - (n \bmod B) - (n \bmod 2) =: a$, reducing the space to even width. This means that we can fit $a/2$ of the (2,1) rectangles, in width, in this free space, however since the space has height 2 we can actually fit two such rows in there, as such we can fit a total of $2 \cdot (a/2) = a$ of the (2,1) rectangles in the free space. So we are left with $l - a$ (2,1) rectangles, which we stack from left to right, creating $\lceil \frac{2l-2a}{B} \rceil$ rows of (2,1) rectangles. Thus, the height ends up being $\lceil \frac{2l-2a}{B} \rceil + 2\lceil \frac{n}{B} \rceil$. Hence, $\mathrm{OPT}_{\mathrm{BOX}}(B,R,w,h) \leq \lceil \frac{2l-2a}{B} \rceil + 2\lceil \frac{n}{B} \rceil$. □

Both lower bound and upper bound increase by 2 for every B of the (1,2) rectangles and by 1 for every $B/2$ of the (2,1) rectangles, as long as we have a certain minimum of (2,1) rectangles. This means we can use induction to show the equivalence of both bounds.

Lemma 4.10. *For every $n, l, B \in \mathbb{N}$, with B, n even, $n \bmod B \neq 0$, $l > a := B - (n \bmod B) - (n \bmod 2)$ and $k = n + l$, we have $\lceil \frac{2l-2a}{B} \rceil + 2\lceil \frac{n}{B} \rceil = \lceil \frac{2k}{B} \rceil$.*

Proof. In the following we use induction, since we have two variables which are relevant to the induction, we use two inductive steps, one for n and one for l, to prove our claim. First we begin with the base case, which is: $0 < n < B, a < l < B$, then

$$\begin{aligned}
\lceil \frac{2l-2a}{B} \rceil + 2\lceil \frac{n}{B} \rceil &= \lceil \frac{2l - 2(B - (n \bmod B) - (n \bmod 2))}{B} \rceil + 2\lceil \frac{n}{B} \rceil \\
&= \lceil \frac{2l - 2(B - n - n \bmod 2)}{B} \rceil + 2(1) \\
&= \lceil \frac{2l - 2B + 2n}{B} + 2 \rceil \\
&= \lceil \frac{2l + 2n}{B} \rceil \\
&= \lceil \frac{2k}{B} \rceil
\end{aligned}$$

Now let us continue with the first inductive step. Assume for an arbitrary n', that $\lceil \frac{2l-2a}{B} \rceil + 2\lceil \frac{n'}{B} \rceil = \lceil \frac{2k}{B} \rceil$, is true.

We will show that $\lceil\frac{2l-2b}{B}\rceil + 2\lceil\frac{n'+B}{B}\rceil = \lceil\frac{2k+2B}{B}\rceil$, where $b := B - ((n'+B) \bmod B) - ((n'+B) \bmod 2)$.:

$$\begin{aligned}
&\lceil\frac{2l-2b}{B}\rceil + 2\lceil\frac{n'+B}{B}\rceil \\
&\quad = \lceil\frac{2l - 2\left(B - ((n'+B) \bmod B) - ((n'+B) \bmod 2)\right)}{B}\rceil + 2\lceil\frac{n'+B}{B}\rceil \\
&\quad = \lceil\frac{2l - 2(B - n' \bmod B - (n'+B) \bmod 2)}{B}\rceil + 2\lceil\frac{n'+B}{B}\rceil \\
&\quad = \lceil\frac{2l - 2(B - n' \bmod B - n' \bmod 2)}{B}\rceil + 2\lceil\frac{n'+B}{B}\rceil \\
&\quad = \lceil\frac{2l-2a}{B}\rceil + 2\lceil\frac{n'+B}{B}\rceil \\
&\quad = \lceil\frac{2l-2a}{B}\rceil + 2\lceil\frac{n'}{B}+1\rceil \\
&\quad = \lceil\frac{2l-2a}{B}\rceil + 2\lceil\frac{n'}{B}\rceil + 2 \\
&\quad = \lceil\frac{2k}{B}\rceil + 2 \\
&\quad = \lceil\frac{2k}{B}+2\rceil \\
&\quad = \lceil\frac{2k+2B}{B}\rceil
\end{aligned}$$

We continue with the second inductive step. Here we assume for an arbitrary l', that $\lceil\frac{2l'-2a}{B}\rceil + 2\lceil\frac{n}{B}\rceil = \lceil\frac{2k}{B}\rceil$, is true.

We will show that $\lceil\frac{2(l'+B)-2a}{B}\rceil + 2\lceil\frac{n}{B}\rceil = \lceil\frac{2k+2B}{B}\rceil$.:

$$\begin{aligned}
\lceil\frac{2(l'+B)-2a}{B}\rceil + 2\lceil\frac{n}{B}\rceil &= \lceil\frac{2l'+2B-2a}{B}\rceil + 2\lceil\frac{n}{B}\rceil \\
&= \lceil\frac{2l'-2a}{B} + 2\lceil\frac{n}{B}\rceil \\
&= \lceil\frac{2l-2a}{B}\rceil + 2\lceil\frac{n}{B}\rceil + 2 \\
&= \lceil\frac{2k}{B}\rceil + 2 \\
&= \lceil\frac{2k}{B}+2\rceil \\
&= \lceil\frac{2k+2B}{B}\rceil
\end{aligned}$$

Thus, it follows that $\lceil\frac{2l-2a}{B}\rceil + 2\lceil\frac{n}{B}\rceil = \lceil\frac{2k}{B}\rceil$ is true for all $n, l, B \in \mathbb{N}$, with n, B even, $n \bmod B \neq 0$ and $l > a$ □

Now if we compile our previous results we obtain the equivalence for the case where both the container width and the number of (1,2) rectangles are even.

Lemma 4.11. *Let (B, R, w, h) be a problem instance, where $l > a := B - (n \bmod B) - (n \bmod 2)$ and n is even. Then* $\mathrm{OPT}_{\mathrm{CUT}}(B,R,w,h) = \mathrm{OPT}_{\mathrm{BOX}}(B,R,w,h)$.

Proof. Lemma 4.9 gives us $\mathrm{OPT}_{\mathrm{BOX}}(B,R,w,h) \leq \lceil\frac{2l-2a}{B}\rceil + 2\lceil\frac{n}{B}\rceil$. Lemma 4.10 shows us that this is equal to $\lceil\frac{2(n+l)}{B}\rceil$. While Corollary 4.3 tells us that $\mathrm{OPT}_{\mathrm{CUT}}(B,R,w,h) \geq \lceil\frac{2(n+l)}{B}\rceil$. Since $\mathrm{OPT}_{\mathrm{CUT}} \leq \mathrm{OPT}_{\mathrm{BOX}}$, it follows that

$$\mathrm{OPT}_{\mathrm{CUT}}(B,R,w,h) = \mathrm{OPT}_{\mathrm{BOX}}(B,R,w,h).$$

□

4.3.2 Construction approach

Rather than showing equivalence through bounds we can also show equivalence by constructing a Box Solution based on an optimal Cutting Solution. The Figure 4.4 visualizes the method used.

We use the x-coordinates of an optimal solution in the Cutting Problem and construct an equivalent Box Solution by ordering rectangles, in a way that is free of overlaps. We achieve such an ordering, by placing width 2 rectangles first and alternating between bottom-up and top-down on even and odd x-coordinates respectively. Since this construction is not dependent on the rectangle height being at most 2, we can expand our case to any set of rectangles where the widths are at most 2.

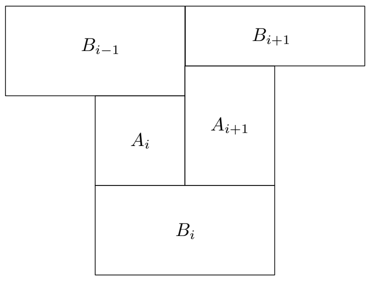

Figure 4.4: An example of the construction. The rectangles A_i, A_{i+1} are bounding boxes for the width 1 rectangles placed at x-coordinates i and $i+1$ respectively. The rectangles B_{i-1}, B_i, B_{i+1} are bounding boxes for the width 2 rectangles placed at x-coordinates $i-1, i$ and $i+1$ respectively. Furthermore, the coordinate i is even and therefore its rectangles are placed bottom-up, the coordinates $i-1$ and $i+1$ are uneven and thus their rectangles are placed placed top-down.

Theorem 4.12. *Let (B, R, w, h) be a problem instance, such that all rectangles have at most width 2, $\forall r \in R : w(r) \leq 2$. Then $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) = \mathrm{OPT}_{\mathrm{BOX}}(B, R, w, h)$.*

Proof. Consider an optimal placement x in the Cutting Problem, we can partition R into disjoint subsets $S_0, ..., S_{B-1}$, such that each subset S_i contains the rectangles with x-coordinate i, $S_i = \{k \in R \mid x(k) = i\}$, where $i \in \{0, ..., B-1\}$. We can completely cover R with the union of these subsets $S_0 \cup ... \cup S_{B-1} = R$, as every rectangle has an x-coordinate in $\{0, ..., B-1\}$. Furthermore the sum of rectangle heights in each subset is at most $\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h)$, $\forall i \in \{0, ..., B-1\} : \sum_{r \in S_i} h(r) \leq \mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h)$, as S_i is a subset of the set of *active* rectangles $C_i := \{r \in R \mid x(r) \leq i < x(r) + w(r)\}$, for every $i \in \{0, ..., B-1\}$ and the Cutting Problem height is the largest sum of active rectangles $\max\{\sum_{r \in C_i} h(r) \mid i \in \{0, ..., B-1\}\}$ (cf. Definition 2.5).

Furthermore we want to separate the width 2 rectangles from the width 1 rectangles, hence consider $A'_i = \{k \in S_i \mid w(k) = 1\}$ and $B'_i = \{k \in R \mid w(k) = 2\}$, then A'_i and B'_i are not only disjoint subsets of S_i, but they also cover S_i, for $i \in \{0, ..., B-1\}$.

Consider rectangles A_i and B_i (cf. Figure 4.4) with $w(A_i) = 1$, $h(A_i) = \sum_{k \in A'_i} h(k)$, $w(B_i) = 2$ and $h(B_i) = \sum_{k \in B'_i} h(k)$. Lemma 3.8 tells us that we can pack every rectangle in A'_i into A_i. Similarly we can apply the same argument to B'_i, since every rectangle in B'_i has width 2, which means we can pack every rectangle in B'_i into B_i.

Now consider the placement (as outlined in Figure 4.4) $y(B_i) = 0$, $y(A_i) = y(B_i) + h(B_i)$, if i is even and $y(B_i) = \mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) - h(B_i)$, $y(A_i) = y(B_i) - h(A_i)$, if i is odd, for

$i \in \{0, ..., B-1\}$. By construction it follows that A_i and B_i do not overlap (cf. Definition 2.4). Furthermore the definition of y ensures, that $y(A_i) + h(A_i) \leq \text{OPT}_{\text{CUT}}(B, R, w, h)$ and $y(B_i) + h(B_i) \leq \text{OPT}_{\text{CUT}}(B, R, w, h)$, for all $i \in \{0, ..., B-1\}$. Now assume that there are A, B, which overlap, then without loss of generality $B = B_i$ and $A = A_{i+1}$. We distinguish the cases odd and even, since $y(A_i)$ and $y(B_i)$ are defined differently depending on whether i is even or odd.

- Case 1: i is even, then $i+1$ is odd and thus $y(A_{i+1}) = y(B_{i+1}) - h(A_{i+1})$. Then the two rectangles overlap, if $y(B_i) + h(B_i) > y(A_{i+1})$, which holds true if and only if $h(B_i) + h(A_{i+1}) + h(B_{i+1}) > \text{OPT}_{\text{CUT}}(B, R, w, h)$, which contradicts x being an optimal placement in the Cutting Problem.
- Case 2: i is odd, then $i+1$ is even and thus $y(A_{i+1}) = h(B_{i+1})$. Similarly the two rectangles overlap, if $y(A_{i+1}) + h(A_{i+1}) > y(B_{i+1})$, which holds true if and only if $h(B_{i+1}) + h(A_{i+1}) + h(B_{i+1}) > \text{OPT}_{\text{CUT}}(B, R, h, w)$, which contradicts x being an optimal placement in the Cutting Problem.

Therefore, there are no overlaps, and we can pack the blocks of rectangles within height $\text{OPT}_{\text{CUT}}(B, R, w, h)$, since we can pack the individual rectangles into the larger blocks, we can also pack the smaller rectangles into the same height. Now, since the Cutting Problem is a lower bound to the Box Problem, it follows that $\text{OPT}_{\text{CUT}}(B, R, w, h) = \text{OPT}_{\text{BOX}}(B, R, w, h)$. □

4.4 Counterexample

Our previous results in equivalence might give rise to the notion that both problems are equivalent. However as it turns out the Cutting Problem and Box Problem are in fact not equivalent in the general case.

Lemma 4.13. *There exists a problem instance* (B, R, w, h)*, such that*

$$\text{OPT}_{\text{CUT}}(B, R, w, h) \neq \text{OPT}_{\text{BOX}}(B, R, w, h)$$

Proof. We divide this proof into two parts, first we present an instance and show that it is feasible in the Cutting Problem. Then we demonstrate that the same instance is infeasible in respect to the same height in the Box Problem. Figure 4.5 visualizes both the instance and the placement in the Cutting Problem.

Let $B = 5$, $R = \{b_1, c_1, c_2, c_3, d_1, d_2, e_1, e_2, e_3, e_4\}$, with

$$\begin{aligned} f(b_1) &= (1,1), & f(c_1) &= (1,2), \\ f(c_2) &= (1,2), & f(c_3) &= (1,2), \\ f(d_1) &= (1,3), & f(d_2) &= (1,3), \\ f(e_1) &= (3,1), & f(e_2) &= (3,1), \\ f(e_3) &= (3,1), & f(e_4) &= (3,1), \end{aligned}$$

where $f(i) = (w(i), h(i))$.

Consider the following placement in Table 4.1, as per Definition of the Cutting Problem (Definition 2.5) the height of the placement is the maximum of the heights, of the individual x-coordinates. The height of a specific x-coordinate i is the sum of all rectangles which cover

x-coordinate	rectangles
0	d_1, e_1, e_2
1	c_1, e_3
2	b_1, e_4
3	d_2
4	c_2, c_3

Table 4.1: The placement of the rectangles in the Cutting Problem.

the coordinate i. The following equations illustrate that the height at each x-coordinate is indeed 5.

$$\sum_{k\in\{k'\in R|x(k)\leq 0<x(k')+w(k')\}} h(k) = h(d_1) + h(e_1) + h(e_2) = 3 + 1 + 1 = 5$$
$$\sum_{k\in\{k'\in R|x(k)\leq 1<x(k')+w(k')\}} h(k) = h(c_1) + h(e_1) + h(e_2) + h(e_3) = 2 + 1 + 1 + 1 = 5$$
$$\sum_{k\in\{k'\in R|x(k)\leq 2<x(k')+w(k')\}} h(k) = h(b_1) + h(e_1) + h(e_2) + h(e_3) + h(e_4) = 5 \cdot 1 = 5$$
$$\sum_{k\in\{k'\in R|x(k)\leq 3<x(k')+w(k')\}} h(k) = h(d_2) + h(e_3) + h(e_4) = 3 + 1 + 1 = 5$$
$$\sum_{k\in\{k'\in R|x(k)\leq 4<x(k')+w(k')\}} h(k) = h(c_2) + h(c_3) + h(e_4) = 2 + 2 + 1 = 5$$

Since every x-coordinate has height 5, the maximum height is also 5 and thus

$$\mathrm{OPT}_{\mathrm{CUT}}(B, R, w, h) = 5$$

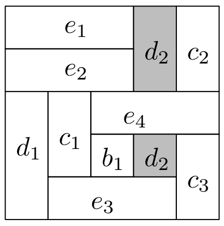

Figure 4.5: A solution which can only be constructed in the Cutting Problem. The gray area is a (1,3) rectangle, which is cut according to the Cutting Problem

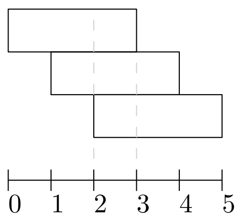

Figure 4.6: The different ways we can place (3,1) rectangles over a strip of width 5. Notice how all rectangles overlap on one x-coordinate.

Now assume that there exists a solution in the Box Problem with height 5. Then this solution cannot contain any free space as the total area of the rectangles is $\sum_{r\in R} h(r) \cdot w(r) = 25$, which is also the surface area of the 5 by 5 bounding box imposed by the Cutting Problem. Furthermore, we know that the rectangles d_1 and d_2 have different x-coordinates, as $h(d_1) + h(d_2) > 5$ and that e_1, e_2, e_3, e_4 have at least one x-coordinate in common, as none

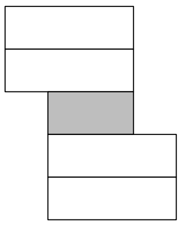

Figure 4.7: A placement, where the (3,1) rectangles share two x-coordinates. The gray rectangle represents an empty space resulting from such a placement.

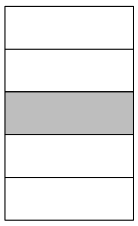

Figure 4.8: The (3,1) rectangles all occupy the same x-coordinates. The gray rectangle represents an empty space resulting from such a placement.

of the aforementioned rectangles can be placed at the same y-coordinate, due to having a larger width, than $B/2$.

There are only three x-coordinates where we can place the rectangles e_1, e_2, e_3, e_4 (cf. Figure 4.6), namely 0, 1 and 2, as placing any of these rectangles on coordinates 3, 4 or 5 violates the bounds of the container (cf. Definition 2.4). Therefore, at least two of the rectangles are placed at the same x-coordinate. Since there are four (3,1) rectangles and only three coordinates ,where e_1, e_2, e_3 and e_4 can be placed. Furthermore, the rectangles e_1, e_2, e_3 and e_4 may only overlap at one x-coordinate. As overlapping on at least two x-coordinate results in height 4 on more than 1 x-coordinate, however the only other height 1 rectangles is b_2, which only has a width of 1 (cf. Figure 4.7). Now this coordinate is $x(b_2) = 2$, since all of the (3,1) rectangles e_1, e_2, e_3, e_4 overlap with x-coordinate 2 (cf. Figure 4.6) and since the x-coordinate has to have a height of 5 the only option is to place the rectangle b_2 as x-coordinate 2. Now we examine the individual cases of how many (3,1) rectangles have the same x-coordinate.

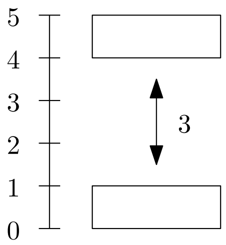

Figure 4.9: The two rectangles are three coordinates apart.

- Case 1: e_1, e_2, e_3, e_4 have the same x-coordinate.

  As we have already noticed, the rectangles e_1, e_2, e_3 and e_4 may only overlap on one x-coordinate. Placing the four rectangles at the same x-coordinate results in them overlapping at three x-coordinates and thus results in the problem, where we have a space where no remaining rectangle can be placed (cf. Figure 4.8).

- Case 2: exactly two out of e_1, e_2, e_3, e_4 have the same x-coordinate.

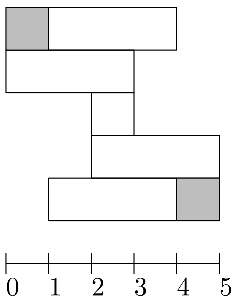

Figure 4.10: The two (3,1) rectangles cannot be placed at x-coordinate 1 and be 3 spaces apart, without creating unusable space.

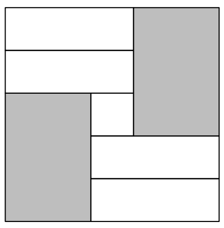

Figure 4.11: The gray areas mark the two spaces we are left with after having placed the (3,1) rectangles and the (1,1) rectangle.

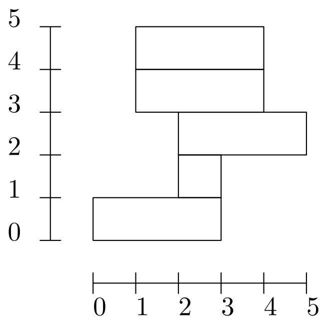

Figure 4.12: The (1,3) rectangles cannot both be packed into such a structure, as there is only one space of height 4, where a (1,3) rectangle would fit.

Without any loss of generality let e_1 and e_2 be the rectangles with the same x-coordinate a. Then e_1 and e_2 are either 0 or 3 y-coordinates *apart* (the height of the tallest rectangle that could be placed between the rectangles, see Figure 4.9). Since both a distance 1 and distance 2 create unusable space, between the two rectangles in the case of distance 1 or at the edge of the bounding box in the case of distance 2, where the space is either above or below one of the rectangles.

– Case 1: $a = 1$. As we have noted before e_1 and e_2 are 0 or 3 y-coordinates apart. However being 3 coordinates apart creates a structure, where we end up with a (1,1) space in a corner, due to one of e_3 and e_4 being adjacent to either e_1 or e_2 (see Figure 4.10). Such a space cannot be filled by any of the remaining rectangles, since b_2 has to be placed at x-coordinate 2. Therefore, the rectangles have no y-coordinates in between. Then the block of e_1 and e_2 is adjacent to the edge of the bounding box (see Figure 4.12), since any other configuration results in one of the two rectangle being a distance of 1 away form the edge of the bounding box resulting in a space of width 3 and height 1, which cannot be used. Furthermore, the other two rectangles e_3 and e_4 have to be placed on opposing sites with one having x-coordinate 1 and the other x-coordinate 2 (see Figure 4.12), to avoid overlapping at more than one x-coordinate. Now the rectangles e_3 and e_4 have to be exactly one y-coordinate apart or we end up with unusable space either at the edge of the bounding box or at the e_1, e_2 block. However this leads to a packing, where only one space of at least height 3 remains (see Figure 4.12). This means we cannot place both (1,3) rectangles, as there is only one space of height greater than 2. Hence $a = 1$ is impossible.

– Case 2: $a \in \{0, 2\}$ Let $a' \in \{0, 2\}$, such that $a \neq a'$, then at least one of e_3 and e_4 has a' as its x-coordinate. As e_1, e_2, e_3 and e_4 would overlap at more than one x-coordinate, if e_3 and e_4 both have x-coordinate 1 (The case where 3 of the rectangle have the same x-coordinate is considered as a separate case). Without any loss of generality let e_3 have a' as its x-coordinate. The rectangles e_1 and e_2 are at most three y-coordinates apart, namely when one of them is at the top and the other at the bottom.

  Now assume that the two rectangles are actually three coordinates apart. Then e_3 must be at y-coordinate 2, as coordinates 1 and 3 create a space of height 1 and width 3 between x-coordinate 2. However such a placement of e_3 forces e_4 into either y-coordinate 1 or 3, which both create the same problem as if we had placed e_3 there. As such we know that e_1 and e_2 are at most two y-coordinates apart.

  Furthermore we now know that e_3 and e_4 also have no y-coordinates in between, as being one coordinate apart creates the same problem where we have an unusable height 1 space. This in turn means e_3 and e_4 have the same x-coordinate (cf. Figure 4.11), as placing the rectangles at different x-coordinates, yet again results in unusable height 1 space at the lower or upper edge of the bounding square.

  After having placed all b_1, e_1, e_2, e_3 and e_4 we are left with two spaces of height 3 and width 2 (cf. Figure 4.11). However we cannot place c_1, c_2, c_3 in a single space of width 2 and height 3, as they are all (1,2) rectangles which create a height of 4 (see Lemma 4.4).

  Now the only remaining option is to pack two of the (1,2) rectangles together with one (1,3) rectangles. However this is also not possible due to the surface area bound (see Corollary 4.3). As such there is no way to pack the rectangles into a 5 by 5 square, if two out of e_1, e_2, e_3, e_4 have the same x-coordinate.

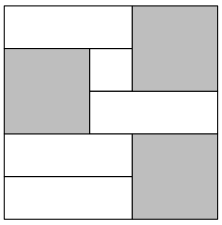

Figure 4.13: The rectangle in the middle separates the right side into two (2,2) spaces. The spaces are marked in gray.

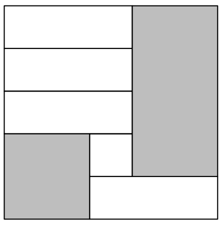

Figure 4.14: The gray (2,4) space to the right cannot be filled completely with (1,2) and (3,1) rectangles.

- Case 3: exactly three out of e_1, e_2, e_3, e_4 have the same x-coordinate. Without any loss of generality let e_1, e_2, e_3 be the rectangles with the same x-coordinate a. Then $a \in \{0, 2\}$, as placing e_1, e_2 and e_3 at the x-coordinate 1 causes the rectangles e_1, e_2, e_3 and e_4 to overlap on at least two x-coordinates. Therefore, $a \in \{0, 2\}$. Now let $a' \in \{0, 2\}$, such that $a \neq a'$, then a' is the x-coordinate of a', $x(e_4) = a'$, since the case $a' = 1$ yet again causes an x-coordinate overlap of at least two.

  Now e_4 has to be either adjacent to the edge of the bounding box or at y-coordinate 2, as being being 1 y-coordinate from the edge results in a space of width 2 and height 1. Furthermore, b_2 has to adjacent to e_4, to avoid the creation of a height 1 width 2 space between either two of the rectangles e_1, e_2, e_3 or one of the three rectangles and the edge of the bounding box. Now we want to examine the cases where e_4 is located on the y-axis. As we have noted before there are two major cases:

  - Case 1: $y(e_4) = 2$, then we can no longer place the (3,1) rectangles, as no space of height 3 remains (see Figure 4.13), due to e_4 separating the only remaining space with at least height 3 into spaces of height 2.
  - Case 2: $y(e_4) \in \{0, 4\}$, then it follows that e_1, e_2 and e_3 are stacked, as b_1 is adjacent to e_4. This means we end up with a packing where there is a (2,2) space and a (2,4) space left (see Figure 4.14). While the (2,2) space can be easily filled by two (1,2) rectangles, the remaining rectangles cannot be placed inside the (2,4) space, without rotating the (1,2) rectangle, which we do not allow (see 2.4).

  Therefore, there cannot be a Box Solution such that three (1,3) rectangles have the same x-coordinate.

Now since all cases fail, there cannot be any two (3,1) rectangles which have the same x-coordinate. However we have four (3,1) rectangles and only three x-coordinates at which those can be placed. Therefore, there is no solution with height 5 in the Box Problem. □

5. Converting Cutting Solutions into Box Solution

While it turned out that Cutting Problem and Box Problem are not equivalent, we can still use the solutions that the Cutting Problem provides. In this chapter we examine two approaches which convert Cutting Solutions into a Box Solutions. The first approach considers a simple algorithm, which initially assign 0 as y-coordinate for every rectangle and then gradually removes overlaps between rectangles, by moving rectangles upwards. The second approach considers mixed integer linear programming, as a method of conversion. Furthermore, we propose another mixed integer linear program to actually solve the Cutting Problem and compare the different approaches using empirical data.

5.1 Rectangle Bubbling

The Cutting Solution provides us with x-coordinates for each rectangle, thus we can construct a feasible Box Solution by moving rectangles along the y-axis. The first approach we consider continually moves overlapping rectangles one y-coordinate upwards, in a sense *bubbling* overlapping rectangles upwards.

Bubbling rectangles as a way of solving the Box Problem

Consider a feasible solution in the Cutting Problem (B, R, w, h, x). We can transform R from a set into a tuple, by numbering the rectangles in R. As such we end up with a rectangle tuple $R' = (r_1, ..., r_k)$, where $k = |R|$. This allows us to distinguish rectangles even if width, height and x-coordinate are equal. We can construct a solution in the Box Problem, for the same problem instance, by placing rectangles at the x-coordinate given by the Cutting Problem solution and y position 0. Naturally this might cause several overlaps between rectangles which are not allowed in the Box Problem. Therefore, we want to move overlapping rectangles one y-coordinate upwards. Now there are several ways to decide which rectangles to move first.

The version we consider first moves rectangles which have the most overlap in relation to their own surface area. If two overlapping rectangles have the same overlap ratio, we move the rectangle which has less surface area, if both rectangles have the same surface area, we move the rectangle which has a lower height. If those are also equal we move the rectangle which has lower width and as a last resort if rectangles have the exact same dimension we move the rectangle which has the lower index, or to be more precise for two rectangles r_i, r_j we move r_m, where $m = \min(i, j)$. This way there is no ambiguity as to which rectangle to move first and we can gradually remove all overlaps. As a result we end up with a feasible solution to the Box Problem (B, R'', w, h, x, y), where $R'' = \bigcup_{i=1}^{k}\{r_i\}$.

At best this solution is optimal, at worst the solution is equivalent to stacking all rectangles on top of each other. As there cannot be anymore overlaps between rectangles once no two rectangle share a height.

An example where bubbling fails to find a good solution

There are cases where the bubbled solution is significantly worse solution than the optimal solution in the Box Problem.

Consider rectangles $r_1, ..., r_n$ with heights $h(r_i) = 1$ and widths $w(r_i) = 2^i$ for each rectangle r_i and $x(r_1) = 0$ and $x(r_i) = x(r_i) + w(r_i) - 1$, where $i \in \{1, ..., n\}$.

If we compare the overlap ratio of the rectangles we find that for $i \in \{2, ..., n-1\}$ the rectangle r_i has an overlap ratio of $2/2^i$, with r_1 and r_n having an overlap ratio of $1/2^1$ and $1/2^n$ respectively. Now bubbling a rectangle r_i changes the overlap ratio of r_{i+1} to $1/2^{i+1}$, however the overlap ratio of r_{i+2} is $2/2^{i+2} = 1/2^{i+1}$, where $i \in \{1, ..., n-3\}$. This means that the leftmost rectangle which has an overlap will be bubbled, which in turn causes the previously bubbled rectangle to also bubble. This creates a stair like pattern, resulting in height $\sum_{k=1}^{n} h(r_k) = \sum_{k=1}^{n} 1 = n$ which is equal to stacking rectangles at the same x-coordinate.

An optimal solution in the Box Problem can be created by using the same x-coordinates and placing every second rectangle at y-coordinate 1, which results in a height of 2.

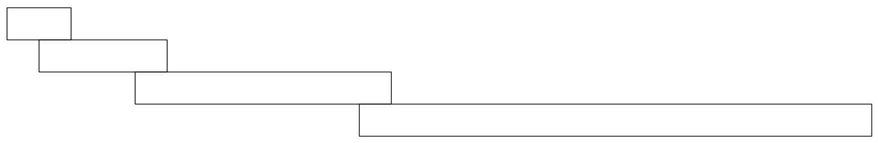

Figure 5.1: An example where bubbling fails to find a good solution. The y-coordinates reflect the actual placement created by the Bubble-Algorithm.

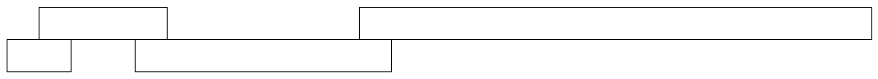

Figure 5.2: For comparison an optimal solution for the example where bubbling fails.

Bubbling large rectangles first

Since the example where bubbling fails relies on bubbling smaller rectangles first, it might seem as though bubbling larger rectangles first will solve this problem.

However we can generalize the example to rectangles $r_1, ..., r_n$ of height 1 and width $w(i)$, where $i \in \{1, .., n\}$ and w satisfies $1/w(i) \geq 2/w(i+1)$ for $i < n$.

Now the only case where the rectangle size is actually relevant is $1/w(i) = 2/w(i+1)$, as such we can simply construct an example where the rectangle width w satisfies $1/w(i) > 2/w(i+1)$, for $i < n$.

An example of such a function is $w(i) = 4^i$, as $\frac{1}{4^i} > \frac{1}{2 \cdot 4^i} = \frac{2}{4 \cdot 4^i} = \frac{2}{4^{i+1}}$.

5.2 Conversion using Mixed Integer Linear Programming

Consider a feasible placement in the Cutting Problem (B, R, w, h, x). We can construct a feasible placement in the Box Problem, by assigning y-coordinates to all rectangles, such that no two rectangles overlap. The problem of finding optimal y-coordinates can be solved using linear programming. We can bound the overall height and thus the rectangle y-coordinates with $M := \sum_{r \in R} h(r)$, since such a solution always exists. The problem can be formulated as a linear programming problem as follows.

$$\begin{aligned}
& \min H && && (5.1)\\
\text{subject to } & y(r) + h(r) \leq y(r') + M \cdot a_{r,r'}, && \forall r, r' \in R : r \neq r' && (5.2)\\
& y(r') + h(r') \leq y(r) + M \cdot b_{r,r'}, && \forall r, r' \in R : r \neq r' && (5.3)\\
& x(r) + w(r) \leq x(r') + B \cdot c_{r,r'}, && \forall r, r' \in R : r \neq r' && (5.4)\\
& x(r') + w(r') \leq x(r) + B \cdot d_{r,r'}, && \forall r, r' \in R : r \neq r' && (5.5)\\
& y(r) + h(r) \leq H, && \forall r \in R && (5.6)\\
& y(r) \geq 0, && \forall r \in R && (5.7)\\
& a_{r,r'} + b_{r,r'} + c_{r,r'} + d_{r,r'} \leq 3, && \forall r, r' \in R && (5.8)\\
& a_{r,r'}, b_{r,r'}, c_{r,r'}, d_{r,r'} \in \{0, 1\}, && \forall r, r' \in R && (5.9)
\end{aligned}$$

Parameters : These are given by the feasible Cutting Solution.

$$\begin{aligned}
B &= \text{the width of the container}\\
R &= \text{the set of rectangles}\\
h(r) &= \text{the height of the rectangle } r \in R\\
w(r) &= \text{the width of the rectangle } r \in R\\
x(r) &= \text{the } x\text{-coordinate of the rectangle } r \in R
\end{aligned}$$

Decision Variables

Objective Function : Minimize the overall height.

$$\text{Minimize } H$$

Constraint Set 1 : For every pair of rectangles r, r', the rectangle r' is above the rectangle r, or $a_{r,r'}$ is greater than 0.

$$y(r) + h(r) \leq y(r') + M \cdot a_{r,r'}, \forall r, r' \in R : r \neq r'$$

Constraint Set 2 : For every pair of rectangles r, r', the rectangle r is above the rectangle r', or $b_{r,r'}$ is greater than 0.

$$y(r') + h(r') \leq y(r) + M \cdot b_{r,r'}, \forall r, r' \in R : r \neq r'$$

Constraint Set 3 : For every pair of rectangles r, r', the rectangle r' is to the right of the rectangle r, or $c_{r,r'}$ is greater than 0.

$$x(r) + w(r) \leq x(r') + B \cdot c_{r,r'}, \forall r, r' \in R : r \neq r'$$

Constraint Set 4 : For every pair of rectangles r, r', the rectangle r is to the right of the rectangle r', or $d_{r,r'}$ is greater than 0.

$$x(r') + w(r') \leq x(r) + B \cdot d_{r,r'}, \forall r, r' \in R : r \neq r'$$

Constraint Set 5 : For every rectangle r, the overall height H is at least the height of the upper end of r.

$$y(r) + h(r) \leq H, \forall r \in R$$

Constraint Set 6 : For every rectangle r, the y-coordinate is at least 0.

$$y(r) \geq 0, \forall r \in R$$

Constraint Set 7 : For every pair of rectangles r, r', the sum of the binary variables $a_{r,r'}, b_{r,r'}, c_{r,r'}, d_{r,r'}$ is at most 3. This ensures that at least one of the variables is 0.

$$a_{r,r'} + b_{r,r'} + c_{r,r'} + d_{r,r'} \leq 3, \forall r, r' \in R$$

Constraint Set 8 : For every pair of rectangles r, r', the variables $a_{r,r'}, b_{r,r'}, c_{r,r'}, d_{r,r'}$ may only take values 0 and 1.

$$a_{r,r'}, b_{r,r'}, c_{r,r'}, d_{r,r'} \in \{0,1\}, \forall r, r' \in R$$

Our goal is to minimize the overall height H (cf. Eq. (5.1)), which is the highest end of all rectangles (cf. Eq. (5.6)). The constraint in Eq. (5.7) ensures that the solution does not violate the container bounds. In order for two rectangles to not overlap it suffices, to not overlap on a single axis. Two rectangles do not overlap on the y-axis if the lower edge of one rectangle r is above or at the same level as the upper edge of the other rectangle r' (cf. Eq. (5.2), Eq. (5.3)). Two rectangles do not overlap on the x-axis if the left edge of one rectangle r is to the right of the right edge of the other rectangle (cf. Eq. (5.4), Eq. (5.5)). Since only one of the four constraints (Eq. (5.2), Eq. (5.3), Eq. (5.4), Eq. (5.5)) has to hold true, we use binary variables $a_{r,r'}, b_{r,r'}, c_{r,r'}, d_{r,r'}$ to disable constraints, by adding the corresponding upper bound to the right hand side of the equation. Thus, fulfilling the constraint if the respective binary variable is 1. Eq. (5.8) ensures that at least one of the four orthogonality (Eq. (5.2), Eq. (5.3), Eq. (5.4), Eq. (5.5)) constraints holds true for any given rectangle.

5.3 Solving the Cutting Problem using Mixed Integer Linear Programming

Now that we have approaches to convert Cutting Solutions into Box Solutions. We want to consider actually solving the Cutting Problem. As with the conversion we are trying to find coordinates for the rectangles. In this case x-coordinates, rather than y-coordinate. This means we can also use linear programming to solve the Cutting Problem. In this case our input consists only of a problem instance (B, R, w, h). The following describes the Cutting Problem as a linear programming problem.

$$\begin{aligned}
&\min H && && (5.10)\\
\text{subject to } & x(r) \geq 0 && \forall r \in R && (5.11)\\
& x(r) + w(r) \leq B, && \forall r \in R && (5.12)\\
& H_i \geq \sum_{r \in R} h(r) \cdot c_{i,r}, && \forall r \in R, i \in \{0,..,B-1\} && (5.13)\\
& H \geq H_i, && \forall i \in \{0,..,B-1\} && (5.14)\\
& a_{i,r} \cdot B + x(r) \geq i + 1 && \forall r \in R, i \in \{0,..,B-1\} && (5.15)\\
& x(r) + w(r) \leq i + B \cdot b_{i,r} && \forall r \in R, i \in \{0,..,B-1\} && (5.16)\\
& c_{i,r} \geq a_{i,r} + b_{i,r} - 1 && \forall r \in R, i \in \{0,..,B-1\} && (5.17)\\
& a_{i,r}, b_{i,r}, c_{i,r} \in \{0,1\} && \forall r \in R, i \in \{0,..,B-1\} && (5.18)
\end{aligned}$$

Parameters : These are given by the instance.

$$\begin{aligned} B &= \text{the width of the container} \\ R &= \text{the set of rectangles} \\ h(r) &= \text{the height of the rectangle } r \in R \\ w(r) &= \text{the width of the rectangle } r \in R \end{aligned}$$

Decision Variables

Objective Function : Minimize the overall height.

$$\text{Minimize } H$$

Constraint Set 1 : For each rectangle $r \in R$, the x-coordinate is at least 0.

$$x(r) \geq 0, \forall r \in R$$

Constraint Set 2 : For each rectangle $r \in R$, the sum of x-coordinate and width is at most B.

$$x(r) + w(r) \leq B, \forall r \in R$$

Constraint Set 3 : For each rectangle x-coordinate i, the height of the x-coordinate H_i is at least the sum of rectangle heights, which overlap with i.

$$H_i \geq \sum_{r \in R} h(r) \cdot c_{i,r}, \forall r \in R, i \in \{0, .., B-1\}$$

Constraint Set 4 : For each x-coordinate i, the overall height is at least the height of the individual x-coordinates.

$$H \geq H_i, \forall i \in \{0, .., B-1\}$$

Constraint Set 5 : For each pair of rectangle r and x-coordinate i, if the x-coordinate of a rectangle r is at most i, then the binary variable $a_{i,r}$ must be greater than 0.

$$a_{i,r} \cdot B + x(r) \geq i + 1, \forall r \in R, i \in \{0, .., B-1\}$$

Constraint Set 6 : For each pair of rectangle r and x-coordinate i, if the sum of x-coordinate and width of a rectangle r is greater than i, then the corresponding binary variable $b_{i,r}$ must be greater than 0.

$$x(r) + w(r) \leq i + B \cdot b_{i,r}, \forall r \in R, i \in \{0, .., B-1\}$$

Constraint Set 7 : For each pair of rectangle r and x-coordinate i, if the binary variables $a_{i,r}$ and $b_{i,r}$ are 1, then the binary variable $c_{i,r}$ should also be 1. This ensures, that the height of the rectangle r is added to the coordinate i, if r overlaps with i.

$$c_{i,r} \geq a_{i,r} + b_{i,r} - 1, \forall r \in R, i \in \{0, .., B-1\}$$

Constraint Set 8 : For each pair of rectangle r and x-coordinate i, the variables $a_{i,r}, b_{i,r}, c_{i,r}$ may only take values 0 and 1.

$$a_{i,r}, b_{i,r}, c_{i,r} \in \{0, 1\}, \forall r \in R, i \in \{0, .., B-1\}$$

Again we want to minimize the overall height H (Eq. (5.10)). Which has to be at least the height of every individual x-coordinate, H_i (Eq. (5.14)). We calculate the individual heights using the binary variables $c_{i,r}$, which indicate whether a rectangle r is *covers* the x-coordinate i. A rectangle r covers an x-coordinate i, if and only if $x(r) \leq i < x(r) + w(r)$. To ensure that the variable $c_{i,r}$ is 1, if r covers i, we use two more binary variables $a_{i,r}$ and $b_{i,r}$. The variable $a_{i,r}$ takes the value 1, if $x(r) < i + 1$ (Eq. (5.15)). Whereas the variable $b_{i,r}$ takes the value 1, if $i < x(r) + w(r)$ (Eq. (5.16)). Now $c_{i,r}$ takes the value 1, if both $a_{i,r}$ and $b_{i,r}$ are 1 (Eq. (5.17)). Note that all the binary variables can also have value 1, if the conditions are not satisfied. However taking value 1, when it is not required only leads to a worse height. Since our goal is to minimize the overall height, a minimal solution will only have $c_{i,r}$ be 1, if and only if $x(r) \leq i < x(r) + w(r)$. The bounds of the container are ensured by the constraints Eq. (5.11) and Eq. (5.12).

5.4 The Box Problem as Linear Programming Problem

Since we have defined the Cutting Problem and the problem of converting a Cutting Solution into a Box Solution as linear programming problems, we can easily define the actual Box Problem as a linear programming problem. In fact the formulation is nearly identical to the conversion problem, with the addition of the two x bounding constraints from the Cutting Problem (Eq. (5.11), Eq. (5.12)).

$$\begin{aligned}
&\min H \\
\text{subject to } & y(r) + h(r) \leq y(r') + M \cdot a_{r,r'}, && \forall r, r' \in R : r \neq r' \\
& y(r') + h(r') \leq y(r) + M \cdot b_{r,r'}, && \forall r, r' \in R : r \neq r' \\
& x(r) + w(r) \leq x(r') + B \cdot c_{r,r'}, && \forall r, r' \in R : r \neq r' \\
& x(r') + w(r') \leq x(r) + B \cdot d_{r,r'}, && \forall r, r' \in R : r \neq r' \\
& y(r) \geq 0, && \forall r \in R \\
& y(r) + h(r) \leq H, && \forall r \in R \\
& x(r) \geq 0 && \forall r \in R \\
& x(r) + w(r) \leq B, && \forall r \in R \\
& a_{r,r'} + b_{r,r'} + c_{r,r'} + d_{r,r'} \leq 3, && \forall r, r' \in R \\
& a_{r,r'}, b_{r,r'}, c_{r,r'}, d_{r,r'} \in \{0, 1\}, && \forall r, r' \in R
\end{aligned}$$

5.5 Empirical Analysis of Linear Programming Models and the Bubble Algorithm

In the following we compare the Bubble Algorithm and the optimal solutions provided by the linear program models using empirical data. The test system was an Intel Core i3-7100U CPU, clocked at 2.4 GHz with 2 cores, 4 threads and 8 GB RAM.

The linear programming models were implemented in C++ using Gurobi, whereas the Bubble Algorithm was implemented in Haskell. The time measurements for the linear programming models were taken using the Chrono library. For the Bubble Algorithm, the measurements were taken using the Criterion Measurement package.

The rectangle widths and heights where chosen randomly from $\{1, ..., B\}$ using uniform distribution for width and power-law distribution for height, where B is the container width. The power-law distribution is a distribution similar to the exponential distribution, with the decay of the power-law distribution being much slower. As such the power-law distribution has very few large values, with the majority of the distribution consisting of smaller values. Like the exponential distribution the power-law distribution also has a

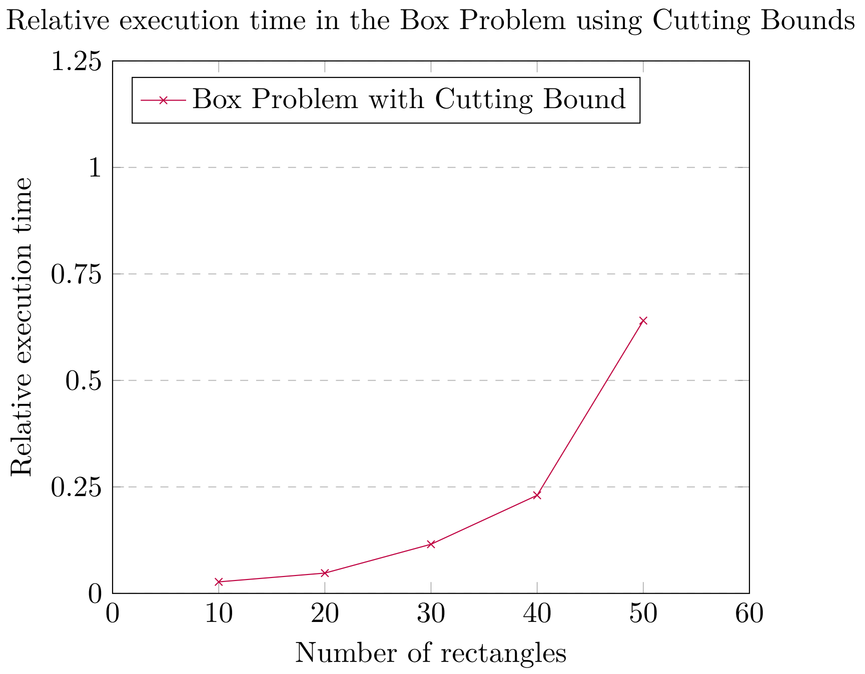

Figure 5.3: The relative execution time of Box Problem using the bound provided by Cutting Problem. The execution time of the Cutting Problem was not factored in for this graph.

parameter, which influences the distribution. In our tests we use −2 as the value for this parameter.

In the following we use the names *Cutting Problem* and *Box Problem* to refer to the corresponding solver rather than the actual problem. The term *Box Conversion* refers to the solver which transforms Cutting Solutions into Box Solutions. Furthermore, we will occasionally use abbreviations for the different solving methods in order to reduce the size of the legend in some graphs. The abbreviations are: *BC* for the Box Conversion solver, *BP* for the Box Problem solver, *BPC* for the Box Problem solver with the bound from the Cutting Solution provided as parameter, *CP* for the Cutting Problem solver and lastly *BA* for the Bubble Algorithm.

Both resulting heights and execution times were averaged over 20 measurements using arithmetic mean for each combination of container width and number of rectangles. For Gurobi the execution time was capped to 30 minutes (1800 seconds). Therefore, the resulting heights, for instances which exceeded the time limit, are not necessarily optimal, whereas the execution time without the cap is likely to be significantly higher. For the Box Conversion solver the height resulting from the Cutting Solution was provided as a parameter to lower bound the solution. For the Box Problem solver the area bound was used as a lower bound for the solution. Both solvers were upper bounded by the height of all rectangles. While the lower bounds are not required for Gurobi to find a solution, they significantly improve the execution times, allowing larger instances to be solved within the time cap.

The execution time of the Box Problem can be reduced significantly by providing the height of the Cutting Problem as a lower bound (cf. Figure 5.3). Though some instances starting at 30 rectangles still require more time than the limit allows.

Figure 5.4 shows a comparison of the average heights of the Bubble Algorithm and the linear programming models. The results show that both the height of the Bubble Algorithm

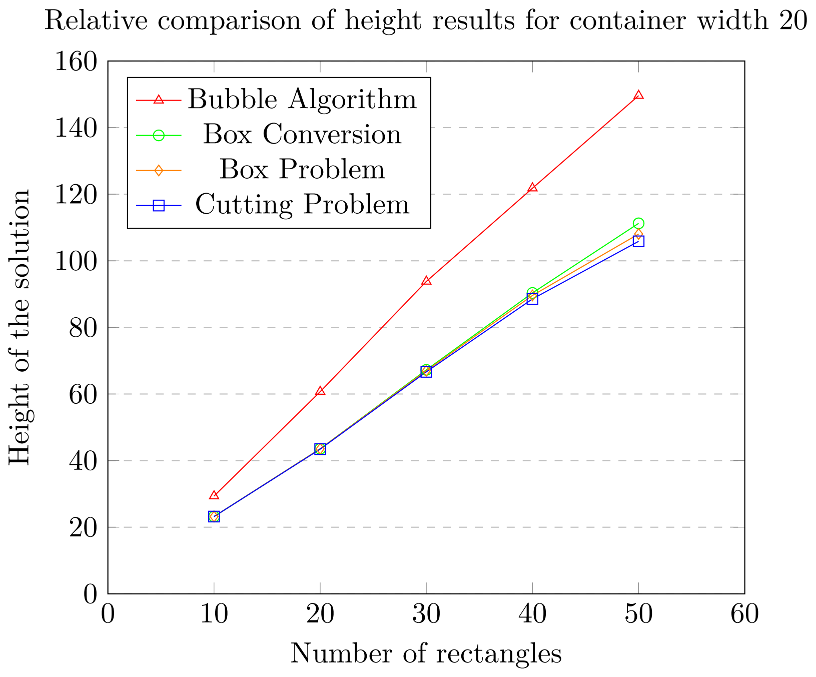

Figure 5.4: A comparison of the average resulting heights in relation to the number of rectangles, with the container width being 20 for each instance.

and the height of the linear programming models increases in a similar manner, with the height of the Bubble Algorithm rising slightly faster, than the linear programming models. The differences between Box Conversion and Box Problem starting at 30 rectangles are a possible result of the time cap, since at 30 rectangles some instances for the Box Conversion hit the time cap, thus not necessarily finding an optimal solution.

The height of the Bubble Solution in relation to the optimal Box Solution is on average well below 1.5, staying close to 1.4 (cf. Figure 5.5). This means that on average the Bubble Algorithm is only 40% worse than the optimal solution, despite having no worst case bound other than the sum of rectangle heights. Whereas the heights of the Cutting Problem and the heights of the Box Conversion are almost identical to the optimal Box Solution. As mentioned before the difference between the Box Conversion and the Box Problem that starts to appear at 30 rectangles is likely a result of the time cap, since the Box Conversion for some instances at 30 rectangles requires more time than the cap allows.

As one might expect the Box Problem with just the area bound requires significantly more time to solve, than the combination of Cutting Problem and Box Conversion (see Figure 5.6). In fact solving the Box Problem with just the area bound is only viable for instances with less than 20 rectangles (cf. Figure 5.7). The Box Problem with the bound provided by the Cutting Problem however, requires noticeably less time than the Box Problem without the bound and even less time, than the Box Conversion for instances beyond 20 rectangles. Despite the lower average execution time of the Box Problem with the Cutting Problem bound the execution time of the Box Conversion is lower for some rare instances (see Figure 5.7 at 30 rectangles) As expected the Bubble Algorithm is faster than all other solving methods, that were considered, barely reaching an average of 11 seconds at 50 rectangles. Similarly the Cutting Problem also requires little computation time, though despite the short average times, the time limit was exceeded for a single instance at 50 rectangles.

An increase in container width and thus an increase in the average size of the rectangles usually results in higher solutions height (see Figure 5.8). The reduction in height seen at container width 50 is a result of the high variance in height at container width 40

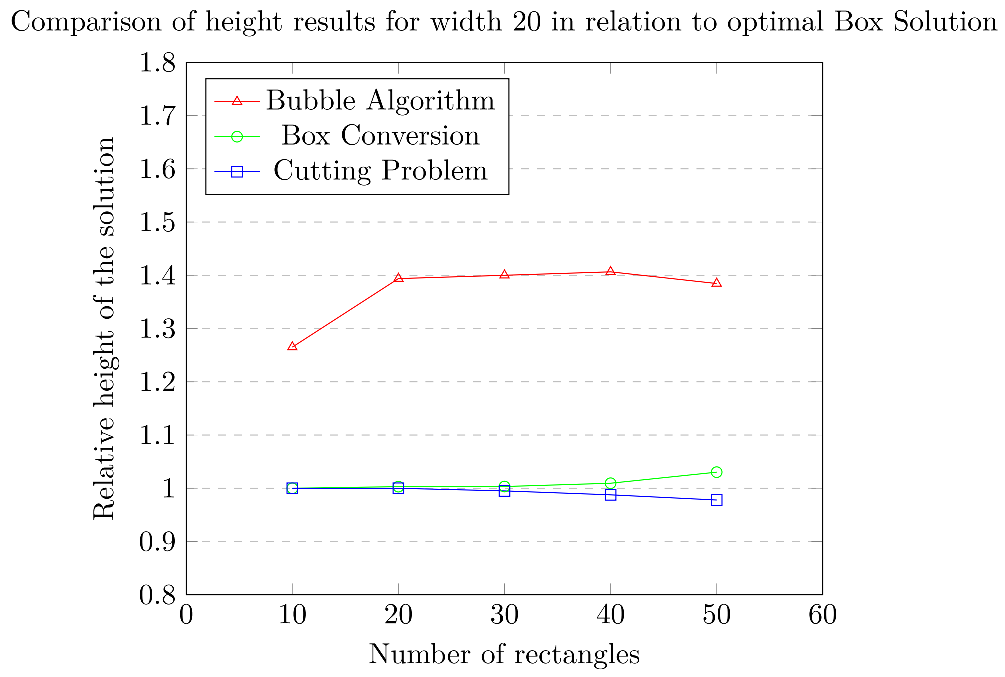

Figure 5.5: A comparison of the height results in relation to the height of the optimal Box Solution.

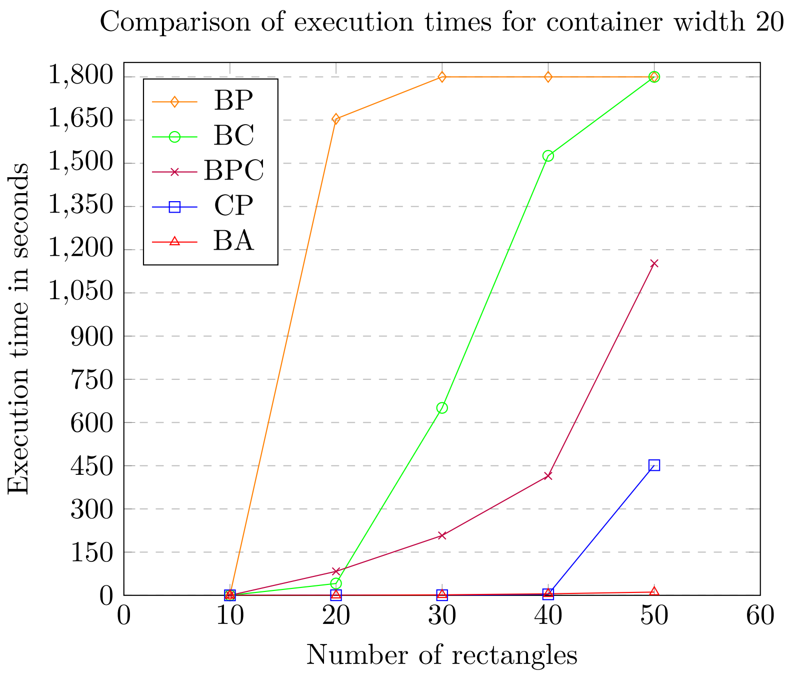

Figure 5.6: A run time comparison between the linear programming models and the Bubble Algorithm. The names of the various solving methods were shortened to reduce the size of the legend.

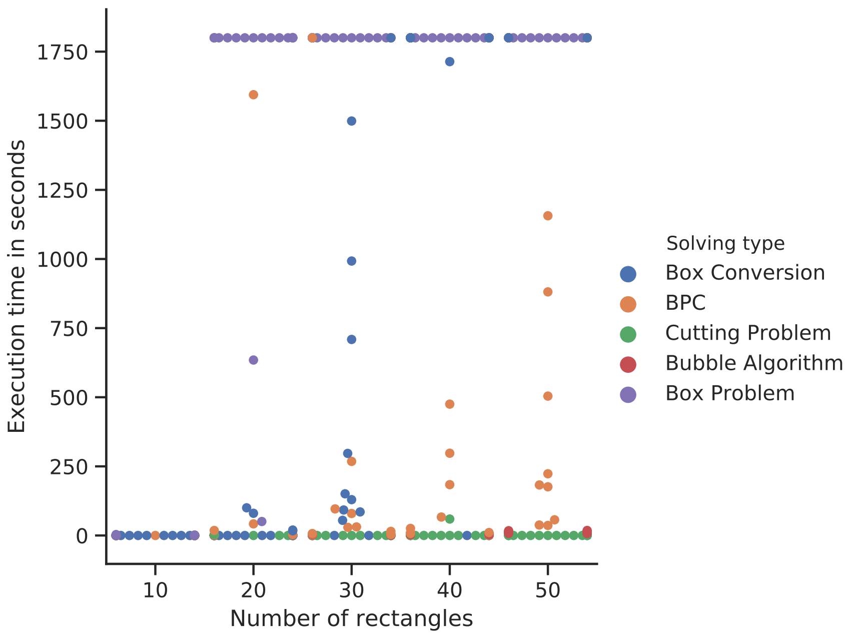

Figure 5.7: The distribution of execution times in relation to the number of rectangles, with the container width being 20 for each data point.

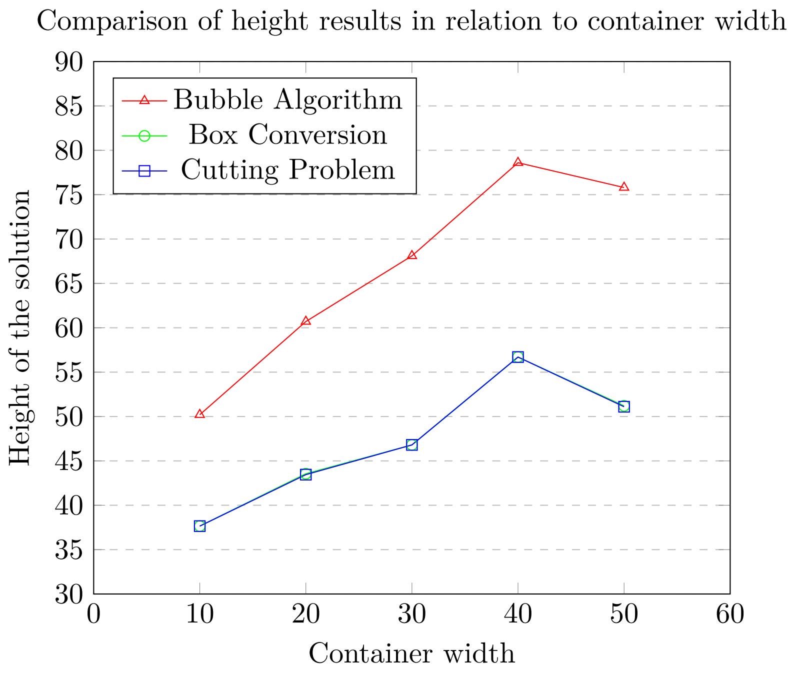

Figure 5.8: Comparison of height results in relation to container width, with the number of rectangles being 20 for each data point. The Box Problem was omitted due to being identical to the Cutting Problem for all data points. Similarly the Box Conversion is equivalent to the Cutting Problem in all but the last data point.

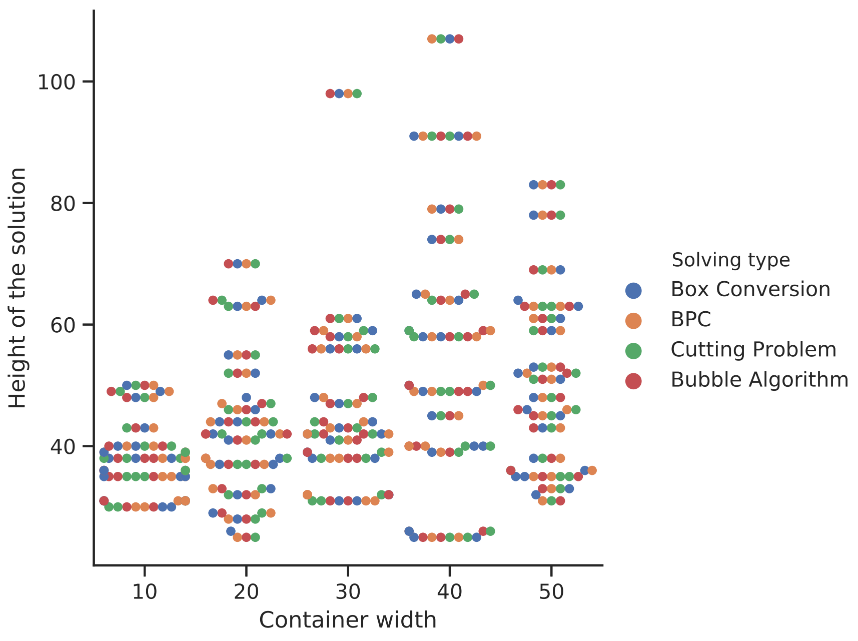

Figure 5.9: The distribution of height results in relation to the container width, with the number of rectangles being 20 for each data point.

(cf. Figure 5.9), without the outliers the general increase in height becomes much more apparent. As a result the solution heights increase with the container width, or rather the size of the rectangles, though much slower than with the number of rectangles.

Unlike with the number of rectangles, there does not seem to be any direct relation between container width and execution time for the Box Problem and the Box Conversion (see Figure 5.10). The execution times for most instances are close to 1 second, though some rare instances take significantly more time for the Box Conversion and the bounded Box Problem (cf. Figure 5.11). For the Cutting Problem and the Bubble Algorithm however the relation between the container width and the execution time is very similar to the relation between the number of rectangles and the execution time (cf. Figure 5.10 and Figure 5.4).

Solving the Box Problem using linear programming models works well for smaller instances, however the average required run time increases significantly with each larger instance. Therefore, for these instances the Bubble Algorithm may be a much better option, if a slight increase in height is acceptable, due to its low run time requirements. For problems where optimal or near optimal height is required, the Bubble Algorithm may still serve as an initial solution, upon which further solutions can be built.

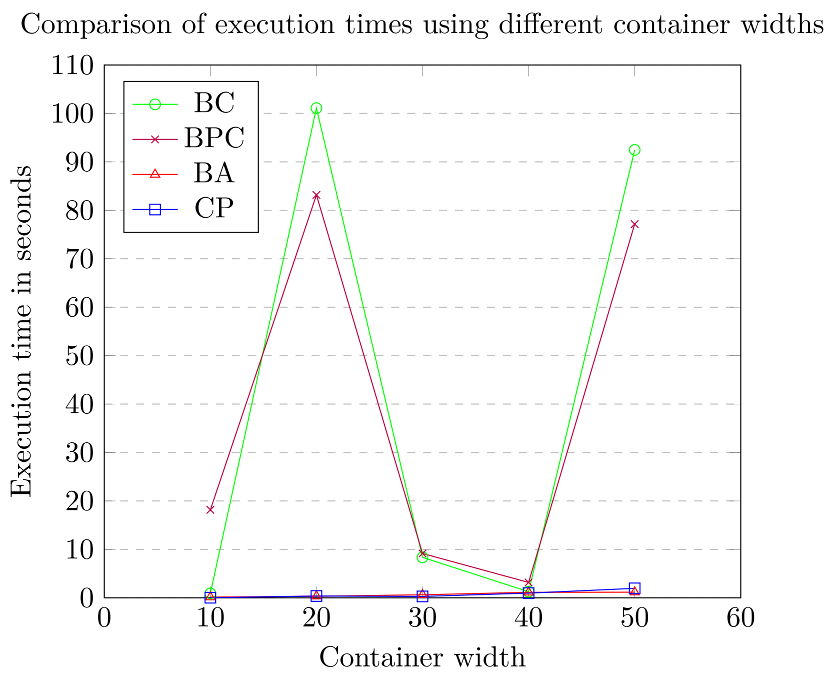

Figure 5.10: Comparison of execution times in relation to container width, with the number of rectangles being 20 for each data point. The spikes at container width 20 and 50 in the Box Problem and Box Conversion are both caused by a single instance exceeding the time limit for the Box Conversion and almost reaching it for the Box Problem.

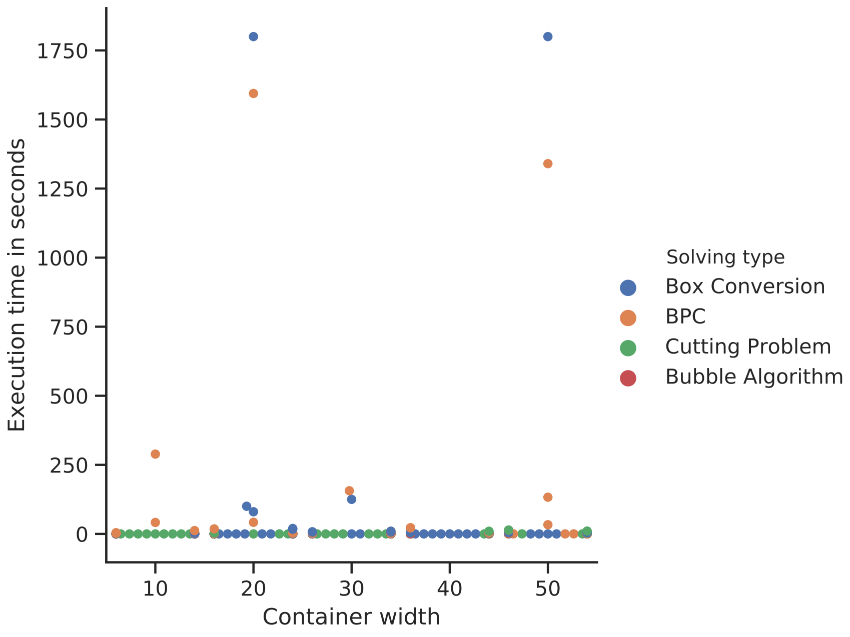

Figure 5.11: The distribution of execution times in relation to container width, with the number of rectangles being 20 for each data point.

6. Conclusion

6.1 Summary

The main objective of this thesis is to examine the relation of the Box Problem and Cutting Problem. The equality of both problems was proven for the case where all rectangles have at most height 1 and for the case where each rectangle has at most width 2. For the latter case, two approaches were considered, first a bounding approach and secondly a construction approach. While the bounding approach was limited to rectangles of area 2 and only proved effective for cases where the container width is even, it also served as a cornerstone for the construction approach, providing construction strategies, such as divide and conquer by rectangle type and combining smaller rectangles into larger rectangles. The upper bounds where obtained through generalizations of the constructions, with the height of the construction providing an upper bound for the Box Problem.

The construction approach was based on the idea of constructing a Box Solutions using x-coordinates from feasible Cutting Solutions. Combined with the idea of merging of smaller rectangles the approach was ultimately used to show the equality of Cutting and Box Problem for the case, where all rectangles have at most width 2. The inequality of both problems was proven for rectangle heights and widths at most 3, with at least one of the two sides being 1. Thus answering the question of whether the two problems are equivalent.

Expanding on the idea of constructing feasible Box Solutions from feasible Cutting Solutions an algorithm was proposed, which at first places all rectangles at the x-coordinate from the Cutting Solution and y-coordinate 0, then gradually removes overlaps by moving rectangles upwards. Furthermore, a mixed integer linear programming model was developed to create optimal Box Solutions based on Cutting Solutions. In order to create optimal Cutting Solutions another linear programming model was developed to solve the Cutting Problem. A linear programming model for the Box Problem was obtained by combining the relevant constraints of both models.

While solving the Box Problem using linear programming models works well for smaller instances, the required run time increases significantly with each larger instance. For these instances the Bubble Algorithm may be a much better option, if a slight increase in height is acceptable, due to its low run time requirements. For problems where optimal or near optimal height is required, the Bubble Algorithm may still serve as an initial solution, upon which further solutions can be built.

6.2 Future Work

It would be interesting to see whether or not the Box Problem and the Cutting Problem are equal for rectangle height at most 2, since the two problems are equal for width at most 2, height at most 1 and not equal for height and width at most 3. Leaving the case of rectangle height at most 2 as an open case. Since the case of rectangle height at most 1 already relied on Dilworth's Theorem it would be very interesting to see how the case could be proven, if at all.

Another interesting topic may be the improvement of the Bubble Algorithm, for example by using the spaces created by the algorithm or even different methods of bubbling altogether. In particular it would be very interesting to see how the worst case bound could be reduced, by such methods.

Bibliography

[dC99] J. M. Valério de Carvalho. Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86(0):629–659, 1999.

[Die17] Reinhard Diestel. *Graph Theory.* Springer Nature, fifth edition, 2017.

[Dyc90] Harald Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145 – 159, 1990.

[KR00] Claire Kenyon and Eric Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25(4):645–656, 2000.

[LBBR15] Ilze Laicane, Dagnija Blumberga, Andra Blumberga, and Marika Rosa. Reducing household electricity consumption through demand side management: The role of home appliance scheduling and peak load reduction. *Energy Procedia*, 72:222 – 229, 2015. International Scientific Conference “Environmental and Climate Technologies, CONECT 2014.

[MMV03] Silvano Martello, Michele Monaci, and Daniele Vigo. An exact approach to the strip-packing problem. *INFORMS Journal on Computing*, 15(3):310–319, 2003.

[MPT00] Silvano Martello, David Pisinger, and Paolo Toth. New trends in exact algorithms for the 0-1 knapsack problem. *European Journal of Operational Research*, 123:325–332, 2000.

[MV98] Silvano Martello and Daniele Vigo. Exact solution of the two-dimensional finite bin packing problem. *Management Science*, 44(3):388–399, 1998.

[PS05] David Pisinger and Mikkel Sigurd. The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optimization*, 2(2):154 – 167, 2005.

[Ste97] A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM J. COMPUT.*, 26(2):401–409, 1997.