9-9-2019 11:30 AM

High Multiplicity Strip Packing

Andrew Bloch-Hansen, *The University of Western Ontario*

Supervisor: Solis-Oba, Roberto, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

Abstract

In the *two-dimensional high multiplicity strip packing problem* (HMSPP), we are given k distinct rectangle types, where each rectangle type T_i has n_i rectangles each with width $0 < w_i \leq 1$ and height $0 < h_i \leq 1$. The goal is to pack these rectangles into a strip of width 1, without rotating or overlapping the rectangles, such that the total height of the packing is minimized.

Let $OPT(I)$ be the optimal height of HMSPP on input I. In this thesis, we consider HMSPP for the case when $k = 3$ and present an $OPT(I) + \frac{5}{3}$ polynomial time approximation algorithm for it. Additionally, we consider HMSPP for the case when $k = 4$ and present an $OPT(I) + \frac{5}{2}$ polynomial time approximation algorithm for it.

Keywords: Approximation algorithms, geometric packing problems, high multiplicity strip packing, linear program rounding, analysis of algorithms

Summary for Lay Audience

Packing problems typically involve maximizing the number of objects that can be placed into a container or minimizing the number of containers needed to hold a set of objects. Many industrial problems can be modeled as packing problems involving rectangles and squares.

Solutions for rectangle packing problems are useful, for example, for loading pallets and shipping containers for storage and transport, for designing transistor layouts for computer chips, and for optimizing workflow in a workplace. There is a quantifiable difference between good and bad solutions in the industrial environment. Wasted space during storage and transport costs companies resources: time might need to be spent re-packing containers, additional containers might need to be shipped, or product might be lost if certain weight restrictions are not satisfied. Many companies still use trial-and-error approaches while packing items for storage and transport.

In this thesis we consider the *two-dimensional high multiplicity strip packing problem*. In this problem we are given k distinct rectangle types, where each rectangle type T_i has n_i rectangles each with width $0 < w_i \leq 1$ and height $0 < h_i \leq 1$. The goal is to pack these rectangles into a strip of width 1, without rotating or overlapping the rectangles, such that the total height of the packing is minimized. We consider the problem for the cases when there are 3 different rectangle types and 4 different rectangle types.

Efficient algorithms for packing problems translate into improved industrial practices that reduce company expenses and improve product delivery. Furthermore, algorithm design techniques specifically developed for rectangle and square packing problems contribute to the design of efficient algorithms for other types of optimization problems. As research into packing problems expands, more industrial problems will be solved using these algorithms and more packing and optimization problems will be able to leverage the algorithmic techniques that are discovered.

Co-Authorship Statement

This work was made possible by the mentorship of Professor Roberto Solis-Oba. The algorithm for the high multiplicity strip packing problem for the case when $k = 3$ was designed in collaboration with Professor Roberto Solis-Oba's master's student Andy Yu. The algorithm for the high multiplicity strip packing problem for the case when $k = 4$ is a new algorithm that extends the first algorithm.

Contents

List of Algorithms

List of Figures

Chapter 1

Introduction

The problem that we study in this thesis is the *two-dimensional high multiplicity strip packing problem* (HMSPP): given k distinct rectangle types, where each rectangle type T_i has n_i rectangles each with width $0 < w_i \leq 1$ and height $0 < h_i \leq 1$, the goal is to pack these rectangles into a strip of width 1, without rotating or overlapping the rectangles, such that the total height of the packing is minimized (see Figure 1.1).

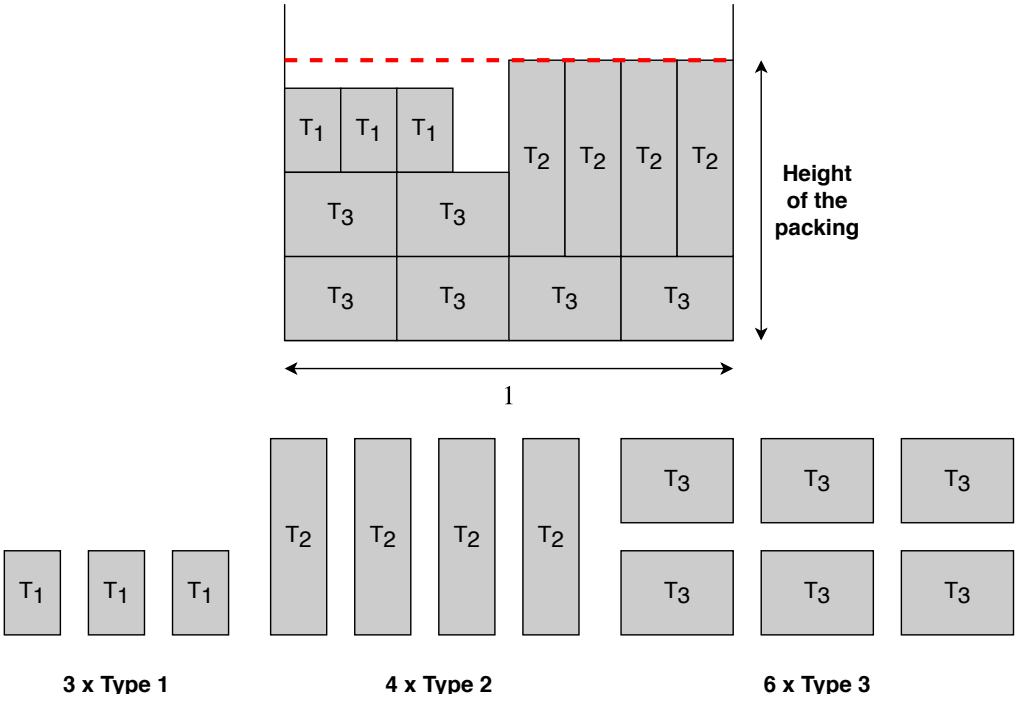


Figure 1.1: An instance of HMSPP where $k = 3$. There are 3 type 1 rectangles, 4 type 2 rectangles, and 6 type 3 rectangles. The total height of the packing is measured from the top of the topmost rectangle to the base of the strip.

The thesis is organized in the following manner. In this chapter we introduce the background concepts needed to understand our research and state our contributions. In Chapter 2 we highlight some related problems and methodologies to solve them. In Chapter 3 we give a formal definition of the high multiplicity strip packing problem and we describe how to use

LP-rounding to find solutions for it. In Chapter 4 we describe an algorithm for the case when $k = 3$, and in Chapter 5 we describe an algorithm for the case when $k = 4$. In Chapter 6 we summarize our findings and describe future research possibilities.

1.1 Fundamental Concepts

In an *optimization problem* the goal is to find the best solution from all *feasible* or possible solutions. A feasible solution for a problem satisfies all the constraints of the problem. An optimization problem can either be *continuous* or *discrete*. The standard form of a continuous optimization problem is:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0, \text{ for all } i = 1, ..., m \\ & h_j(x) = 0, \text{ for all } j = 1, ..., p \\ \text{and} & x \in \mathbb{R}^n \end{array}$$

where x is a vector of variables, f is an *objective function*[1], $g_i(x) \leq 0$ is an *inequality constraint*, and $h_j(x) = 0$ is an *equality constraint*. Finding the best solution from all feasible solutions is equivalent to minimizing the objective function. Maximization problems can be solved by negating the objective function.

A *combinatorial optimization problem* is a discrete optimization problem in which the variables used to express a feasible solution take on values from a *discrete* set. Finding an optimal solution to a combinatorial optimization problem might involve selecting an optimal combination, ordering, arrangement, or set of variables. Many researchers in the fields of mathematics, computer science, and operations research study combinatorial optimization problems due to their theoretical and practical importance.

A special kind of combinatorial optimization problems are *packing problems*[2]. Some packing problems involve maximizing the number of objects that can be placed into a container, while some others involve minimizing the number of containers needed to hold a set of objects. In *geometric packing problems*, the objects being packed are regular geometric shapes. Many problems that do not involve regular geometric shapes or containers, such as scheduling tasks on processors, can be represented as packing problems by encoding their input into geometric shapes. For example, in a scheduling problem requiring the processing of a set of tasks by a group of processors where each task needs to be processed by a contiguous subset of processors, the number of processors required by a task can be represented by the width of a rectangle and the time needed to process the task can be represented by the length of the rectangle. A schedule of the tasks on the processors then corresponds to a packing of the corresponding set of rectangles into a container of the same width as the number of processors (see Figure 1.2).

Many optimization problems, including geometric packing problems, are considered to be very complex computer science problems. An entire field of research, called *computa-*

[1] Sometimes referred to as a *loss function*, *reward function*, or *cost function*.

[2] Without loss of generality we can assume that there is not an infinite number of possible positions for the objects being packed as we can assume that we minimize space between adjacent objects. That makes the problem discrete.

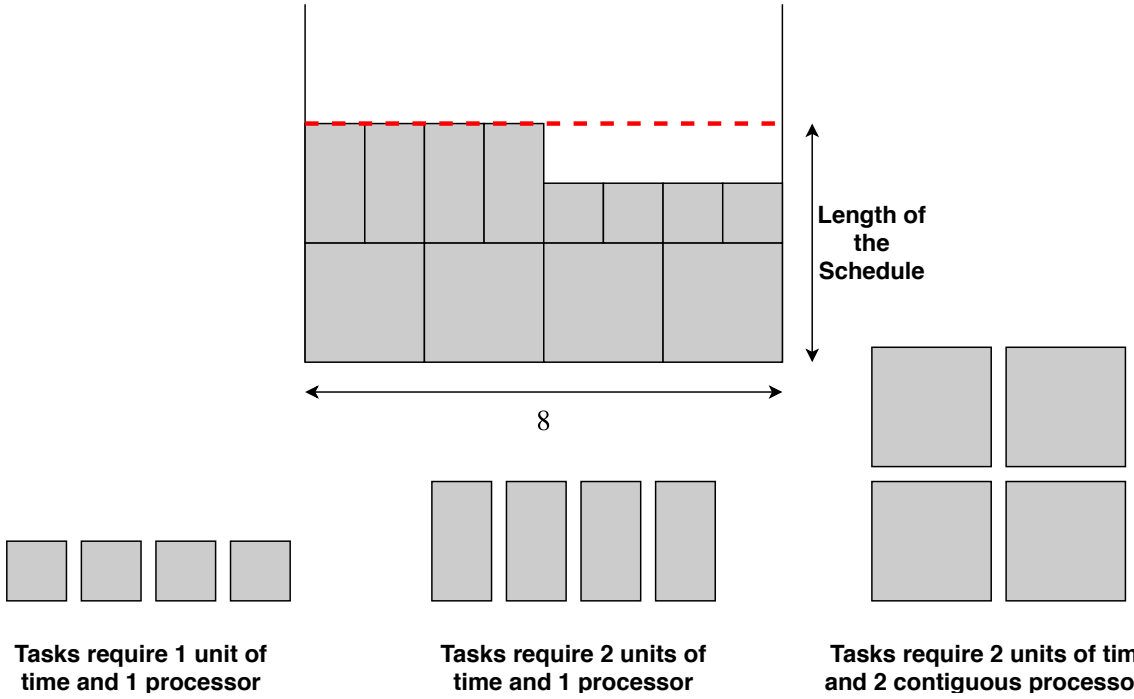


Figure 1.2: Scheduling the processing of a set of tasks by a group of processors can be encoded into a packing problem: the number of contiguous processors required by a task can be represented by the width of a rectangle, and the time needed to process the task can be represented by the length of the rectangle. Note that in this example we have assumed that the processors are indexed and that a process being scheduled on multiple processors is assigned to a contigous block of the processors.

tional complexity theory, aims to group problems into classes according to their complexity, or difficulty to solve them. The difficulty of a problem can be measured by the amount of computational resources required to solve it, such as the amount of storage or the time needed to compute a solution. The *time complexity* function measures the number of computational operations performed by an algorithm. The *space complexity* function measures the amount of memory required by an algorithm.

In this thesis we are most interested in these complexity classes:

- *P*
- *NP*
- *NP-hard*
- *APX*

The complexity class *P* contains all problems that can be solved by algorithms with time complexity functions that are polynomial in the size of the input (hereafter described as *in polynomial time*). The class *NP* contains all problems for which there are algorithms that when provided a potential answer can verify the answer's correctness in polynomial time. Problems in the class *NP-hard* are considered at least as hard as the hardest problems in *NP*. A problem *H* is *NP-hard* if every other problem in *NP* can be transformed into *H* in polynomial time: therefore, if *H* were solvable in polynomial time, then by the polynomial time transformation from every problem in *NP* to *H*, all problems in *NP* would be solvable in polynomial time. Currently, there are no known polynomial time algorithms to compute exact solutions for any

problems in the complexity class *NP-hard*.

Computational resources commonly need to be restricted when developing practical algorithms. A computer does not have infinite storage capacity and an algorithm is probably intended to finish during the lifetime of its designer. Algorithms are called *efficient* if their time complexity is a polynomial function of the size of the input. The complexity class *APX* is the set of *NP-hard* optimization problems for which there are efficient algorithms that can produce solutions that are within a constant factor of the optimal solutions.

Many important practical and theoretical problems belong to the complexity class *NP-hard*; therefore, we do not know any efficient algorithms to compute exact solutions to these problems. To overcome this, some researchers design *approximation algorithms* for these problems. Instead of computing exact solutions, approximation algorithms aim to compute in polynomial time solutions that are provably within a certain factor of the optimal solutions. Algorithms that compute approximate solutions but do not provide a guarantee on the maximum difference between the solutions that they compute and optimum solutions are commonly called *heuristics*.

Let $OPT(I)$ be an optimal solution for a problem on instance I, and let $SOL(I)$ be a solution output by an approximation algorithm for the same instance I. The *approximation ratio* of an approximation algorithm measures how close the solution output by the algorithm is to an optimal solution. For minimization problems, *approximation ratio* = $\frac{SOL(I)}{OPT(I)}$ and for maximization problems, *approximation ratio* = $\frac{OPT(I)}{SOL(I)}$. For some problems, a *polynomial time approximation scheme* can be constructed, which is an algorithm that accepts as input the parameter $\epsilon > 0$ such that $\frac{SOL(I)-OPT(I)}{OPT(I)} \leq \epsilon$ (minimization problems) or $\frac{OPT(I)-SOL(I)}{OPT(I)} \leq \epsilon$ (maximization problems) and has a time complexity that is a polynomial function of both the size of the input and $\frac{1}{\epsilon}$.

Asymptotic analysis describes the quality of an approximation algorithm as the value of an optimal solution becomes very large. Many approximation ratios have the form $A + \frac{C}{OPT(I)}$, where A is factor not dependent on $OPT(I)$ called the asymptotic approximation ratio and C is an *additive constant*. When $OPT(I)$ is small, the additive constant of the approximation ratio might be significant; however, in the asymptotic setting, additive constants are insignificant.

1.2 Applications

Many industrial problems can be modeled as geometric packing problems. One of the most famous packing problems, the cutting-stock problem, can model the industrial practice of cutting raw material such as paper, wood, and metal into smaller units [23]. Solutions for rectangle packing problems can be applied, for instance, to loading pallets and shipping containers for storage and transport, to designing transistor layouts for computer chips, and to optimizing workflow in a workplace [11].

Many practical applications require a combination of packing algorithms to produce a useful solution. For example, a company that creates many products and ships them might have to a) pick a subset of product to be included in a specific shipment, b) pack products onto wooden skids while respecting weight restrictions and product fragility and minimizing wasted space on the skid, c) pack wooden skids of various sizes into metal shipping containers while maximizing the number of skids that are placed in a container, and (d) pack metal shipping

containers onto large shipping vessels while minimizing the number of vessels needed to ship all containers.

There is a quantifiable difference between good and bad solutions in the industrial environment. Wasted space during storage and transport costs companies resources: time might be spent re-packing containers, additional containers might need to be shipped, or product might be lost if weight restrictions are not satisfied. Many companies still use trial-and-error approaches while packing items for storage and transport [4]. Improved algorithms for packing gates onto microchips and scheduling processes among processors lead to better performing computers [30]; better algorithms for scheduling tasks to employees or to machines allows an optimized workflow in factories [54]; and efficient algorithms for cutting units out of stock material creates less wasted materials[55].

Efficient algorithms for geometric packing problems translate into improved industrial practices that reduce company expenses and improve product delivery. Furthermore, algorithm design techniques specifically developed for geometric packing problems contribute to the design of efficient algorithms for other types of optimization problems. As research into packing problems expands, more industrial problems will be solved using these algorithms and more packing and optimization problems will be able to leverage the algorithmic techniques that are discovered.

1.3 Related Work

In [47] an approximation algorithm is presented for HMSPP using *linear program rounding* which they prove can be done in polynomial time. They introduce the concepts of a fractional packing, configurations, parititioning a packing into common and uncommon portions, and rounding up fractional rectangles. As these are concepts that we use in our algorithm, we will explain each of these in detail in Chapters 3 and 4.

Given an instance I of HMSPP we denote with $OPT(I)$ the height of an optimal solution for instance I, and we denote with $SOL(I)$ the height of the solution output by some algorithm for the same instance I; the algorithm that $SOL(I)$ refers to will be clear from the context. In [47] an algorithm for HMSPP is presented for the case when $k = 2$ for which $SOL(I) \leq OPT(I) + 1$ and a second algorithm is also given that for any fixed value k it computes a solution of value $SOL(I) \leq OPT(I) + (k - 1)$.

1.4 Our Contributions

The algorithms that we present in this thesis divide a fractional packing into vertical sections and examine one vertical section at a time from left to right. Depending on the sizes of the fractional rectangles in each vertical section, we use different rounding techniques to transform fractional rectangles into whole rectangles. This approach is explained in more detail in Chapters 4 and 5.

In this thesis we present an algorithm designed in collaboration with Yu and Solis-Oba for HMSPP when $k = 3$ for which $SOL(I) \leq OPT(I) + \frac{5}{3}$. This algorithm can be generalized

for any fixed value k to get $SOL(I) \leq OPT(I) + k - \frac{4}{3}$. Note that this algorithm has a better approximation ratio than the algorithm in [47] for all $k > 2$.

Additionally, in this thesis we present a new algorithm for HMSPP when $k = 4$ for which $SOL(I) \leq OPT(I) + \frac{5}{2}$. This algorithm can be generalized for any fixed value k to get $SOL(I) \leq OPT(I) + k - \frac{3}{2}$.

Chapter 2

Related Problems

Packing problems come in many shapes and sizes, and small variations in the definition of a packing problem can produce many seemingly similar but many times very different problems. In this chapter we describe several classic packing problems: the bin packing problem, the cutting stock problem, the rectangle packing problem, and the strip packing problem.

The rectangle packing and strip packing problems are very similar to HMSPP; each of these problems involves packing rectangles into a rectangular container. The bin packing and cutting stock problems are widely known, and the relationship between them is similar to the relationship between the strip packing problem and HMSPP; in each case, the latter problem is a high multiplicity version of the former. We describe the term high multiplicity in more detail later on in this chapter.

2.1 The Bin Packing Problem

One of the earliest packing problems considered in the literature is the *one-dimensional bin packing problem*. In this problem the goal is to pack a set $A = \{a_1, a_2, ..., a_n\}$ of n items, each of size $0 < size(a_i) \leq 1$, into the smallest possible number of unit capacity bins [37]. The one-dimensional bin packing problem is NP-hard [19].

A practical application of this problem encodes transport trucks as bins and products as items. In this model, the size of an item is the product's weight, and the capacity of a bin is the truck's carrying capacity. By minimizing the total number of bins needed to hold all the items, we also minimize the number of transport trucks needed to carry all of the product; therefore, algorithms that minimize the number of required bins can help a transport company minimize its delivery expenses.

The one-dimensional bin packing problem is also theoretically significant: as one of the earlier NP-hard problems studied, many of the classical approaches that are used to evaluate the performance of approximation algorithms (approximation ratios and average-case behaviour) were first designed for algorithms for the one-dimensional bin packing problem [8].

Johnson et al. [37] presented a simple algorithm for this problem called *first-fit decreasing* (FFD). FFD first sorts the items $A = \{a_1, a_2, ..., a_n\}$ by non-increasing values of $size(a_i)$. Then, FFD repeatedly takes the largest un-packed item and packs it into the first bin where it fits. Johnson et al. proved that $FFD(I) \leq \frac{11}{9}OPT(I) + 2$, where $FFD(I)$ is the number of bins used

by the algorithm FFD. Other researchers have further analyzed this algorithm, and it has been proven that $FFD(I) \leq \frac{3}{2}OPT(I)$, which is the best possible approximation ratio for this problem unless $P = NP$ [50].

When the one-dimensional bin packing problem is considered in the asymptotic setting, which involves instances of the problem with very large optimal values, algorithms have been designed with better approximation ratios. In 2007, Dósa [10] proved that $FFD(I) \leq \frac{11}{9}OPT(I)+\frac{6}{9}$. In 1980, Yao [56] presented the *refined first-fit decreasing* algorithm, that produces solutions of value $RFFD(I) \leq \frac{11}{9}OPT(I) - 10^{-7}$. In 1985, Garey and Johnson [38] presented the *modified first-fit decreasing* algorithm, with asymptotic approximation ratio $\frac{71}{60}$. In 1991, Friesen and Langson [18] presented an algorithm combining the *best two-fit* algorithm and the *first fit decreasing* algorithm, with asymptotic approximation ratio that matches that of the modified first-fit descreasing algorithm. Each of these algorithms attempts to enhance FFD by identifying bins containing large items and pairing these large items with small items to fill each bin as much as possible.

In 1981, Fernandez de la Vega and Lueker [53] presented an asymptotic PTAS where $SOL(I) \leq (1 + \epsilon)OPT(I)$ for any $\epsilon > 0$. In 1982, Karmarkar and Karp [40] presented an algorithm that computes solutions of values at most $OPT(I) + O(\log^2 OPT(I))$. In 2013, Rothvoß [48] presented an algorithm that computes solutions of values at most $OPT(I) + O(\log OPT(I) * \log\log OPT(I))$.

2.2 The Cutting Stock Problem

In the *one-dimensional cutting stock problem* we are given a set $A = \{A_1, A_2, ..., A_d\}$ of d item types and a set $N = \{n_1, n_2, ..., n_d\}$ of d item multiplicities. All n_i items of type A_i have size $0 < size(A_i) \leq 1$. In this problem the goal is to pack the items into the smallest number of unit capacity bins. The cutting stock problem is NP-hard [20].

This problem can model many industrial challenges of cutting raw materials into smaller units for customers. For example, a business might stock standard length pieces of wood, but customers might request non-standard lengths of wood. The business wants to minimize the cost of cutting their standard material to the requested size. A solution to the cutting stock problem is equivalent to selecting the minimum number of standard-length materials needed to fulfill the customers orders.

Note that the one-dimensional cutting stock problem is the one-dimensional bin packing problem where the input objects are partitioned into d types. However, there is a significant difference between the cutting stock problem and the bin packing problem; while the input to the bin packing problem consists of a vector of n item sizes, the input to the cutting stock problem consists of a vector of d item sizes and a vector of d item multiplicities. Thus, even when the value of n is very large, if the value of d is small the size of the input for the cutting stock problem is small (see Figure 2.1).

Recall that an algorithm is efficient if its time complexity is polynomial in its input size. Algorithms for the bin packing problem that consider individually each item would not run in polynomial time if applied to the cutting stock problem. Bin packing algorithms are given n items as input, so considering each item individually incurs a number of operations that depends on the value of n, the size of the input; however, cutting stock algorithms are given d

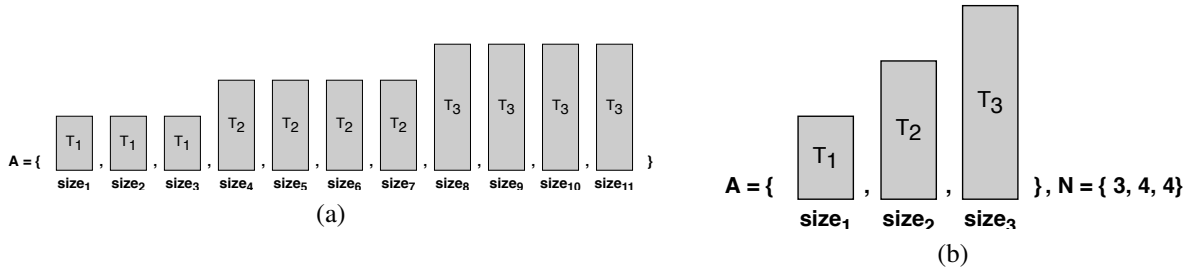


Figure 2.1: a) In a bin packing problem n sizes of items must be given in the input. b) In a cutting stock problem only d sizes of items and d multiplicites must be given in the input. Even when the value of n is very large, if the value of d is small the size of the input for the cutting stock problem is small.

item types as input, so when the total number of items is larger than d, considering each item individually incurs a number of operations that depends on the value of n, but a polynomial time cutting stock algorithm must incur a number of operations that polynomially depends on the value of d, the size of the input. When n is much larger than d, the time complexities of bin packing algorithms applied to the cutting stock problem can be exponential in their input sizes. Polynomial time algorithms for the cutting stock problem must consider each object type and not individual objects.

The cutting stock problem first appeared under the name of the *trim problem* in 1957 [12] and some of the first work on solving this problem was done by Gilmore and Gomory [21, 22, 23], who formulated this problem as an integer program. In 2005, Filippi and Agnetis [14] presented a polynomial-time algorithm that computes solutions of values at most $OPT(I) + (d - 2)$ bins, where d is a fixed number of object types. Note that when there are only 2 object types, this algorithm solves the cutting stock problem exactly. In 2007, Filippi and Agnetis [13] improved their algorithm to use at most $OPT(I) + 1$ bins for the case when $2 < d \leq 6$ and at most $OPT(I) + 1 + \lfloor \frac{d-1}{3} \rfloor$ bins for the case when $d > 6$. In 2010, Jansen and Solis-Oba [34] presented a polynomial-time algorithm that computes solutions of values at most $OPT(I) + 1$, for any fixed value of d. In 2013, Goemans and Rothvoß [24] presented a polynomial-time algorithm that solves the cutting stock problem exactly for any fixed value of d.

In 1982, Karmarkar and Karp [40] presented an algorithm that computes solutions of values at most $OPT(I) + O(\log^2 d)$, for arbitrary values of d. In 2013, Rothvoß [48] presented an algorithm that computes solutions of values at most $OPT(I) + O(\log d * \log\log d)$, for arbitrary values of d. Similarly to the bin packing problem, no approximation algorithm for the cutting stock problem can have an approximation ratio smaller than $\frac{3}{2}$ unless $P = NP$ [34].

2.3 The Rectangle Packing Problem

In the *two-dimensional rectangle packing problem*, we are given a set $R = \{r_1, r_2, ..., r_n\}$ of n rectangles, where each rectangle r_i has width w_i and height h_i, and a rectangular container of width W and height H. The goal is to determine the maximum subset of rectangles from R that can be packed within the rectangular container, without rotating or overlapping any of the rectangles [3]. The two-dimensional rectangle packing problem is NP-hard [2].

This problem is very useful in modeling the problem of cutting patterns out of raw material. Consider a rectangular piece of fabric of width W and height H, and a set $R = \{r_1, r_2, ..., r_n\}$ of n rectangular cutting patterns, where each cutting pattern r_i has width w_i and height h_i. Solving the rectangle packing problem is equivalent to maximizing the number of patterns that can be cut from the rectangular piece of fabric, which in turn leads to a cost-effective strategy of cutting the fabric.

Similarly to the bin packing problem, algorithms such as FFD can be modified to work on the two-dimensional rectangle packing problem. When FFD is applied to the two-dimensional rectangle packing problem, it is considered to be a *level oriented* algorithm. In a level oriented algorithm, the bottom of the rectangular container is considered the first level of the packing. Rectangles are packed in the first level until a rectangle r is found that is too wide to be packed in the remaining space at the bottom of the container. The bottom of the second level of the packing is defined by a horizontal line drawn from the top of the tallest rectangle packed in the first level. Subsequent levels are defined in the same way. When FFD is applied to the two-dimensional rectangle packing problem, it has an approximation ratio of 1.7 [7].

In 1983, Baker et al. [2] presented an algorithm that computes solutions of values at most $\frac{4}{3}OPT(I)$ for packing unit-weight squares into a rectangle. In 2004, Caprara and Monaci [6] considered the problem when rectangles have profits and the goal is to maximize the profit of the rectangles packed in the rectangular container; they presented an algorithm that computes solutions of values at most $(3 + \epsilon)OPT(I)$ for any $\epsilon > 0$. In 2007, Jansen and Zhang [36] improved this result by presenting an algorithm that computes solutions of values at most $(2 + \epsilon)OPT(I)$ for any $\epsilon > 0$. In 2009, Harren [27] presented an algorithm for packing squares with profits into a rectangular container that computes solutions of values at most $(\frac{5}{4}+\epsilon)OPT(I)$ for any $\epsilon > 0$. In 2008, Jansen and Solis-Oba [33] presented a PTAS for packing squares with profits that computes solutions of values at most $(1 + \epsilon)OPT(I)$ for any $\epsilon > 0$.

In 2005, Fishkin et al. [16] considered two special cases of the problem of packing a set of rectangles with profits into a unit size square frame. They presented a PTAS for the case when each square has profit equal to its area. Additionally, they presented a PTAS for the square packing problem with augmentation [15], in which the size of the container can be increased by a small factor $\epsilon > 0$. In 2012, Lan et al. [44] presented an algorithm for the case when the rectangles have side lengths at most $\frac{1}{k}$, where $k \geq 1$ is an integer, that computes solutions of values at most $\frac{k^2+3k+2}{k^2}OPT(I)$. Additionally, Lan et al. gave a PTAS for the square packing problem without profits.

2.4 The Strip Packing Problem

In the *two-dimensional strip packing problem* we are given a set $R = \{r_1, r_2, ..., r_n\}$ of n rectangles, where each rectangle r_i has width w_i and height h_i, and a rectangular strip with a fixed width and infinite height. The goal is to pack all of the rectangles within the strip while minimizing the total height of the packing [3]. Rectangles cannot be rotated and rectangles cannot overlap with other rectangles. Note that this problem is different than the two-dimensional rectangle packing problem; in the rectangle packing problem, the input rectangles are not all guaranteed to fit within the container, but in the strip packing problem, all of the input rectangles must fit within the container and the total height of the packing must be minimized. The

two-dimensional strip packing problem is NP-hard [19].

A simple application of the two-dimensional strip packing problem involves scheduling computer processes to be run on a multi-core computer. Computer processes that require different amounts of time can be encoded as rectangles that have different heights. Processes that require a single computer core can be encoded as rectangles of width 1 and processes that require multiple cores can be encoded as rectangles of width equal to the number of cores. The width of the strip represents the number of computer cores available. Therefore, finding an optimal solution for the two-dimensional strip packing problem is equivalent to scheduling computer processes to CPU cores to minimize the total time needed to complete every process.

Baker et al. [3] designed a class of algorithms for the two-dimensional strip packing problem called *bottom-up left-justified* algorithms (BL). BL algorithms pack rectangles one-by-one into the lowest possible location in the packing where they fit, and then rectangles are left-justified within the packing. Baker et al. proved that this approach achieves a 3-approximation algorithm when the rectangles are sorted by decreasing width.

Coffman et al. [9] presented an 2.7-approximation algorithm, Sleator [51] designed a 2.5-approximation algorithm, Schiermeyer [49] and Steinberg [52] independently presented 2-approximation algorithms, and Harren and van Stee [29] gave a 1.9396-approximation algorithm. Harren et al. [28] presented the best known approximation algorithm for this problem in 2014 with an approximation ratio of $\frac{5}{3} + \epsilon$. There is no approximation algorithm for the two-dimensional strip packing problem with an approximation ratio better than $\frac{3}{2}$ unless $P = NP$ [28].

When the two-dimensional strip packing problem is considered in the *asymptotic* setting, Golan [25] proved that the BL algorithm has asymptotic approximation ratio $\frac{4}{3}$, and Baker et al. [1] proved that it has asymptotic approximation ratio $\frac{5}{4}$. Kenyon and Rémila [42] gave an FPTAS for the problem where $SOL(I) \leq (1 + \epsilon)OPT(I) + O(\frac{1}{\epsilon^2})$ for any $\epsilon > 0$ and Jansen and Solis-Oba [32] gave a PTAS for the problem where $SOL(I) \leq (1 + \epsilon)OPT(I) + 1$ for any $\epsilon > 0$.

2.5 High Multiplicity Problems

Practical problems that can be solved using packing algorithms often include a small number of different types of items. For example, a company might have a shipment containing only a few different products, but they ship these products in bulk. Therefore, a packing algorithm can group identical products together and pack the groups, instead of packing individual items.

As defined by Hochbaum and Shamir [31], a *high multiplicity* problem has its input "partitioned into relatively few groups (or types), and in each group all the [inputs] are identical." The number of items within a type is called the *multiplicity* of that type. Note that high multiplicity problems represent the input of a problem very compactly, as we only need a list of types and a list of multiplicities and not a list of the individual items in the input. For example, consider the difference between the bin packing and cutting stock problems described above: the bin packing problem takes as input a list of n items, while the cutting stock problem takes as input a list of d types and a list of d multiplicities. Ideally, an algorithm for a high multiplicity problem takes advantage of the fact that there are many items with identical dimensions to find a better solution than a general algorithm that considers every item to be unique.

Each of the problems listed above has high multiplicity variants. In general, it seems that

approximation algorithms for high multiplicity problems produce solutions closer to the optimal ones than approximation algorithms for their respective general problems.

In 2014, Goemans and Rothvoß [24] presented an algorithm for solving the one-dimensional cutting stock problem in polynomial time for a constant number of item types. Recall that bin packing can also be considered as a scheduling problem where the processing times correspond to the item sizes. Goemans and Rothvoß also solved the high multiplicity variant of minimizing the makespan for unrelated machines with machine-dependent release dates for a fixed number of job types and machine types.

In 2001 McCormick et al. [45] presented a polynomial-time algorithm to solve the *multiprocessor scheduling problem with* 2 *job lengths*. In this problem, each job has one of two possible processing times. In 2018, Fitzsimmons and Hemaspaandra [17] proved that the multiprocessor scheduling problem with C job lengths is solvable in polynomial time for any fixed value of C.

Chapter 3

High Multiplicity Strip Packing

Given k distinct rectangle types, where each rectangle type T_i has n_i rectangles with width $0 < w_i \leq 1$ and height $0 < h_i \leq 1$, we wish to pack all of the rectangles into a strip of width 1 in such a way that the height of the packing is minimized (see Figure 1.1). The height of the packing is measured from the top of the topmost rectangle to the base of the strip.

Rectangles must be packed such that they are fully contained within the strip, rectangles cannot overlap with other rectangles, and rectangles cannot be rotated. The rectangles are *oriented*; that is, each rectangle has a top, bottom, left, and right side. When packing a rectangle, the bottom side must be parallel to the base of the strip. This restriction applies to practical scenarios such as cutting wood with or against the grain.

3.1 Linear Program Rounding

A useful approach for designing approximation algorithms for NP-hard problems uses a strategy called *linear program rounding* (LP-rounding). To understand LP-rounding, we must first introduce the notions of linear programs and integer programs.

A *linear program* is an encoding of a mathematical model, and can be expressed in the general form:

$$\begin{array}{ll} \text{maximize} & c^T x \\ \text{subject to} & Ax \leq b, \\ \text{and} & x \geq 0 \end{array}$$

where x is a vector of variables, c and b are vectors of coefficients, and A is a matrix of coefficients. The expression $c^T x$ is known as the objective function. We can use a linear program to solve minimization problems by negating the objective function. The inequalities $Ax \leq b$ and $x \geq 0$ are constraints on the mathematical model.

The set of solutions that satisfy a linear program's constraints, called *feasible solutions*, form an *n-polytope*, which is an n-dimensional geometric shape where n is the number of variables in the linear program. This n-polytope is a *convex set* of points, which means that a line segment joining any two points from the set is completely contained within the n-polytope. The problem of solving a linear program is equivalent to finding a point within the n-polytope that optimizes the objective function.

Solutions to the linear program that correspond to vertices of the n-polytope are known as *basic feasible solutions*. If a linear program has an optimal solution, then it has at least one basic feasible solution that optimizes the objective function; this is important, as basic feasible solutions have a valuable property: in any basic feasible solution, the number of nonzero variables is at most the lesser of the number of constraints and the number of variables [39]. This property can be used to bound the number of non-zero variables in a solution of a linear program.

An *integer program* is a linear program where all the variables are restricted to be integers. The problem of finding optimal solutions for integer programs is NP-hard [41]; however, solutions for linear programs can be computed in polynomial time [43].

Now we can describe the process of LP-rounding:

1. Formulate an NP-hard problem that we wish to solve as an integer program and then relax the integrality constraints of the integer program to obtain a linear program
2. Solve the linear program
3. Transform the solution of the linear program into a feasible solution for the NP-hard problem by rounding the values of non-integral variables to integer values

3.2 Rounding a Solution for the Linear Program for HMSPP

Consider the version of the strip packing problem where rectangles can be sliced horizontally: this is the *fractional strip packing problem* (FSPP). Since rectangles in FSPP can be cut into smaller pieces called *fractional rectangles*, which are rectangles that do not have the full height of a whole rectangle of its type, these fractional rectangles can be packed into regions of the strip that otherwise might have been left empty if slicing rectangles were not allowed (see Figure 3.1). Note that we shade rectangles in a darker color either when they are fractional rectangles or when those rectangles will be transformed.

A *configuration* C_j consists of a group of rectangles packed side-by-side whose total width is at most 1. Rectangles can be packed one-on-top of another within a configuration from the base of the configuration until they reach the configuration's height. Any horizontal line parallel to the base of the packing drawn through the configuration intersects the same multiset of rectangle types (see Figure 3.2). In other words, within a configuration, two rectangles that are packed one-on-top-of-another must be of the same rectangle type, but two rectangles that are packed side-by-side can be of different rectangle types. The fractional packing in Figure 3.1 consists of two configurations: C_1 contains rectangles of types T_2 and C_2 contains rectangles of types T_1 and T_3.

A fractional packing consists of a finite number of configurations stacked one on top of the other (see Figure 3.3). Within a packing, a configuration C_j is only used once; for example, you would not find a packing consisting of C_j, C_{j+1}, and C_j again. We index the configurations such that the top configuration is C_1, the configuration beneath C_1 is C_2, and so on (see Figure 3.3). The total height of a packing is equal to the sum of the heights of its configurations.

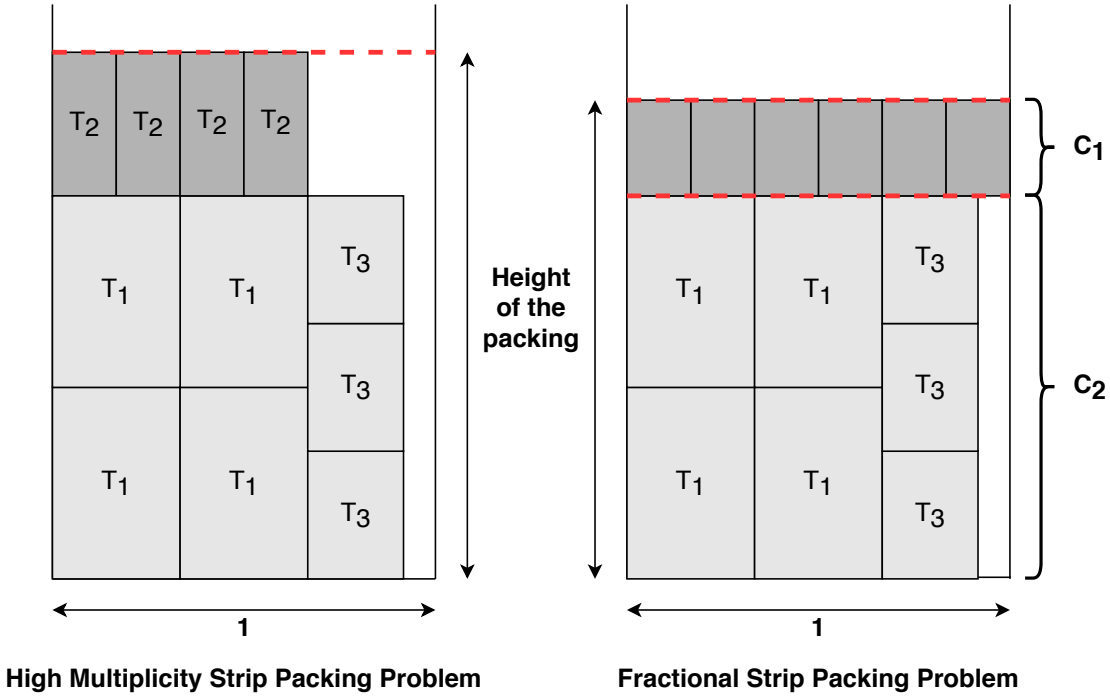


Figure 3.1: An instance of HMSPP (left) compared to an instance of FSPP (right). Rectangles in FSPP might be sliced horizontally to form smaller pieces; a solution to FSPP might have a lower height due to packing fractional pieces in regions where whole rectangles cannot fit. The darker shaded rectangles from the left have been sliced horizontally into the darker shaded smaller fractional pieces on the right.

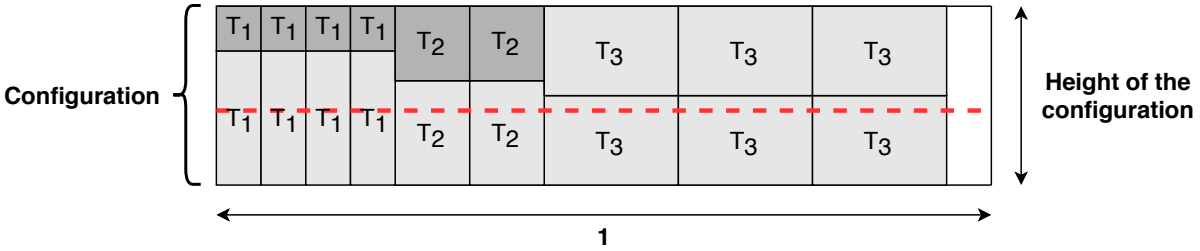


Figure 3.2: A *configuration* is a horizontal strip of a packing where any horizontal line (red dashed line) parallel to the base of the packing drawn through the configuration intersects the same multiset of rectangle types. The fractional rectangles are shaded in a darker color.

A linear program formulation for FSPP can be obtained in the following manner:

- Let a solution to FSPP be expressed using a vector x with J dimensions, where J is the set of all possible configurations. Each vector coordinate x_j represents the height of configuration C_j. Configurations that are used in a solution to FSPP should have a positive height and configurations that are not used in a solution to FSPP should have a height of 0.
- Let n_i be the total number of rectangles of type T_i and let h_i be the height of each rectangle of type T_i.
- Let $n_{i,j}$ be the number of rectangles of type T_i in configuration C_j.

Figure 3.3: The configurations are stacked one on top of the other to form a packing. The vector x represents the height of each configuration as determined by the solution of the linear program.

The following linear program defines FSPP:

$$\begin{aligned} &\text{Minimize} \sum_{j \in J} x_j \\ &\text{Subject to:} \sum_{j \in J} x_j n_{i,j} \geq n_i h_i, \text{ for each rectangle type } i \\ &\qquad x_j \geq 0, \text{ for each } j \in J \end{aligned} \tag{3.1}$$

where the objective function aims to minimize the total height of the packing, the first constraint ensures that all input rectangles are included in the fractional packing, and the second constraint ensures that no configurations have negative height.

A solution to linear program (3.1) is stored in the vector x. Each coordinate x_j with a positive value indicates that configuration C_j is used in the solution of FSPP (see Figure 3.3). The value of x_j is the height of configuration C_j. Therefore, the sum of x's coordinates is the total height of the fractional packing.

In [47] it is proven that a basic feasible solution to this linear program uses at most k configurations, i.e., at most k of the variables x_j have positive value, and it can be found in polynomial time. The process involves using the Grötschel-Lovász-Schrijver [26] algorithm to find an optimal solution to the dual linear program corresponding to (3.1) and using the

algorithm of Karmarkar and Karp [40] to transform this solution into an optimal basic feasible solution for (3.1). The basic feasible solution obtained from this process is used as the input for our approximation algorithms described below.

3.3 A Simple Approximation Algorithm for HMSPP

Transforming a basic feasible solution S of FSPP into a feasible solution for HMSPP requires transforming fractional rectangles, rectangles that do not have the full width or the full height of whole rectangles of their same types, into whole ones. Perhaps the simplest way to do this is to replace each fractional rectangle by a whole rectangle of its type, shifting rectangles upwards in the packing to make space as needed (see Figure 3.4). Clearly, after this transformation, all rectangles must be whole. Since the height of any rectangle is at most 1, there is at most one layer of fractional rectangles in each configuration, and there are at most k configurations in S, the total height of the packing produced in this manner is at most k plus the height of the fractional packing produced by solving the linear program.

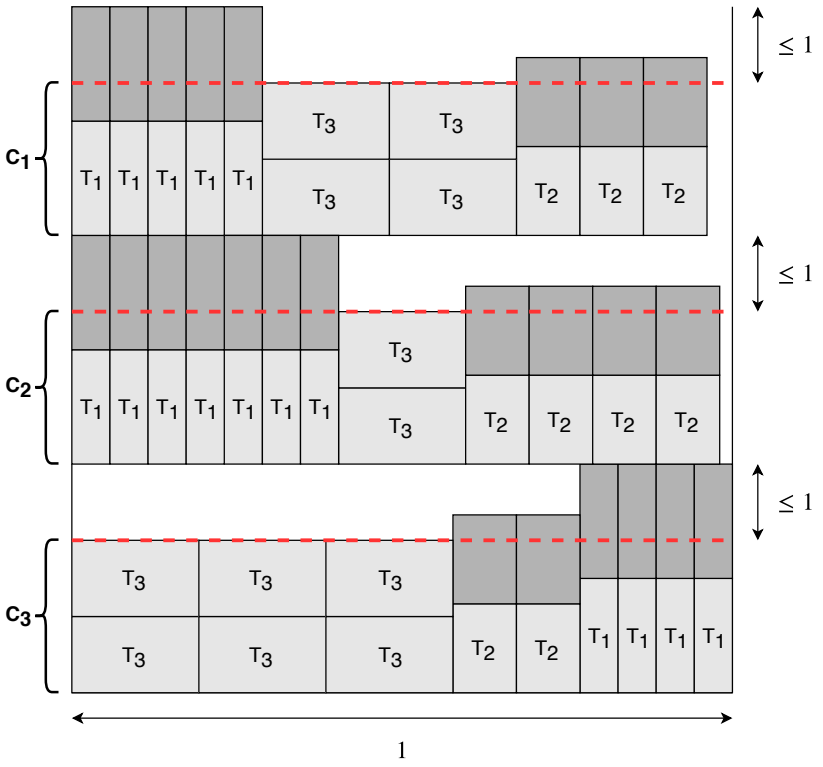


Figure 3.4: A simple approximation algorithm for HSMPP replaces each fractional rectangle by a whole rectangle of the corresponding type, shifting rectangles upwards in the packing to make space as needed.

Note that the height of an optimal solution for FSPP is a lower bound on the height of an optimal solution for HMSPP; a solution using whole rectangles cannot achieve a lower total height than a solution using fractional rectangles, as those fractional rectangles can potentially be packed in regions where whole rectangles would not fit. We use the height of an optimal

fractional packing as a lower bound for the height of an optimal solution for HMSPP when evaluating the performance of our approximation algorithms.

The simple approximation algorithm described above produces a solution where $SOL(I) \leq OPT(I) + k$. Note that the above algorithm which transforms fractional rectangles into whole rectangles runs in polynomial time: for each rectangle type, fractional rectangles have their fractions changed to 1. As proven in [47], solving the linear program (3.1) can be done in polynomial time; therefore, this approximation algorithm runs in polynomial time.

However, algorithms with better approximation ratios can be designed for HMSPP; intuition suggests that while replacing fractional rectangles with whole rectangles is a good strategy when the size of a fractional rectangle is almost the size of a whole rectangle of its type, replacing small fractional rectangles with whole rectangles probably increases the height of the packing more than it needs to. We show two algorithms in the next two chapters that improve upon the simple algorithm described above.

Chapter 4

Strip Packing with Three Rectangle Types

4.1 Overview of the Algorithm

As discussed in Chapter 3, when $k = 3$ there can be either three configurations, two configurations, or only one configuration in the solution obtained from the linear program. Our algorithm must correctly round all these possible cases to integer solutions in polynomial time. In this section we assume that there are three configurations in the fractional solution. We consider the cases when the fractional solution has two configurations and one configuration in Sections 4.3 and 4.4.

The three configurations are packed one on top of the other as shown in Figure 4.1; fractional rectangles are shaded in a darker color. Let the configuration packed at the top be called C_{Top}, the configuration packed in the middle be called C_{Mid}, and the configuration packed at the bottom be called C_{Bot}. Note that we can change the order of the configurations if necessary.

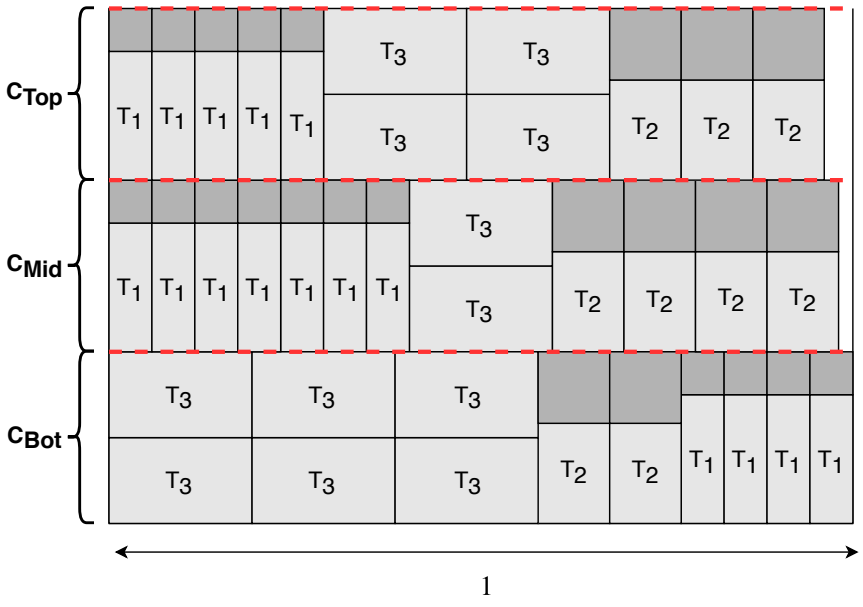


Figure 4.1: The three configurations are packed one on top of the other. Fractional rectangles are shaded in a darker color.

4.1.1 The Common Portion of the Packing

Rectangles can be rearranged horizontally within the configurations so that rectangles of the same type appearing in all three configurations are packed together; these rectangles appear to the left of the packing, and we label this section of the packing the *common* portion of the packing. In the common portion of the packing, the fractional rectangles in C_{Bot} can be combined with rectangles in C_{Mid} of the same type to form either larger fractional rectangles or whole rectangles, and the same can be done for the fractional rectangles in C_{Mid} and the rectangles in C_{Top} of the same type.

In Figure 4.2 a solid vertical line has been drawn between the common portion and the uncommon portion. Note that in the common portion of the packing, fractional rectangles must be the topmost rectangles in C_{Top}, as fractional rectangles lower in the packing have bee combined together (see Figure 4.2).

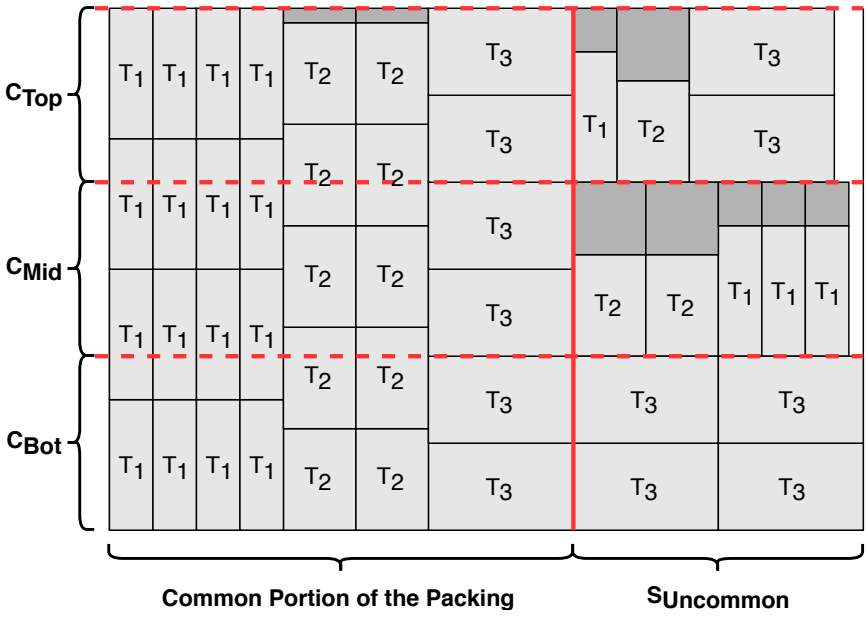


Figure 4.2: Common and uncommon portions of the packing.

Recall that the height of any rectangle r is at most 1; therefore, we can replace the fractional rectangles in the common portion of the packing with whole rectangles of the same type (see Figure 4.3), similar to the process used in the simple algorithm presented in Chapter 3. This process increases the height of the common portion of the packing by at most 1 (see Figure 4.4).

Corollary 4.1.1. *If all rectangles are packed in the common portion of the packing, then there will be no leftover fractional rectangles after processing the common portion of the packing as described above. Therefore, in this case our algorithm produces an integer packing of height at most* 1 *plus the height of the fractional packing produced by the solution of the linear program.*

Proof. All fractional rectangles in the common portion of the packing, which must be located in C_{Top}, are replaced by whole rectangles of the corresponding type. Therefore, there will be no leftover fractional rectangles. Recall that the height of any rectangle is at most 1; therefore, the

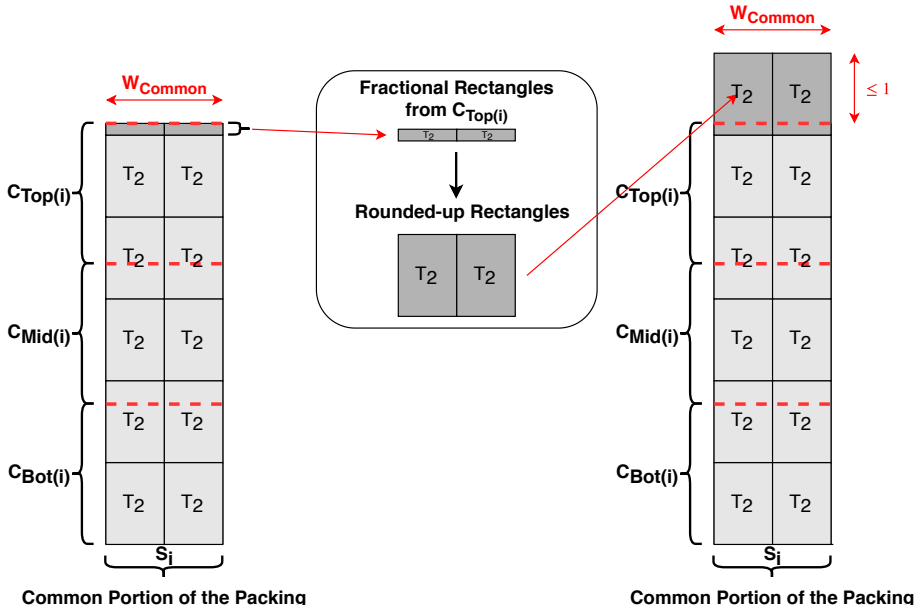


Figure 4.3: For every section $s_i \in S_{Common}$, the fractional rectangles in $R_{Top(i)}$ are replaced by whole rectangles.

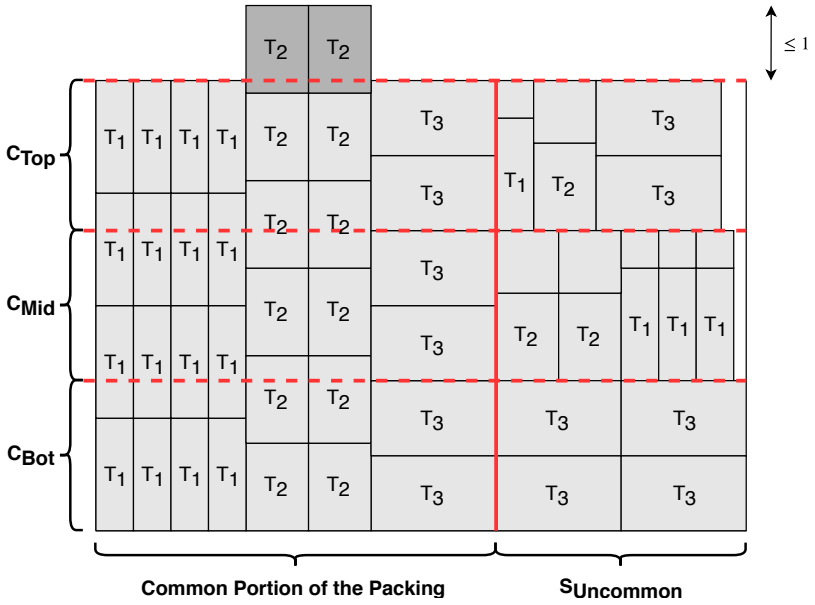


Figure 4.4: After the fractional rectangles in S_{Common} have been replaced by whole rectangles, the height of the packing has increased by at most 1. The newly packed whole rectangles are shaded.

total increase in height from replacing the fractional rectangles in C_{Top} with whole rectangles is at most 1. □

Note that the common portion of the packing can be rounded independently of the uncommon portion of the packing, because there is no rectangle r that spans both the common portion of the packing and the uncommon portion of the packing. For the rest of this chapter,

we discuss how to round the uncommon portion of the packing.

4.1.2 The Uncommon Portion of the Packing

After horizontally rearranging rectangles in each configuration to create the common portion of the packing, the *uncommon* portion of the packing does not contain a single rectangle type that appears in all three configurations (see Figure 4.2). Therefore, fractional pieces at the top of C_{Bot} might not be of the same type as the rectangles at the bottom of C_{Mid}.

The algorithm in this chapter describes how to transform fractional rectangles located in the uncommon portion of the packing into whole rectangles.

4.1.3 Sorting the Configurations

Within a configuration C at most k distinct rectangle types are packed. Let f_i represent the fraction of each rectangle of type T_i that appears at the top of C. Fraction f_i is calculated by dividing the height of a fractional rectangle of type T_i in C by the height of that rectangle type (see Figure 4.5a). Within each configuration, the rectangles in the uncommon portion of the packing will be packed left to right in non-decreasing value of f_i. Figure 4.5 shows how the rectangles are sorted in the configurations.

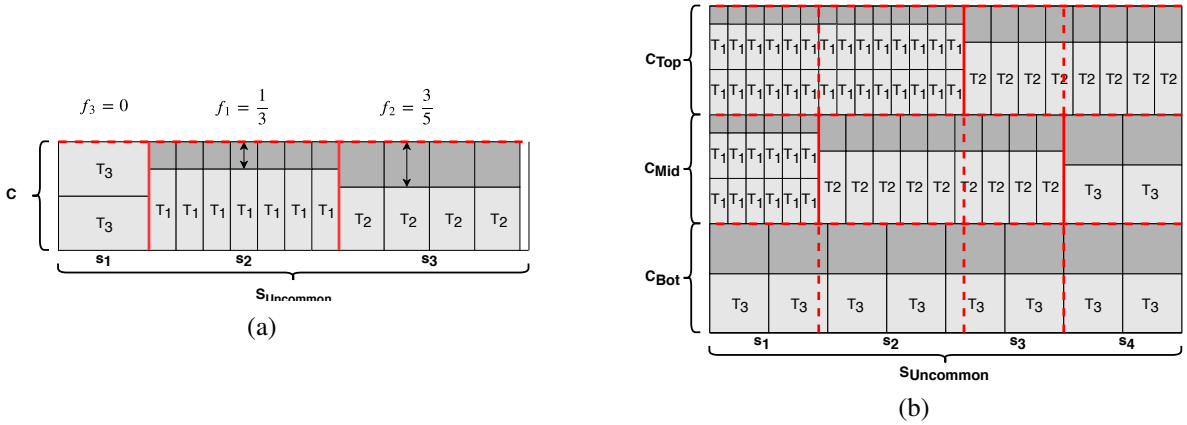


Figure 4.5: The rectangles in each configuration are sorted according to f_i.

For configurations that have more than one type of rectangle, draw a solid vertical line at the point where the rectangle type changes. This vertical line will extend through all of the other configurations and may divide rectangles in other configurations into two or more pieces. In the figures we draw a dashed vertical line to show the extension of a vertical division into the other configurations (see Figure 4.6). We say that a configuration creates a vertical division where it changes rectangle type. Vertical divisions separate the uncommon portion of the packing into multiple vertical sections. Sections in the uncommon portion of the packing are indexed from left to right starting at index 1 (see Figure 4.6).

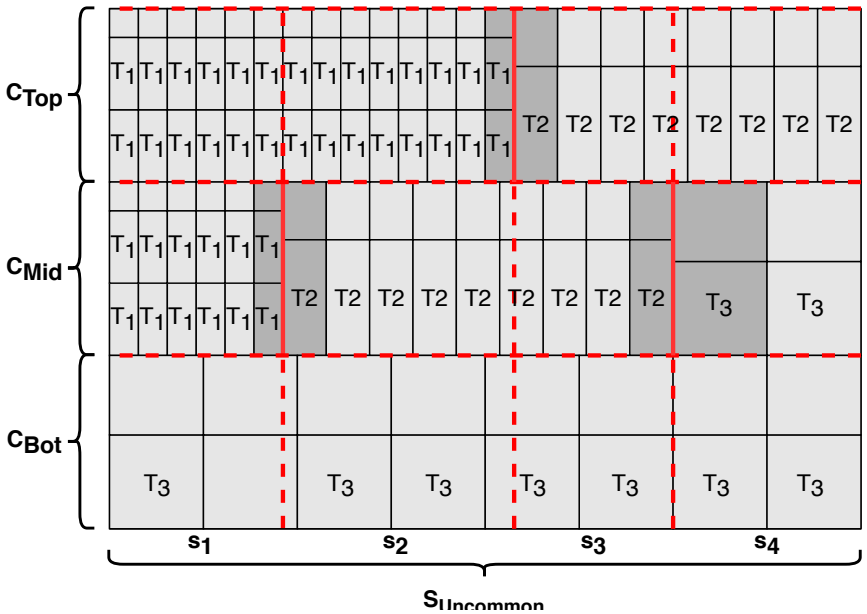


Figure 4.6: A vertical division is created between rectangles of different types within a configuration. The horizontally adjacent rectangles that are responsible for creating a vertical division are shaded.

4.1.4 Rounding Fractional Rectangles

To transform the fractional rectangles into whole rectangles we consider each vertical section in the uncommon portion of the packing one at a time from left to right. Within a section s_i, each configuration only has a single rectangle type. Let $f_{Top(i)}$, $f_{Mid(i)}$, and $f_{Bot(i)}$ represent the fraction of the rectangles packed in s_i at the top of C_{Top}, C_{Mid}, and C_{Bot}, respectively. Figure 4.7 shows fractions $f_{Top(4)}$, $f_{Mid(4)}$, and $f_{Bot(4)}$ within section s_4; note that we sometimes simplify the figures by not showing all of the rectangles in each configuration.

Algorithm 3TypeRounding, described below, transforms a fractional packing obtained by solving the linear program into an integer packing. This transformation increases the total height of the packing by at most $\frac{5}{3}$, as we show later. Since the fractional packing obtained by the GLS algorithm may have three configurations, two configurations, or only one configuration, we use different algorithms depending on the number of configurations in the fractional solution.

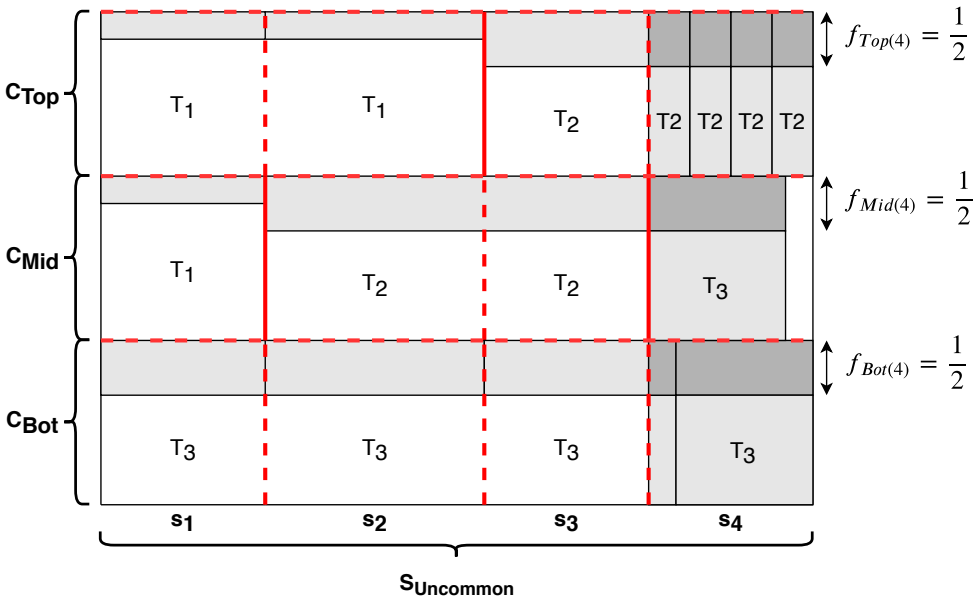


Figure 4.7: Fractions $f_{Top(4)}$, $f_{Mid(4)}$, and $f_{Bot(4)}$ are labeled for section s_4.

Algorithm 4.1 3TypeRounding(*FractionalSolution*)

Input: Optimal fractional packing *FractionalSolution*.
Output: The height of the integer packing h obtained from *FractionalSolution*.
Divide *FractionalSolution* into common and uncommon portions by horizontally rearranging rectangles as described in Section 4.2.1.
Round up the fractional rectangles of the common portion of the packing as described in Section 4.2.1.
for each configuration C in *FractionalSolution* **do**
    Sort the uncommon portion of C by non-decreasing value of f_i.
end for
if number of configurations in *FractionalSolution* = 3 **then**
    return *3ConfigurationRounding*(*FractionalSolution*).
else if number of configurations in *FractionalSolution* = 2 **then**
    return *2ConfigurationRounding*(*FractionalSolution*).
else
    return 1*ConfigurationRounding*(*FractionalSolution*).
end if

4.2 Three Configurations

When the solution to the linear program has three configurations, our algorithm considers three cases, depending on the values of the three fractions $f_{Top(i)}$, $f_{Mid(i)}$, and $f_{Bot(i)}$ for each section s_i:

- **Case 1:** $f_{Top(i)} \leq \frac{1}{3}$, $f_{Mid(i)} \leq \frac{1}{3}$, and $f_{Bot(i)} \leq \frac{1}{3}$.

Let i be the smallest index for which $f_{Bot(i)} > \frac{1}{3}$, $f_{Top(i)} > \frac{1}{3}$, or $f_{Mid(i)} > \frac{1}{3}$. If such an index does not exist then Case 2 and Case 3 do not need to be considered; otherwise, we (re)-order the configurations so that $f_{Bot(i)} > \frac{1}{3}$ for all $j > i$. Now we can define Case 2 and Case 3:

- **Case 2:** $f_{Bot(i)} > \frac{1}{3}$ and $f_{Top(i)} + f_{Mid(i)} \leq 1$, and
- **Case 3:** $f_{Bot(i)} > \frac{1}{3}$ and $f_{Top(i)} + f_{Mid(i)} > 1$.

We first flip C_{Top} upside down as shown in Figure 4.8. The algorithm for rounding fractional rectangles into whole ones when there are three configurations is shown below.

Algorithm 4.2 3ConfigurationRounding(*FractionalSolution*)

Input: An optimal fractional packing *FractionalSolution* with three configurations C_{Top}, C_{Mid}, and C_{Bot}.
Output: The height of an integer packing h obtained from *FractionalSolution*.
Order the configurations as described above.
Flip C_{Top} upside down.
Initialize set S to contain all the vertical sections in the uncommon portion of *FractionalSolution*.
Initialize sets S_{Case1}, S_{Case2}, and S_{Case3} to be empty sets.
for $s_i \in S$ **do**
    if $f_{1(i)} \leq \frac{1}{3}$, $f_{2(i)} \leq \frac{1}{3}$, and $f_{3(i)} \leq \frac{1}{3}$ **then**
        $S_{Case1} = S_{Case1} \cup s_i$.
    else if $f_{1(i)} > \frac{1}{3}$ and $f_{2(i)} + f_{3(i)} \leq 1$ **then**
        $S_{Case2} = S_{Case2} \cup s_i$.
    else
        $S_{Case3} = S_{Case3} \cup s_i$.
    end if
end for
h_0 = height of *FractionalSolution*.
h_1 = height increase after processing S_{Case1} according to Section 4.2.3.
h_2 = height increase after processing S_{Case2} according to Section 4.2.4.
h_3 = height increase after processing S_{Case3} according to Section 4.2.5.
return $h_0 + \max(h_1, h_2, h_3)$.

Note that this algorithm only depends on the number of configurations and not on the number of rectangle types.

Theorem 4.2.1. *Algorithm 3ConfigurationRounding produces an integer packing of height at most $\frac{5}{3}$ plus the height of the fractional packing produced by the solution of the linear program.*

We prove this theorem in the Sections 4.2.3 - 4.2.5.

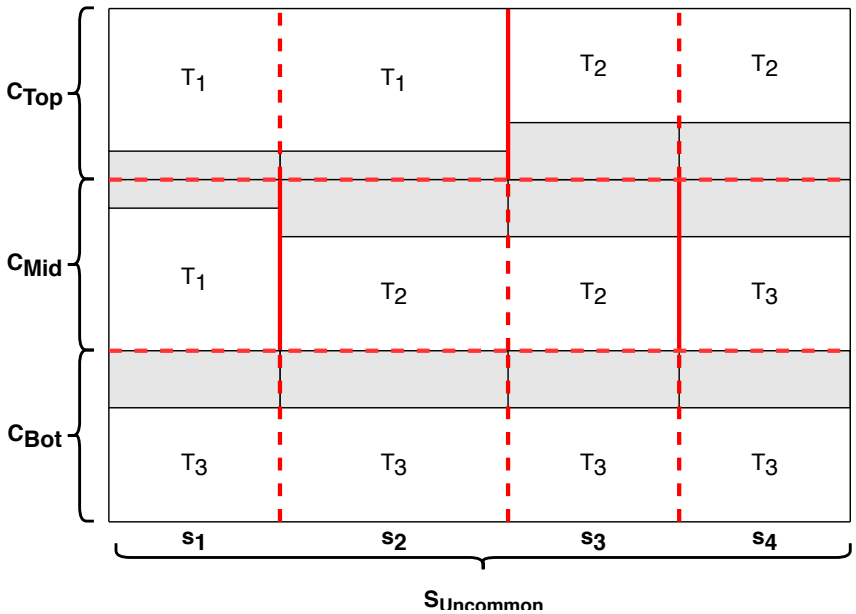


Figure 4.8: C_{Top} is flipped upside down. Recall that we sometimes simplify the figures by not showing all of the rectangles in each configuration.

4.2.1 Notation

Let S be the set of all vertical sections in the uncommon portion of the packing. We use S_{Case1}, S_{Case2}, and S_{Case3} to represent the sets of all sections classified as Case 1, Case 2, and Case 3, respectively (see Figure 4.9). Additionally, when we write "rectangle $r \in S_{Case1}$" we mean that rectangle r is packed in a section within S_{Case1}.

Within some section s_i, let $C_{Top(i)}$, $C_{Mid(i)}$, and $C_{Bot(i)}$ refer to the part of C_{Top}, C_{Mid}, and C_{Bot} that is located within s_i, respectively. Also, let $R_{Top(i)}$, $R_{Mid(i)}$, and $R_{Bot(i)}$ represent the fractional rectangles packed at the top of $C_{Top(i)}$, $C_{Mid(i)}$, and $C_{Bot(i)}$, respectively. When we write "R_{Top}", "R_{Mid}", and R_{Bot}", without including a section, we are referring to the fractional rectangles packed at the top of C_{Top}, C_{Mid}, and C_{Bot}, respectively.

A rectangle r might belong to two or more sections. We say, for example, that $r \in R_{Top} \cap S_{Case2} \cap S_{Case3}$ if r is a fractional rectangle packed at the top of C_{Top} and has been split by a vertical division that separates Case 2 and Case 3. Hereafter, we call such a rectangle a *vertically split* fractional rectangle. We use r_{Case2} to refer to the part of r that is packed within S_{Case2} and r_{Case3} to refer to the part of r that is packed within S_{Case3}. Figure 4.9 shows an example of this notation. A fractional rectangle $r \in R_{Top} \cap S_{Case2} \cap S_{Case3}$ has been highlighted in sections s_2 and s_3. Note that we actually cut rectangle r into the parts r_{Case2} and r_{Case3} so that the parts can be packed independently of each other.

4.2.2 Shifting Rectangles

Within a section s_i, we say that a rectangle r_1 is packed below a rectangle r_2 if the distance from the bottom of the packing to the top of r_1 is less than or equal to the distance from the bottom of the packing to the bottom of r_2. Similarly, we say that r_1 is packed above r_2 if the distance from

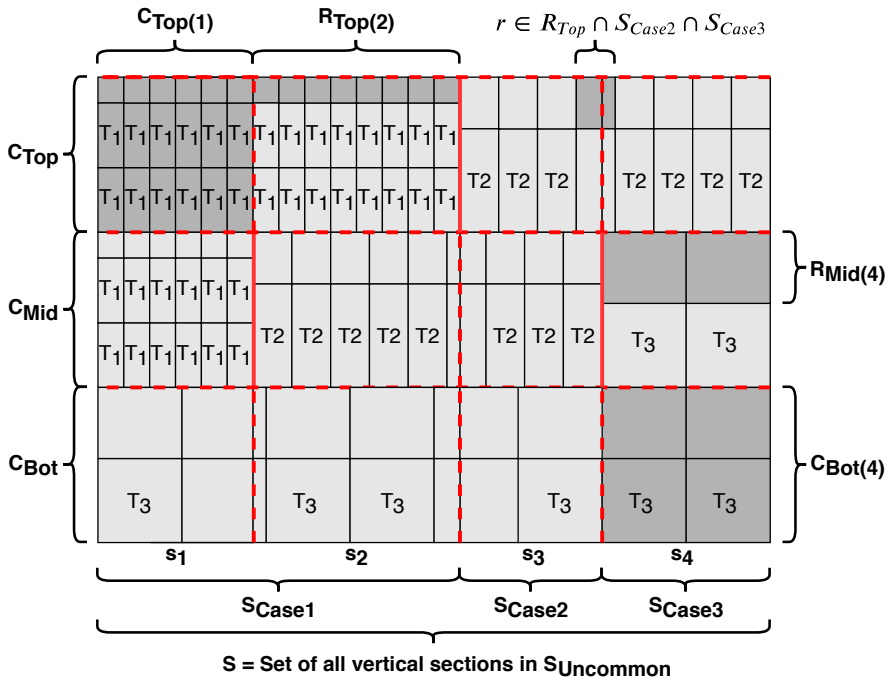
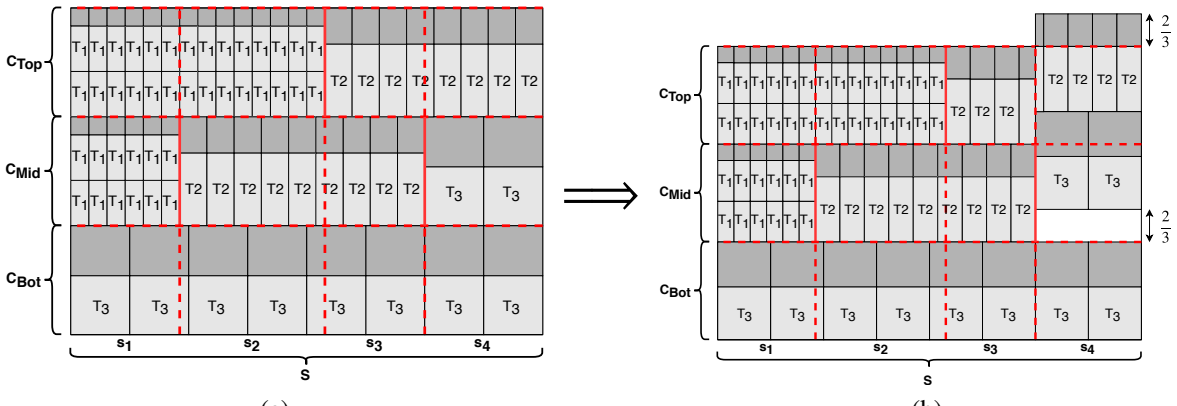


Figure 4.9: Notation used when referring to the uncommong portion of the packing.

the bottom of the packing to the bottom of r_1 is greater than or equal to the distance from the bottom of the packing to the top of r_2. We use these same definitions of "above" and "below" when referring to configurations.

Assume that for some section s_i we want to shift all the rectangles in $C_{Mid(i)}$ upwards by a distance of $\frac{2}{3}$. To do this we must move all the rectangles in $C_{Mid(i)}$ and in $C_{Top(i)}$ a distance of $\frac{2}{3}$ higher than their original locations. This leaves empty space of height $\frac{2}{3}$ between the top of $R_{Bot(i)}$ and the bottom of $C_{Mid(i)}$. Figure 4.10 shows how $C_{Mid(4)}$ is shifted upwards by a distance of $\frac{2}{3}$. Note that a vertically split rectangle must be cut along the vertical division so that its parts can move independently.



(a) (b)

Figure 4.10: $C_{Mid(4)}$ and $C_{Top(4)}$ are shifted upwards by a distance of $\frac{2}{3}$.

4.2.3 Case 1. $f_{Top(i)} \leq \frac{1}{3}$, $f_{Mid(i)} \leq \frac{1}{3}$, and $f_{Bot(i)} \leq \frac{1}{3}$

Let the width of S_{Case1} be W_1. For every section $s_i \in S_{Case1}$ we remove the fractional rectangles in $R_{Top(i)}$, $R_{Mid(i)}$, and $R_{Bot(i)}$, including the parts r_{Case1} for vertically split fractional rectangles r; re-shape them so that they have the full height of a rectangle of the same type but only a fraction of its width; and pack them side-by-side in a region of width W_1 and height 1. This region, hereafter referred to as C_A (see Figure 4.11), is created by shifting all rectangles in C_{Top} upwards, including rectangles in S_{Case2} and S_{Case3}, until there is empty space of height 1 between C_{Top} and C_{Mid} (see Figure 4.12). After shifting the rectangles the tops of the topmost rectangles in C_{Top} must lie on a common line.

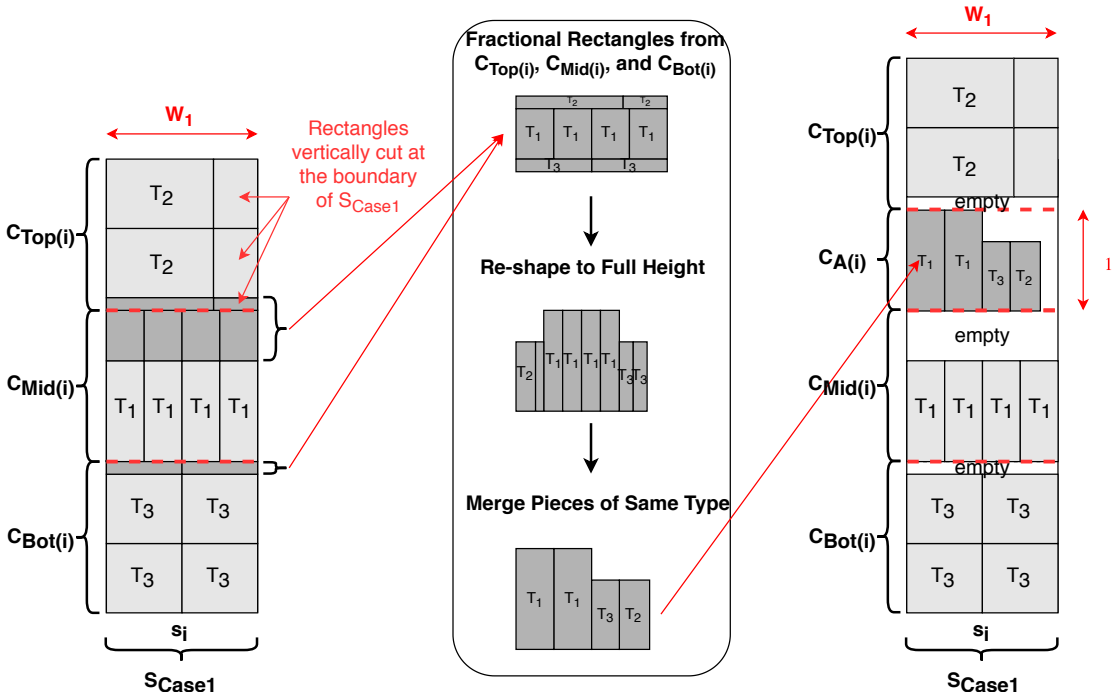


Figure 4.11: For every section $s_i \in S_{Case1}$ the fractional rectangles in $R_{Top(i)}$, $R_{Mid(i)}$, and $R_{Bot(i)}$ are removed, re-shaped, and packed side-by-side in C_A; a whole rectangle is formed whenever a sufficient number of fractional pieces of the same type have been packed in C_A. All rectangles in C_{Top} are shifted upwards until there is empty space of height 1 between C_{Top} and C_{Mid}.

Lemma 4.2.2. *Let section $s_i \in S_{Case1}$ have width W_i, and let $C_{A(i)}$ be the portion of C_A contained within s_i. The fractional rectangles in $R_{Top(i)}$, $R_{Mid(i)}$, and $R_{Bot(i)}$ can be packed in $C_{A(i)}$ using at most height 1 and width W_i.*

Proof. The total available area A_i in $C_{A(i)}$ is $A_i = W_i * 1 = W_i$. Since each of $C_{Top(i)}$, $C_{Mid(i)}$, and $C_{Bot(i)}$ has only one rectangle type, the total area a_i of $R_{Top(i)}$, $R_{Mid(i)}$, and $R_{Bot(i)}$ is

$$a_i \leq (W_i * f_{Top(i)}) + (W_i * f_{Mid(i)}) + (W_i * f_{Bot(i)}) \leq W_i = A_i,$$

as the height of each rectangle is at most 1 and $f_{Top(i)} \leq \frac{1}{3}$, $f_{Mid(i)} \leq \frac{1}{3}$, and $f_{Bot(i)} \leq \frac{1}{3}$.

In order to pack the fractional rectangles from $R_{Top(i)}$, $R_{Mid(i)}$, and $R_{Bot(i)}$ in $C_{A(i)}$, we reshape them so that they have the full height of a rectangle of the same type but only a fraction of its

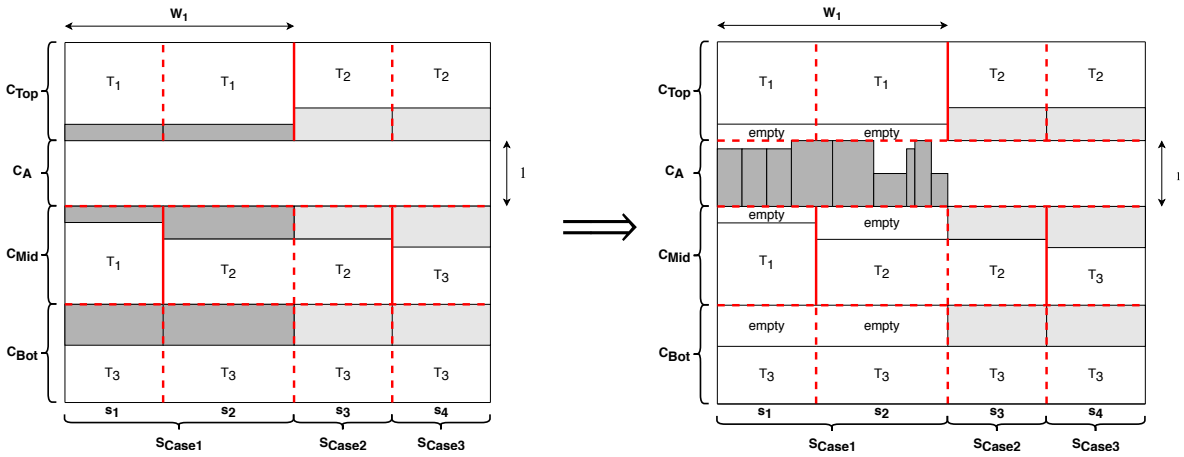


Figure 4.12: (a) C_A is created by shifting all rectangles in C_{Top} upwards, including rectangles in S_{Case2} and S_{Case3}, until there is empty space of height 1 between C_{Top} and C_{Mid}. (b) For every section $s_i \in S_{Case1}$ the fractional rectangles from $R_{Top(i)}$, $R_{Mid(i)}$, and $R_{Bot(i)}$ are removed, re-shaped, and packed side-by-side in C_A within S_{Case1}.

width. This does not change the total area of the fractional rectangles and it ensures that all of the reshaped fractional rectangles when packed side-by-side will fit in $C_{A(i)}$, because the total width of these reshaped rectangles is $W_i * f_{Top(i)} + W_i * f_{Mid(i)} + W_i * f_{Bot(i)} \leq W_i$, and the height of each rectangle is at most 1 and $f_{Top(i)} \leq \frac{1}{3}$, $f_{Mid(i)} \leq \frac{1}{3}$, and $f_{Bot(i)} \leq \frac{1}{3}$. □

By Lemma 4.2.2, the fractional rectangles from S_{Case1} that were removed, reshaped, and packed in C_A have total width at most W_1. Since the width of C_A within S_{Case1} is W_1, the re-shaped rectangles can be packed in C_A.

Lemma 4.2.3. *After processing the fractional rectangles from S_{Case1} as explained in Lemma 4.2.2 there is at most one fractional rectangle of each type in C_A.*

Proof. For each $s_i \in S_{Case1}$, we pack the fractional rectangles of s_i in C_A such that a whole rectangle is formed whenever a sufficient number of pieces of the same type have been packed. When fractional rectangles of the same type do not form a whole rectangle, they merge to become one larger fractional rectangle. Therefore, at most one fractional rectangle of each type may remain in C_A. □

Corollary 4.2.4. *If $S = S_{Case1}$, then any leftover fractional rectangles after processing Case 1 as described in this section can be discarded. Therefore, in this case algorithm 3ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program.*

Proof. For every section $s_i \in S_{Case1}$ the fractional rectangles in $R_{Top(i)}$, $R_{Mid(i)}$, and $R_{Bot(i)}$ are reshaped and packed side-by-side in C_A. By Lemma 4.2.3 there can be at most one fractional rectangle of each type in C_A. Let F be the set of these fractional rectangles.

The fractional rectangles in the solution of the linear program must sum to an integer number of whole rectangles. Therefore, for each rectangle $r \in F$ there must have been more

fractional rectangles in the solution of the linear program of the same type as r in the common portion of the packing of sufficient size to form an integer number of whole rectangles. Recall that any fractional rectangles in the common portion of the packing were rounded up. Therefore, fractions of rectangles of the same type as the rectangles in F and of total area equal to the area of F have been added when rounding up the fractional rectangles in the common portion of the packing. Thus, the fractional rectangles in F can be discarded.

Therefore, if $S = S_{Case1}$, there are no leftover fractional rectangles after processing Case 1 as described in this section. Algorithm 3ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program. □

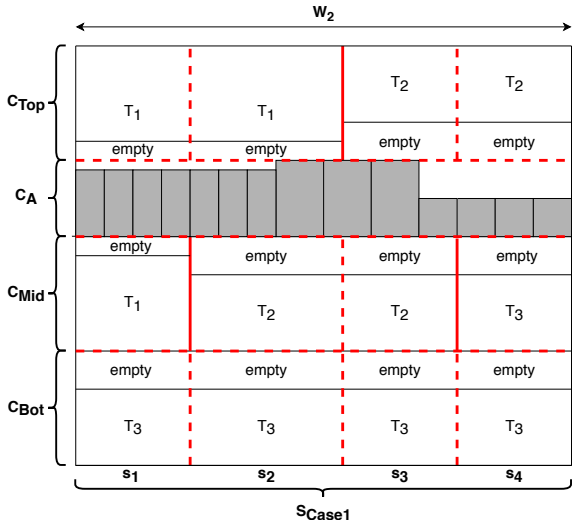


Figure 4.13: When $S = S_{Case1}$, there will be no leftover fractional rectangles after the process described in Corollary 4.2.4.

4.2.4 Case 2. $f_{Bot(i)} > \frac{1}{3}$ and $f_{Top(i)} + f_{Mid(i)} \leq 1$

Let the width of S_{Case2} be W_2. For every section $s_i \in S_{Case2}$ we remove the fractional rectangles in $R_{Top(i)}$ and $R_{Mid(i)}$, including the parts r_{Case2} for vertically split fractional rectangles r; reshape them so that they have the full height of a rectangle of the same type but only a fraction of its width; and pack them side-by-side in a region of width W_2 and height 1 in C_A. Each fractional rectangle fully contained within $R_{Bot(i)}$ is replaced by a whole rectangle of the same type (see Figure 4.14); the parts r_{Case2} for vertically split fractional rectangles $r \in R_{Bot(i)}$ are replaced by fractional rectangles that have the full height of a rectangle of the same type but the same width as r_{Case2}. Hereafter the process of replacing a fractional rectangle r with a rectangle that has the full height as rectangles of the same type as r and shifting other rectangles as needed to make room for the larger rectangle will simply be called *rounding up* a fractional rectangle r.

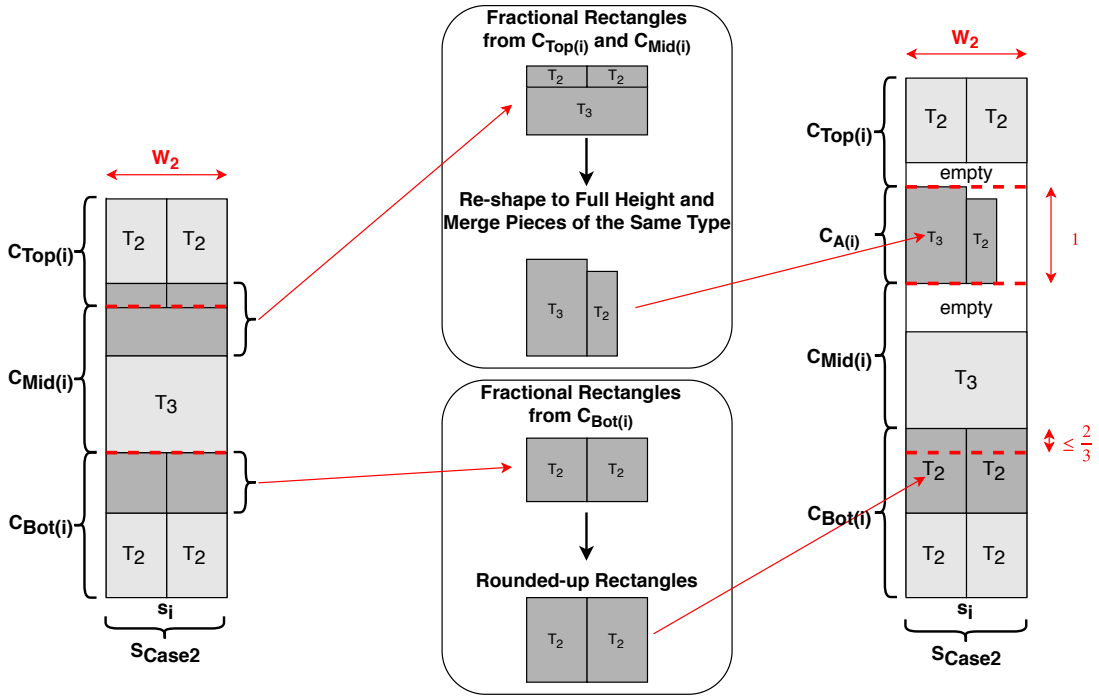


Figure 4.14: For every section $s_i \in S_{Case2}$ the fractional rectangles in $R_{Top(i)}$ and $R_{Mid(i)}$ are removed, re-shaped, and packed side-by-side in C_A; a whole rectangle is formed whenever a sufficient number of fractional pieces of the same type have been packed in C_A. The fractional rectangles in $R_{Bot(i)}$ are rounded up. All rectangles in $C_{Top} \cup C_{Mid}$ are shifted upwards until there is empty space of height 1 between C_{Top} and C_{Mid} and until the rounded up rectangles in $R_{Bot(i)}$ fit between $C_{Mid(i)}$ and $C_{Bot(i)}$.

All rectangles in $C_{Top} \cup C_{Mid}$ are shifted upwards, including rectangles in S_{Case1} and S_{Case3}, until there is empty space of height 1 between C_{Top} and C_{Mid} and the rounded up rectangles in $R_{Bot(i)}$ fit between $C_{Mid(i)}$ and $C_{Bot(i)}$ (see Figure 4.15). After shifting the rectangles the bottoms of the bottommost rectangles in C_{Mid} must lie on a common line and the tops of the topmost rectangles in C_{Top} must also lie on a common line.

Recall that the height of any rectangle is at most 1, therefore, the height increase caused by rounding up for the fractional rectangles in $R_{Bot(i)}$ is at most $\frac{2}{3}$, as $f_{Bot(i)} > \frac{1}{3}$. Combining this

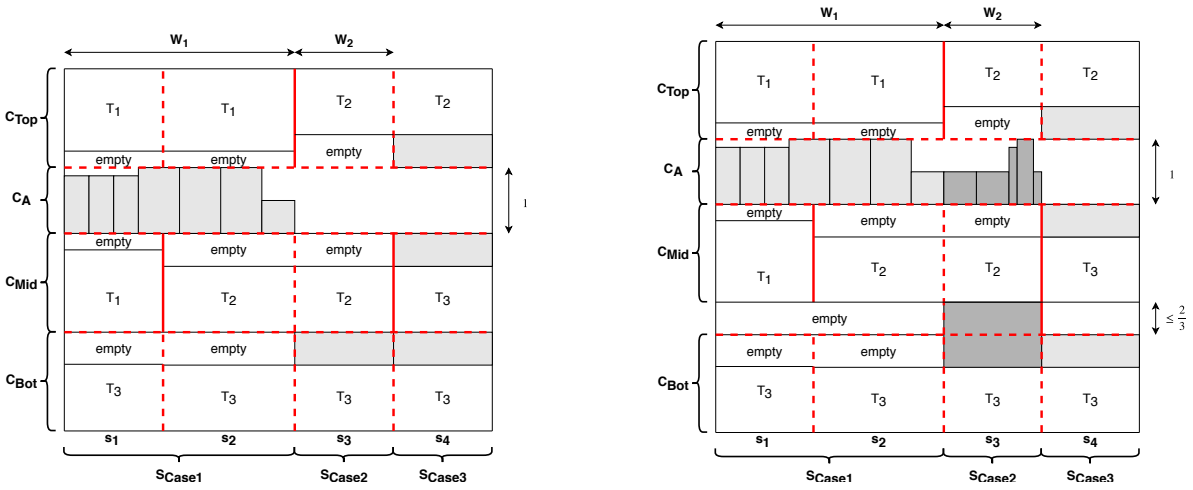


Figure 4.15: (a) All rectangles in $C_{Top} \cup C_{Mid}$ are shifted upwards, including rectangles in S_{Case1} and S_{Case3}, until there is empty space of height 1 between C_{Top} and C_{Mid} and the rounded up rectangles in $R_{Bot(i)}$ fit. (b) For every section $s_i \in S_{Case2}$, the fractional rectangles from $R_{Top(i)}$ and $R_{Mid(i)}$ are removed, re-shaped, and packed side-by-side in C_A. The fractional rectangles in $R_{Bot(i)}$ are rounded up.

with the height increase caused by leaving space of height 1 between C_{Top} and C_{Mid}, the total increase in height of each $s_i \in S_{Case2}$ is at most $\frac{5}{3}$.

In order to pack in C_A the fractional rectangles from $R_{Top(i)}$ and $R_{Mid(i)}$, we first reshape them so that they have the full height of a rectangle of the corresponding type but only a fraction of its width, similar to the process in Section 4.2.3. Recall that by Lemma 4.2.2 these re-shaped rectangles have total width at most W_2. Since the width of C_A within S_{Case2} is W_2, the re-shaped rectangles can be packed in C_A.

Lemma 4.2.5. *If there is a fractional rectangle $r \in S_{Case1} \cap S_{Case2}$ located in C_{Bot} in the solution of the linear program, after processing the fractional rectangles as described above at most one whole rectangle of the same type as r can be formed using the fractional rectangles within C_A and the fractional part r_{Case2}. This whole rectangle can be packed without further increasing the height of the packing and without overlapping any other rectangles.*

Proof. Recall that within C_A when sufficient fractional rectangles of the same type are packed, they merge to become whole rectangles. By Lemma 4.2.3 there can be at most one fractional rectangle of each type leftover in C_A after merging fractional pieces. Let F be the set of fractional rectangles in C_A.

There is empty space within S_{Case1} right beside r_{Case2} of sufficient width to extend the width of r_{Case2} to the width of a whole rectangle of the same type as r_{Case2}, because the fractional rectangle r was originally packed in both S_{Case1} and S_{Case2} in the solution of the linear program (see Figure 4.16a). Furthermore, there is empty space between C_{Mid} and C_{Bot} within S_{Case1} and S_{Case2} of sufficient height to pack a whole rectangle of the same type as r_{Case2}, because the fractional part r_{Case2} was rounded up so that it had the full height of a rectangle of its type (see Figure 4.16b). If there is a fractional rectangle $r' \in F$ of the same type as r_{Case2} such that r' and the rounded up r_{Case2} form a whole rectangle, then r' is removed from F and from C_A and

r_{Case2} is replaced by a whole rectangle of its same type. Therefore, the fractional part r_{Case2} can be replaced by a whole rectangle without further increasing the height of the packing and without overlapping any other rectangles (see Figure 4.16c).

If $S_{Case3} = \emptyset$, then the remaining rectangles in F are discarded because fractional rectangles of the same type in the solution of the linear program sum to an integer number of whole rectangles. Therefore, for each rectangle $r \in F$, there must be more fractional rectangles of the same type as r in either R_{Bot} or the common portion of the packing of sufficient size to form an integer number of whole rectangles. Since all fractional rectangles in $R_{Bot(i)}$ and in the common portion of the packing are rounded up, then it must be the case that fractions of rectangles of the same type as the rectangles in F and of total area at least the area of F have been added when rounding up the fractional rectangles in R_{Bot} and the common portion of the packing. Thus, the fractional rectangles in F can be discarded.

Note that if $S_{Case3} \neq \emptyset$, then we do not discard any leftover fractional rectangles yet as we might need to use them later. □

Corollary 4.2.6. *If $S = S_{Case2}$, then any leftover fractional rectangles after processing Case 2 as described in this section can be discarded. Therefore, in this case algorithm 3ConfigurationRounding produces an integer packing of height at most $\frac{5}{3}$ plus the height of the fractional packing produced by the solution of the linear program.*

Proof. For every section $s_i \in S_{Case2}$ the fractional rectangles in $R_{Bot(i)}$ are rounded up and the fractional rectangles in $R_{Top(i)}$ and $R_{Mid(i)}$ are re-shaped and rectangles of the same type are packed side-by-side in C_A. By Lemma 4.2.3 there can be at most one fractional rectangle of each type leftover after this merging. Let F be the set of these leftover fractional rectangles.

Recall that the fractional rectangles of the same type in the solution of the linear program sum to an integer number of whole rectangles. Therefore, for each rectangle $r \in F$, there must be more fractional rectangles of the same type as r in either R_{Bot} or the common portion of the packing of sufficient size to form an integer number of whole rectangles. Since all fractional rectangles in $R_{Bot(i)}$ and in the common portion of the packing are rounded up, then it must be the case that fractions of rectangles of the same type as the rectangles in F and of total area at least the area of F have been added when rounding up the fractional rectangles in R_{Bot} and the common portion of the packing. Thus, the fractional rectangles in F can be discarded.

Therefore, if $S = S_{Case2}$, there are no leftover fractional rectangles after processing Case 2 as described in this section (see Figure 4.17). Algorithm 3ConfigurationRounding produces an integer packing of height at most $\frac{5}{3}$ plus the height of the fractional packing produced by the solution of the linear program. □

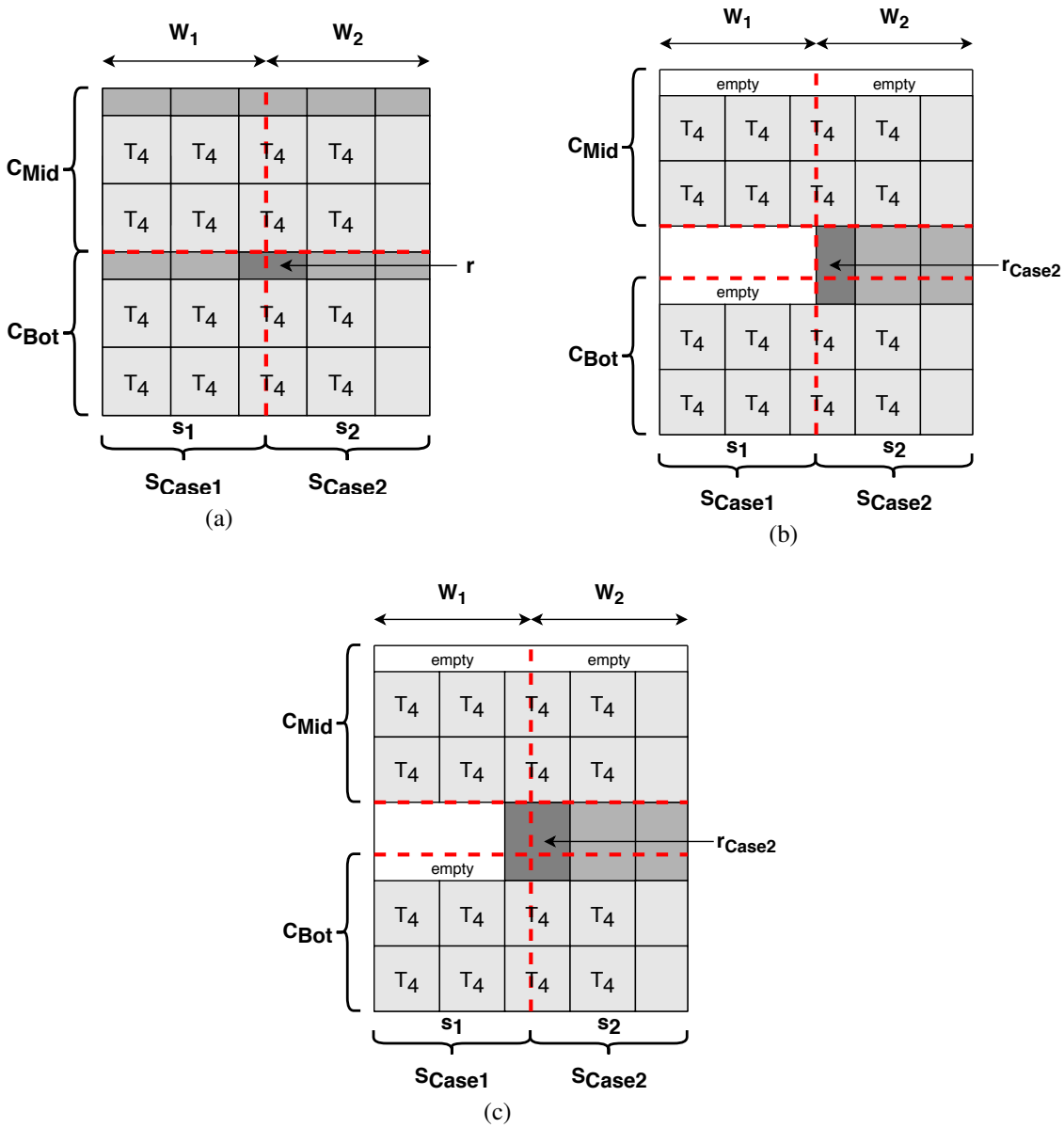


Figure 4.16: (a) A fractional rectangle $r \in S_{Case1} \cap S_{Case2}$ located in C_{Bot} in the solution of the linear program. (b) The rounded up part r_{Case2}. (c) The fractional part r_{Case2} can be replaced by a whole rectangle without further increasing the height of the packing and without overlapping any other rectangles.

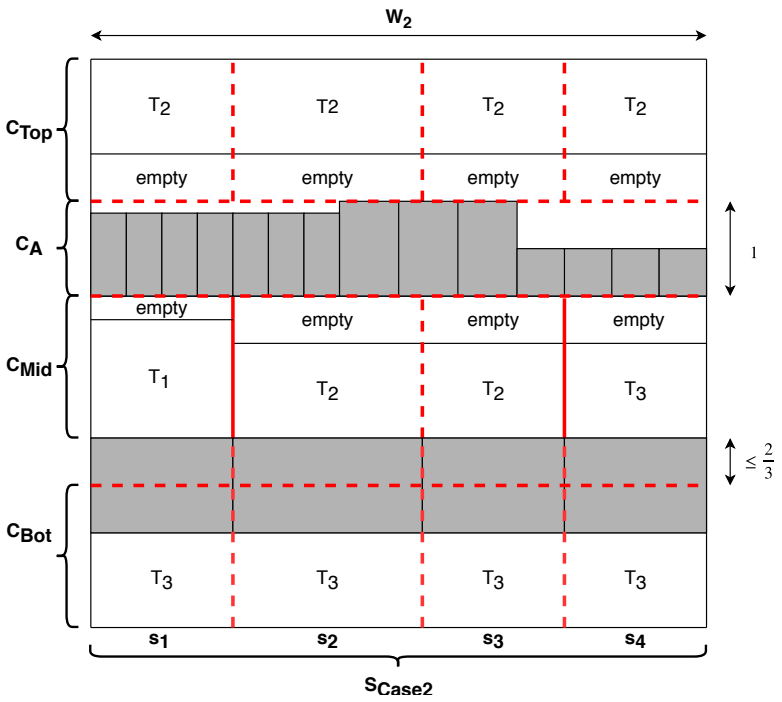


Figure 4.17: When $S = S_{Case2}$, there will be no leftover fractional rectangles after the process described in Corollary 4.2.6.

4.2.5 Case 3. $f_{Bot(i)} > \frac{1}{3}$ and $f_{Top(i)} + f_{Mid(i)} > 1$

For every section $s_i \in S_{Case3}$ the fractional rectangles in $R_{Top(i)}$, $R_{Mid(i)}$, and $R_{Bot(i)}$ are rounded up, including the parts r_{Case3} for vertically split fractional rectangles r (see Figure 4.18). After the fractional rectangles in $R_{Top(i)}$, $R_{Mid(i)}$, and $R_{Bot(i)}$ are rounded up, all rectangles in $C_{Top} \cup C_{Mid}$ are shifted upwards, including rectangles in S_{Case1} and S_{Case2}, until the rounded up rectangles fit between C_{Top} and C_{Mid} and between C_{Mid} and C_{Bot} (see Figure 4.19). After shifting the rectangles the bottoms of the bottommost rectangles in C_{Mid} must lie on a common line and the tops of the topmost rectangles in C_{Top} must also lie on a common line. Since the height of any rectangle is at most 1, this process increases the height of the packing by at most $\frac{5}{3}$.

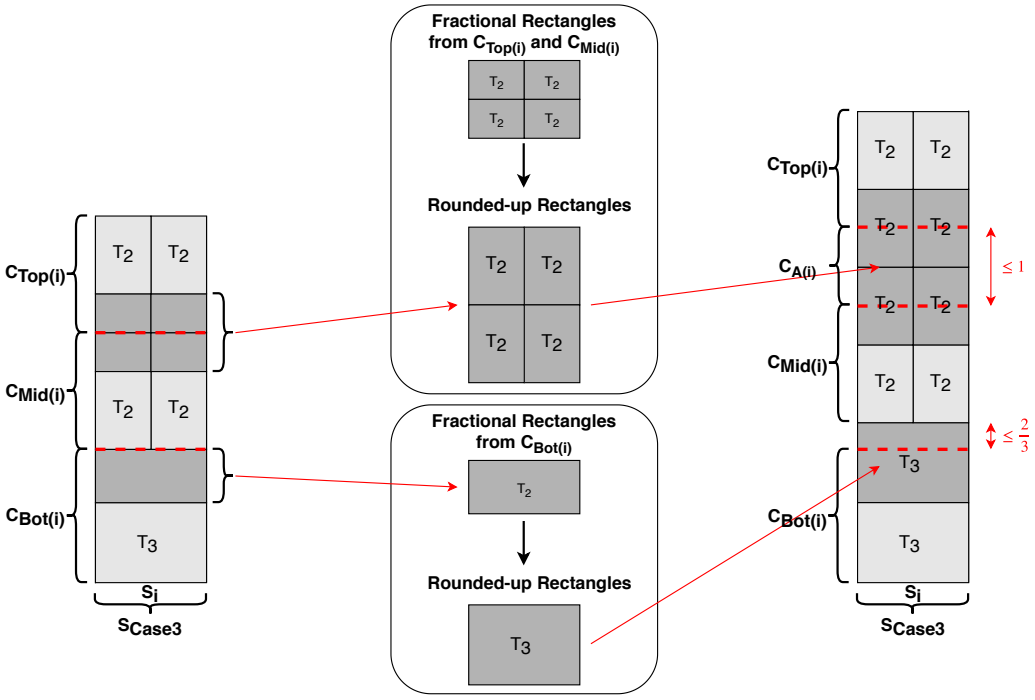


Figure 4.18: For every section $s_i \in S_{Case3}$ the fractional rectangles in $R_{Top(i)}$, $R_{Mid(i)}$, and $R_{Bot(i)}$ are rounded up. All rectangles in $C_{Top} \cup C_{Mid}$ are shifted upwards until the rounded up rectangles fit between C_{Top} and C_{Mid} and between C_{Mid} and C_{Bot}.

Lemma 4.2.7. *If $S_{Case2} \neq \emptyset$ and $S_{Case3} \neq \emptyset$ then the rectangles in C_{Mid} create a vertical division separating Case 2 and Case 3.*

Proof. Let $s_i \in S_{Case2}$ and $s_j \in S_{Case3}$, then $f_{Bot(i)} > \frac{1}{3}$, $f_{Top(i)} + f_{Mid(i)} \leq 1$, $f_{Bot(j)} > \frac{1}{3}$, and $f_{Top(j)} + f_{Mid(j)} > 1$. Therefore, either $f_{Top(j)} > f_{Top(i)}$ holds or $f_{Mid(j)} > f_{Mid(i)}$ holds.

Recall that a vertical division is created at the point where a configuration has packed adjacent rectangles that are not of the same type. This is important; in order for $f_{Top(j)} > f_{Top(i)}$ to hold, the rectangles packed in $C_{Top(j)}$ must be of a different type than the ones packed in $C_{Top(i)}$, and for $f_{Mid(j)} > f_{Mid(i)}$ to hold, the same must be true of the rectangles packed in $C_{Mid(j)}$ and $C_{Mid(i)}$. Therefore, the rectangles in C_{Top} or C_{Mid} must create a vertical division separating Case 2 and Case 3. If needed, we swap C_{Top} and C_{Mid} so that the rectangles in C_{Mid} create a vertical division separating Case 2 and Case 3. □

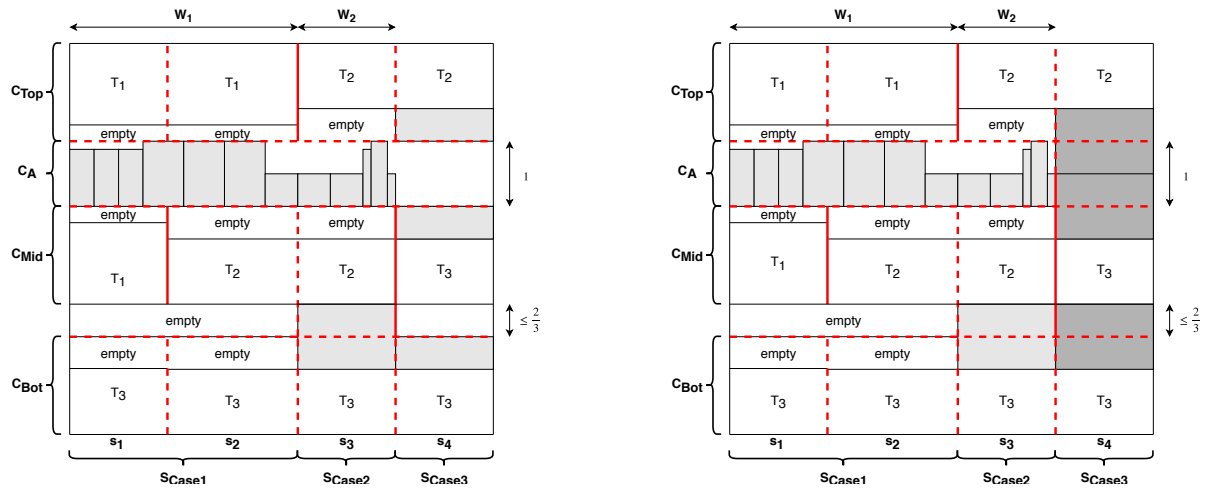


Figure 4.19: After the fractional rectangles in $R_{Top(i)}$, $R_{Mid(i)}$, and $R_{Bot(i)}$ are rounded up, all rectangles in $C_{Top} \cup C_{Mid}$ are shifted upwards, incuding rectangles in S_{Case1} and S_{Case2}, until the rounded up rectangles fit.

Lemma 4.2.8. *If $S_{Case1} \neq \emptyset$, $S_{Case3} \neq \emptyset$, and $S_{Case2} = \emptyset$, then the rectangles in both C_{Mid} and C_{Bot} create a common vertical division separating Case 1 and Case 3.*

Proof. Let $s_i \in S_{Case1}$ and $s_j \in S_{Case3}$, then $f_{Top(i)} \leq \frac{1}{3}$, $f_{Mid(i)} \leq \frac{1}{3}$, $f_{Bot(i)} \leq \frac{1}{3}$, $f_{Bot(j)} > \frac{1}{3}$, and $f_{Top(j)} + f_{Mid(j)} > 1$. Therefore, the rectangles in C_{Bot} create a vertical division separating Case 1 and Case 3. Moreover, since $f_{Top(i)} \leq \frac{1}{3}$ and $f_{Mid(i)} \leq \frac{1}{3}$, then $f_{Top(j)} + f_{Bot(j)} < 1$; therefore, the rectangles in C_{Top} or C_{Mid} must also create a vertical division separating Case 1 and Case 3. If needed, we swap C_{Top} and C_{Mid} so that the rectangles in C_{Mid} create a vertical division separating Case 1 and Case 3. □

Lemma 4.2.9. *Let $S_{Case2} = \emptyset$. If there is a fractional rectangle $r \in S_{Case1} \cap S_{Case3}$ in the solution of the linear program, after processing the fractional rectangles as described above at most one whole rectangle of the same type as r can be formed using the fractional rectangles within C_A and the fractional part r_{Case3}. This whole rectangle can be packed without further increasing the height of the packing and without overlapping any other rectangles.*

Proof. By Lemma 4.2.8 rectangle r must be located in C_{Top}. Recall that within C_A when sufficient fractional rectangles of the same type are packed, they merge to become whole rectangles. By Lemma 4.2.3 there can be at most one fractional rectangle of each type leftover in C_A after this merging. Let F be the set of these leftover fractional rectangles.

The part r_{Case3} was rounded up so that it has the full height of a rectangle of its type but only a fraction of its width (see Figure 4.20b). If there is a fractional rectangle $r' \in F$ of the same type as r_{Case3} such that r' and the rounded up r_{Case3} form a whole rectangle, then r' is removed from F; then r' is shifted rightwards until it is side-by-side with the rounded up r_{Case3} (see Figure 4.20c). Next, r' is shifted upwards to form a whole rectangle with the rounded up r_{Case3} (see Figure 4.20d). Note that r' can be shifted upwards because the fractional rectangle r was packed by the solution of the linear program between S_{Case1} and S_{Case3} so there is enough

empty space beside r_{Case3} to put r' (see Figure 4.20a). Therefore, r' and the rounded up r_{Case3} form a whole rectangle without further increasing the height of the packing and without overlapping any other rectangles.

The remaining rectangles in F are discarded because fractional rectangles of the same type in the solution of the linear program sum to an integer number of whole rectangles. Therefore, for each fractional rectangle $r \in F$, there must be other fractional rectangles of the same type elsewhere in the packing that were rounded up by at least the same area as r. Thus, the fractional rectangles in F can be discarded. Note that if the rounded up r_{Case3} and r' do not form a whole rectangle, then r_{Case3} is discarded as well. □

Lemma 4.2.10. *If there is a fractional rectangle $r \in S_{Case2} \cap S_{Case3}$ located in C_{Top} in the solution of the linear program, after processing the fractional rectangles as described above at most one whole rectangle of the same type as r can be formed using the fractional rectangles within C_A and the fractional part r_{Case3}. This whole rectangle can be packed without further increasing the height of the packing and without overlapping any other rectangles.*

Proof. A similar proof as that of Lemma 4.2.9 can be used. By Lemma 4.2.3 there can be at most one fractional rectangle of each type leftover in C_A after merging fractional pieces. Let F be the set of fractional rectangles in C_A.

If there is a fractional rectangle $r' \in F$ of the same type as r_{Case3} such that r' and the rounded up r_{Case3} form a whole rectangle, then r' is removed from F; then r' is shifted rightwards until it is side-by-side with the rounded up r_{Case3}, and finally r' is shifted upwards to form a whole rectangle with the rounded up r_{Case3} (see Figure 4.20). Refer to Lemma 4.2.9 for an explanation on why there must be sufficient empty space beside r_{Case3} to form a whole rectangle. Therefore, r' and the rounded up r_{Case3} form a whole rectangle without further increasing the height of the packing and without overlapping any other rectangles.

As shown in the proof of Lemma 4.2.9, the remaining rectangles in F can be discarded. If the rounded up r_{Case3} and r' do not form a whole rectangle, then r_{Case3} is discarded as well. □

Corollary 4.2.11. *If there are fractional rectangles $r \in S_{Case1} \cap S_{Case2} \cap S_{Case3}$, then they must be located in C_{Top} or C_{Bot} in the solution of the linear program. After processing the fractional rectangles as described above, at most two whole rectangles of the same types as r can be formed. These whole rectangles can be packed without further increasing the height of the packing and without overlapping any other rectangles.*

Proof. By Lemma 4.2.7, the rectangles in C_{Mid} create a vertical division separating Case 2 and Case 3; therefore, a fractional rectangle $r \in S_{Case1} \cap S_{Case2} \cap S_{Case3}$ cannot be located in C_{Mid}, as the rectangles in S_{Case2} and S_{Case3} of C_{Mid} are of different types. There is at most one vertically split fractional rectangle $r \in S_{Case1} \cap S_{Case2} \cap S_{Case3} \cap C_{Top}$ and at most one vertically split fractional rectangle $r' \in S_{Case1} \cap S_{Case2} \cap S_{Case3} \cap C_{Bot}$.

Recall that the fractional rectangles from S_{Case1} and $S_{Case2} \cap (R_{Top} \cup R_{Mid})$ were packed in C_A. By Lemma 4.2.3 there can be at most one fractional rectangle of each type in C_A. Let F be the set of these fractional rectangles.

Note that Lemma 4.2.10 holds even if $r \in S_{Case1} \cap S_{Case2} \cap S_{Case3} \cap C_{Top}$, because the fractional parts r_{Case1} and r_{Case2} were both re-shaped and packed in C_A; hence, the same argument used to prove Lemma 4.2.10 shows that either r_{Case3} can be replaced by a whole rectangle of

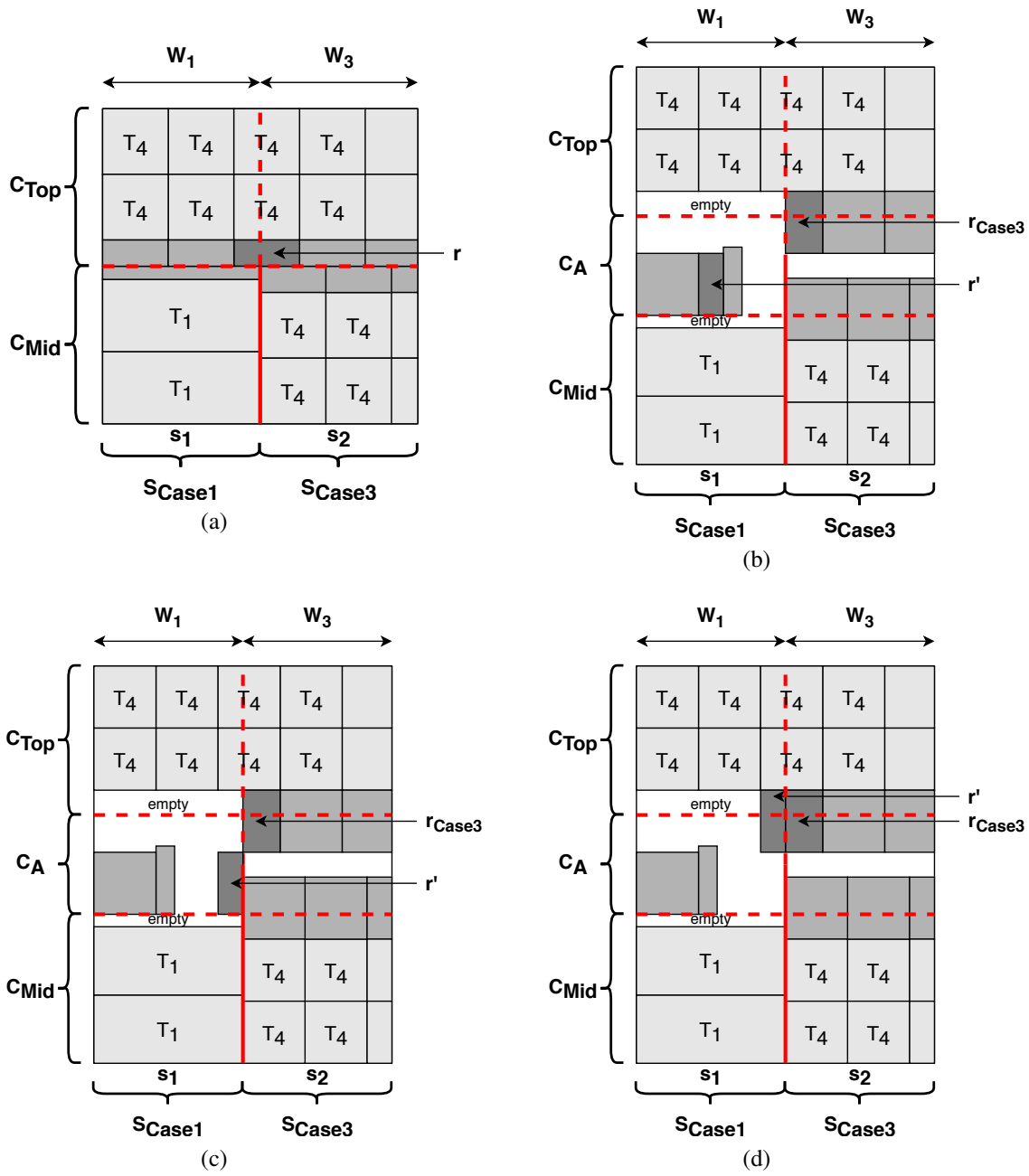


Figure 4.20: (a) A fractional rectangle $r \in S_{Case1} \cap S_{Case3} \cap C_{Top}$ was packed by the solution of the linear program. (b) A fractional rectangle $r' \in F$ of the same type as the rounded up r_{Case3} can form a whole rectangle with the rounded up r_{Case3}. (c) The fractional rectangle r' is shifted rightwards until it is side-by-side with the rounded up r_{Case3}. (d) The fractional rectangle r' is shifted upwards until it forms a whole rectangle with the rounded up r_{Case3}.

its same type without further increasing the height of the packing and without overlapping any other rectangles, or the rounded up r_{Case3} and the rectangles in F of the same type as r_{Case3} can be discarded.

Note that Lemma 4.2.5 holds even if $r \in S_{Case1} \cap S_{Case2} \cap S_{Case3} \cap C_{Bot}$, because the fractional parts r_{Case2} and r_{Case3} are both rounded up and are essentially one larger fractional rectangle; hence, the same argument used to prove Lemma 4.2.5 shows that r_{Case2} and r_{Case3} can be be replaced by a whole rectangle of its same type without further increasing the height of the packing and without overlapping any other rectangles. If the rounded up parts r_{Case2}, r_{Case3}, and a fractional rectangle from F do not form a whole rectangle, then r_{Case2}, r_{Case3}, and the fractional rectangle of the same type as r_{Case2} and r_{Case3} can be discarded. □

Corollary 4.2.12. *If $S = S_{Case3}$, then there will be no leftover fractional rectangles after processing Case 3 as described in this section. Therefore, in this case algorithm 3ConfigurationRounding produces an integer packing of height at most $\frac{5}{3}$ plus the height of the fractional packing produced by the solution of the linear program.*

Proof. For every section $s_i \in S_{Case3}$ the fractional rectangles in $R_{Top(i)}$, $R_{Mid(i)}$, and $R_{Bot(i)}$ are rounded up to become whole rectangles; therefore, there will be no leftover fractional rectangles (see Figure 4.21). Recall that the height of any rectangle is at most 1; therefore, the total increase in height from rounding up the fractional rectangles in $R_{Top(i)}$, $R_{Mid(i)}$, and $R_{Bot(i)}$ is at most $\frac{5}{3}$. □

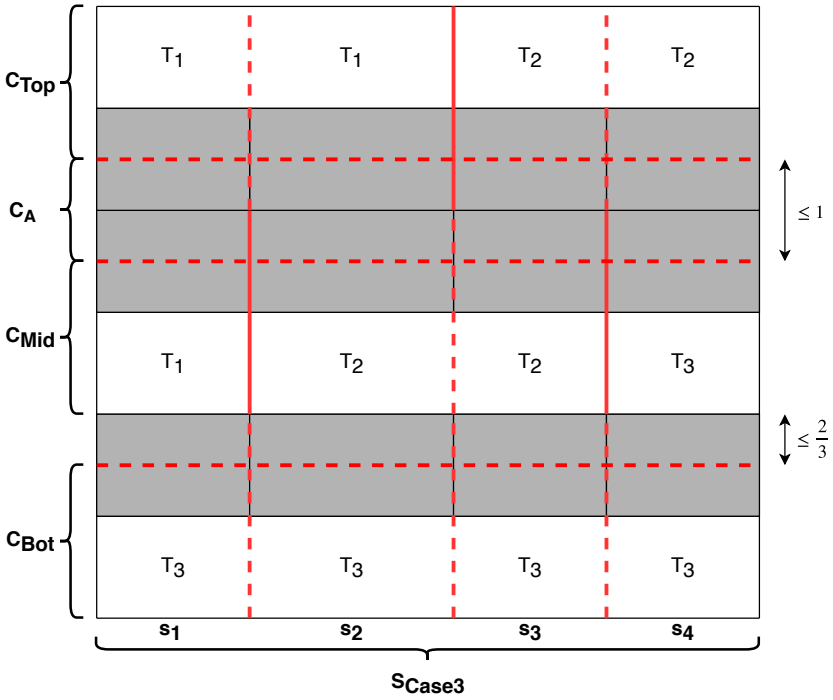


Figure 4.21: When $S = S_{Case3}$, there will be no leftover fractional rectangles after processing Case 3.

We can now prove Theorem 4.2.1.

Proof. By Corollary 4.1.1, if all rectangles are packed in the common portion of the packing, algorithm 3ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program.

Recall that the common portion of the packing can be rounded independently of the uncommon portion of the packing. For the rest of the proof we assume that the common portion of the packing has been processed so it does not have any fractional rectangles and we already know how much the height of the packing in this portion has increased.

By Corollary 4.2.4, if $S = S_{Case1}$, algorithm 3ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program. By Corollary 4.2.6, if $S = S_{Case2}$, algorithm 3ConfigurationRounding produces an integer packing of height at most $\frac{5}{3}$ plus the height of the fractional packing produced by the solution of the linear program. By Corollary 4.2.12, if $S = S_{Case3}$, algorithm 3ConfigurationRounding produces an integer packing of height at most $\frac{5}{3}$ plus the height of the fractional packing produced by the solution of the linear program.

If $S_{Case1} \neq \emptyset$, $S_{Case2} \neq \emptyset$, and $S_{Case3} = \emptyset$, then after processing Case 1 and Case 2 as described in this section, the height increase caused by leaving space of height 1 between C_{Top} and C_{Mid} is 1, and the height increase caused by rounding up for the fractional rectangles in R_{Bot} within S_{Case2} is at most $\frac{2}{3}$. By Lemma 4.2.5, any fractional rectangle $r \in S_{Case1} \cap S_{Case2} \cap C_{Bot}$ in the solution of the linear program is either replaced by a whole rectangle of the same type without further increasing the height of the packing and without overlapping any other rectangles or it is discarded. Note that for fractional rectangles $r \in S_{Case1} \cap S_{Case2}$ in C_{Top} or C_{Mid}, the parts r_{Case1} and r_{Case2} are both removed, re-shaped, and packed side-by-side in C_A. For any whole rectangle $r' \in S_{Case1} \cap S_{Case2}$ the parts r_{Case1} and r_{Case2} remain side-by-side and thus still form a whole rectangle in the final packing. Using the same proof as in Corollary 4.2.6, it can be shown that any leftover fractional rectangles at this point can be discarded. Therefore, in this case algorithm 3ConfigurationRounding produces an integer packing of height at most $\frac{5}{3}$ plus the height of the fractional packing produced by the solution of the linear program.

If $S_{Case1} \neq \emptyset$, $S_{Case2} = \emptyset$, and $S_{Case3} \neq \emptyset$, then after processing Case 1 and Case 3 as described in this section, the height increase caused by leaving space of height 1 between C_{Top} and C_{Mid} is 1, and fractional rectangles in R_{Top} and R_{Mid} within S_{Case3} can be rounded up without further increasing the height of the packing. Rounding up the fractional rectangles in R_{Bot} within S_{Case3} further increases the height of the packing by at most $\frac{2}{3}$. By Lemma 4.2.9, any fractional rectangle $r \in S_{Case1} \cap S_{Case3} \cap C_{Top}$ in the solution of the linear program is either transformed into a whole rectangle without further increasing the height of the packing or overlapping any other rectangles or it is discarded. By Lemma 4.2.8, there can be no fractional rectangle $r \in S_{Case1} \cap S_{Case3}$ in C_{Mid} or C_{Bot}. For any whole rectangle $r' \in S_{Case1} \cap S_{Case3}$ the parts r_{Case1} and r_{Case3} remain side-by-side and thus still form a whole rectangle in the final packing. Using the same proof as in Corollary 4.2.6, it can be shown that any leftover fractional rectangles at this point can be discarded. Therefore, in this case algorithm 3ConfigurationRounding produces an integer packing of height at most $\frac{5}{3}$ plus the height of the fractional packing produced by the solution of the linear program.

If $S_{Case1} = \emptyset$, $S_{Case2} \neq \emptyset$, and $S_{Case3} \neq \emptyset$, then after processing Case 2 and Case 3 as described in this section, the height increase caused by leaving space of height 1 between C_{Top}

and C_{Mid} is 1, and fractional rectangles in R_{Top} and R_{Mid} within S_{Case3} can be rounded up without further increasing the height of the packing. Rounding up the fractional rectangles in R_{Bot} within S_{Case3} further increases the height of the packing by at most $\frac{2}{3}$. By Lemma 4.2.9, any fractional rectangle $r \in S_{Case2} \cap S_{Case3} \cap C_{Top}$ in the solution of the linear program is either transformed into a whole rectangle without further increasing the height of the packing or overlapping any other rectangles or it is discarded. By Lemma 4.2.7, there can be no fractional rectangle $r \in S_{Case2} \cap S_{Case3} \cap C_{Mid}$. Note that for a fractional rectangle $r \in S_{Case2} \cap S_{Case3} \cap C_{Bot}$, the parts r_{Case2} and r_{Case3} are both rounded up and form a whole rectangle. For any whole rectangle $r' \in S_{Case2} \cap S_{Case3}$ the parts r_{Case2} and r_{Case3} remain side-by-side and thus still form a whole rectangle in the final packing. Using the same proof as in Corollary 4.2.6, it can be shown that any leftover fractional rectangles at this point can be discarded. Therefore, in this case algorithm 3ConfigurationRounding produces an integer packing of height at most $\frac{5}{3}$ plus the height of the fractional packing produced by the solution of the linear program.

If $S_{Case1} \neq \emptyset$, $S_{Case2} \neq \emptyset$, and $S_{Case3} \neq \emptyset$, then after processing Case 1, Case 2, and Case 3 as described in this section, the height increase caused by leaving space of height 1 between C_{Top} and C_{Mid} is 1, and fractional rectangles in R_{Top} and R_{Mid} within S_{Case3} can be rounded up without further increasing the height of the packing. Rounding up the fractional rectangles in R_{Bot} within S_{Case2} and S_{Case3} further increases the height of the packing by at most $\frac{2}{3}$. By Lemma 4.2.5, a fractional rectangle $r \in S_{Case1} \cap S_{Case2} \cap C_{Bot}$ in the solution of the linear program can be transformed into a whole rectangle without further increasing the height of the packing and without overlapping any other rectangles. By Lemma 4.2.9, any fractional rectangle $r \in S_{Case2} \cap S_{Case3} \cap C_{Top}$ in the solution of the linear program is either transformed into a whole rectangle without further increasing the height of the packing or overlapping any other rectangles or it is discarded. By Corollary 4.2.11, a fractional rectangle $r \in S_{Case1} \cap S_{Case2} \cap S_{Case3}$ located in C_{Top} or C_{Bot} in the solution of the linear program can either be transformed into a whole rectangle without further increasing the height of the packing or overlapping any other rectangles or it can be discarded. Note that for fractional rectangles $r \in S_{Case1} \cap S_{Case2} \cap (C_{Top} \cup C_{Mid})$, the parts r_{Case1} and r_{Case2} are both removed, re-shaped, and packed side-by-side in C_A. Additionally, note that for a fractional rectangles $r \in S_{Case2} \cap S_{Case3} \cap C_{Bot}$, the parts r_{Case2} and r_{Case3} are both rounded up and form a whole rectangle. By Lemma 4.2.7, there can be no fractional rectangle $r \in S_{Case2} \cap S_{Case3} \cap C_{Mid}$. For any vertically split whole rectangle r' its parts remain side-by-side and thus still form a whole rectangle in the final packing. Using the same proof as in Corollary 4.2.6, it can be shown that any leftover fractional rectangles at this point can be discarded. Therefore, in this case algorithm 3ConfigurationRounding produces an integer packing of height at most $\frac{5}{3}$ plus the height of the fractional packing produced by the solution of the linear program.

Therefore, algorithm 3ConfigurationRounding produces an integer packing of height at most $\frac{5}{3}$ plus the height of the fractional packing produced by the solution of the linear program. □

4.3 Two Configurations

When the solution to the linear program has two configurations we refer to them as C_{Top} and C_{Bot}. The two configurations are packed one on top of the other (see Figure 4.22); note that

C_{Top} has again been flipped upside down. Our algorithm considers two cases, depending on the values of the two fractions $f_{Top(i)}$ and $f_{Bot(i)}$ for each section s_i:

- **Case 1:** $f_{Top(i)} + f_{Bot(i)} \leq 1$, and
- **Case 2:** $f_{Top(i)} + f_{Bot(i)} > 1$.

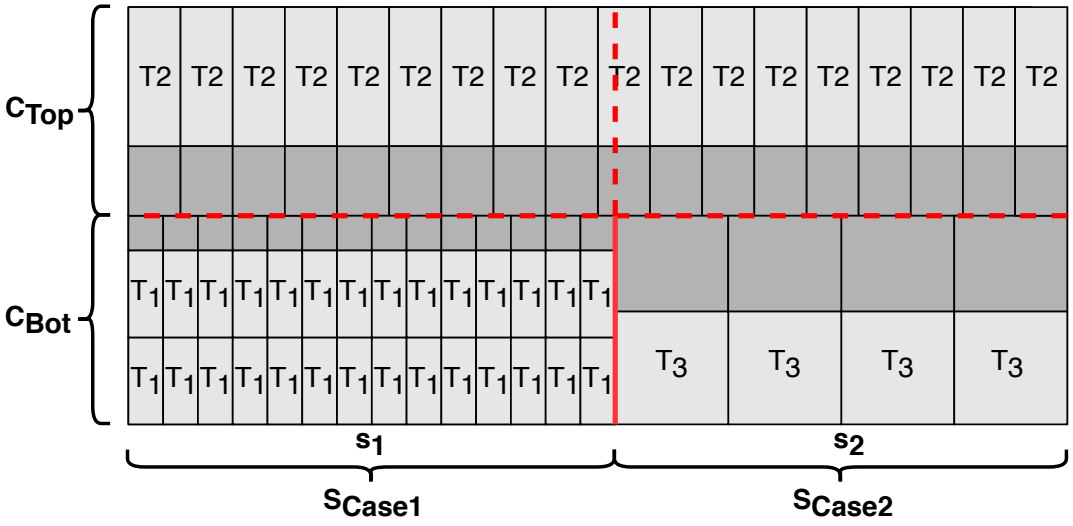


Figure 4.22: C_{Top} is flipped upside down.

If needed, we change the order of the configurations so that if there is at least one section $s_i \in S_{Case1}$ and at least one section $s_j \in S_{Case2}$, then the rectangles in C_{Bot} create a vertical division separating Case 1 and Case 2 (see Figure 4.22). The algorithm for rounding fractional rectangles into whole ones when there are two configurations is shown below.

Note that this algorithm only depends on the number of configurations and not on the number of rectangle types.

Theorem 4.3.1. *Algorithm 2ConfigurationRounding produces an integer packing of height at most* 1 *plus the height of the fractional packing produced by the solution of the linear program.*

We prove this theorem in the following two subsections.

Algorithm 4.3 2ConfigurationRounding(*FractionalSolution*)

Input: An optimal fractional packing *FractionalSolution* with two configurations C_{Top} and C_{Bot}.
Output: The height of an integer packing h obtained from *FractionalSolution*.
$S_{Case1} = \emptyset$.
$S_{Case2} = \emptyset$.
for $s_i \in S$ **do**
    if $f_{Top(i)} + f_{Bot(i)} \leq 1$ **then**
        $S_{Case1} = S_{Case1} \cup s_i$.
    else
        $S_{Case2} = S_{Case2} \cup s_i$.
    end if
end for
Order the configurations so that if there is at least one section $s_i \in S_{Case1}$ and at least one section $s_j \in S_{Case2}$, then the rectangles in C_{Bot} create a vertical division separating Case 1 and Case 2.
h_0 = height of *FractionalSolution*.
h_1 = height increase after processing S_{Case1} according to Section 4.3.1.
h_2 = height increase after processing S_{Case2} according to Section 4.3.2.
return $h_0 + \max(h_1, h_2)$.

4.3.1 Case 1. $f_{Top(i)} + f_{Bot(i)} \leq 1$

Let the width of S_{Case1} be W_1. For every section $s_i \in S_{Case1}$ we remove the fractional rectangles in $R_{Top(i)}$ and $R_{Bot(i)}$, including the parts r_{Case1} for vertically split fractional rectangles; re-shape them so that they have the full height of a rectangle of the same type but only a fraction of its width; and pack them side-by-side in a region of width W_1 and height 1. This region, hereafter is referred to as C_A (see Figure 4.23), is created by shifting all rectangles in C_{Top} upwards, including rectangles in S_{Case2}, until there is empty space of height 1 between C_{Top} and C_{Bot}. After shifting the rectangles the tops of the topmost rectangles in C_{Top} must lie on a common line.

In order to pack in C_A the fractional rectangles from $R_{Top(i)}$ and $R_{Bot(i)}$, we first reshape them so that they have the full height of a rectangle of the corresponding type but only a fraction of its width. Recall that by Lemma 4.2.2 these re-shaped rectangles have total width at most W_1. Since the width of C_A within S_{Case1} is W_1, the re-shaped rectangles can be packed in C_A.

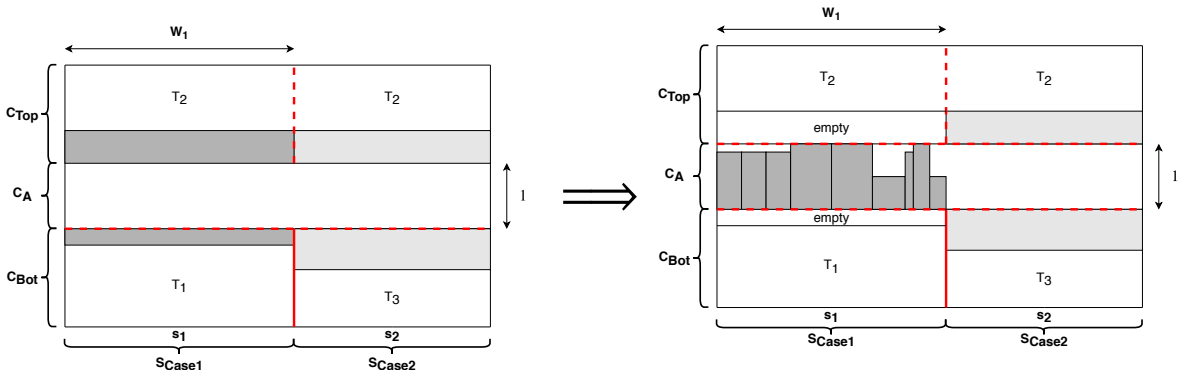


Figure 4.23: (a) C_A is created by shifting all rectangles in C_{Top} upwards, including rectangles in S_{Case2}, until there is empty space of height 1 between C_{Top} and C_{Bot}. (b) The fractional rectangles from $R_{Top(i)}$ and $R_{Bot(i)}$ are removed, re-shaped, and packed side-by-side in C_A.

4.3.2 Case 2. $f_{Top} + f_{Bot} > 1$

For every section $s_i \in S_{Case2}$ the fractional rectangles in $R_{Top(i)}$ and $R_{Bot(i)}$ are rounded up, including the parts r_{Case2} for vertically split fractional rectangles. After the fractional rectangles in $R_{Top(i)}$ and $R_{Bot(i)}$ are rounded up, all rectangles in C_{Top} are shifted upwards, including rectangles in S_{Case1}, until the rounded up rectangles fit between C_{Top} and C_{Bot} (see Figure 4.24). After shifting the rectangles the tops of the topmost rectangles in C_{Top} must lie on a common line. Since the height of any rectangle is at most 1, this process increases the height of the packing by at most 1.

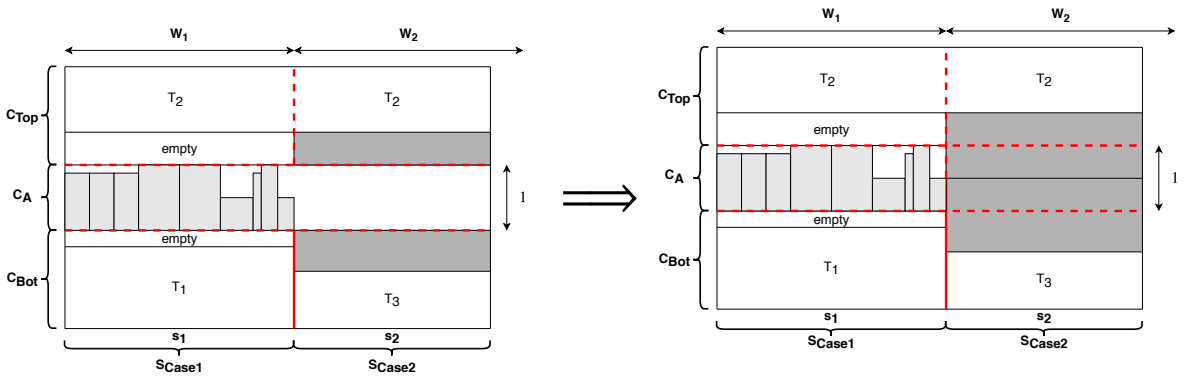


Figure 4.24: After the fractional rectangles in $R_{Top(i)}$ and $R_{Bot(i)}$ are rounded up, all rectangles in C_{Top} are shifted upwards, including rectangles in S_{Case1}, until the rounded up rectangles fit.

We can now prove Theorem 4.3.1.

Proof. By Corollary 4.1.1, if all rectangles are packed in the common portion of the packing, algorithm 2ConfigurationRounding produces an integer packing of height at most 1 plus the

height of the fractional packing produced by the solution of the linear program.

Recall that the common portion of the packing can be rounded independently of the uncommon portion of the packing. For the rest of the proof we assume that the common portion of the packing has been processed so it does not have any fractional rectangles and we already know how much the height of the packing in this portion has increased.

By Corollary 4.2.4, if $S = S_{Case1}$, algorithm 2ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program. By Corollary 4.2.12, if $S = S_{Case2}$, algorithm 2ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program.

If $S_{Case1} \neq \emptyset$ and $S_{Case2} \neq \emptyset$, the height increase caused by leaving space between C_{Top} and C_{Bot} is 1, and fractional rectangles within S_{Case2} can be rounded up without further increasing the height of the packing. If there is a fractional rectangle $r \in S_{Case1} \cap S_{Case2}$ located in C_{Top} in the solution of the linear program, after processing the fractional rectangles as described above at most one whole rectangle of the same type as r can be formed using fractional rectangles within C_A and the fractional part r_{Case2}. By Lemma 4.2.9, either this whole rectangle is packed without further increasing the height of the packing and without overlapping any other rectangles, or the rounded up r_{Case2} and the remaining fractional rectangles are discarded. For any whole rectangle $r' \in S_{Case1} \cap S_{Case2}$ the parts r_{Case1} and r_{Case2} remain side-by-side and thus still form a whole rectangle in the final packing. Using the same proof as in Corollary 4.2.6, it can be shown that any leftover fractional rectangles at this point can be discarded. Therefore, in this case algorithm 2ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program.

Therefore, algorithm 2ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program. □

4.4 One Configuration

When the solution to the linear program has one configuration we refer to it as C (see Figure 4.25). Note that $f_i \leq 1$ for each section s_i.

The algorithm for rounding fractional rectangles into whole ones when there is one configuration is shown below.

Algorithm 4.4 1ConfigurationRounding(*FractionalSolution*)

1: **Input:** An optimal fractional solution *FractionalSolution* with one configuration C.
2: **Output:** The height of an integer packing h obtained from *FractionalSolution*.
3: S = the set of all sections in *FractionalSolution*.
4: h_0 = height of *FractionalSolution*.
5: h_1 = height increase after processing S according to Theorem 4.4.1.
6: **return** $h_0 + h_1$.

Note that this algorithm only depends on the number of configurations and not on the number of rectangle types.

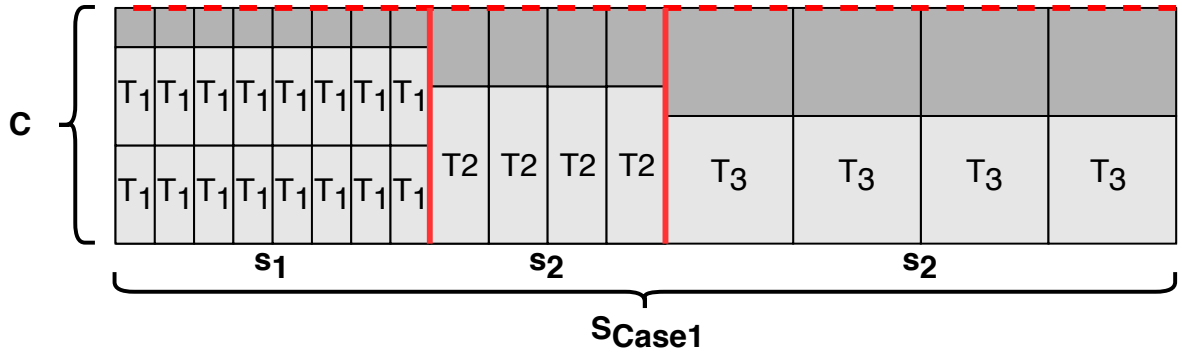


Figure 4.25: One configuration.

Theorem 4.4.1. *Algorithm 1ConfigurationRounding produces an integer packing of height at most* 1 *plus the height of the fractional packing produced by the solution of the linear program.*

Proof. For every section $s_i \in S$ the fractional rectangles in R are rounded up. Since the height of any rectangle is at most 1, this process increases the height of the packing by at most 1.

By Corollary 4.2.12, algorithm 1ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program. □

4.5 Approximation Ratio

As described above, when there are $k = 3$ rectangle types in the fractional solution computed by the GLS algorithm there can be either three configurations, two configurations, or only one configuration.

By Theorems 4.2.1, 4.3.1, and 4.4.1, when there are three configurations, two configurations, or only one configuration, algorithm 3TypeRounding increases the height of the fractional packing produced by the solution of the linear program by at most $\frac{5}{3}$, 1, or 1, respectively.

Therefore, when there are $k = 3$ rectangle types, algorithm 3TypeRounding produces an integer packing of height at most $\frac{5}{3}$ plus the height of the fractional packing produced by the solution of the linear program. Since the height of the fractional packing obtained by solving the linear program is no larger than the height OPT of an optimum solution for HMSP, then algorithm 3TypeRounding produces a packing of height at most $OPT + \frac{5}{3}$.

4.6 Running Time

Throughout the thesis we have provided many figures describing the structure of the packings computed by algorithm 3TypeRounding. We now show that these packings can be represented in a very compact manner. Recall that the input to HMSP is represented as a list of $3k$ numbers: the input contains, for $1 \leq i \leq k$, the width of a rectangle of type T_i, the height of a rectangle of type T_i, and the number of rectangles of type T_i (see Figure 4.26).

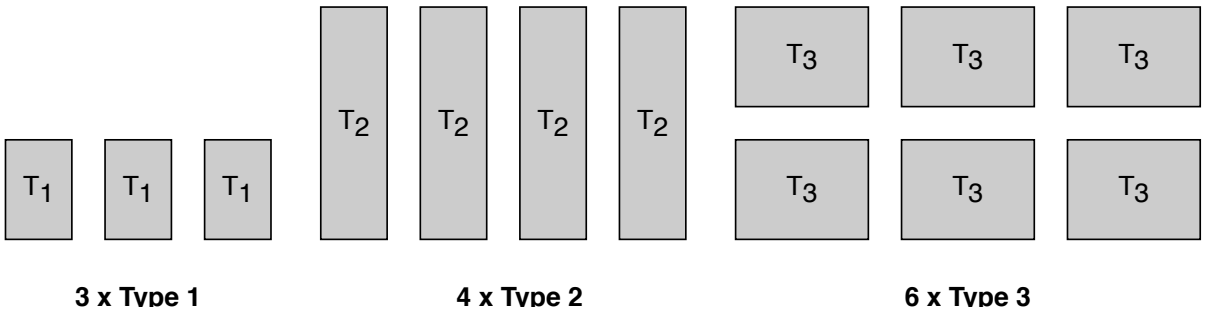


Figure 4.26: The above input to HMSP is specified as a list of $3k = 9$ numbers: $\{(4, 6, 3), (4, 14, 4), (8, 6, 6)\}$. There are 3 type T_1 rectangles, each with width 4 and height 6; there are 4 type T_2 rectangles, each with width 4 and height 14; and there are 6 type T_3 rectangles, each with width 8 and height 6.

The input of algorithm 3TypeRounding is different than the input of HMSP. The vector output from the linear program is transformed into a list of numbers that specifies the rectangles that are packed in the configurations of the fractional solution. A configuration is specified using $O(k)$ numbers: for $1 \leq i \leq k$, we specify the rectangle type T_i, the number of rectangles of type T_i packed side-by-side, and the number of rectangles of type T_i packed one on top of the other (see Figure 4.27). Note that the number of rectangles packed one on top of the other is a rational number; for example, if 2.5 rectangles are packed one on top of the other, then two whole rectangles and one fractional rectangle with height equal to $\frac{1}{2}$ of the height of a whole rectangle of the corresponding type are packed one on top of the other, with the fractional rectangle packed on top.

Therefore, since the fractional solution determined by the linear program has at most k configurations, and since each configuration uses $O(k)$ numbers, then the input to algorithm 3TypeRounding uses $O(k^2)$ numbers to specify all the configurations.

The output of algorithm 3TypeRounding is also specified in a compact manner. The common portion of the packing is specified as a list of $O(k)$ numbers: for $1 \leq i \leq k$, we specify the rectangle type T_i, the number of rectangles of type T_i packed side-by-side, and the number of rectangles of type T_i packed one on top of the other (see Figure 4.28).

The part of a configuration in the uncommon portion of the packing is specified as a list of $O(k)$ numbers: we specify the rectangle type T_i, the number of rectangles of type T_i packed side-by-side, and the number of rectangles of type T_i packed one on top of the other (see Figure 4.29). Note that the order of the rectangle types in the uncommon portion of the packing does not need to be 1, 2, 3. Additionally, at most one rectangle type might be repeated in the list if some rectangles of that type are rounded up and some others are not (see Figure 4.29).

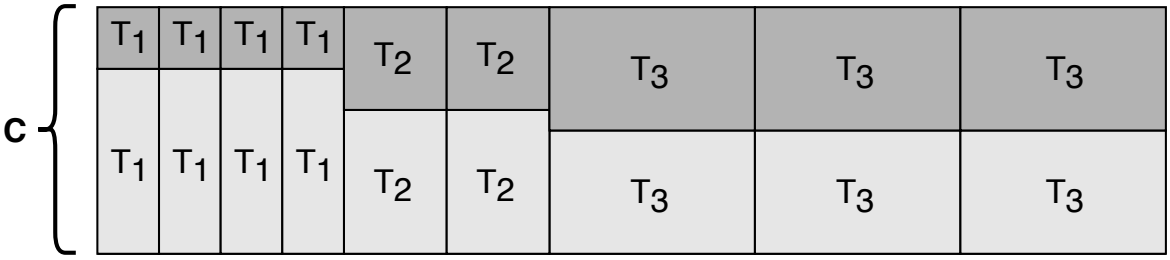


Figure 4.27: The above configuration of a fractional solution is specified as a list of $O(k)$ numbers: $\{(1,4,1.33),(2,2,1.71),(3,3,2)\}$. For example, the first rectangle is of type T_1, there are 4 type T_1 rectangles packed side-by-side, and there are 1.33 type T_1 rectangles packed one on top of the other. That is, there is 1 whole rectangle and one fractional rectangle with height equal to $\frac{1}{3}$ of the height of a whole rectangle of type T_1 packed one on top of the other.

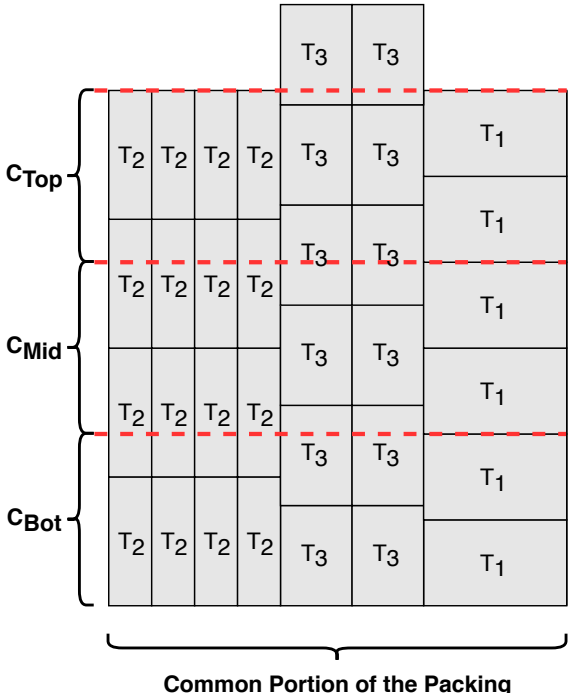


Figure 4.28: The common portion of the above packing is specified as a list of $O(k)$ numbers: $\{(2,4,4),(3,2,6),(1,1,6)\}$. For example, the first rectangle is of type T_1, there are 4 type T_1 rectangles packed side-by-side, and there are 4 type T_1 rectangles packed one on top of the other.

Therefore, since the fractional solution determined by the linear program has at most k configurations, we only add a single configuration C_A, and each configuration uses $O(k)$ numbers, then the output produced by algorithm 3TypeRounding uses $O(k^2)$ numbers to specify all the configurations (see Figure 4.30).

Algorithm 3TypeRounding transforms a fractional solution into an integral one by performing operations on the list of numbers given as input.

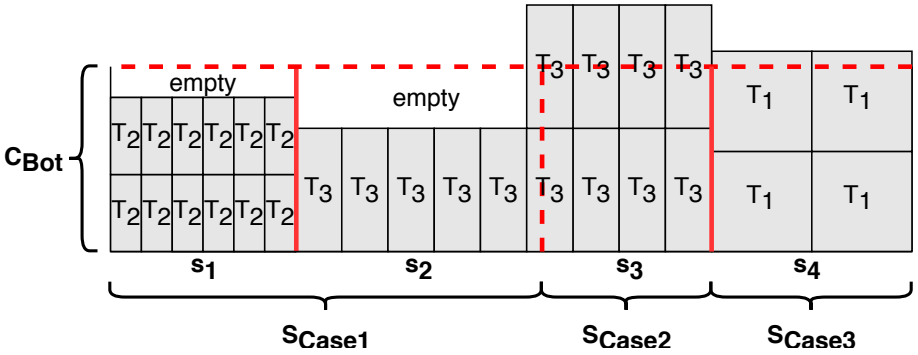


Figure 4.29: The part of a configuration in the uncommon portion of the packing is specified as a list of $O(k)$ numbers: $\{(2,6,2),(3,5,1),(3,4,2),(1,2,2)\}$. Note how rectangle type T_2 is repeated in this list; the fractional rectangles in S_{Case1} are removed, re-shaped, and packed in C_A and so there is only 1 rectangle of type T_2 packed one on top of the other in C_{Bot}, but the rectangles in S_{Case2} located in C_{Bot} are rounded up and so there are 2 rectangles of type T_2 packed one on top of the other in C_{Bot}.

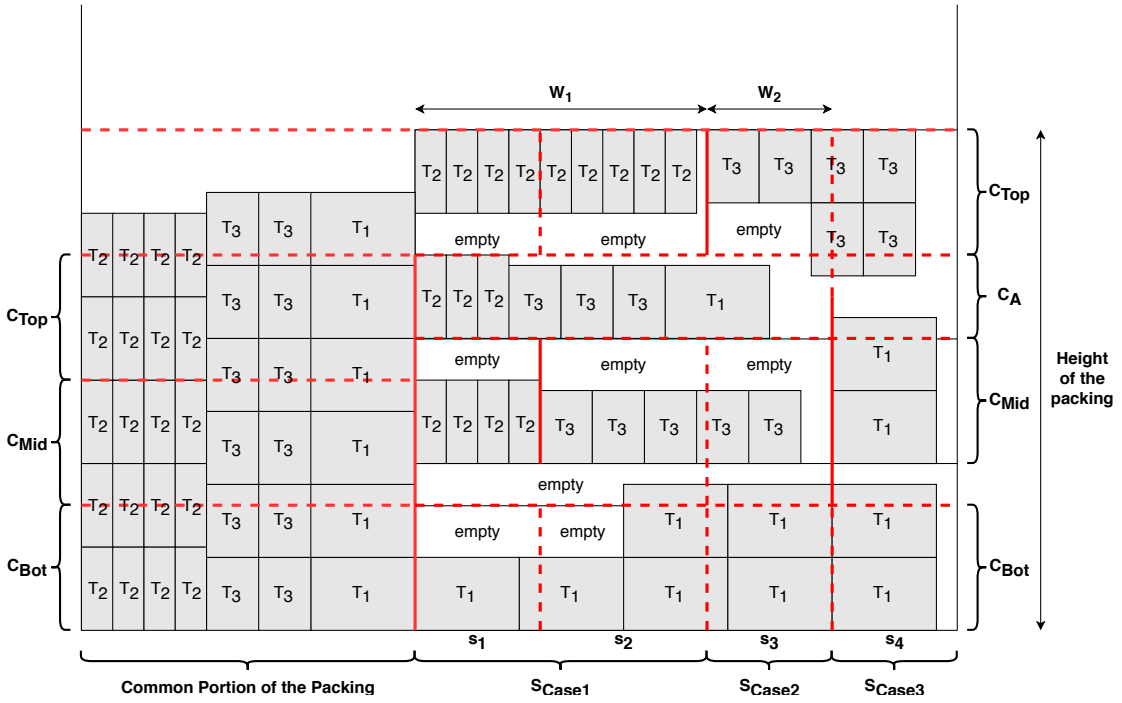


Figure 4.30: A compact representation of the above packing uses $O(k^2)$ numbers, broken down into sections for easier reading. The common portion of the packing is specified as $\{(2,4,5),(3,2,6),(1,1,6)\}$. For the uncommon portion of the packing, the rectangles in each configuration are as follows: C_{Top} is specified as $\{(2,9,1),(3,2,1),(3,2,2)\}$, C_A is specified as $\{(2,3,1),(3,3,1),(1,1,1)\}$, C_{Mid} is specified as $\{(2,4,1),(3,5,1),(1,1,2)\}$, and C_{Bot} is specified as $\{(1,2,1),(1,3,2)\}$.

Theorem 4.6.1. *Algorithm 3TypeRounding produces an integer packing of height at most $\frac{5}{3}$ plus the height of the fractional packing produced by the solution of the linear program using at most $O(k^3)$ operations plus the time needed to compute the solution of the linear program.*

Proof. Algorithm 3TypeRounding first partitions the list of numbers specifying the fractional solution into common and uncommon portions. Identifying the number of rectangles of type T_1 that belong in the common portion of the packing requires identifying the number of rectangles of type T_1 that are packed side-by-side and one on top of the other in each of C_{Top}, C_{Mid}, and C_{Bot}.

The number of rectangles of type T_1 that should be packed side-by-side in the common portion of the packing is equal to the minimum of the number of rectangles of type T_1 that are packed side-by-side in each of C_{Top}, C_{Mid}, and C_{Bot}. The number of rectangles of type T_1 that should be packed one on top of the other in the common portion of the packing is equal to the sum of the number of rectangles of type T_1 that are packed one on top of the other in each of C_{Top}, C_{Mid}, and C_{Bot}. Hence, finding the number of rectangles of type T_1 that belong in the common portion of the packing requires $O(k)$ operations. Therefore, finding the number of rectangles of each type that belong in the common portion of the packing requires $O(k^2)$ operations.

Processing the common portion of the packing according to Section 4.1.1 requires $O(k)$ operations as for $1 \leq i \leq k$, algorithm 3TypeRounding only needs to round up the fractional numbers for each rectangle type T_i.

Sorting the rectangles in each configuration in the uncommon portion of the packing by the fractions f_i requires at most k comparisons per configuration. Since this needs to be done for each configuration, sorting the rectangles in the uncommon portion of the packing requires $O(k^2)$ operations.

Processing a section $s_i \in S_{Case1}$ according to Lemma 4.2.2 requires $O(k)$ operations: the number of fractional rectangles from $C_{Top(i)}$ is computed by multiplying the corresponding number of rectangles packed side-by-side by the decimal part of the number of rectangles packed one on top of the other. This is repeated for $C_{Mid(i)}$ and $C_{Bot(i)}$. Products corresponding to the same rectangle types are added together.

Processing a section $s_i \in S_{Case2}$ according to Section 4.2.4 requires $O(k)$ operations: $C_{Top(i)}$ and $C_{Mid(i}$ are processed as described above; fractional numbers specifying the number of rectangles packed one on top of the other in $C_{Bot(i)}$ are rounded up.

Processing a section $s_i \in S_{Case3}$ according to Section 4.2.5 also requires $O(k)$ operations: for each of $C_{Top(i)}$, $C_{Mid(i)}$, and $C_{Bot(i)}$, fractional numbers specifying the number of rectangles packed one on top of the other are rounded up.

Note that for the vertically split fractional rectangles only a constant number of operations are needed to check whether one part needs to be re-shaped and the other part rounded up. When a rounded up fractional rectangle is combined with a fractional rectangle from C_A to form a whole rectangle, a constant number of numbers in the list need to be changed to reflect the additional rectangle.

Recall that in the uncommon portion of the packing, for configurations that contain more than one type of rectangle, a new section is created at the point where the rectangle type changes in the configuration. Also recall that in the uncommon portion of the packing, within a configuration at most k distinct rectangle types are packed. Therefore, one configuration can contain

at most k sections, one for each rectangle type. Since there are at most k configurations, and each configuration can contain at most k sections, there are $O(k^2)$ sections in the uncommon portion of the packing.

Note that processing a section requires $O(k)$ operations, regardless of which case that section belongs to. In the worst case, $O(k^2)$ sections need to be processed, each requiring $O(k)$ operations, for a total of $O(k^3)$ operations. Partitioning the fractional packing into common and uncommon sections and sorting the rectangles in the uncommon portion of the packing each require $O(k^2)$ operations. Processing the common portion of the packing requires $O(k)$ operations. Therefore, algorithm 3TypeRounding requires $O(k^3 + k^2 + k) = O(k^3)$ operations plus the time needed to compute the solution of the linear program, which was proven in [47] to be polynomial. □

Chapter 5

Strip Packing with Four Rectangle Types

5.1 Overview of the Algorithm

As discussed in Chapter 3, when $k = 4$ there can be either four configurations, three configurations, two configurations, or only one configuration in the solution obtained from the linear program. Our algorithm must correctly round all these possible cases to integer solutions in polynomial time. We consider the cases when the fractional solution has three configurations, two configurations, and only one configuration in Sections 5.3-5.5.

The configurations are packed one on top of the other as shown in Figure 5.1; recall that we sometimes simplify the figures by not showing all of the rectangles in each configuration. Let the configuration packed at the top be called C_1, the configuration packed beneath C_1 be called C_2, and so on. Note that we can change the order of the configurations if necessary.

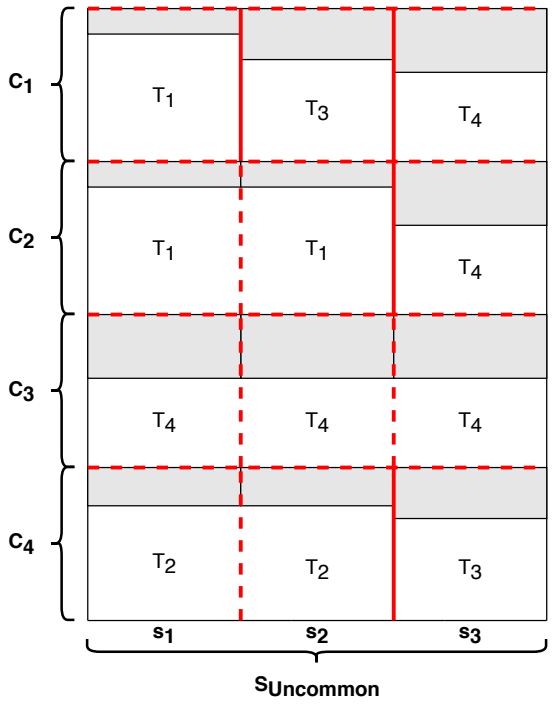


Figure 5.1: The four configurations are packed one on top of the other.

5.1.1 Rounding Fractional Rectangles

We divide the fractional solution obtained from the linear program into common and uncommon sections by horizontally rearranging the rectangles within each configuration, as described in Chapter 4. The fractional rectangles in the common section are rounded up, increasing the height of the fractional solution by at most 1.

The uncommon portion of the packing is divided into vertical sections, as described in Chapter 4, and again we consider one vertical section at a time from left to right. Recall that within a section s_i, each configuration will only have a single rectangle type. For each section s_i let $f_{1(i)}$, $f_{2(i)}$, $f_{3(i)}$, and $f_{4(i)}$ represent the fraction of the rectangle packed at the top of $C_{1(i)}$, $C_{2(i)}$, $C_{3(i)}$, and $C_{4(i)}$, respectively (see Figure 5.2).

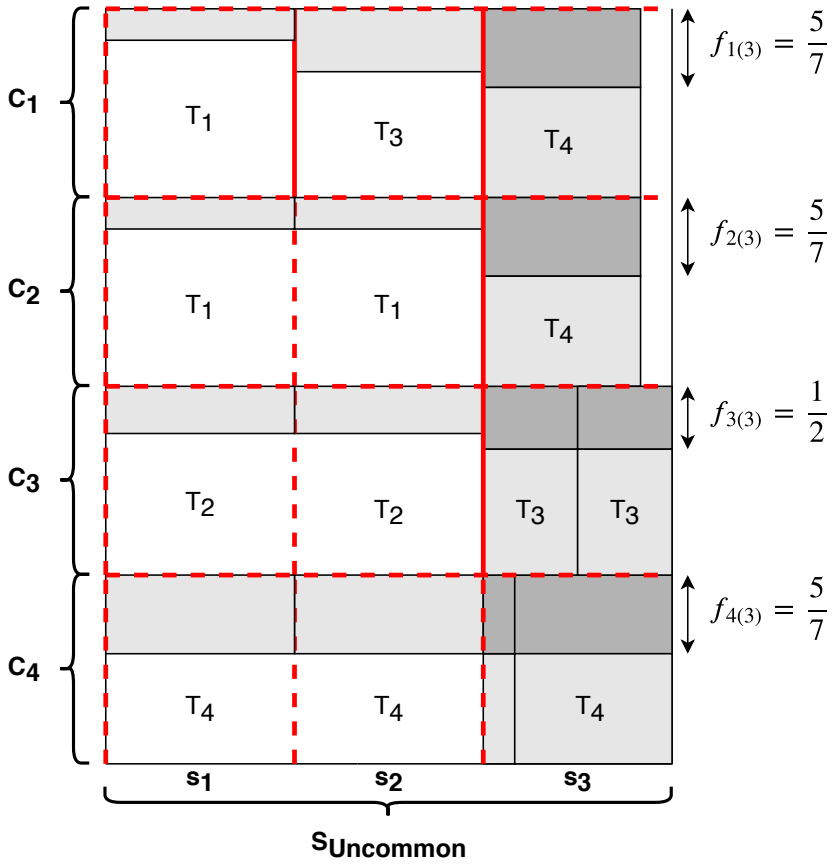


Figure 5.2: Fractions $f_{1(3)}$, $f_{2(3)}$, $f_{3(3)}$, and $f_{4(3)}$ for section s_3.

Algorithm 4TypeRounding, described below, transforms a fractional packing obtained by solving the linear program into an integer packing. This transformation increases the total height of the packing by at most $\frac{5}{2}$, as we show later.

Algorithm 5.1 4TypeRounding(*FractionalSolution*)

Input: An optimal fractional packing *FractionalSolution*.
Output: The height of the integer packing h obtained from *FractionalSolution*.
Divide *FractionalSolution* into common and uncommon portions by horizontally rearranging rectangles as described in Chapter 4.
Round up the fractional rectangles of the common portion of the packing.
for each configuration C in *FractionalSolution* **do**
    Sort the uncommon portion of C by non-decreasing value of f_i.
end for
if number of configurations in *FractionalSolution* = 4 **then**
    return $4ConfigurationRounding$(*FractionalSolution*).
else if number of configurations in *FractionalSolution*= 3 **then**
    return $3ConfigurationRounding$(*FractionalSolution*).
else if number of configurations in *FractionalSolution*= 2 **then**
    return $2ConfigurationRounding$(*FractionalSolution*).
else
    return $1ConfigurationRounding$(*FractionalSolution*).
end if

5.2 Four Configurations

When the solution to the linear program has four configurations, our algorithm considers three cases, depending on the values of the four fractions $f_{1(i)}$, $f_{2(i)}$, $f_{3(i)}$, and $f_{4(i)}$ for each section $s_i \in S$:

- **Case 1:** $f_{1(i)} + f_{2(i)} \leq \frac{1}{2}$ and $f_{3(i)} + f_{4(i)} \leq \frac{1}{2}$.

Let i be the smallest index for which $f_{1(i)} + f_{2(i)} > \frac{1}{2}$ or $f_{3(i)} + f_{4(i)} > \frac{1}{2}$. If such an index does not exist then Case 2 and Case 3 do not need to be considered; otherwise, we (re)order the configurations so that $f_{3(j)} + f_{4(j)} > \frac{1}{2}$ for all $j \geq i$. Now we can define Case 2 and Case 3:

- **Case 2:** $f_{1(i)} + f_{2(i)} \leq 1$ and $f_{3(i)} + f_{4(i)} > \frac{1}{2}$, and
- **Case 3:** $f_{1(i)} + f_{2(i)} > 1$ and $f_{3(i)} + f_{4(i)} > \frac{1}{2}$.

Note that for a section $s_i \in S_{Case1}$, $f_{3(i)}+f_{4(i)} \leq \frac{1}{2}$, but for a section $s_j \in S_{Case2}$, $f_{3(j)}+f_{4(j)} > \frac{1}{2}$. Therefore, the rectangles in either C_3 or C_4 create a vertical division separating Case 1 and Case 2. We (re)order C_3 and C_4 so that the rectangles in C_4 create a vertical division separating Case 1 and Case 2. Similarly, for a section $s_i \in S_{Case2}$ and a section $s_j \in S_{Case3}$, we (re)order C_1 and C_2 to ensure that the rectangles in C_1 create a vertical division between Case 2 and Case 3. Finally, for a packing with a section $s_i \in S_{Case1}$ and an adjacent section $s_j \in S_{Case3}$, we (re)order C_1 and C_2 to ensure that the rectangles in C_1 create a vertical division between Case 1 and Case 3, and we (re)order C_3 and C_4 to ensure that the rectangles in C_4 create a vertical division between Case 1 and Case 3.

We flip C_1 and C_3 upside down as shown in Figure 5.3. The algorithm for rounding fractional rectangles into whole ones when there are four configurations is shown below.

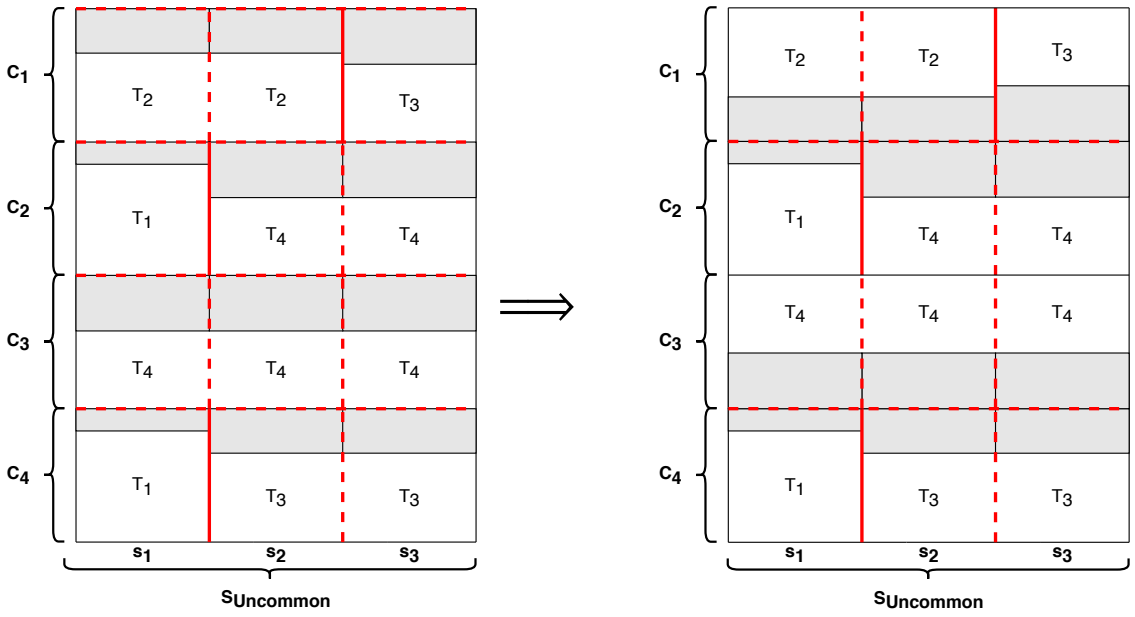


Figure 5.3: C_1 and C_3 are flipped upside down.

Note that this algorithm only depends on the number of configurations and not on the number of rectangle types.

Algorithm 5.2 4ConfigurationRounding(*FractionalSolution*)

Input: An optimal fractional solution *FractionalSolution* with 4 configurations C_1, C_2, C_3, and C_4.
Output: The height of an integer packing h obtained from $FractionalSolution$.
Flip C_1 and C_3 upside down.
Initialize set S to contain all the vertical sections in the uncommon portion of $FractionalSolution$.
Initialize sets S_{Case1}, S_{Case2}, and S_{Case3} to be empty sets.
for $s_i \in S$ **do**
    if $f_{1(i)} + f_{2(i)} \leq \frac{1}{2}$ and $f_{3(i)} + f_{4(i)} \leq \frac{1}{2}$ **then**
        $S_{Case1} = S_{Case1} \cup s_i$.
    else if $f_{1(i)} + f_{2(i)} \leq 1$ and $f_{3(i)} + f_{4(i)} > \frac{1}{2}$ **then**
        $S_{Case2} = S_{Case2} \cup s_i$.
    else if $f_{1(i)} + f_{2(i)} > 1$ and $f_{3(i)} + f_{4(i)} > \frac{1}{2}$ **then**
        $S_{Case3} = S_{Case3} \cup s_i$.
    end if
end for
h_0 = height of *FractionalSolution*.
h_1 = height increase after processing S_{Case1} according to Section 5.2.1.
h_2 = height increase after processing S_{Case2} according to Section 5.2.2.
h_3 = height increase after processing S_{Case3} according to Section 5.2.3.
return $h_0 + \max(h_1, h_2, h_3)$.

Theorem 5.2.1. *Algorithm 4ConfigurationRounding produces an integer packing of height at most $\frac{5}{2}$ plus the height of the fractional packing produced by the solution of the linear program.*

We prove this theorem in the following sections of this chapter.

5.2.1 Case 1. $f_{1(i)} + f_{2(i)} \leq \frac{1}{2}$ and $f_{3(i)} + f_{4(i)} \leq \frac{1}{2}$

Let the width of S_{Case1} be W_1. For every section $s_i \in S_{Case1}$ we remove the fractional rectangles in $R_{1(i)}$, $R_{2(i)}$, $R_{3(i)}$, and $R_{4(i)}$, including the parts r_{Case1} for vertically split fractional rectangles r; re-shape them so that they have the full height of a rectangle of the same type but only a fraction of its width; and pack them side-by-side in a region of width W_1 and height 1. This region, hereafter referred to as C_{A1} (see Figure 5.4), is created by shifting all rectangles in C_1 upwards, including rectangles in S_{Case2} and S_{Case3}, until there is empty space of height 1 between C_1 and C_2 (see Figure 5.5). After shifting the rectangles the tops of the topmost rectangles in C_1 must lie on a common line.

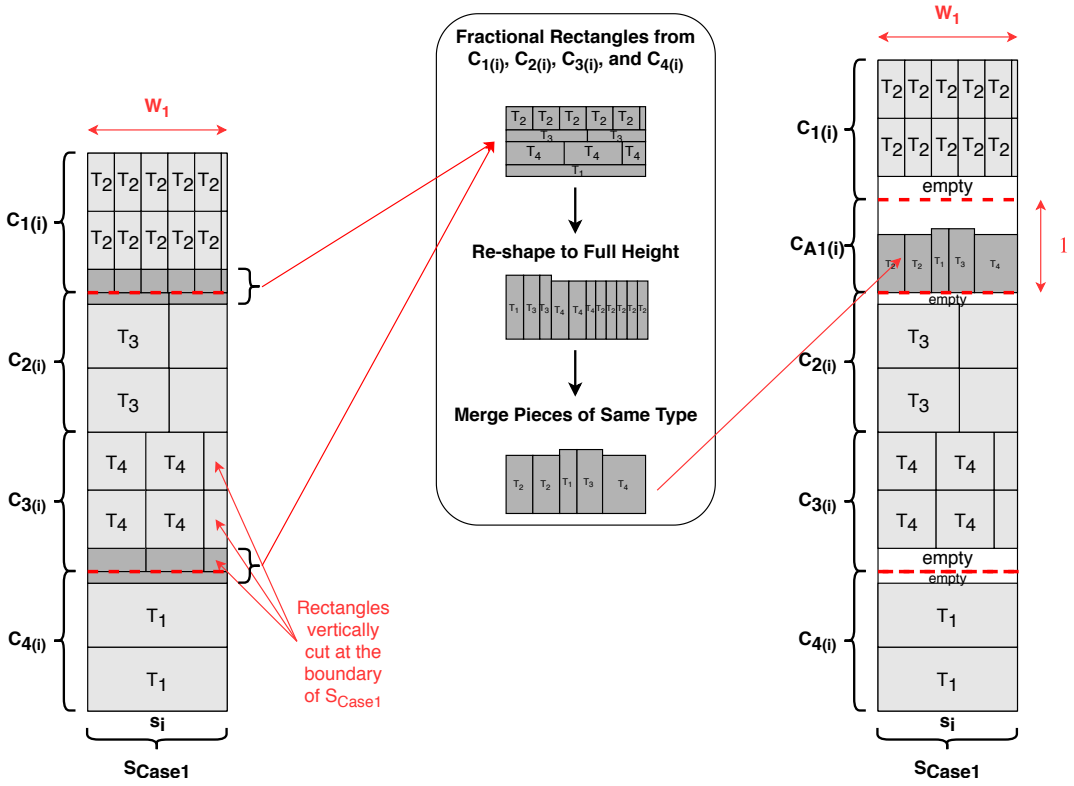


Figure 5.4: For every section $s_i \in S_{Case1}$ the fractional rectangles in $R_{1(i)}$, $R_{2(i)}$, $R_{3(i)}$, and $R_{4(i)}$ are removed, re-shaped, and packed side-by-side in C_{A1}; a whole rectangle is formed whenever a sufficient number of fractional pieces of the same type have been packed in C_{A1}. All rectangles in C_1 are shifted upwards until there is empty space of height 1 between C_1 and C_2.

In order to pack the fractional rectangles from $R_{1(i)}$, $R_{2(i)}$, $R_{3(i)}$, and $R_{4(i)}$ in C_{A1}, we first reshape them so that they have the full height of a rectangle of the corresponding type but only a fraction of its width, similar to Chapter 4 (see Figure 5.4). By Lemma 4.2.2, the fractional rectangles from S_{Case1} that were removed, reshaped, and packed in C_{A1} have total width at most W_1. Since the width of C_{A1} within S_{Case1} is W_1, the re-shaped rectangles can be packed in C_{A1}.

Corollary 5.2.2. *If $S = S_{Case1}$, then any leftover fractional rectangles after processing Case 1 as described in this section can be discarded. Therefore, in this case algorithm 4ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program.*

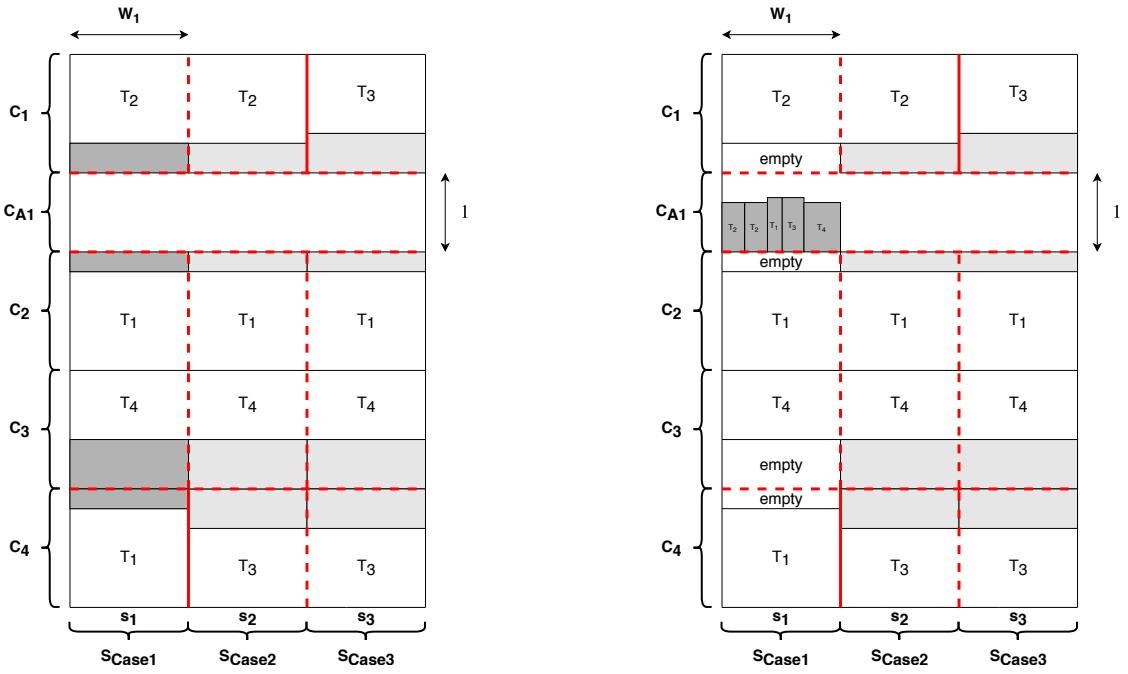


Figure 5.5: (a) C_{A1} is created by shifting all rectangles in C_1 upwards, including rectangles in S_{Case2} and S_{Case3}, until there is empty space of height 1 between C_1 and C_2. (b) For every section $s_i \in S_{Case1}$ the fractional rectangles in $R_{1(i)}$, $R_{2(i)}$, $R_{3(i)}$, and $R_{4(i)}$ are removed, re-shaped, and packed side-by-side in C_{A1} within S_{Case1}.

Proof. For every section $s_i \in S_{Case1}$ the fractional rectangles in $R_{1(i)}$, $R_{2(i)}$, $R_{3(i)}$, and $R_{4(i)}$ are reshaped and packed side-by-side in C_{A1}. By Lemma 4.2.3 there can be at most one fractional rectangle of each type in C_{A1}. Let F be the set of these fractional rectangles.

The fractional rectangles in the solution of the linear program must sum to an integer number of whole rectangles. Therefore, for each rectangle $r \in F$ there must have been more fractional rectangles in the solution of the linear program of the same type as r in the common portion of the packing of sufficient size to form an integer number of whole rectangles. Recall that any fractional rectangles in the common portion of the packing were rounded up. Therefore, fractions of rectangles of the same type as the rectangles in F and of total area equal to the area of F have been added when rounding up the fractional rectangles in the common portion of the packing. Thus, the fractional rectangles in F can be discarded.

Therefore, if $S = S_{Case1}$, there are no leftover fractional rectangles after processing Case 1 as described in this section. Algorithm 4ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program. □

5.2.2 Case 2. $f_{1(i)} + f_{2(i)} \leq 1$ and $f_{3(i)} + f_{4(i)} > \frac{1}{2}$

Let the width of S_{Case2} be W_2. For every section $s_i \in S_{Case2}$ we remove the fractional rectangles in $R_{1(i)}$ and $R_{2(i)}$, including the parts r_{Case2} for vertically split fractional rectangles r; re-shape them so that they have the full height of a rectangle of the same type but only a fraction of its width; and pack them side-by-side in a region of width W_2 and height 1 in C_{A1}. The fractional rectangles in $R_{3(i)}$ and $R_{4(i)}$, including the parts r_{Case2} for vertically split fractional rectangles, are rounded up; hereafter the region of the packing created to fit the rounded up rectangles is referred to as C_{A2} (see Figure 5.6).

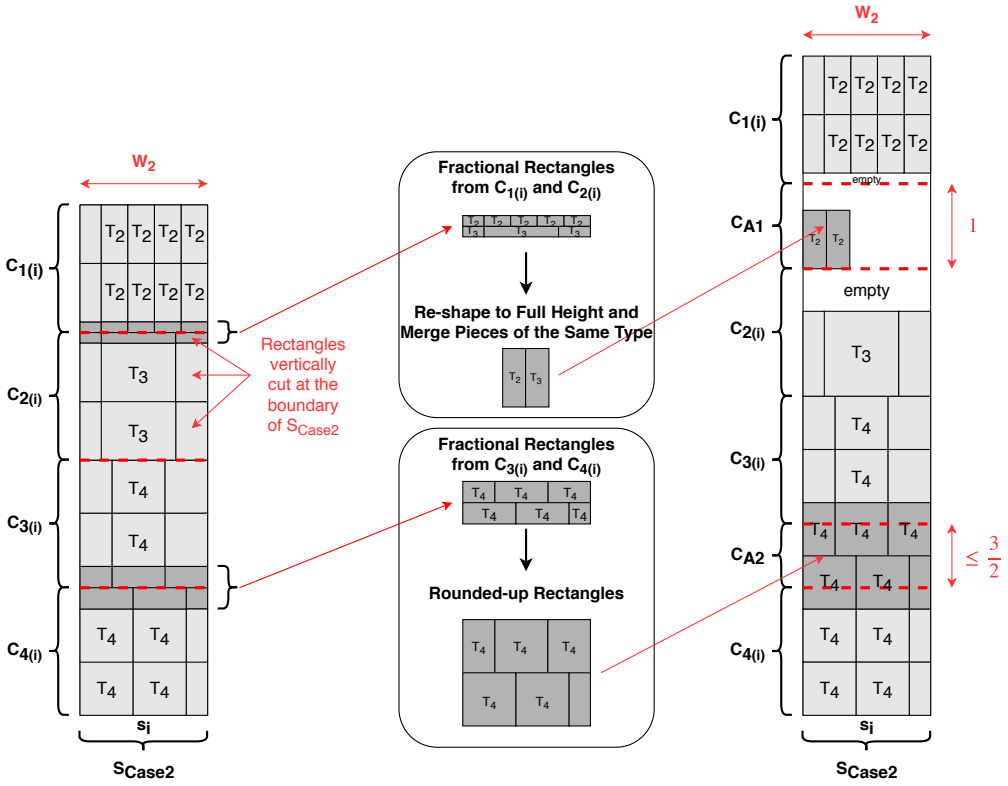


Figure 5.6: For every section $s_i \in S_{Case2}$ the fractional rectangles in $R_{1(i)}$ and $R_{2(i)}$ are removed, reshaped, and packed side-by-side in C_{A1}; a whole rectangle is formed whenever a sufficient number of fractional pieces of the same type have been packed in C_{A1}. The fractional rectangles in $R_{3(i)}$ and $R_{4(i)}$ are rounded up. All rectangles in $C_1 \cup C_2 \cup C_3$ are shifted upwards until there is empty space of height 1 between C_1 and C_2 and until the rounded up rectangles in $R_{3(i)}$ and $R_{4(i)}$ fit between $C_{3(i)}$ and $C_{4(i)}$.

After the fractional rectangles in $R_{1(i)}$ and $R_{2(i)}$ are removed and the fractional rectangles in $R_{3(i)}$ and $R_{4(i)}$ are rounded up, all rectangles in $C_1 \cup C_2 \cup C_3$ are shifted upwards, including rectangles in S_{Case1} and S_{Case3}, until there is empty space of height 1 between C_1 and C_2 and the rounded up rectangles in $R_{3(i)}$ and $R_{4(i)}$ fit between $C_{3(i)}$ and $C_{4(i)}$ (see Figure 5.7). After shifting the rectangles the tops of the topmost rectangles in C_1 must lie on a common line, the bottoms of the bottommost rectangles in C_2 must lie on a common line, and the tops of the topmost rectangles in C_3 must also lie on a common line.

Recall that the height of any rectangle is at most 1; therefore, the height increase caused by rounding up for the fractional rectangles in $R_{3(i)}$ and $R_{4(i)}$ is at most $\frac{3}{2}$. Combining this with the

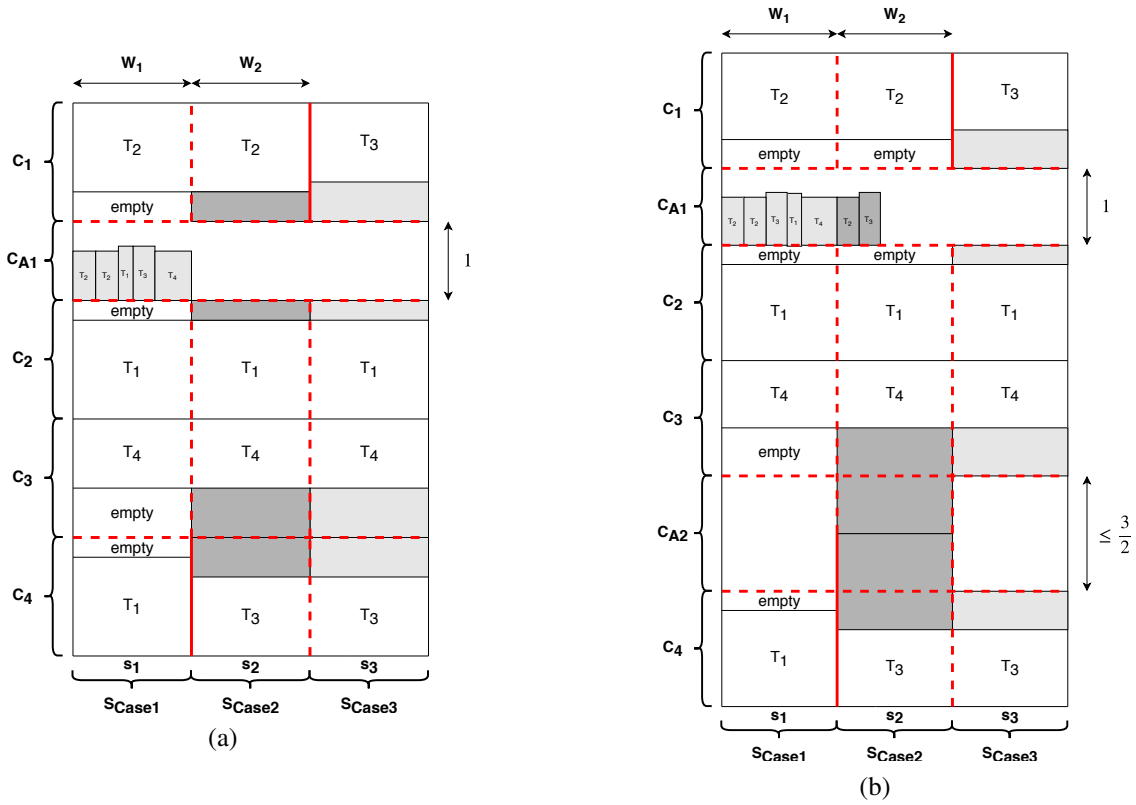


Figure 5.7: (a) After the fractional rectangles in $R_{1(i)}$ and $R_{2(i)}$ are removed and the fractional rectangles in $R_{3(i)}$ and $R_{4(i)}$ are rounded up, all rectangles in $C_1 \cup C_2 \cup C_3$ are shifted upwards, including rectangles in S_{Case1}, S_{Case3}, and S_{Case4}, until there is empty space of height 1 between C_1 and C_2 and the rounded up rectangles in $R_{3(i)}$ and $R_{4(i)}$ fit. (b) For every section $s_i \in S_{Case2}$, the fractional rectangles from $R_{1(i)}$ and $R_{2(i)}$ are re-shaped and packed side-by-side in C_{A1}.

height increase caused by leaving space of height 1 between C_1 and C_2, the total increase in height of each $s_i \in S_{Case2}$ is at most $\frac{5}{2}$.

In order to pack in C_{A1} the fractional rectangles from $R_{1(i)}$ and $R_{2(i)}$, we first reshape them so that they have the full height of a rectangle of the corresponding type but only a fraction of its width. Recall that by Lemma 4.2.2 these re-shaped rectangles have total width at most W_2. Since the width of C_{A1} within S_{Case2} is W_2, the re-shaped rectangles can be packed in C_{A1}.

Lemma 5.2.3. *If there is a fractional rectangle $r \in S_{Case1} \cap S_{Case2}$ located in C_3 or C_4 in the solution of the linear program, after processing the fractional rectangles as described above at most one whole rectangle of the same type as r can be formed using the fractional rectangles within C_{A1} and the fractional part r_{Case2}. This whole rectangle can be packed without further increasing the height of the packing and without overlapping any other rectangles.*

Proof. Recall that we ordered the configurations so that the rectangles in C_4 create a vertical division separating Case 1 and Case 2. Therefore, there is at most one vertically split fractional rectangle $r \in S_{Case1} \cap S_{Case2}$ and it must be located in C_3.

The fractional rectangles from S_{Case1} were packed in C_{A1} within S_{Case1} and C_{A2} within S_{Case1} is empty. Recall that within C_{A1} when sufficient fractional rectangles of the same type

are packed, they merge to become whole rectangles. By Lemma 4.2.3 there can be at most one fractional rectangle of each type leftover in C_{A1} after merging fractional pieces. Let F be the set of fractional rectangles in C_{A1}.

There is empty space in C_{A2} within S_{Case1} right beside r_{Case2} of sufficient width to extend the width of r_{Case2} to the width of a whole rectangle of the same type as r_{Case2}, because the fractional rectangle r was originally packed in both S_{Case1} and S_{Case2} in the solution of the linear program (see Figure 5.8a). Furthermore, there is empty space between C_3 and C_4 within S_{Case1} and S_{Case2} of sufficient height to pack a whole rectangle of the same type as r_{Case2}, because the fractional part r_{Case2} was rounded up so that it has the full height of a rectangle of its type (see Figure 5.8b). If there is a fractional rectangle $r' \in F$ of the same type as r_{Case2} such that r' and the rounded up r_{Case2} form a whole rectangle, then r' is removed from F and from C_{A1} and r_{Case2} is replaced by a whole rectangle of its same type. Therefore, the fractional part r_{Case2} can be replaced by a whole rectangle without further increasing the height of the packing and without overlapping any other rectangles. (see Figure 5.8c).

If $S_{Case3} = \emptyset$, then the remaining rectangles in F are discarded because fractional rectangles of the same type in the solution of the linear program sum to an integer number of whole rectangles. Therefore, for each rectangle $r \in F$, there must be more fractional rectangles of the same type as r in either $R_{3(i)}$, $R_{4(i)}$, or the common portion of the packing of sufficient size to form an integer number of whole rectangles. Since all fractional rectangles in $R_{3(i)}$, in $R_{4(i)}$, and in the common portion of the packing are rounded up, then it must be the case that fractions of rectangles of the same type as the rectangles in F and of total area at least the area of F have been added when rounding up the fractional rectangles in $R_{3(i)}$, $R_{4(i)}$, and the common portion of the packing. Thus, the fractional rectangles in F can be discarded.

Note that if $S_{Case3} \neq \emptyset$, then we do not discard any leftover fractional rectangles yet as we might need to use them later. □

Corollary 5.2.4. *If $S = S_{Case2}$, then any leftover fractional rectangles after processing Case 2 as described in this section can be discarded. Therefore, in this case algorithm 4ConfigurationRounding produces an integer packing of height at most $\frac{5}{2}$ plus the height of the fractional packing produced by the solution of the linear program.*

Proof. For every section $s_i \in S_{Case2}$ the fractional rectangles in $R_{3(i)}$ and $R_{4(i)}$ are rounded up and the fractional rectangles in $R_{1(i)}$ and $R_{2(i)}$ are re-shaped and rectangles of the same type are packed side-by-side in C_{A1}. By Lemma 4.2.3 there can be at most one fractional rectangle of each type leftover after this merging. Let F be the set of these leftover fractional rectangles.

Recall that the fractional rectangles of the same type in the solution of the linear program sum to an integer number of whole rectangles. Therefore, for each rectangle $r \in F$, there must be more fractional rectangles of the same type as r in either $R_{3(i)}$, $R_{4(i)}$, or the common portion of the packing of sufficient size to form an integer number of whole rectangles. Since all fractional rectangles in $R_{3(i)}$, $R_{4(i)}$, and in the common portion of the packing are rounded up, then it must be the case that fractions of rectangles of the same type as the rectangles in F and of total area at least the area of F have been added when rounding up the fractional rectangles in $R_{3(i)}$, $R_{4(i)}$, and the common portion of the packing. Thus, the fractional rectangles in F can be discarded.

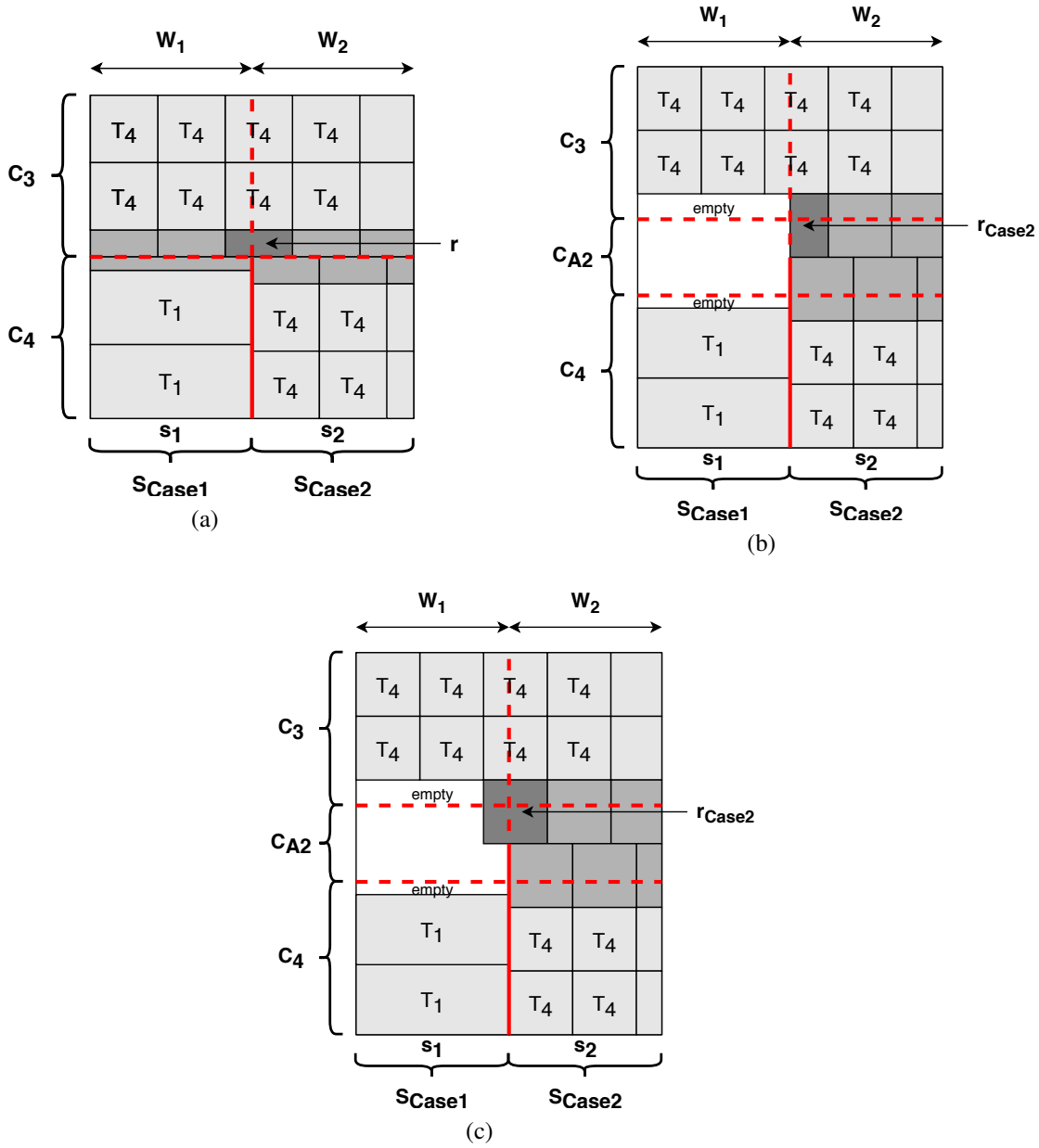


Figure 5.8: (a) A fractional rectangle $r \in S_{Case1} \cap S_{Case2}$ located in C_3 in the solution of the linear program. (b) The rounded up part r_{Case2}. (c) The fractional part r_{Case2} can be replaced by a whole rectangle without further increasing the height of the packing and without overlapping any other rectangles.

Therefore, if $S = S_{Case2}$, there are no leftover fractional rectangles after processing Case 2 as described in this section (see Figure 4.17). Algorithm 4ConfigurationRounding produces an integer packing of height at most $\frac{5}{2}$ plus the height of the fractional packing produced by the solution of the linear program. □

5.2.3 Case 3. $f_{1(i)} + f_{2(i)} > 1$ and $f_{3(i)} + f_{4(i)} > \frac{1}{2}$

For every section $s_i \in S_{Case3}$ the fractional rectangles in $R_{1(i)}$, $R_{2(i)}$, $R_{3(i)}$, and $R_{4(i)}$ are rounded up, including the parts r_{Case3} for vertically split fractional rectangles r (see Figure 5.9). After the fractional rectangles in $R_{1(i)}$, $R_{2(i)}$, $R_{3(i)}$, and $R_{4(i)}$ are rounded up, all rectangles in $C_1 \cup C_2 \cup C_3$ are shifted upwards, including rectangles in S_{Case1} and S_{Case2}, until the rounded up rectangles fit between C_1 and C_2 and between C_3 and C_4 (see Figure 5.10). After shifting the rectangles the tops of the topmost rectangles in C_1 must lie on a common line, the bottoms of the bottommost rectangles in C_2 must lie on a common line, and the tops of the topmost rectangles in C_3 must lie on a common line. Recall that the height of any rectangle is at most 1; therefore, this process increases the height of the packing by at most $\frac{5}{2}$.

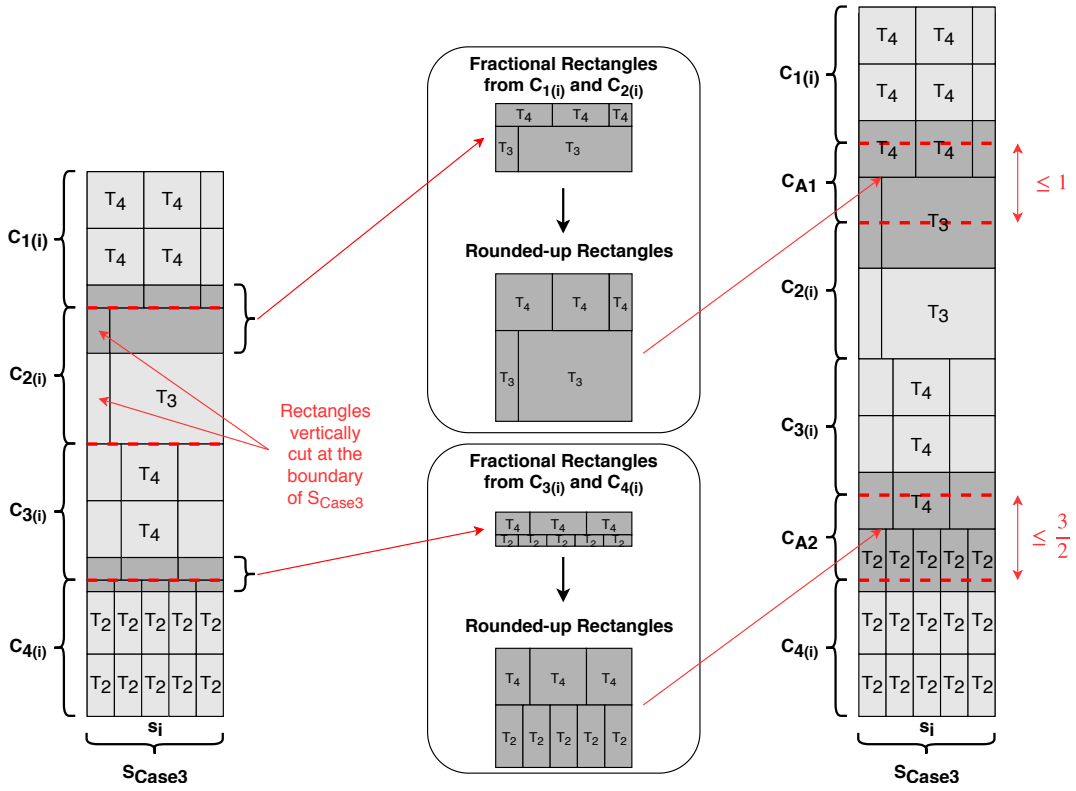


Figure 5.9: For every section $s_i \in S_{Case3}$, the fractional rectangles in $R_{1(i)}$, $R_{2(i)}$, $R_{3(i)}$, and $R_{4(i)}$ are rounded up. All rectangles in $C_1 \cup C_2 \cup C_3$ are shifted upwards until the rounded up rectangles fit between C_1 and C_2 and between C_3 and C_4.

Lemma 5.2.5. *Let $S_{Case2} = \emptyset$. If there is a fractional rectangle $r \in S_{Case1} \cap S_{Case3}$ located in C_3 or C_4 in the solution of the linear program, or a fractional rectangle $r \in S_{Case1} \cap S_{Case3}$ located in C_1 or C_2 in the solution of the linear program, after processing the fractional rectangles as described above at most two whole rectangles of the same type as r can be formed using the fractional rectangles within C_{A1} and the fractional parts r_{Case3}. These whole rectangles can be packed without further increasing the height of the packing and without overlapping any other rectangles.*

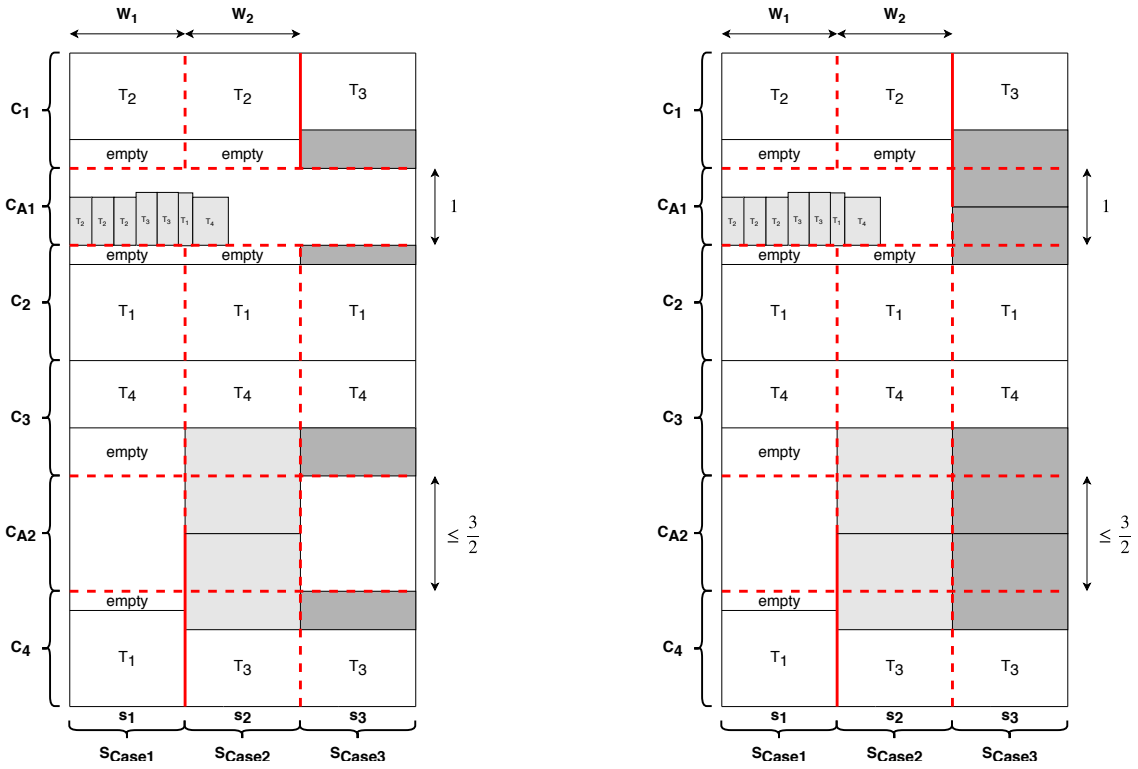


Figure 5.10: After the fractional rectangles in $R_{1(i)}$, $R_{2(i)}$, $R_{3(i)}$, and $R_{4(i)}$ are rounded up, all rectangles in $C_1 \cup C_2 \cup C_3$ are shifted upwards, including rectangles in S_{Case1} and S_{Case2}, until the rounded up rectangles fit.

Proof. Recall that we ordered the configurations so that the rectangles in C_4 create a vertical division separating Case 1 and Case 3. Therefore, there is at most one vertically split fractional rectangle $r \in S_{Case1} \cap S_{Case3}$ in C_3 or C_4 and it must be located in C_3. Additionally, recall that the fractional rectangles from S_{Case1} were packed in C_{A1} within S_{Case1} and that C_{A2} within S_{Case1} is empty.

Note that a similar lemma as Lemma 5.2.3 holds for $r \in S_{Case1} \cap S_{Case3} \cap C_3$, because the fractional part r_{Case1} was re-shaped and packed in C_{A1} and the fractional part r_{Case3} was rounded up; hence, the same argument used to prove Lemma 5.2.3 shows that r_{Case3} can either be replaced by a whole rectangle of its same type without further increasing the height of the packing and without overlapping any other rectangles or it can be discarded.

Recall that we ordered the configurations so that the rectangles in C_1 create a vertical division separating Case 1 and Case 3. Therefore, there is at most one vertically split fractional rectangle $r \in S_{Case1} \cap S_{Case3}$ in C_1 or C_2 and it must be located in C_2. Additionally, recall that within C_{A1} when sufficient fractional rectangles of the same type are packed, they merge to become whole rectangles. By Lemma 4.2.3, there can be at most one fractional rectangle of each type leftover in C_{A1} after this merging. Let F be the set of these leftover fractional rectangles.

The part r_{Case3} was rounded up so that it has the full height of a rectangle of its type but

only a fraction of its width (see Figure 5.11b). If there is a fractional rectangle $r' \in F$ of the same type as r_{Case3} such that r' and the rounded up r_{Case3} form a whole rectangle, then r' is removed from F; then r' is shifted rightwards until it is side-by-side with the rounded up r_{Case3} (see Figure 5.11c). Next, r' is shifted downwards to form a whole rectangle with the rounded up r_{Case3} (see Figure 5.11d). Note that r' can be shifted downwards because the fractional rectangle r was packed by the solution of the linear program between S_{Case1} and S_{Case3} so there is enough empty space beside r_{Case3} to put r' (see Figure 5.11a). Therefore, r' and the rounded up r_{Case3} form a whole rectangle without further increasing the height of the packing and without overlapping any other rectangles. □

Note that rectangles $r \in S_{Case1} \cap S_{Case3} \cap C_2$ and $r' \in S_{Case1} \cap S_{Case3} \cap C_3$ can be present at the same time. Together, at most two whole rectangles can be formed that must be packed. Only after attempting to pack both of these whole rectangles can the remaining fractional rectangles from F and the unused fractional parts r_{Case3} be discarded.

Corollary 5.2.6. *If there is a fractional rectangle $r \in S_{Case2} \cap S_{Case3}$ located in C_1 or C_2 in the solution of the linear program, after processing the fractional rectangles as described above at most one whole rectangle of the same type as r can be formed using the fractional rectangles within C_{A1} and the fractional part r_{Case3}. This whole rectangle can be packed without further increasing the height of the packing and without overlapping any other rectangles.*

Proof. A similar proof as that of Lemma 5.2.5 can be used. Recall that we ordered the configurations so that the rectangles in C_1 create a vertical division separating Case 2 and Case 3. Therefore, there is at most one vertically split fractional rectangle $r \in S_{Case2} \cap S_{Case3}$ in C_1 or C_2 and it must be located in C_2. By Lemma 4.2.3 there can be at most one fractional rectangle of each type leftover in C_{A1} after merging fractional pieces. Let F be the set of fractional rectangles in C_{A1}.

If there is a fractional rectangle $r' \in F$ of the same type as r_{Case3} such that r' and the rounded up r_{Case3} form a whole rectangle, then r' is removed from F; then r' is shifted rightwards until it is side-by-side with the rounded up r_{Case3}, and finally r' is shifted downwards to form a whole rectangle with the rounded up r_{Case3} (see Figure 5.11). Refer to Lemma 5.2.5 for an explanation on why there must be sufficient empty space beside r_{Case3} to form a whole rectangle. Therefore, r' and the rounded up r_{Case3} form a whole rectangle without further increasing the height of the packing and without overlapping any other rectangles.

The remaining rectangles in F are discarded because fractional rectangles of the same type in the solution of the linear program sum to an integer number of whole rectangles. Therefore, for each fractional rectangle $r \in F$, there must be other fractional rectangles of the same type elsewhere in the packing that were rounded up by at least the same area as r. Thus, the fractional rectangles in F can be discarded. Note that if the rounded up r_{Case3} and r' do not form a whole rectangle, then r_{Case3} is discarded as well. □

Corollary 5.2.7. *If there are fractional rectangles $r \in S_{Case1} \cap S_{Case2} \cap S_{Case3}$, then they must be located in C_2 or C_3 in the solution of the linear program. After processing the fractional rectangles as described above, at most two whole rectangles of the same types as r can be formed. These whole rectangles can be packed without further increasing the height of the packing and without overlapping any other rectangles.*

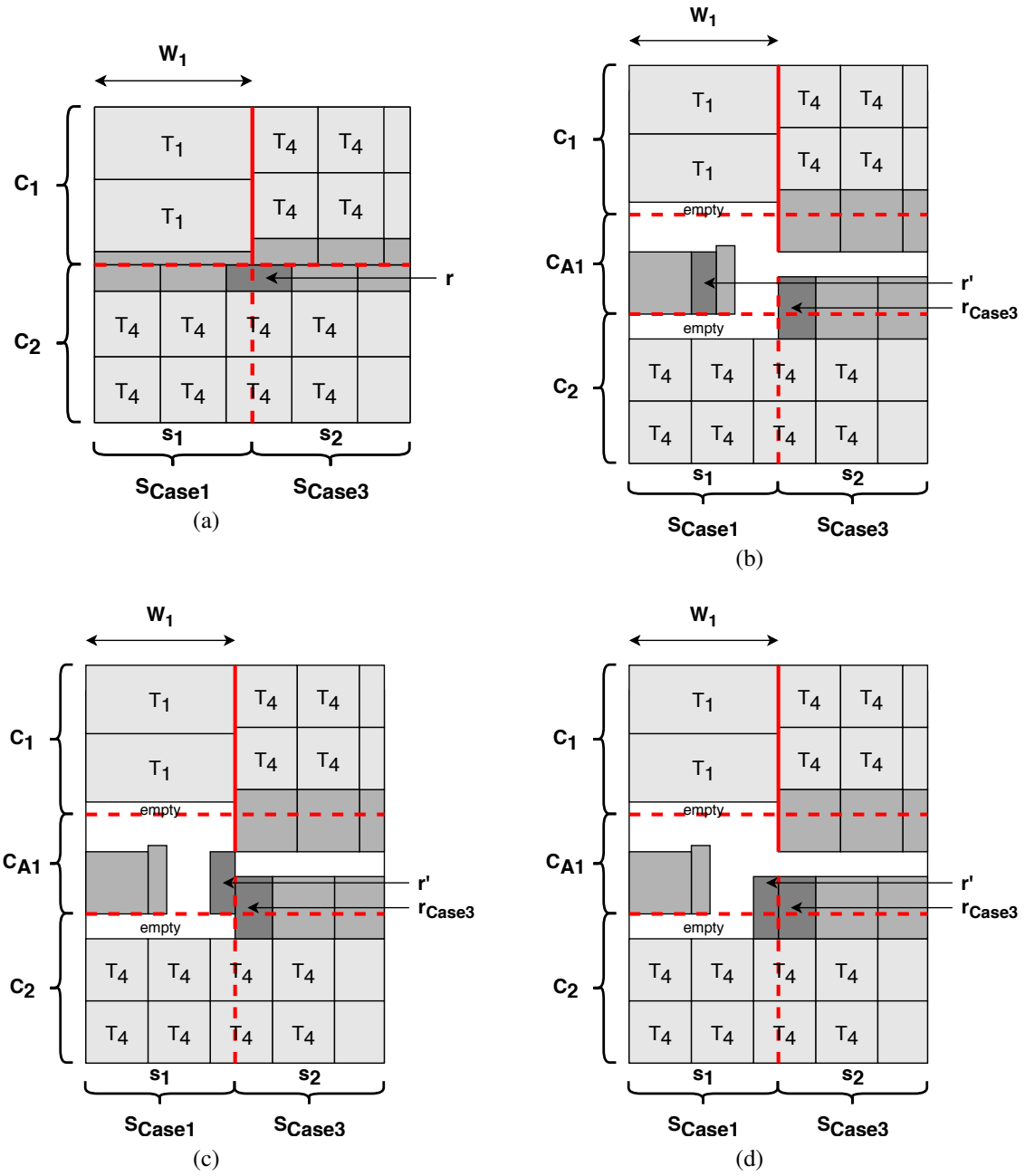


Figure 5.11: (a) A fractional rectangle $r \in S_{Case1} \cap S_{Case3} \cap C_2$ was packed by the solution of the linear program. (b) A fractional rectangle $r' \in F$ of the same type as the rounded up r_{Case3} can form a whole rectangle with the rounded up r_{Case3}. (c) The fractional rectangle r' is shifted rightwards until it is side-by-side with the rounded up r_{Case3}. (d) The fractional rectangle r' is shifted downwards until it forms a whole rectangle with the rounded up r_{Case3}.

Proof. Recall that we ordered the configurations so that the rectangles in C_4 create a vertical division separating Case 1 and Case 2. Additionally, recall that we ordered the configurations so that the rectangles in C_1 create a vertical division separating Case 2 and Case 3. Therefore, a

fractional rectangle $r \in S_{Case1} \cap S_{Case2} \cap S_{Case3}$ cannot be located in C_1 or C_4, as the rectangles in S_{Case1} and S_{Case2} of C_4 are of different types and the rectangles in S_{Case2} and S_{Case3} of C_1 are of different types. There is at most one vertically split fractional rectangle $r \in S_{Case1} \cap S_{Case2} \cap S_{Case3} \cap C_2$ and at most one vertically split fractional rectangle $r' \in S_{Case1} \cap S_{Case2} \cap S_{Case3} \cap C_2$.

Also recall that the fractional rectangles from S_{Case1} and the fractional rectangles from S_{Case2} in C_1 or C_2 were packed in C_{A1}. By Lemma 4.2.3 there can be at most one fractional rectangle of each type in C_{A1}. Let F be the set of these fractional rectangles.

Note that Lemma 5.2.5 holds even if $r \in S_{Case1} \cap S_{Case2} \cap S_{Case3} \cap C_2$, because the fractional parts r_{Case1} and r_{Case2} were both re-shaped and packed in C_{A1}; hence, the same argument used to prove Lemma 5.2.5 shows that either r_{Case3} can be replaced by a whole rectangle of its same type without further increasing the height of the packing and without overlapping any other rectangles, or the rounded up r_{Case3} and the rectangles in F of the same type as r_{Case3} can be discarded.

Note that Lemma 5.2.3 holds even if $r \in S_{Case1} \cap S_{Case2} \cap S_{Case3} \cap C_3$, because the fractional parts r_{Case2} and r_{Case3} are both rounded up and are essentially one larger fractional rectangle; hence, the same argument used to prove Lemma 5.2.3 shows that either r_{Case2} and r_{Case3} can be be replaced by a whole rectangle of its same type without further increasing the height of the packing and without overlapping any other rectangles, or the rounded up r_{Case2}, the rounded up r_{Case3}, and the fractional rectangles in F of the same type as r_{Case2} and r_{Case3} can be discarded. □

Lemma 5.2.8. *If $S = S_{Case3}$, then there will be no leftover fractional rectangles after processing Case 3 as described in this section. Therefore, in this case algorithm 4ConfigurationRounding produces an integer packing of height at most $\frac{5}{2}$ plus the height of the fractional packing produced by the solution of the linear program.*

Proof. For every section $s_i \in S_{Case3}$, the fractional rectangles in $R_{1(i)}$, $R_{2(i)}$, $R_{3(i)}$, and $R_{4(i)}$ are rounded up to become whole rectangles; therefore, there will be no leftover fractional rectangles. Recall that the height of any rectangle is at most 1; therefore, the total increase in height from rounding up the fractional rectangles in $R_{1(i)}$, $R_{2(i)}$, $R_{3(i)}$, and $R_{4(i)}$ is at most $\frac{5}{2}$. □

We can now prove Theorem 5.2.1.

Proof. By Corollary 4.1.1, if all rectangles are packed in the common portion of the packing, algorithm 4ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program.

Recall that the common portion of the packing can be rounded independently of the uncommon portion of the packing. For the rest of the proof we assume that the common portion of the packing has been processed so it does not have any fractional rectangles and we already know how much the height of the packing in this portion has increased.

By Corollary 5.2.2, if $S = S_{Case1}$, algorithm 4ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program. By Corollary 5.2.4, if $S = S_{Case2}$, algorithm 4ConfigurationRounding produces an integer packing of height at most $\frac{5}{2}$ plus the height of the fractional packing produced by the solution of the linear program. By Corollary 5.2.8, if $S = S_{Case3}$, algorithm 4ConfigurationRounding produces an integer packing of height at most $\frac{5}{2}$ plus the height of the fractional packing produced by the solution of the linear program.

If $S_{Case1} \neq \emptyset$, $S_{Case2} \neq \emptyset$, and $S_{Case3} = \emptyset$, then after processing Case 1 and Case 2 as described in this section, the height increase caused by leaving space of height 1 between C_1 and C_2 is 1, and the height increase caused by rounding up for the fractional rectangles in R_3 and R_4 within S_{Case2} is at most $\frac{3}{2}$. By Lemma 5.2.3, any fractional rectangle $r \in S_{Case1} \cap S_{Case2} \cap C_3$ in the solution of the linear program is either replaced by a whole rectangle of the same type without further increasing the height of the packing and without overlapping any other rectangles or it is discarded. Note that for fractional rectangles $r \in S_{Case1} \cap S_{Case2}$ in C_1 or C_2, the parts r_{Case1} and r_{Case2} are both removed, re-shaped, and packed side-by-side in C_{A1}. For any whole rectangle $r' \in S_{Case1} \cap S_{Case2}$ the parts r_{Case1} and r_{Case2} remain side-by-side and thus still form a whole rectangle in the final packing. Using the same proof as in Corollary 5.2.4, it can be shown that any leftover fractional rectangles at this point can be discarded. Therefore, in this case algorithm 4ConfigurationRounding produces an integer packing of height at most $\frac{5}{2}$ plus the height of the fractional packing produced by the solution of the linear program.

If $S_{Case1} \neq \emptyset$, $S_{Case2} = \emptyset$, and $S_{Case3} \neq \emptyset$, then after processing Case 1 and Case 3 as described in this section, the height increase caused by leaving space of height 1 between C_1 and C_2 is 1, and fractional rectangles in R_1 and R_2 within S_{Case3} can be rounded up without further increasing the height of the packing. Rounding up the fractional rectangles in R_3 and R_4 within S_{Case3} further increases the height of the packing by at most $\frac{3}{2}$. By Lemma 5.2.5, any fractional rectangle $r \in S_{Case1} \cap S_{Case3} \cap C_2$ in the solution of the linear program is either replaced by a whole rectangle of the same type without further increasing the height of the packing or overlapping any other rectangles or it is discarded. By Lemma 5.2.5, any fractional rectangle $r \in S_{Case1} \cap S_{Case3} \cap C_3$ in the solution of the linear program is either replaced by a whole rectangle of the same type without further increasing the height of the packing or overlapping any other rectangles or it is discarded. Recall that we ordered the configurations such that there can be no fractional rectangle $r \in S_{Case1} \cap S_{Case3} \cap C_1$ or $r \in S_{Case1} \cap S_{Case3} \cap C_4$. For any whole rectangle $r' \in S_{Case1} \cap S_{Case3}$ the parts r_{Case1} and r_{Case3} remain side-by-side and thus still form a whole rectangle in the final packing. Using the same proof as in Corollary 5.2.4, it can be shown that any leftover fractional rectangles at this point can be discarded. Therefore, in this case algorithm 4ConfigurationRounding produces an integer packing of height at most $\frac{5}{2}$ plus the height of the fractional packing produced by the solution of the linear program.

If $S_{Case1} = \emptyset$, $S_{Case2} \neq \emptyset$, and $S_{Case3} \neq \emptyset$, then after processing Case 2 and Case 3 as described in this section, the height increase caused by leaving space of height 1 between C_1 and C_2 is 1, and fractional rectangles in R_1 and R_2 within S_{Case3} can be rounded up without further increasing the height of the packing. Rounding up the fractional rectangles in R_3 and R_4 within S_{Case3} further increases the height of the packing by at most $\frac{3}{2}$. By Lemma 5.2.6, any fractional rectangle $r \in S_{Case2} \cap S_{Case3} \cap C_2$ in the solution of the linear program is either transformed into a whole rectangle without further increasing the height of the packing or overlapping any other rectangles or it is discarded. Recall that we ordered the configurations such that there can be no fractional rectangle $r \in S_{Case2} \cap S_{Case3} \cap C_1$ or $r \in S_{Case2} \cap S_{Case3} \cap C_4$. Note that for a fractional rectangle $r \in S_{Case2} \cap S_{Case3} \cap C_3$, the parts r_{Case2} and r_{Case3} are both rounded up form a whole rectangle. For any whole rectangle $r' \in S_{Case2} \cap S_{Case3}$ the parts r_{Case2} and r_{Case3} remain side-by-side and thus still form a whole rectangle in the final packing. Using the same proof as in Corollary 5.2.4, it can be shown that any leftover fractional rectangles at this point can be discarded. Therefore, in this case algorithm 4ConfigurationRounding produces an integer packing of height at most $\frac{5}{2}$ plus the height of the fractional packing produced by the

solution of the linear program.

If $S_{Case1} \neq \emptyset$, $S_{Case2} \neq \emptyset$, and $S_{Case3} \neq \emptyset$, then after processing Case 1, Case 2, and Case 3 as described in this section, the height increase caused by leaving space of height 1 between C_1 and C_2 is 1, and fractional rectangles in R_1 and R_2 within S_{Case3} can be rounded up without further increasing the height of the packing. Rounding up the fractional rectangles in R_3 and R_4 within S_{Case2} and S_{Case3} further increases the height of the packing by at most $\frac{3}{2}$. By Lemma 5.2.3, any fractional rectangle $r \in S_{Case1} \cap S_{Case2} \cap C_3$ in the solution of the linear program is transformed into a whole rectangle without further increasing the height of the packing and without overlapping any other rectangles. By Lemma 5.2.6, any fractional rectangle $r \in S_{Case2} \cap S_{Case3} \cap C_2$ in the solution of the linear program is either transformed into a whole rectangle without further increasing the height of the packing or overlapping any other rectangles or it is discarded. By Corollary 5.2.7, any fractional rectangle $r \in S_{Case1} \cap S_{Case2} \cap S_{Case3}$ located in C_2 or C_3 in the solution of the linear program is either transformed into a whole rectangle without further increasing the height of the packing or overlapping any other rectangles or it is discarded. Note that for fractional rectangles $r \in S_{Case1} \cap S_{Case2}$ in C_1 or C_2, the parts r_{Case1} and r_{Case2} are both removed, re-shaped, and packed side-by-side in C_{A1}. Additionally, note that for fractional rectangles $r \in S_{Case2} \cap S_{Case3} \cap C_3$, the parts r_{Case2} and r_{Case3} are both rounded up and form a whole rectangle. Recall that we ordered the configurations such that there can be no fractional rectangle $r \in S_{Case2} \cap S_{Case3} \cap C_1$ or $r \in S_{Case2} \cap S_{Case3} \cap C_4$. For any whole rectangle $r' \in S_{Case2} \cap S_{Case3}$ the parts r_{Case2} and r_{Case3} remain side-by-side and thus still form a whole rectangle in the final packing. Using the same proof as in Corollary 5.2.4, it can be shown that any leftover fractional rectangles at this point can be discarded. Therefore, in this case algorithm 4ConfigurationRounding produces an integer packing of height at most $\frac{5}{2}$ plus the height of the fractional packing produced by the solution of the linear program.

Therefore, algorithm 4ConfigurationRounding produces an integer packing of height at most $\frac{5}{2}$ plus the height of the fractional packing produced by the solution of the linear program. □

5.3 Three Configurations

When the solution to the linear program has three configurations, we use algorithm 3ConfigurationRounding described in Chapter 4. By Theorem 4.2.1, when there are three configurations, algorithm 3ConfigurationRounding produces an integer packing of height at most $\frac{5}{3}$ plus the height of the fractional packing produced by the solution of the linear program.

Note that the presence of four distinct rectangle types does not have any impact on the algorithms described in Chapter 4. Algorithms 3ConfigurationRounding, 2ConfigurationRounding, and 1ConfigurationRounding each consider one section s_i at a time such that only a single rectangle type is packed in that portion of the configuration; it does not make a difference to any of these algorithms which rectangle type is currently being considered, these algorithms simply consider the value of the fractions and apply a particular rounding technique. The increase in height caused by these rounding techniques also does not depend on the number of rectangle types.

5.4 Two Configurations

When the solution to the linear program has two configurations, we use algorithm 2ConfigurationRounding described in Chapter 4. By Theorem 4.3.1, when there are two configurations, algorithm 2ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program. Note that the presence of four distinct rectangle types does not have any impact on the algorithms described in Chapter 4.

5.5 One Configuration

When the solution to the linear program has only one configuration, we use algorithm 1ConfigurationRounding described in Chapter 4. By Theorem 4.4.1, when there is only one configuration, algorithm 1ConfigurationRounding produces an integer packing of height at most 1 plus the height of the fractional packing produced by the solution of the linear program. Note that the presence of four distinct rectangle types does not have any impact on the algorithms described in Chapter 4.

5.6 Approximation Ratio

As described above, when there are $k = 4$ rectangle types in the fractional solution computed by the GLD algorithm there can be either four configurations, three configurations, two configurations, or only one configuration.

By Theorems 5.2.1, 4.2.1, 4.3.1, and 4.4.1, when there are four configurations, three configurations, two configurations, or only one configuration, algorithm 4TypeRounding increases the height of the fractional packing produced by the solution of the linear program by at most $\frac{5}{2}$, $\frac{5}{3}$, 1, or 1, respectively.

Therefore, when there are $k = 4$ rectangles types, algorithm 4TypeRounding produces an integer packing of height at most $\frac{5}{2}$ plus the height of the fractional packing produced by the solution of the linear program. Since the height of the fractional packing obtained by solving the linear program is no larger than the height OPT of an optimum solution for HSMP, then algorithm 4TypeRounding produces a packing of height at most $OPT + \frac{5}{2}$.

5.7 Running Time

The vector output from the linear program is transformed into a list of numbers that specifies the rectangles that are packed in the configurations of the fractional solution. A configuration is specified using $O(k)$ numbers: for $1 \leq i \leq k$, we specify the rectangle type T_i, the number of rectangles of type T_i packed side-by-side, and the number of rectangles of type T_i packed one on top of the other (see Chapter 4, Figure 4.27).

Therefore, since the fractional solution determined by the linear program has at most k configurations, and since each configuration uses $O(k)$ numbers, then the input to algorithm 4TypeRounding uses $O(k^2)$ numbers to specify all the configurations.

The output of algorithm 4TypeRounding first specifies the common portion of the packing as a list of $O(k)$ numbers: for $1 \leq i \leq k$, we specify the rectangle type T_i, the number of rectangles of type T_i packed side-by-side, and the number of rectangles of type T_i packed one on top of the other (see Chapter 4, Figure 4.28).

The part of a configuration in the uncommon portion of the packing is specified as a list of $O(k)$ numbers: we specify the rectangle type T_i, the number of rectangles of type T_i packed side-by-side, and the number of rectangles of type T_i packed one on top of the other (see Chapter 4, Figure 4.29).

Therefore, since the fractional solution determined by the linear program has at most k configurations, we only add two configurations C_{A1} and C_{A2}, and each configuration uses $O(k)$ numbers, then the output produced by algorithm 4TypeRounding uses $O(k^2)$ numbers to specify all the configurations (see Chapter 4, Figure 4.30).

Algorithm 4TypeRounding transforms a fractional solution into an integral one by performing operations on the list of numbers given as input.

Theorem 5.7.1. *Algorithm 4TypeRounding produces an integer packing of height at most $\frac{5}{2}$ plus the height of the fractional packing produced by the solution of the linear program using at most $O(k^3)$ operations plus the time needed to compute the solution of the linear program.*

Proof. A similar proof as that of Theorem 4.6.1 can be used. Partitioning the list of numbers specifying the fractional solution into common and uncommon portions requires identifying the number of rectangles of type T_1, T_2, T_3, and T_4 that belong in the common portion of the packing.

The number of rectangles of type T_1 that should be packed side-by-side in the common portion of the packing is equal to the minimum of the number of rectangles of type T_1 that are packed side-by-side in each of C_1, C_2, C_3, and C_4. The number of rectangles of type T_1 that should be packed one on top of the other in the common portion of the packing is equal to the sum of the number of rectangles of type T_1 that are packed one on top of the other in each of C_1, C_2, C_3, and C_4. Hence, finding the number of rectangles of type T_1 that belong in the common

portion of the packing requires $O(k)$ operations. Therefore, finding the number of rectangles of each type that belong in the common portion of the packing requires $O(k^2)$ operations.

Processing the common portion of the packing according to Section 4.1.1 requires $O(k)$ operations as for $1 \leq i \leq k$, algorithm 4TypeRounding only needs to round up the fractional numbers for each rectangle type T_i.

Sorting the rectangles in each configuration in the uncommon portion of the packing by the fractions f_i requires at most k comparisons per configuration. Since this needs to be done for each configuration, sorting the rectangles in the uncommon portion of the packing requires $O(k^2)$ operations.

Processing a section $s_i \in S_{Case1}$ according to Lemma 4.2.2 requires $O(k)$ operations: the number of fractional rectangles from $C_{1(i)}$ is computed by multiplying the corresponding number of rectangles packed side-by-side by the decimal part of the number of rectangles packed one on top of the other. This is repeated for $C_{2(i)}$, $C_{3(i)}$, and $C_{4(i)}$. Products corresponding to the same rectangle types are added together.

Processing a section $s_i \in S_{Case2}$ according to Section 5.2.2 requires $O(k)$ operations: $C_{1(i)}$ and $C_{2(i}$ are processed as described above; fractional numbers specifying the number of rectangles packed one on top of the other in each of $C_{3(i)}$ and $C_{4(i)}$ are rounded up.

Processing a section $s_i \in S_{Case3}$ according to Section 5.2.3 also requires $O(k)$ operations: for each of $C_{1(i)}$, $C_{2(i)}$, $C_{3(i)}$, and $C_{4(i)}$, fractional numbers specifying the number of rectangles packed one on top of the other are rounded up.

Note that for the vertically split fractional rectangles only a constant number of operations are needed to check whether one part needs to be re-shaped and the other part rounded up. When a rounded up fractional rectangle is combined with a fractional rectangle from C_{A1} to form a whole rectangle, a constant number of numbers in the list need to be changed to reflect the additional rectangle.

Recall that in the uncommon portion of the packing, for configurations that contain more than one type of rectangle, a new section is created at the point where the rectangle type changes in the configuration. Also recall that in the uncommon portion of the packing, within a configuration at most k distinct rectangle types are packed. Therefore, one configuration can contain at most k sections, one for each rectangle type. Since there are at most k configurations, and each configuration can contain at most k sections, there are $O(k^2)$ sections in the uncommon portion of the packing.

Note that processing a section requires $O(k)$ operations, regardless of which case that section belongs to. In the worst case, $O(k^2)$ sections need to be processed, each requiring $O(k)$ operations, for a total of $O(k^3)$ operations. Partitioning the fractional packing into common and uncommon sections and sorting the rectangles in the uncommon portion of the packing each require $O(k^2)$ operations. Processing the common portion of the packing requires $O(k)$ operations. Therefore, algorithm 3TypeRounding requires $O(k^3 + k^2 + k) = O(k^3)$ operations plus the time needed to compute the solution of the linear program, which was proven in [47] to be polynomial. □

Chapter 6

Conclusion

In this thesis we considered the two-dimensional high multiplicity strip packing problem: given k distinct rectangle types, where each rectangle type T_i has n_i rectangles each with width $0 < w_i \leq 1$ and height $0 < h_i \leq 1$, the goal is to pack these rectangles into a strip of width 1, without rotating or overlapping the rectangles, such that the total height of the packing is minimized.

We presented algorithm 3TypeRounding designed in collaboration with Yu for HMSPP when $k = 3$ for which $SOL(I) \leq OPT(I) + \frac{5}{3}$. Algorithm 3TypeRounding can be generalized for any fixed value k to get $SOL(I) \leq OPT(I) + k - \frac{4}{3}$.

Additionally, we presented algorithm 4TypeRounding for HMSPP when $k = 4$ for which $SOL(I) \leq OPT(I) + \frac{5}{2}$. Algorithm 4TypeRounding can be generalized for any fixed value k to get $SOL(I) \leq OPT(I) + k - \frac{3}{2}$.

Algorithms 3TypeRounding and 4TypeRounding run in polynomial time; these algorithms compute their solutions using at most $O(k^3)$ operations plus the time needed to compute the solution to the linear program, which was proven in [47] to be polynomial.

Modifying these algorithms to work for any fixed value of k is simple. For example, when $k = 5$ and the solution to the linear program contains five configurations, C_1, C_2, C_3, and C_4 can be processed using an unmodified version of algorithm 4TypeRounding producing a height of at most $\frac{5}{2}$ plus the height of the fractional packing produced by the solution of the linear program. Each fractional rectangle in C_5 is rounded up, further increasing the height by at most 1 (see Figure 6.1). Therefore, the above algorithm for HMSPP when $k = 5$ produces solutions of value $SOL(I) \leq OPT(I) + \frac{7}{2}$.

Each additional configuration beyond C_1, C_2, C_3, and C_4 has its fractional rectangles rounded up and further increases the height of the packing by at most 1.

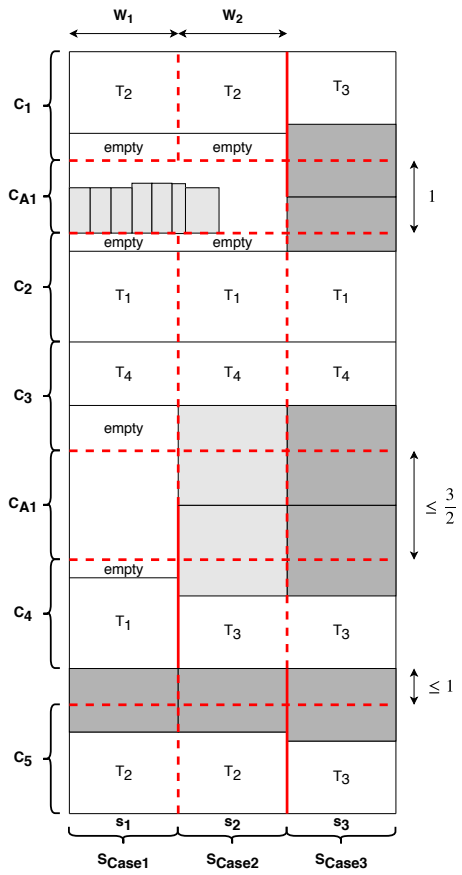


Figure 6.1: When $k = 5$ and the solution to the linear program contains five configurations, C_1, C_2, C_3, and C_4 can be processed using an unmodified version of algorithm 4TypeRounding. Each fractional rectangle in C_5 is rounded up.

6.1 Future Work

In Chapter 2 we discussed some classic packing problems such as the bin packing problem and the cutting stock problem. Algorithms designed for the cutting stock problem, which is the high multiplicity version of the bin packing problem, produce solutions that are much closer to the optimal solutions. In fact, in [24] an algorithm was presented that solves the cutting stock problem exactly for a fixed value of d. Our future work includes determining whether we can decrease the heights of the packings producted by algorithms 3TypeRounding and 4TypeRounding and whether we can design an algorithm that solves HMSPP exactly for a fixed value of k.

Many variants of packing problems exist that permit rotations of the objects. This distinction is important in certain application such as cutting wood with the grain or against the grain, or in other applications where the material is featureless and its orientation does not matter. Since many applications require cuts along vertical or horizontal lines, many packing problems allow only 90 degree rotations. Allowing a rectangle to be rotated might allow a better solution to be found, as there might not have been an ideal location to pack a tall and skinny rectangle but there could be a perfect location to pack a short and wide rectangle. However,

allowing rotations also increases the total number of possible packings.

Jansen and van Stee [35] presented a PTAS for the strip packing problem with 90 degree rotations that produces solutions of value $SOL(I) \leq (1+\epsilon)OPT(I) + O(\frac{1}{\epsilon^2})$ for any $\epsilon > 0$. In contrast, recall that Jansen and Solis-Oba [32] presented a PTAS for the strip packing problem with no rotations that produces solutions of value $SOL(I) \leq (1+\epsilon)OPT(I) + 1$ for any $\epsilon > 0$. The algorithm designed for the strip packing problem that permits rotations has a better asymptotic approximation ratio. Our future work includes determining whether an approximation algorithm or a PTAS can be designed for HMSPP that permits 90 degree rotations.

Variants of packing problems also exist in more than just two dimensions. Bortfeldt and Mack [5] presented a heuristic for the three dimensional strip packing problem, which has applications in the steel industry. Their algorithm uses a *layer-building* approach; a layer of a packing consists of several cubes packed side-by-side, much like our configurations, and they stack layers one on top of the other to fill their container. Additionally, Miyazawa and Wakabayashi [46] presented an algorithm for the three dimensional strip packing problem with an approximation ratio of 2.76. Our future work includes designing an approximation algorithm for the three dimensional version of HMSPP.

Bibliography

[1] B. Baker, D. Brown, and H. Katseff. A $\frac{5}{4}$ algorithm for two-dimensional packing. *Journal of Algorithms*, 2(4):348–368, 1981.

[2] B. Baker, R. Calderbank, E. Coffman, and J. Lagarias. Approximation algorithms for maximizing the number of squares packed into a rectangle. *SIAM Journal on Algebraic Discrete Methods*, 4(3):383–397, 1983.

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

[4] J. Beirão. *Packing Problems in Industrial Environments: Application to the Expedition Problem at INDASA*. PhD thesis, Universidade Technica de Lisboa, 2009.

[5] A. Bortfeldt and D. Mack. A heuristic for the three-dimensional strip packing problem. *European Journal of Operational Research*, 183(3):1267–1279, 2007.

[6] A. Caprara and M. Monaci. On the two-dimensional knapsack problem. *Operations Research Letters*, 32(1):5–14, 2004.

[7] P. Chen, Y. Chen, M. Goel, and F. Mang. Approximation of two-dimensional rectangle packing. *CS270 Project Report*, 1999.

[8] E. Coffman, M. Garey, and D. Johnson. Approximation algorithms for bin packing: a survey. *Approximation Algorithms for NP-hard Problems*, pages 46–93, 1996.

[9] E. Coffman, M. Garey, D. Johnson, and R. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 9(4):808–826, 1980.

[10] G. Dósa. The tight bound of first fit decreasing bin-packing algorithm is $ffd(i) \leq \frac{11}{9}opt(i) + \frac{6}{9}$. In *International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pages 1–11. Springer, 2007.

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

[12] K. Eisemann. The trim problem. *Management Science*, 3(3):279–284, 1957.

[13] C. Filippi. On the bin packing problem with a fixed number of object weights. *European Journal of Operational Research*, 181(1):117–126, 2007.

[14] C. Filippi and A. Agnetis. An asymptotically exact algorithm for the high-multiplicity bin packing problem. *Mathematical Programming*, 104(1):21–37, 2005.

[15] A. Fishkin, O. Gerber, K. Jansen, and R. Solis-Oba. On packing squares with resource augmentation: Maximizing the profit. In *Proceedings of the 2005 Australasian Symposium on Theory of Computing-Volume 41*, pages 61–67. Australian Computer Society, Inc., 2005.

[16] A. Fishkin, O. Gerber, K. Jansen, and R. Solis-Oba. Packing weighted rectangles into a square. In *International Symposium on Mathematical Foundations of Computer Science*, pages 352–363. Springer, 2005.

[17] Z. Fitzsimmons and E. Hemaspaandra. High-multiplicity election problems. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1558–1566. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[18] D. Friesen and M. Langston. Analysis of a compound bin packing algorithm. *SIAM Journal on Discrete Mathematics*, 4(1):61–79, 1991.

[19] M. Garey and D. Johnson. "strong"np-completeness results: Motivation, examples, and implications. *Journal of the ACM (JACM)*, 25(3):499–508, 1978.

[20] M. Gary and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman and Company, New York, 1979.

[21] P. Gilmore and R. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961.

[22] P. Gilmore and R. Gomory. A linear programming approach to the cutting stock problem—part ii. *Operations Research*, 11(6):863–888, 1963.

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

[24] M. Goemans and T. Rothvoß. Polynomiality for bin packing with a constant number of item types. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 830–839. SIAM, 2014.

[25] I. Golan. Performance bounds for orthogonal oriented two-dimensional packing algorithms. *SIAM Journal on Computing*, 10(3):571–582, 1981.

[26] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[27] R. Harren. Approximation algorithms for orthogonal packing problems for hypercubes. *Theoretical Computer Science*, 410(44):4504–4532, 2009.

[28] R. Harren, K. Jansen, L. Prädel, and R. Van Stee. A $(\frac{5}{3} + \varepsilon)$-approximation for strip packing. *Computational Geometry*, 47(2):248–267, 2014.

[29] R. Harren and R. Van Stee. Improved absolute approximation ratios for two-dimensional packing problems. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 177–189. Springer, 2009.

[30] D. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *Journal of the ACM (JACM)*, 32(1):130–136, 1985.

[31] D. Hochbaum and R. Shamir. Strongly polynomial algorithms for the high multiplicity scheduling problem. *Operations Research*, 39(4):648–653, 1991.

[32] K. Jansen and R. Solis-Oba. New approximability results for 2-dimensional packing problems. In *International Symposium on Mathematical Foundations of Computer Science*, pages 103–114. Springer, 2007.

[33] K. Jansen and R. Solis-Oba. A polynomial time approximation scheme for the square packing problem. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 184–198. Springer, 2008.

[34] K. Jansen and R. Solis-Oba. An opt + 1 algorithm for the cutting stock problem with constant number of object lengths. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 438–449. Springer, 2010.

[35] K. Jansen and R. van Stee. On strip packing with rotations. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of computing*, pages 755–761. ACM, 2005.

[36] K. Jansen and G. Zhang. Maximizing the total profit of rectangles packed into a rectangle. *Algorithmica*, 47(3):323–342, 2007.

[37] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[38] D. Johnson and M. Garey. A $\frac{71}{60}$ theorem for bin packing. *Journal of Complexity*, 1(1):65–106, 1985.

[39] H. Karloff. *Linear programming*. Springer Science & Business Media, 2008.

[40] N. Karmarkar and R. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 312–320. IEEE, 1982.

[41] R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

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

[43] L. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[44] Y. Lan, G. Dósa, X. Han, C. Zhou, and A. Benko. 2d knapsack: Packing squares. *Theoretical Computer Science*, 508:35–40, 2013.

[45] S. McCormick, S. Smallwood, and F. Spieksma. A polynomial algorithm for multiprocessor scheduling with two job lengths. *Mathematics of Operations Research*, 26(1):31–49, 2001.

[46] F. Miyazawa and Y. Wakabayashi. Packing problems with orthogonal rotations. In *Latin American Symposium on Theoretical Informatics*, pages 359–368. Springer, 2004.

[47] D. Price. High multiplicity strip packing. Master's thesis, Western University, 2014.

[48] T. Rothvoß. Approximating bin packing within $o(logopt * loglogopt)$ bins. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 20–29. IEEE, 2013.

[49] I. Schiermeyer. Reverse-fit a 2-optimal algorithm for packing rectangles. In *European Symposium on Algorithms*, pages 290–299. Springer, 1994.

[50] D. Simchi-Levi. New worst-case results for the bin-packing problem. *Naval Research Logistics (NRL)*, 41(4):579–585, 1994.

[51] D. Sleator. A 2.5 times optimal algorithm for packing in two dimensions. *Information Processing Letters*, 10(1):37–40, 1980.

[52] A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997.

[53] F. De La Vega and G. Lueker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[54] L. Wang, R. Duan, X. Li, S. Lu, T. Hung, R. Calheiros, and R. Buyya. An iterative optimization framework for adaptive workflow management in computational clouds. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1049–1056. IEEE, 2013.

[55] P. Wang. Two algorithms for constrained two-dimensional cutting stock problems. *Operations Research*, 31(3):573–586, 1983.

[56] A. Yao. New algorithms for bin packing. *Journal of the ACM (JACM)*, 27(2):207–227, 1980.

Curriculum Vitae

Name:	Andrew Bloch-Hansen
Post-Secondary Education and Degrees:	Western University Ontario, Canada 2013 - 2017 Honors Specialization in Computer Science University of Western Ontario London, ON 2017 - 2019 M.Sc. in Theoretical Computer Science
Related Work Experience:	Teaching Assistant Western University 2017 - 2019 Research Assistant University of Western Ontario 2017 - 2019