11-15-2019 12:30 PM

High Multiplicity Strip Packing Problem With Three Rectangle Types

Andy Yu
The University of Western Ontario

Supervisor
Solis-Oba, Roberto
The University of Western Ontario

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

Abstract

The two-dimensional strip packing problem (2D-SPP) involves packing a set $R = \{r_1, ..., r_n\}$ of n rectangular items into a strip of width 1 and unbounded height, where each rectangular item r_i has width $0 < w_i \leq 1$ and height $0 < h_i \leq 1$. The objective is to find a packing for all these items, without overlaps or rotations, that minimizes the total height of the strip used. 2D-SPP is strongly **NP**-hard and has practical applications including stock cutting, scheduling, and reducing peak power demand in smart-grids.

This thesis considers a special case of 2D-SPP in which the set of rectangular items R has three distinct rectangle sizes or types. We present a new $OPT + 5/3$ polynomial-time approximation algorithm, where OPT is the value of an optimum solution. This algorithm is an improvement over the previously best $OPT + 2$ polynomial-time approximation algorithm for the problem.

Keywords: high multiplicity, strip packing, approximation algorithm, optimization

Summary for Lay Audience

Consider a set of rectangles of three different sizes. The goal of the three-type strip packing problem (3T-SPP) is to pack these rectangles as densely as possible without overlaps or rotations into a single two-dimensional container of fixed width. In an optimum solution for 3T-SPP where we pack rectangles as densely as possible, we denote with *OPT* the minimum possible height within which these rectangles can be packed into the container.

Many real-life problems, from industrial manufacturing to scheduling, can be modelled in terms of packing rectangular items of various sizes into a two-dimensional container. One example is stock cutting, which involves cutting a roll of flat material into different rectangle sizes while minimizing the material wasted. Another example is scheduling tasks of various types on processors while minimizing the total completion time; each type of task requires a certain number of adjacent processors for a given amount of time. If we can present a procedure that produces an optimum solution to 3T-SPP, then this procedure can also produce optimum solutions to some of these real-life problems.

The two-dimensional strip packing problem (2D-SPP) is a generalization of 3T-SPP that allows any number of different rectangle sizes. 2D-SPP belongs to a class of problems that currently lack any procedure to efficiently produce optimum solutions. As 3T-SPP is a special case of 2D-SPP, we focus on algorithms or procedures that efficiently find solutions that are not optimal, but are always close to the optimum solutions. We present a procedure for 3T-SPP that efficiently finds a packing of height at most $OPT + 5/3 \cdot h_{max}$, where h_{max} is the largest height of the rectangles. This procedure is an improvement over the previously best algorithm that finds a packing of height at most $OPT + 2 \cdot h_{max}$. Although this improvement looks small, $OPT + 2 \cdot h_{max}$ is already close to *OPT* and we expect further improvements to be hard to obtain.

Acknowledgements

I would like to thank my supervisor, Professor Roberto Solis-Oba for his guidance and assistance. He was very willing to give me time to discuss my research and any problems that I had issues with. I would also like to thank his master's student Andrew Bloch-Hansen, who contributed to the idea of shifting the boundary between segments S_2 and S_3 based on some criteria which I incorporated into the solution presented in this thesis for 3T-SPP. Finally, I would like to thank the Department of Computer Science at the University of Western Ontario for general support.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

The two-dimensional strip packing problem (2D-SPP) involves packing a set $R = \{r_1, ..., r_n\}$ of n rectangular items into a bin of width W and unbounded height, which we call a strip. Each item r_i has width $w_i \leq W$ and height h_i. The objective is to pack the n rectangles into the strip such that they do not overlap and that the total height of the strip used is minimal. Figure 1.1 shows an instance of 2D-SPP.

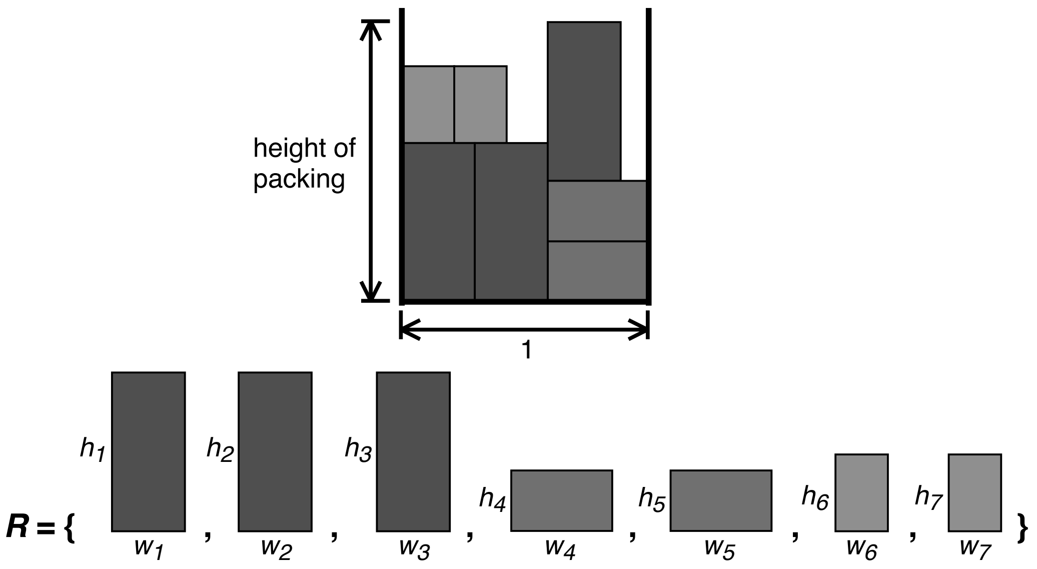

FIGURE 1.1: An instance of 2D-SPP with seven rectangles. The width of the strip is 1. Note how the set R has three distinct rectangle sizes or types.

This thesis assumes that packings for 2D-SPP are orthogonal, oriented, and normalized.

In an orthogonal packing, every edge of every rectangle is parallel to either the bottom or the vertical side of the strip [1]. We assume that rectangles are oriented, so that the bottom edge of each rectangle is parallel to the bottom of the strip—this problem does not allow rectangles to be rotated. We also assume, without loss of generality, that the width of the strip and the height of the tallest rectangle in R are normalized to 1 so that each rectangle r_i has width $0 < w_i \leq 1$ and height $0 < h_i \leq 1$.

Martello et al. [2] prove that 2D-SPP is strongly **NP**-hard via a reduction from the one-dimensional bin packing problem (1D-BPP), which is known to be strongly **NP**-hard [3]. The objective of 1D-BPP is to pack n items of different sizes into a minimal number of bins of capacity W so that the sum of sizes of the items in each bin is at most W. If in 2D-SPP we set all rectangle heights to 1, then we get an instance of 1D-BPP. Therefore, since 1D-BPP is strongly **NP**-hard, 2D-SPP must also be strongly **NP**-hard [2].

Currently, there is no known polynomial-time algorithm for any **NP**-hard problem. A **NP**-hard problem is at least as hard to solve as the hardest problems in **NP**, where **NP** is the class of problems that admit algorithms that can verify the feasibility of a solution in polynomial time [4]. In his seminal paper, Cook [5] proved that if we can design a polynomial-time algorithm for a **NP**-hard problem, then we can transform that algorithm into a polynomial-time algorithm for any **NP** problem. However, unless **P**, the class of problems that admit deterministic polynomial-time algorithms, is equal to **NP**, no algorithm exists for a **NP**-hard problem that is (1) optimal and (2) has polynomial running time in the worst case [4].

In this thesis we relax condition (1) by focusing on approximation algorithms. An approximation algorithm can produce in polynomial time solutions of values that are provably within some factor β of the value of optimum solutions. In this research we do not focus on heuristic-based approaches, because unlike approximation algorithms, they do not have a provable worst-case guarantee in regards to the quality of their solutions and/or run times. Section 1.1 further explains approximation algorithms.

In 1991, Hochbaum and Shamir [6] first defined high multiplicity problems as problems having inputs that can be partitioned into relatively few groups (or types) of items, and in each group all the items are identical. The multiplicity of a type is the number of input items of that given type [6]. In the context of 2D-SPP, the rectangular items in the set R can be partitioned into distinct rectangle sizes or types, where the multiplicity for a given rectangle type is the number of rectangles of that type that need to be packed. Rectangles of the same type have the same dimensions. In Figure 1.1, the number of distinct rectangles types is 3, the multiplicity of the tallest rectangle type is 3 and the multiplicities of the other two rectangle types are both 2.

In real-life problems where 2D-SPP arises, the number of distinct rectangle types is typically a small number. This thesis considers the high multiplicity version of 2D-SPP (HM-SPP) in which the number of distinct rectangle types in the set R is a fixed, positive integer K. In particular, we focus on the special case where $K = 3$, i.e., the three-type strip packing problem (3T-SPP).

As the number of distinct rectangle types in 3T-SPP is constant, we can express the input for an instance of the problem in a very compact way because we only need to list the dimensions and multiplicity for each rectangle type. To be more formal, we can represent the input of an instance of HM-SPP as a set of rectangle types $T = \{T_1, ..., T_K\}$, where each rectangle type T_i has a multiplicity n_i, width w_i, and height h_i; note that we can represent these rectangle types with $3K$ positive numbers, regardless of the number of rectangles. Similarly, we can specify the input of any instance of 3T-SPP with only 9 positive numbers.

Unlike 2D-SPP, approximation algorithms for 3T-SPP and HM-SPP must run in polynomial time on the number of rectangle types instead of in polynomial time on the number of rectangles. This is a very important distinction: because the input is so compact, some polynomial-time algorithms for 2D-SPP are no longer polynomial-time algorithms for 3T-SPP or HM-SPP. For example, an algorithm for 2D-SPP can specify the individual position of each rectangle within the strip. However, an algorithm for HM-SPP cannot individually specify the position of each rectangle as the number of rectangles is not a polynomial function of the number of rectangle types [7]. Although HM-SPP might appear to be simpler than 2D-SPP due to the many rectangle types in the latter problem, because the input of HM-SPP is so small Price [7] states that it is more difficult to compute a solution for HM-SPP as it must have time complexity bounded by a polynomial function of the small size of the input.

1.1 Approximation Algorithms

Let *A(I)* denote the value of the solution produced by an approximation algorithm A for a instance I of a problem. Let *OPT(I)* denote the value of an optimum solution for instance I. In this thesis, *A(I)* is the height of the packing that our approximation algorithm generates for an instance I of 3T-SPP; $OPT(I)$ is the minimum possible height within which the rectangles in I can be packed. Note that we sometimes write *OPT* instead of *OPT(I)* if the meaning is clear.

Even if we do not know the value of *OPT*, we can prove that an approximation algorithm A produces a solution that is within a factor β of *OPT* by computing a lower bound for *OPT*. If an approximation algorithm A always produces a solution whose value is at most a factor β larger than that lower bound, then A is also within a factor β from *OPT*. For an arbitrary instance I of a minimization problem, where $\beta \geq 1$, if $A(I) \leq \beta \cdot OPT(I)$ for every I, then approximation algorithm A has *absolute approximation ratio* β. 2D-SPP, HM-SPP, and 3T-SPP are all minimization problems because their objective is to minimize the total height of the strip used by the packing. If $A(I) \leq \beta \cdot OPT(I) + \gamma$ for every I, where $\beta \geq 1, \gamma \geq 0$, and *OPT(I)* is much larger than γ, then approximation algorithm A is said to have *asymptotic approximation ratio* β. Note how the *absolute approximation ratio* does not have the additive constant term γ; if $\beta = 1$, then A is known as *asymptotically exact*. Another notation that we use is β-approximation algorithm and asymptotic β-approximation algorithm.

If an **NP**-hard problem has an approximation algorithm with constant approximation ratio, then that problem belongs to the class **APX**. If a problem in **APX** has a $(1 + \epsilon)$-approximation algorithm that runs in polynomial time for any constant $\epsilon > 0$ and any instance I of a problem, then that problem has a *polynomial-time approximation scheme* (PTAS). We call an approximation scheme *fully polynomial* (FPTAS) if the running time of that approximation algorithm is bounded by a polynomial function in n and $\frac{1}{\epsilon}$. Note that a PTAS and a FPTAS can be asymptotic as well, which we can call APTAS or AFPTAS.

1.2 Applications

Recall that the objective of 2D-SPP is to pack the n rectangular items into the strip such that they do not overlap and that the total height of the strip used is minimal. If we can model a real-life problem in terms of 2D-SPP, then an optimal solution to 2D-SPP is also an optimal solution to that real-life problem. Generally, if we can model a real-life problem in terms of 2D-SPP, we can also represent it in terms of HM-SPP with the following caveat: only a limited number of distinct item types can appear in the input.

A typical optimization problem in industrial manufacturing is stock cutting. Suppose we are given a standard-sized roll of material (such as cloth, paper, or metal) where the width of the roll is fixed but the length is unspecified. For manufacturing purposes, it is desirable to cut this roll of material into rectangular pieces of specified sizes while minimizing the material wasted: this problem directly corresponds to the strip packing problem. In the context of the high multiplicity version, consider a mass production scenario where we need large quantities of standardized rectangular stock units. A client could request arbitrary numbers of certain rectangular pieces: this problem directly corresponds to HM-SPP.

An important optimization problem is scheduling parallel tasks on contiguous processors. Suppose an instance of the problem has a set of tasks, where each task requires a certain number of contiguous processors for a certain amount of time. The goal is to schedule the parallel tasks so that the makespan of the schedule (the total time it takes to complete all tasks) is minimal. This problem directly corresponds to the strip packing problem: the height of the packing is the makespan, and the width of the strip is the number of processors. In the context of the high multiplicity version, suppose we know the numbers and types of tasks that need to be scheduled: this problem directly corresponds to HM-SPP.

One of the goals of a smart electric grid is to better schedule electric consumption loads. In response to increasing peak power demand, existing electric grids typically invest in new infrastructure. However, peak power demand occurs in a relatively short fraction of the entire year; by minimizing peak power demand, we reduce the need to unnecessarily invest in expensive infrastructure [8]. We can model an instance of the problem as a set of rectangles, where each rectangle represents a demand for power. The width of the rectangle is the duration for which the power is needed; the height of the rectangle is the amount of power requested. As a simplification, we assume that each job requires a constant amount of power over its duration; this is true for the charging of certain types of electric vehicles [8]. This problem corresponds to the strip packing problem: the height of the packing is the total power that all demands require. In the context of the high multiplicity version, suppose we know the numbers and types of demands: this problem directly corresponds to HM-SPP.

1.3 Related Work

Since the 1980s, researchers have designed approximation algorithms for 2D-SPP. Baker et al. [1] designed the Bottom-Left algorithm, which sorts rectangles by decreasing widths and has an *asymptotic approximation ratio* of 3. Because 2D-SPP includes 1D-BPP as a special case [2], no polynomial-time approximation algorithm for 2D-SPP can have an *absolute approximation ratio* better than 3/2 unless **P** = **NP** [3]. However, *asymptotic approximation ratios* can have better approximation ratios than 3/2 if $OPT(I) \gg 1$, where 1 is the height of the tallest rectangle in an instance I of the problem. Note that many of the papers discussed in this section assume the height of the tallest rectangle is normalized to 1.

For absolute approximations, Coffman et al. [9] extended the First-Fit Decreasing Height (FFDH) algorithm for 1D-BPP to 2D-SPP and showed that the algorithm has an *absolute approximation ratio* of 2.7. Sleator [10] designed an approximation algorithm that has an *absolute* *approximation ratio* of 2.5. Schiermeyer [11] and Steinberg [12] both improved the *absolute approximation ratio* to 2. Harren and van Stee [13] then designed an algorithm with an *absolute approximation ratio* of 1.9396. The best current result is by Harren et al. [14] which has an *absolute approximation ratio* of $5/3 + \epsilon$ for any $\epsilon > 0$.

For asymptotic approximations, where $OPT(I)$ is very large, Coffman et al. [9] extended two approximation algorithms for 1D-BPP to 2D-SPP: the Next-Fit Decreasing Height algorithm and the FFDH algorithm. Coffman et al. [9] proved that these algorithms produce solutions of height at most $2 \cdot OPT(I) + 1$ and $1.7 \cdot OPT(I) + 1$, respectively. Golan [15] designed an approximation algorithm that produces solutions of height at most $4/3 \cdot OPT(I) + \frac{307}{18}$. Baker et al. [16] presented an approximation algorithm that produces solutions of height at most $5/4 \cdot OPT(I) + \frac{53}{8}$.

In their seminal paper, Kenyon and Rémila [17] designed an AFPTAS for 2D-SPP that produces solutions with height at most $(1+\epsilon)OPT + O(\frac{1}{\epsilon^2})$, for any $\epsilon > 0$. Jansen and Solis-Oba [18] designed an APTAS that improves upon [17] by reducing the additive term from $O(\frac{1}{\epsilon^2})$ to 1, but this algorithm has a higher running time. Using a modification of the algorithm of Kenyon and Rémila and using $\epsilon = \sqrt{\frac{log\ OPT}{OPT}}$, Sviridenko [19] obtained a polynomial-time approximation algorithm that produces solutions of height at most $OPT + \sqrt{OPT\ log\ OPT}$.

We say an algorithm runs in pseudo-polynomial time if the algorithm runs in polynomial time in the numeric values of the input, but not necessarily polynomial in the number of bits needed to encode the input. Because 2D-SPP is strongly **NP**-hard, no pseudo-polynomial time algorithm exists that is optimal. Jansen and Thöle [20] designed an approximation algorithm for 2D-SPP that runs in pseudo-polynomial time and has an approximation ratio of $3/2 + \epsilon$. Nadiradze and Wiese [21] designed a pseudo-polynomial approximation algorithm with approximation ratio $1.4 + \epsilon$. Gálvez et al. [22] and Jansen and Rau [23] both independently improved the approximation ratio to $4/3 + \epsilon$ with pseudo-polynomial running time. The best current result by Jansen and Rau [24] has approximation ratio $5/4 + \epsilon$ and pseudo-polynomial running time.

A variant of 2D-SPP is the multiple strip packing problem (M-SPP), where the objective is to pack n rectangles into a constant number of strips. One application of M-SPP is to find a schedule of minimal makespan for a set of parallel tasks to be executed on disjoint clusters of continguous processors [25]. Zhuk [26] proved that unless **P** = **NP**, there is no algorithm for M-SPP with *absolute approximation ratio* better than 2. Bougeret et al. [27] designed an approximation algorithm that has an *absolute approximation ratio* of 2, which is an improvement over the algorithm by Ye et al. [28] that has an *absolute approximation ratio* of $2+\epsilon$ for any $\epsilon > 0$. Bougeret et al. [27] also present an AFPTAS with additive constant *O(1)* if the number of strips is sufficiently large. This AFPTAS is the best possible asymptotic result unless **P** = **NP** [26].

A closely related problem to HM-SPP is HM-BPP. Recall that the objective of HM-BPP is to pack a set $I = \{I_1, ..., I_n\}$ of n items of different sizes into the minimal number of bins of capacity W so that the sum of sizes of the items in each bin is at most W. The items in I can only have K distinct types $T = \{T_1, ..., T_K\}$, where K is a positive, fixed integer, and each item size w_i for a type T_i has a multiplicity n_i. McCormick et al. [29] designed an exact polynomial-time algorithm for HM-BPP when $K = 2$ that runs in $O((\log W)^2)$. In 2005, Filippi and Agnetis [30] designed an *asymptotically exact*, polynomial-time approximation algorithm that uses at most $OPT(I) + K - 2$ bins. They also presented an exact polynomial-time algorithm that runs in $O(\log W)$ time when $K = 2$ [30]. Filippi [31] then further improved the approximation algorithm so that it uses at most $OPT + 1$ bins for $2 < K \leq 6$, and $OPT + 1 + \lfloor (K-1)/3 \rfloor$ bins for $K > 6$. In 2010, Jansen and Solis-Oba [32] designed a polynomial-time approximation algorithm that uses at most $OPT + 1$ bins for any constant K. Finally, Goemans and Rothvoß [33] recently proved that HM-BPP can be solved in time $O((\log \Delta)^{2^K})$, where Δ is the largest item size; this is an exact algorithm that runs in polynomial time.

Regarding HM-SPP, Price [7] designed a polynomial-time approximation algorithm that produces a solution of height at most $OPT(I) + K - 1$. If $OPT(I) \gg K$, this algorithm is *asymptotically exact* [7].

1.4 Contributions

The main contribution of this thesis is a new polynomial-time approximation algorithm for HM-SPP that produces packings of height at most $OPT(I) + 5/3$ for the special case where the number of different rectangle types is 3. If $OPT(I) \gg 5/3$, then our algorithm is *asymptotically exact*.

Our algorithm is an improvement over the algorithm in [7] for 3T-SPP; the algorithm in [7] produces a packing of height at most $OPT(I) + 2$ for the case when $K = 3$. Although this improvement looks small, note that $OPT(I) + 2$ is already very close to OPT and we would expect further improvements to be hard to obtain. Our research centers on the fundamental question: how closely can we approximate the optimum solution of a problem that is not known to be polynomially solvable? The work in [33] recently proved that HM-BPP is in the class **P**; perhaps HM-SPP is also in the class **P**? Note that HM-BPP involves uni-dimensional items, so a proof for HM-SPP being in the class **P** would likely be more complex than that in [33]. Furthermore, we propose several new techniques that might lead to improved algorithms for other versions of HM-SPP. This research leaves open the question of whether we can design algorithms with an even better performance guarantee for 3T-SPP or for other versions of HM-SPP.

1.5 Outline of the Thesis

In Chapter 2 we introduce definitions and notations used throughout the thesis; we discuss the fractional strip packing problem and present a high-level overview of our algorithm. This chapter also briefly describes an important subroutine of our algorithm, namely a polynomial-time algorithm in [7] that solves the fractional strip packing problem with a constant number of rectangle types.

In Chapter 3 we present a detailed description of our algorithm for 3T-SPP. In Chapter 4 we prove the correctness of the algorithm and analyze its time complexity. Finally, in Chapter 5 we give our conclusions, present some open questions, and explore areas of future work.

Chapter 2

Overview of the Algorithm

2.1 The Fractional Strip Packing Problem

The fractional strip packing problem (F-SPP) is a relaxed version of the strip packing problem where we allow rectangles to be horizontally cut. Our algorithm first computes a solution for the fractional version of 3T-SPP and then the algorithm converts it to a solution for 3T-SPP by eliminating any rectangles that have been horizontally cut. Let $C = \{C_1, C_2, ..., C_J\}$ be all possible subsets of rectangles from R such that the sum of widths of the rectangles in one of these sets C_j is at most 1. Note then that all the rectangles in one of these sets C_j fit together side-by-side in the strip. Also observe that each set C_j might contain several rectangles of the same type. Recall from Chapter 1 that we can represent an instance I of 3T-SPP as a set of three rectangle types $T = \{T_1, T_2, T_3\}$, where each rectangle type T_i has multiplicity n_i, width w_i, and height h_i; the multiplicity of a type T_i is the number of rectangles of type T_i. Let $\alpha_{i,j}$ be the number of rectangles of type T_i in set $C_j \in C$. We can formulate the fractional 3T-SPP as the following linear program:

$$\text{Minimize:} \quad \sum_{C_j \in C} x_j \tag{2.1}$$

$$\text{Subject to:} \quad \sum_{C_j \in C} x_j \alpha_{i,j} \geq n_i h_i, \text{ for all } i = 1, 2, 3 \tag{2.2}$$

$$x_j \geq 0$$

In this linear program there is a variable x_j for each set $C_j \in C$. We know that if we put all rectangles in C_j side by side their total width is at most 1 so they fit within the strip. Imagine that for each rectangle r of C_j we stack on top of it more rectangles of the same type as r until the stack has height x_j, horizontally slicing the last rectangle in each stack if needed as shown in Figure 2.1.

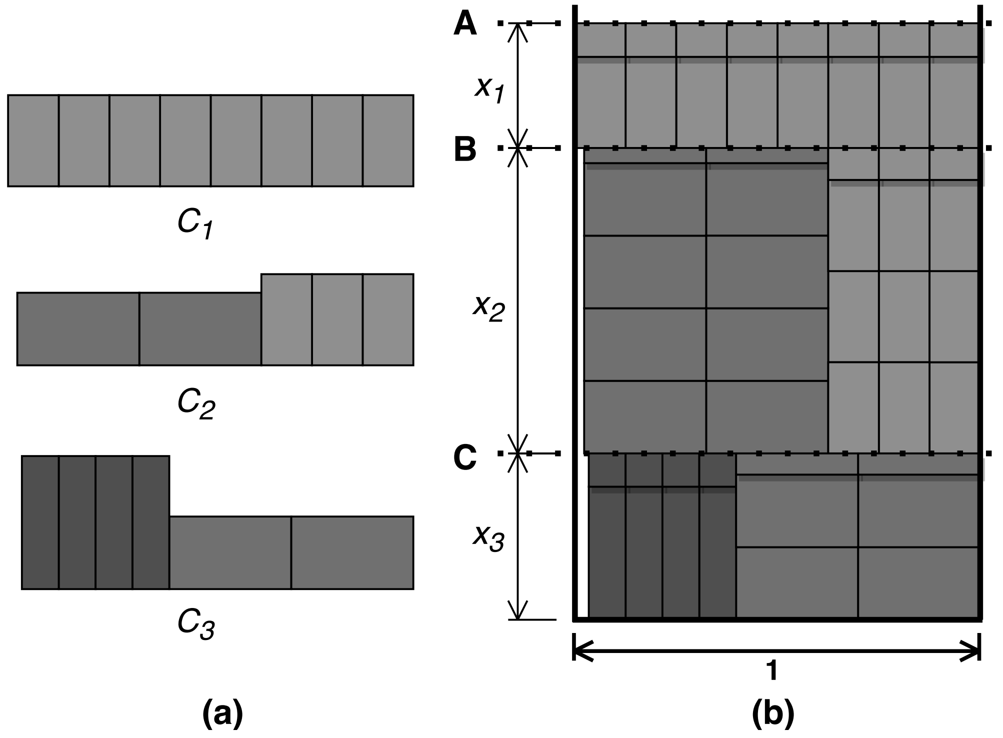

Figure 2.1: A fractional solution where some rectangles are cut horizontally. (a) The sets C_1, C_2, and C_3 in this fractional solution. (b) Rectangles of the same type as those that belong to each set C_j are stacked up to height x_j. For example, the rectangles in C_3 are stacked on top of each other until the stack has height x_3, horizontally cutting some of the rectangles at point C. We pack the rectangles from C_2 directly on top of these rectangles. The height of the fractional solution is $x_1 + x_2 + x_3$.

If we put all these stacks of rectangles in the bin, one on top of each other for all sets $C_j \in C$ we obtain a packing of total height equal to $\sum_{C_j \in C} x_j$; the above linear program minimizes this value, thus minimizing the height of the packing. Note that $n_i h_i$ represents the total height for all rectangles of type T_i that need to be packed, hence the constraint $\sum_{C_j \in C} x_j \alpha_{i,j} \geq n_i h_i$ ensures that all rectangles of type T_i are (fractionally) packed in the solution.

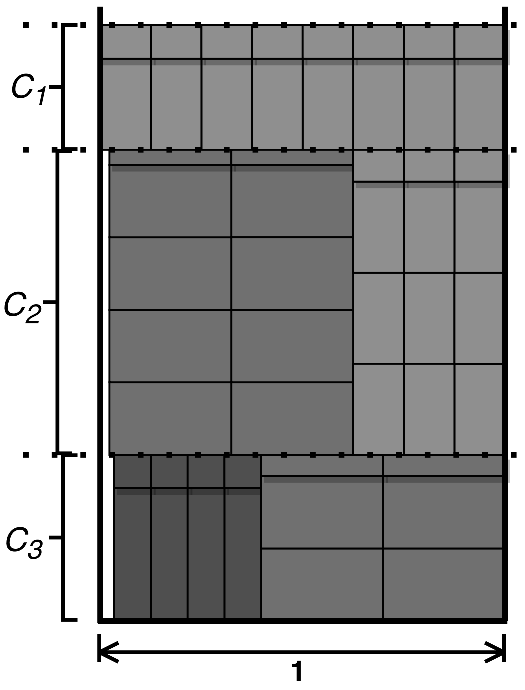

Figure 2.2: A fractional solution with three configurations, C_1, C_2, C_3, and three rectangle types.

In the linear program (2.1), the number of constraints or conditions of the form (2.2) that a solution must satisfy is equal to the number of rectangle types $K = 3$. The number of variables is equal to the number of sets $C_j \in C$, which is $O(n^3)$ because each set C_j can be expressed as a vector $(n_{j,1}, n_{j,2}, n_{j,3})$ where $n_{j,i}$ is the number of rectangles of type T_i in C_j. Since each $n_{j,i}$ can have at most $n + 1$ values, the number of sets $C_j \in C$ is at most $(n + 1)^3$. In Figure 2.1 note how the fractional solution has three distinct parts where rectangles are packed according to some set $C_j \in C$. We define a *configuration* to be a part of the packing where any horizontal line drawn across the bin intersects the same multiset of rectangle types. For example, a fractional solution with three configurations is shown in Figure 2.2. The number of possible configurations is equal to the number of sets in C and it is also equal to the number of variables; we index the configurations as C_1, C_2, C_3, ..., C_J.

The set of feasible solutions for the linear program (2.1) consists of all solutions that satisfy the conditions of the form (2.2). Basic feasible solutions are solutions where the number of non-zero variables is at most the minimum between the number of constraints (excluding non-negativity constraints such as $x_j \geq 0$) and the number of variables of the linear program [34]. If the linear program (2.1) has an optimal solution, then it must have at least one basic feasible solution that minimizes $\sum_{C_j \in C} x_j$ [34]. As the number of constraints of linear program (2.1) is $K = 3$, then an optimum basic feasible solution for this linear program has at most 3 nonzero variables, and so it arranges the rectangles in at most three configurations.

For an instance I of 3T-SPP, let $s(I)$ be the total area of the rectangles in I and let $lin(I)$ be the height of an optimal fractional packing for I (or, in other words, the value of an optimal solution for the linear program (2.1)). Recall that $OPT(I)$ is the minimum possible height within which the rectangles in I can be packed in the bin. Observe that $s(I) = \sum_i w_i h_i n_i$. Clearly, $s(I) \leq lin(I) \leq OPT(I)$; the first inequality holds because all rectangles must be packed by any solution of the linear program, and the second inequality holds because the optimal fractional solution packs the rectangles at least as efficiently as the optimum solution since a fractional solution is allowed to cut the rectangles horizontally. Thus, solving the linear program (2.1) gives a lower bound for the value of an optimal solution for 3T-SPP.

2.2 High-Level Description of the Algorithm

Our algorithm first uses the algorithm in [7] as a subroutine to solve the fractional 3T-SPP. The algorithm in [7] solves the linear program (2.1) in polynomial time by using a modified version of the Grötschel-Lovász-Schrijver (GLS) algorithm [35] and by borrowing several techniques from Karmarkar and Karp [36]. First, the algorithm in [7] uses the GLS algorithm to solve the dual linear program for the linear program (2.1). Then the algorithm of Karmarkar and Karp is used to transform the solution of the dual linear program into a basic feasible solution for the linear program (2.1). This fractional solution consists of at most three configurations. Note that in a fractional solution, if we add all rectangles of some type T_i, including rectangles of fractional height, over all configurations, we must get an integer value equal to the number n_i of rectangles of type T_i. We show below how to transform a fractional solution for 3T-SPP into an integral solution (i.e., a solution without fractional rectangles).

If the fractional solution obtained by solving the linear program (2.1) only uses one configuration then we simply round up every rectangle with fractional height to its full height as shown in Figure 2.3. Note that the integral solution in Figure 2.3, obtained after rounding up the rectangles of fractional height, has more rectangles than the fractional solution. This solution can easily be transformed into a solution for the original instance by discarding the extra rectangles.

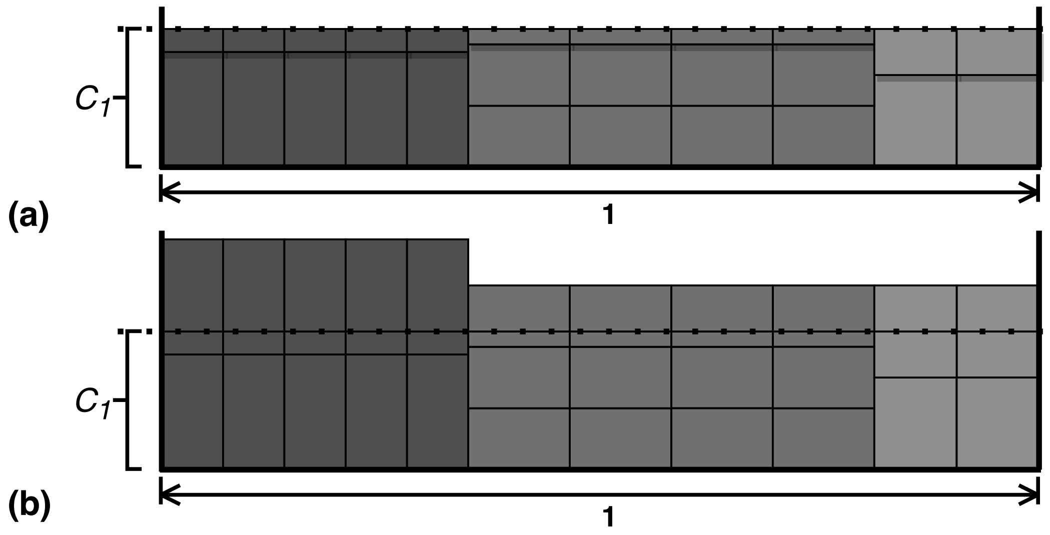

Figure 2.3: A fractional and integral solution with one configuration and three rectangle types. (a) A fractional solution. (b) An integral solution obtained by rounding up the height of each rectangle with fractional height in (a) to its full height.

If the fractional solution uses two configurations, we use the algorithm for HM-SPP in [7]. We briefly describe the algorithm below. Let the two configurations of the fractional solution be C_1 and C_2. Rearrange rectangles horizontally within each configuration so that a common part S_1 exists where rectangles of the same type line up between configurations, and hence, are not cut at the boundaries between C_1 and C_2 (see Figure 2.4). Let the non-common part be S_2, where no rectangle type appears in both C_1 and C_2.

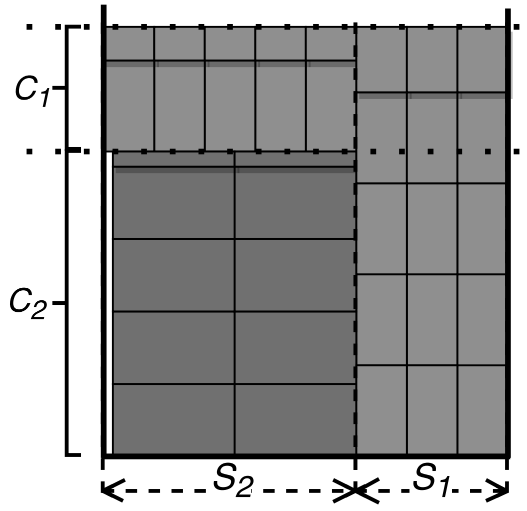

Figure 2.4: Horizontally rearranging rectangles so that a common part S_1 exists where rectangles of the same type line up between configurations. Rectangles in S_1 are not cut at the boundaries between C_1 and C_2. Note how in S_2, no rectangle type can appear in both C_1 and C_2.

Definition 2.1. Denote with $f_{i,j}$ the fraction of each rectangle of type T_i that lies just below the top of a configuration C_j.

For each configuration C_1 and C_2 sort the rectangles in S_2 by nondecreasing order of the value $f_{i,j}$. For part S_2 of C_1 and C_2, round up rectangles with fractional height to their full height as described in [7] so that they fit within a right triangular shape on top of each configuration. The part S_2 of C_1 is then turned 180 degrees and placed above part S_2 of C_2 so that the triangular regions fit together without overlaps (see Figure 2.5).

In [7], Price proved that the total increase in the height of the packing caused by the above process is at most 1. For the common part S_1 of C_1 and C_2, the rectangles of fractional height are simply rounded up to their full height and this increases the height of S_1 by at most 1. Because the heights of both S_1 and S_2 increase by at most 1, the height of the whole packing increases by at most 1.

If the fractional solution has K configurations, [7] suggests performing the above process on two adjacent configurations and then rounding up the rectangles with fractional height in the remaining configurations; this produces a packing of height at most $OPT(I) + K - 1$. If $K = 2$, as mentioned above, the algorithm produces a packing of height at most $OPT(I) + 1$.

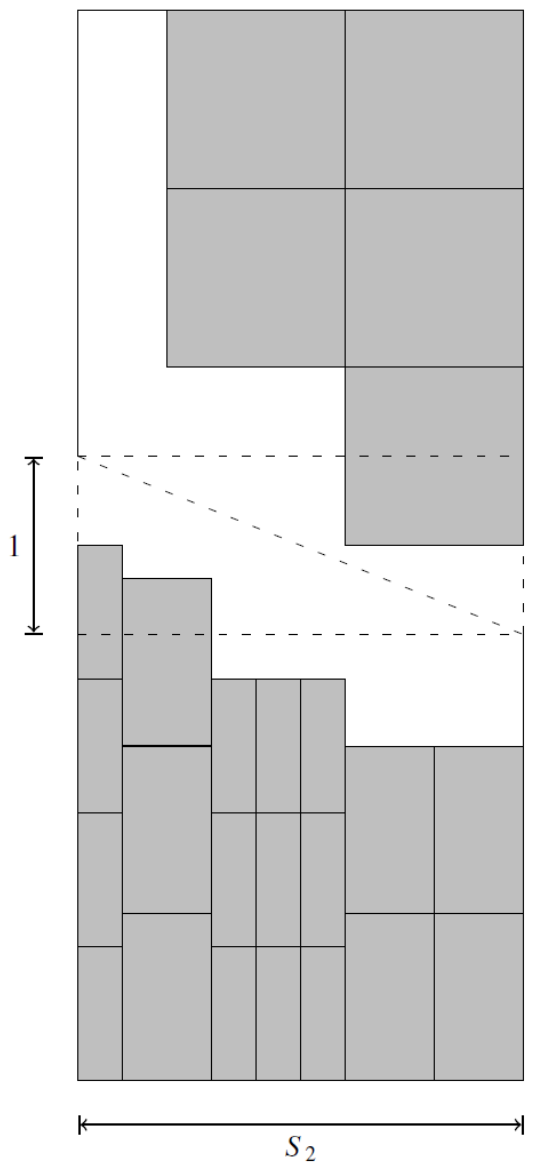

Figure 2.5: Figure from [7] that shows how the algorithm arranges the rectangles in S_2. The height increase is at most 1.

If the fractional solution produced by the solution of linear program (2.1) uses three configurations, then we use the algorithm described in Chapter 3 to transform it into a solution for 3T-SPP.

Chapter 3

Algorithm for the Three-Type Strip Packing Problem

This chapter presents an algorithm that converts an optimal fractional packing for 3T-SPP into an integral packing with a height increase of at most 5/3 i.e. a performance bound of $OPT + 5/3$. The input to this algorithm is an optimal fractional solution F for 3T-SPP that uses at most three configurations, obtained by computing a basic feasible solution for the linear program (2.1). As mentioned above, if the fractional solution F only uses one configuration, then we round up the rectangles of fractional height, as described in Section 2.2, so they become whole and this produces a solution of height at most $OPT + 1$. If F uses two configurations, then we use the algorithm for HM-SPP described in Section 2.2 to pack the rectangles in a solution of height at most $OPT + 1$. If F uses three configurations, we convert it into an integral solution by using the algorithms described in Sections 3.1 – 3.7.

3.1 Partitioning the Fractional Packing

Let the fractional solution F use three configurations: C_1, C_2, and C_3. As explained in Chapter 2, rectangles are packed according to these configurations into 3 sections, which are stacked one on top of the other, with the section corresponding to C_3 at the bottom, C_2 at the middle, and C_1 at the top. The rectangles are arranged within each one of these three sections into columns, where each column is formed by rectangles of the same type. The three sections, or configurations, are placed so that the top of each configuration touches the bottom of the configuration on top of it. Figure 3.1 shows an example of three configurations in a fractional solution with three rectangle types.

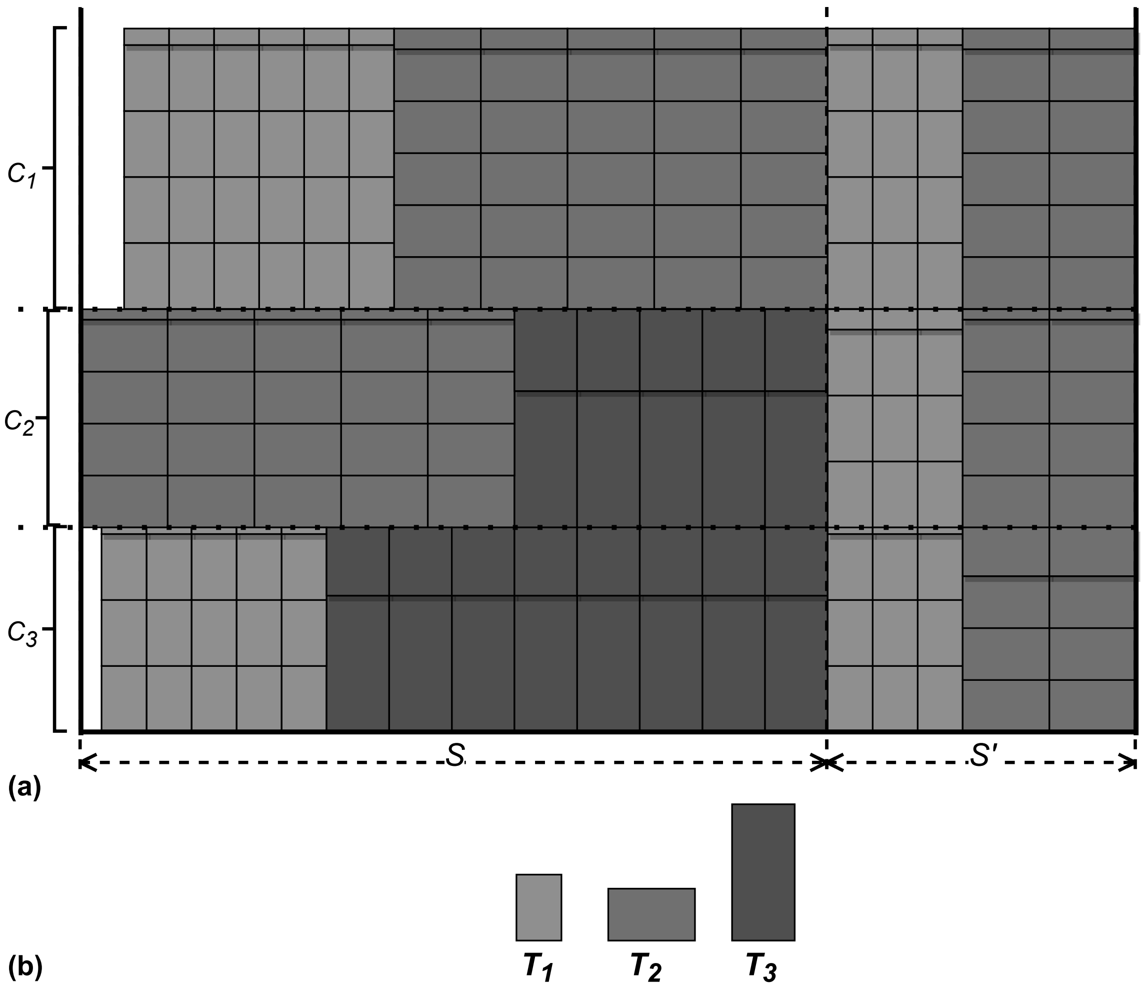

Figure 3.1: (a) Partitioning a fractional packing into two parts: S' contains rectangles common in all configurations, and S contains the remaining rectangles. (b) The three rectangle types used in (a).

We first show that we can rearrange the rectangles within each configuration so that a common part S' exists where rectangles of the same type line up between configurations and, hence, are not cut at the boundaries between them. Outside this common part, no type of rectangle appears in all three configurations.

For each rectangle type T_i, let $t_{i,j}$ be the total number of rectangles of type T_i across configuration C_j: here, the word "across" means that if we draw a horizontal line across C_j, this line would intersect $t_{i,j}$ rectangles of type T_i. Let $m_i = \min\left\{t_{i,j} \mid j = 1, 2, 3\right\}$. In Figure 3.1, for example, the value of m_1 is 3 because $t_{1,1} = 9$, $t_{1,2} = 3$, and $t_{1,3} = 8$. For each $i = 1, 2, 3$, we shift m_i columns of rectangles of type T_i to the rightmost position in all configurations. We call the part of the solution containing the rectangles that were shifted, S'. The part containing the remaining rectangles is called S (see Figure 3.1). For all $i = 1, 2, 3$, the value of $t_{i,j}$ - m_i is zero for at least one configuration C_j. In Figure 3.1 for example, the value of $t_{1,2}$ - m_1 is zero. Therefore, no type T_i will appear in S in all configurations.

Depending on the input, configurations could have different types of rectangles in S. For example, in Figure 3.1 each configuration has two rectangle types in S. For a different input, one configuration could have, for example, three rectangle types while the other two configurations could have only one rectangle type.

If S' is empty, the problem is simpler as only rectangles in S need to be packed. Similarly, if S is empty, the problem is simpler as only rectangles in S' need to be packed. Our algorithm for converting a fractional solution for 3T-SPP into an integral solution is given below:

Algorithm 3.1 roundThreeTypes(F)

```
 1: In: An optimal fractional solution $F$ for fractional 3T-SPP.
 2: Out: An integral packing
 3: Arrange the configurations $C_1$, $C_2$, and $C_3$ in $F$ so that they are stacked one on top of the
    other, with $C_3$ at the bottom, $C_2$ at the middle, and $C_1$ at the top.
 4: for all rectangle types $T_i$ do
 5:     $m_i \leftarrow \min\left\{t_{i,j} \mid j = 1, 2, 3\right\}$
 6:     if $m_i > 0$ then
 7:         Horizontally rearrange rectangles by shifting $m_i$ columns of rectangles of type $T_i$
            in all configurations to the rightmost position of the strip.
 8:         Round up the rectangles with fractional height of type $T_i$ located at the top of $S'$,
            as described in Section 3.2.
 9: if the non-common part $S$ is not empty then
10:     partitionPartS($T$, $C$) // algorithm described in Section 3.4
11:     packSegments($T$, $C$) // algorithm described in Sections 3.5 and 3.6
12:     removeFractionalRectangles($T$, $C$) // algorithm described in Section 3.7
13: return the integral packing produced
```

3.2 Packing the Common Part S'

In S' we pack rectangles of the same type on top of each other (see Figure 3.2). Hence, rectangles now are not cut between adjacent configurations; only rectangles in the uppermost configuration, C_1, might be cut at the top of the packing (see Figure 3.2). Therefore, we only need to round up any rectangles with fractional height at the top of S' so that they become whole. This rounding increases the height of S' by at most 1, because the tallest rectangle type has height 1.

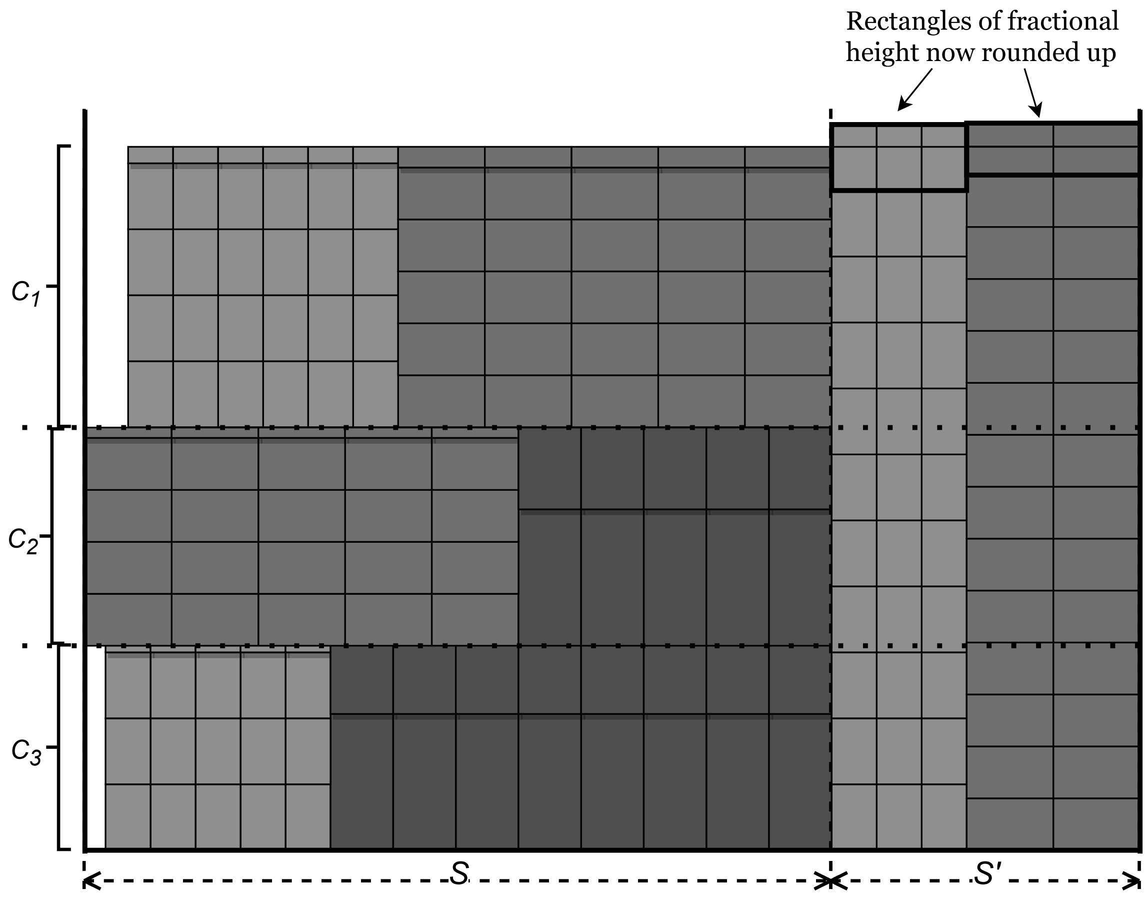

Figure 3.2: Partition of a fractional packing where rectangles in S' with fractional height are rounded up. Note how in S' only the rectangles with fractional height at the top of the packing are rounded up.

3.3 Rounding Up Rectangles with Fractional Height in the non-Common Part *S*

In part *S*, rectangles do not necessarily line up between configurations and hence they might be cut at the boundaries between configurations. Rectangles in the uppermost configuration, C_1, might also be cut at the top of the packing. We cannot just round up all the rectangles that have fractional height without increasing the height of part *S* by up to 3 units.

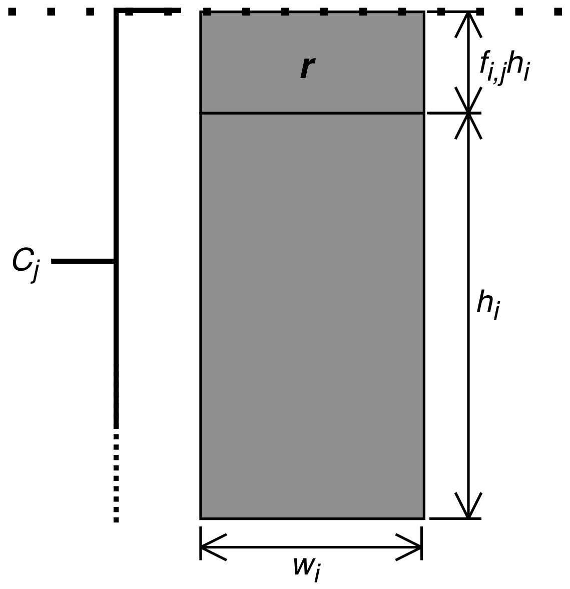

Figure 3.3: Two rectangles of type T_i where one is a whole rectangle and the other is a rectangle of fractional height. The horizontal dotted line indicates where the rectangle r is cut at the top of the configuration C_j. From Definition 2.1 $f_{i,j}$ is the fraction of the rectangle r of type T_i that lies just below the top of C_j. Then $f_{i,j}h_i$ is the height of r.

Let $a_{i,j}$ be the total area of the rectangles of type T_i that the fractional solution packs in configuration C_j. Recall that w_i and h_i are the width and height of rectangle type T_i, respectively. Then $n_{i,j} = a_{i,j} / w_ih_i$ is the (fractional) number of rectangles of type T_i that are in C_j. Although this number might be fractional, the total number, $n_i = n_{i,1} + n_{i,2} + n_{i,3}$, of rectangles of type T_i over all configurations is integer for any type T_i.

Recall from Definition 2.1 that $f_{i,j}$ is the fraction of each rectangle of type T_i that lies just below the top of C_j. Hence, $f_{i,j}h_i$ is the height of the rectangles of type T_i that lie just below the top of C_j (see Figure 3.3). We call a fractional rectangle *short* if $f_{i,j} < 1/3$ and we call it *tall* if $f_{i,j} \geq 1/3$. For example, in Figure 3.4, fractional rectangle r' of type $T_{i'}$ is *tall* as $f_{i',j} \geq 1/3$.

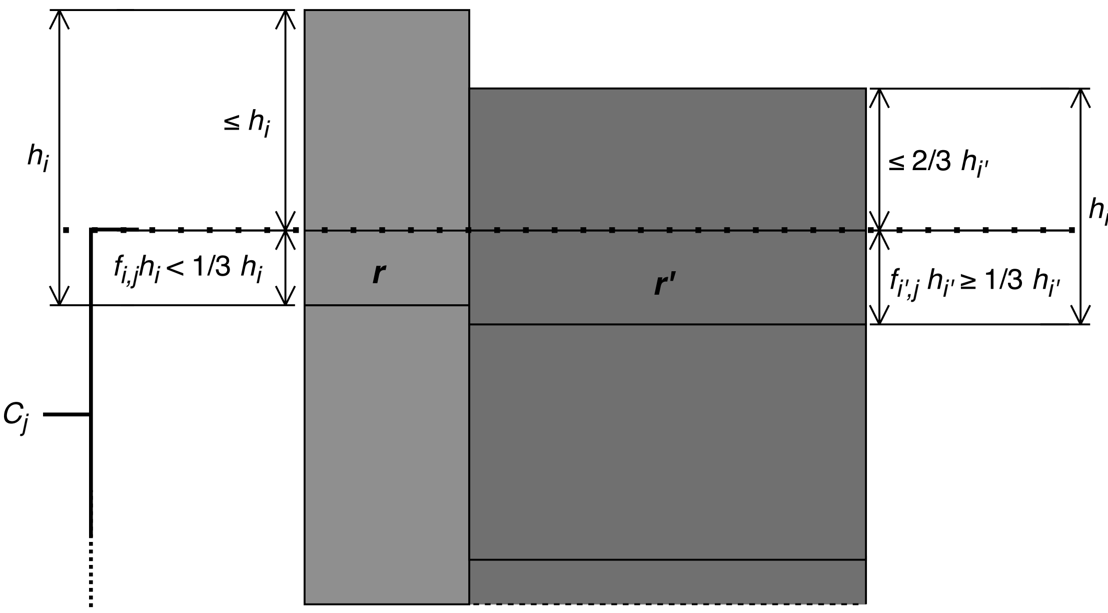

FIGURE 3.4: Rounding up *short* and *tall* fractional rectangles. A *short* fractional rectangle r is to the left; a *tall* fractional rectangle r' is to the right.

If we round up a *short* fractional rectangle to form a whole rectangle, the height of the packing might increase by up to 1; however, if we round up a *tall* fractional rectangle, the height of the packing will increase by at most 2/3 units (see Figure 3.4).

3.4 Partitioning the non-Common Part *S*

For each configuration C_j, we calculate the fraction $f_{i,j}$ for each rectangle type T_i in part S. We then sort these fractions in nondecreasing order so that rectangle types with smaller fractions $f_{i,j}$ are to the left while rectangle types with larger fractions are to the right. For example, in configuration C_1 of Figure 3.5, rectangles of type T_1 are to the left of rectangles of type T_2 because $f_{1,1} = 1/4$ is less than $f_{2,1} = 2/5$.

In part S we identify the positions where each configuration switches from either empty space or some rectangle type to a different rectangle type. From these positions, we create several columns denoted by k_c. Within a column, each configuration has at most one rectangle type. The subscript c takes values from 1 to the total number of columns (see Figure 3.5).

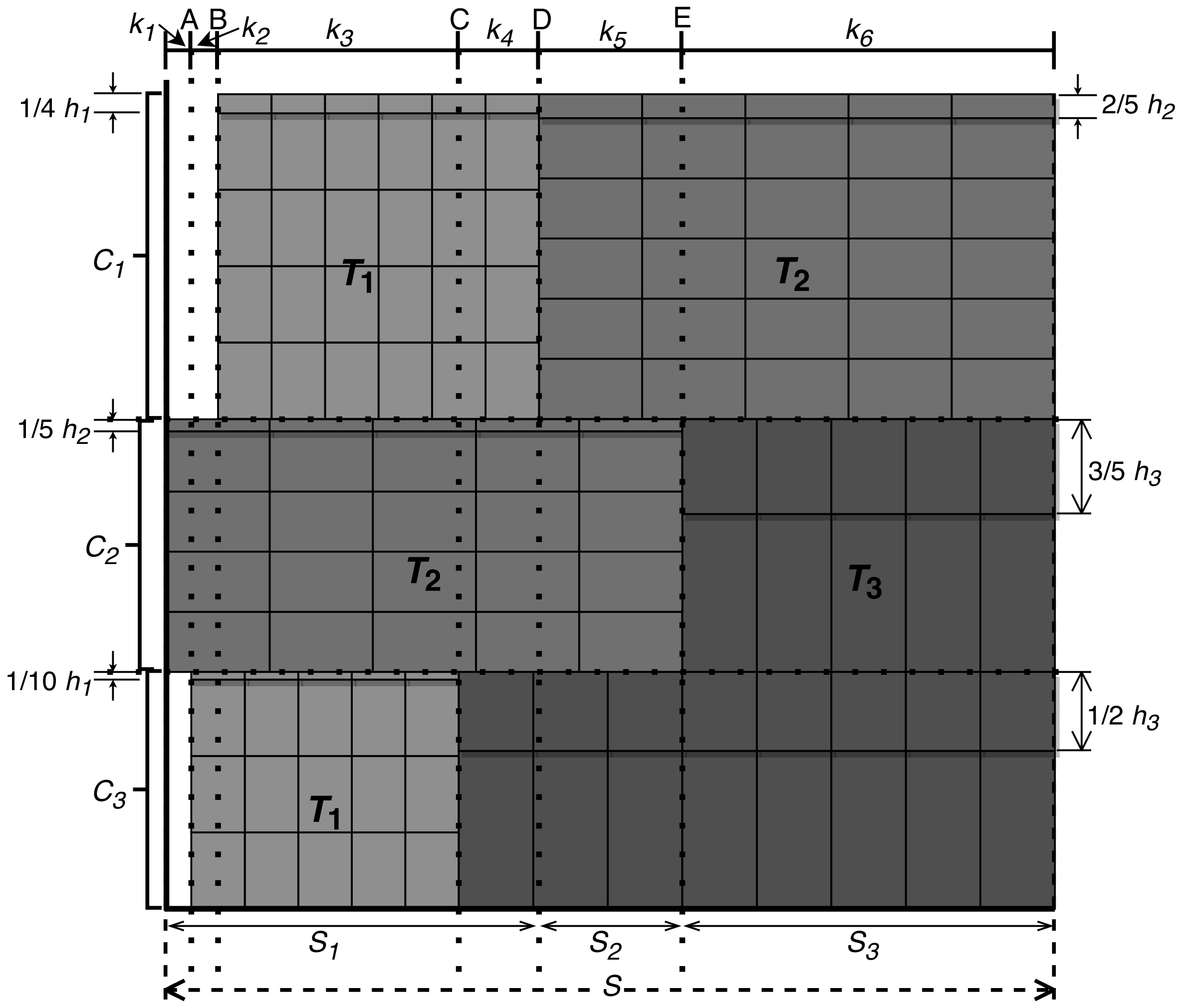

Figure 3.5: Three partitions of S: S_1, S_2, and S_3. Rectangle types in S are sorted for each configuration C_j in nondecreasing order of $f_{i,j}$. As we scan S from left to right within a configuration, rectangles with small fractional values are to the left whereas rectangles with larger fractional values are to the right. We draw vertical lines at points A, B, C, D, E where a configuration switches from either empty space or some rectangle type to a different rectangle type; k_c denotes the columns created by these vertical lines, where subscript c increases as we scan S from left to right. In each column, a configuration has at most one rectangle type.

For each configuration in part S there are at most two positions where it switches from a rectangle type to a different rectangle type because a configuration can have at most three rectangle types. For illustrative purposes, in Figure 3.5 we scan S from left to right with a vertical sweepline and draw a vertical line at each point where a configuration switches from either empty space or some rectangle type to a different rectangle type. For example, we draw a vertical line at point **A** because in C_3 the vertical sweepline touches the left side of the leftmost rectangle in C_3. We draw a vertical line at point **C** because in C_3 the vertical sweepline touches the border between a rectangle of type T_1 and a rectangle of type T_3. We use the vertical lines discussed above to create columns k_c. Note how for each border of a column there is at least one configuration whose rectangles are not split by that border.

Using these columns, we partition S into at most three segments, S_1, S_2, and S_3, as follows. Let $F_{j,c}$ be the fraction of each rectangle that lies just below the cutting line in configuration C_j for column k_c. Segment S_1 contains all those columns k_c for which $\sum_{j=1}^{3} F_{j,c} \leq 1$, segment S_2 contains those columns for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$, and columns for which $\sum_{j=1}^{3} F_{j,c} > 4/3$ can be in either segment S_2 or S_3 as described in Subsection 3.4.1. For example, in Figure 3.5, S_1 contains columns k_1, k_2, k_3 and k_4 because $\sum_{j=1}^{3} F_{j,1}$ and $\sum_{j=1}^{3} F_{j,2}$ clearly have value less than $\sum_{j=1}^{3} F_{j,3} = 1/4 + 1/5 + 1/10 = 11/20 = 0.55$, and $\sum_{j=1}^{3} F_{j,4} = 1/4 + 1/5 + 1/2 = 19/20 = 0.95$. Segment S_2 contains column k_5 because $\sum_{j=1}^{3} F_{j,5} = 2/5 + 1/5 + 1/2 = 11/10 = 1.1$. Segment S_3 contains column k_6 as explained in Subsection 3.4.1.

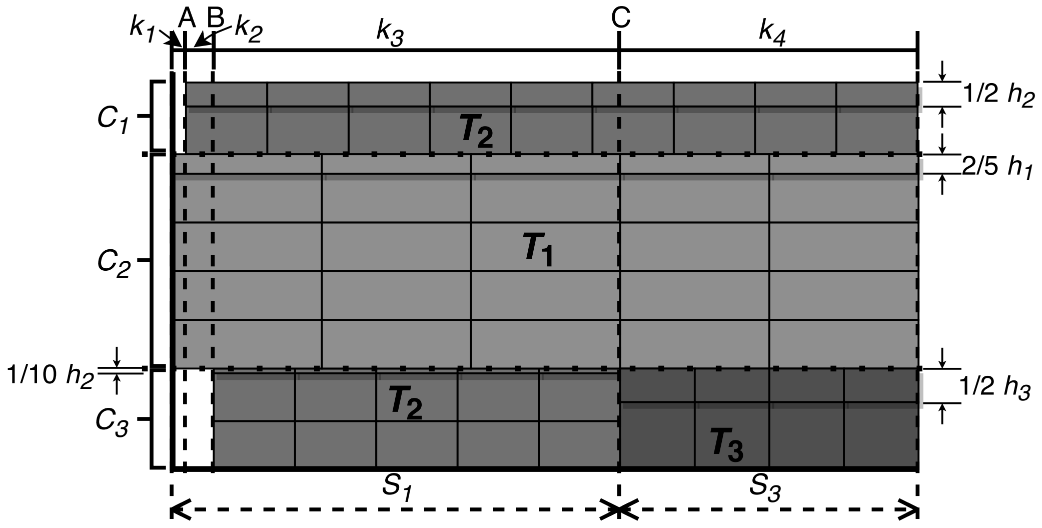

FIGURE 3.6: Two partitions of S where no columns k_c exist for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$. We draw vertical lines at points A, B, C where a configuration switches from either empty space or a rectangle type to a different rectangle type. Columns for which $\sum_{j=1}^{3} F_{j,c} \leq 1$ are to the left of point C, and columns for which $\sum_{j=1}^{3} F_{j,c} > 4/3$ are to the right. Only one rectangle with fractional height is vertically cut at point C, so S_3 contains column k_4 as explained in Subsection 3.4.1.

3.4.1 Determining Which Columns are in S_2 or S_3

Definition 3.1. Denote with P_σ the position of the left border of the leftmost column k_c for which $\sum_{j=1}^{3} F_{j,c} > 4/3$.

If $\sum_{j=1}^{3} F_{j,c} > 4/3$ for each column $k_c \in S$, then all these columns are in segment S_3. If fewer than two rectangles with fractional height cross the vertical cutting line at P_σ, then those columns for which $\sum_{j=1}^{3} F_{j,c} > 4/3$ are all in S_3. For example, one rectangle with fractional height in Figure 3.5 crosses the vertical cutting line at point **E**, so S_3 contains column k_6 because $\sum_{j=1}^{3} F_{j,6} = 2/5 + 3/5 + 1/2 = 3/2 = 1.5$. In Figure 3.6 one rectangle with fractional height crosses the vertical cutting line at point **C**, so S_3 contains column k_4 because $\sum_{j=1}^{3} F_{j,4} = 1/2 + 2/5 + 1/2 = 7/5 = 1.4$.

However, if two rectangles with fractional height cross the vertical cutting line at P_σ, then we need to consider two cases. Let r_1 and r_2 be the rectangles of fractional height that cross the vertical line at P_σ, and let r_1 and r_2 belong to configurations C_k and $C_{k'}$ respectively.

Definition 3.2. If two rectangles r_1 and r_2 of fractional height cross the vertical cutting line at P_σ, let P_{min} be the leftmost position between the right sides of r_1 and r_2.

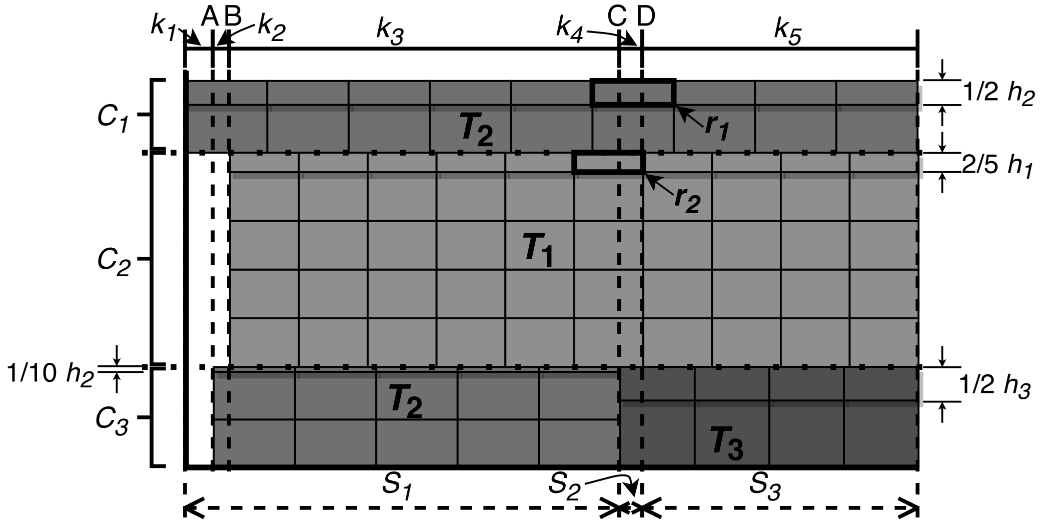

Figure 3.7: Three partitions of S where no columns k_c exist for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$. Rectangles r_1 and r_2 with fractional height cross the vertical line at point C. We draw a vertical line at point D because the right border of r_2 is closest to the left border of S; column k_4 is in S_2.

1. If no column exists for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$, then segment S_2 contains all columns between P_σ and P_{min}, and segment S_3 contains all columns that are to the right of P_{min}. For example, in Figure 3.7, the right border of r_2 is closest to the left border of part S so we draw a vertical line at point **D** to create column k_4. Segment S_2 contains column k_4 because it is between points **C** and **D**. Segment S_3 contains column k_5 because it is to the right of point **D**. Note how S_2 cannot be empty in this case.

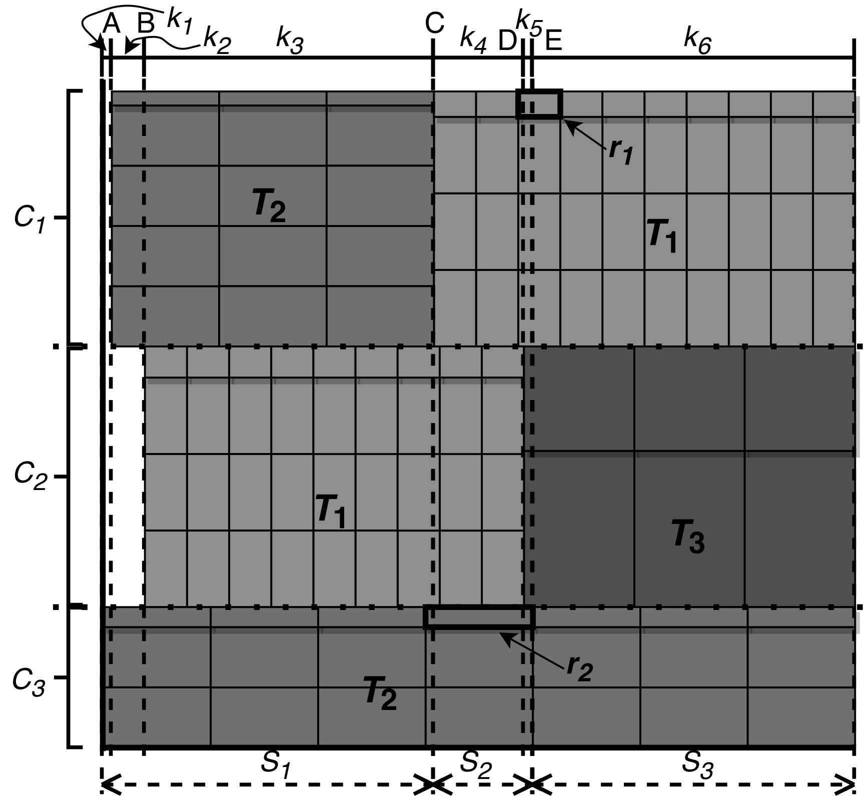

FIGURE 3.8: Three partitions of S where S_2 has a column where $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$ and also has a column where $\sum_{j=1}^{3} F_{j,c} > 4/3$. Rectangles r_1 and r_2 with fractional height cross the vertical line at point D. Note how r_1 and r_2 are not in C_2; in this figure, C_2 has the largest fractional value $F_{j,c}$ in column k_4. We draw a vertical line at point E because the right border of r_2 is closest to the left border of S; column k_5 is in S_2. Because C_3 has no rectangles that cross the boundary between S_2 and S_3, we rename and reorder configurations so that C_3 becomes C_2.

2. If columns exist for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$, then define configuration C_α as follows:

Definition 3.3. If columns exist for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$, then we denote with C_α the configuration with the largest fractional value $F_{\alpha,c}$ in the leftmost column k_c for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$.

In Figure 3.8, the leftmost column k_c for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$ is k_4 and so C_α is C_2. If r_1 or r_2 are in C_α, then all those columns for which $\sum_{j=1}^{3} F_{j,c} > 4/3$ are in segment S_3. Otherwise, (recall from Definition 3.2 that P_{min} is the leftmost position between the right sides of r_1 and r_2) we place in segment S_2 all columns between P_σ and P_{min} and we place in segment S_3 all columns that are to the right of P_{min}.

For example, in Figure 3.8, the right border of r_2 is closest to the left border of part S so we draw a vertical line at point **E** to create column k_5. Segment S_2 contains column k_5 because it is between points **D** and **E**. Segment S_3 contains column k_6 because it is to the right of point **E**.

There cannot be three rectangles with fractional height that cross the vertical line at P_σ because for each border of a column there is at least one configuration whose rectangles are not split by that border. For example, the border at P_{min} will not split either r_1 or r_2.

3.4.2 Reordering Configurations

We vertically cut rectangles of fractional height at the boundary between two adjacent segments; whole rectangles that cross the boundary between two segments are not vertically cut as explained in Section 3.6.

Lemma 3.4.1 *We can rename and reorder configurations so that configuration C_2 does not have rectangles that cross the boundary between segment S_3 and its adjacent segment.*

Proof Recall that for any border of a column there is at least one configuration whose rectangles are not split by that border. Thus, for each segment S_1, S_2, S_3, at least one configuration does not have rectangles that cross the boundary between that segment and an adjacent one. If S_3 and another segment are not empty, we rename and reorder the configurations so that the configuration that has no rectangles that cross the boundary between the two rightmost segments becomes C_2. □

In Figure 3.5, rectangles of type T_1 in configuration C_1 do not cross the boundary between segments S_1 and S_2 (see point **D**); rectangles of type T_3 in configurations C_2 and C_3 do not cross the boundary between segments S_2 and S_3 (see point **E**). Therefore, we do not need to reorder the configurations because C_2 does not have any rectangles that cross the boundary between S_2 and S_3. In Figure 3.8, C_3 has no rectangles that cross the boundary between S_2 and S_3, so we rename and reorder configurations so that C_3 becomes C_2 and vice versa.

Lemma 3.4.2 *Assume segments S_1 and S_3 are not empty and segment S_2 is empty. We can rename and reorder configurations so that:*

1. *Lemma 3.4.1 holds, and*
2. *at most one rectangle with fractional height crosses the boundary between S_1 and S_3, and this rectangle is in configuration C_1.*

Proof First, we rename and reorder configurations according to Lemma 3.4.1 so that configuration C_2 does not have any rectangle of fractional height that crosses the boundary between S_1 and S_3. In this case, no column k_c exists for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$ because as explained in Section 3.4 then S_2 would contain all these columns, but S_2 is empty. Since S_2 is empty then, as explained in Subsection 3.4.1, there cannot be two rectangles with fractional height that cross the vertical cutting line at P_σ. Therefore, at most one rectangle with fractional height crosses the vertical cutting line at P_σ (the cutting line is the left border of S_3). If configuration C_3 has a rectangle of fractional height that crosses the boundary between S_1 and S_3, we rename and reorder the configurations so that C_3 becomes C_1 and vice versa. □

Lemma 3.4.3 *Assume segment S_2 is not empty and it contains no columns k_c for which* $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$. *Then two rectangles with fractional height cross the vertical cutting line at P_σ.*

Proof In this case, at least one column k_c exists for which $\sum_{j=1}^{3} F_{j,c} \leq 4/3$ because otherwise S_2 would be empty as all columns in part S would be in segment S_3 as described in Subsection 3.4.1. Therefore, by the assumption of the lemma k_c must belong to segment S_1. As described in Subsection 3.4.1, if fewer than two rectangles with fractional height cross the vertical cutting line at P_σ, then S_2 would be empty because S_3 would contain all columns for which $\sum_{j=1}^{3} F_{j,c} > 4/3$. Therefore, two rectangles with fractional height must cross the vertical cutting line at P_σ. □

Definition 3.4. Assume rectangles r_1 and r_2 with fractional height cross the vertical cutting line at P_σ, where P_σ is the position of the left border of the leftmost column for which $\sum_{j=1}^{3} F_{j,c} > 4/3$ as stated in Definition 3.1. Let r_1 and r_2 belong to configurations C_k and $C_{k'}$ respectively. We denote with C_β the configuration that is neither C_k nor $C_{k'}$.

Lemma 3.4.4 *Assume segment S_2 is not empty and contains no columns k_c for which* $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$. *Then $F_{\beta,c} > 1/3$ and $\sum_{j=1}^{3} F_{j,c} - F_{\beta,c} < 1$ for each column $k_c \in S_2$.*

Proof In this case two rectangles r_1 and r_2 with fractional height cross the vertical cutting line at P_σ according to Lemma 3.4.3. As stated in Definition 3.2, let P_{min} be the leftmost position between the right sides of r_1 and r_2. Because no column k_c exists for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$, S_2 only contains columns between P_σ and P_{min} as stated in Subsection 3.4.1. Thus, $\sum_{j=1}^{3} F_{j,c} > 4/3$ for each column in S_2; these columns must have part of r_1 and part of r_2 because these rectangles intersect or touch P_σ and P_{min}.

Since r_1 and r_2 cross the vertical cutting line at P_σ and columns to the left of P_σ are in S_1, then part of r_1 and part of r_2 must be in a column for which $\sum_{j=1}^{3} F_{j,c} \leq 1$. Therefore, $F_{k,c} + F_{k',c} < 1$ for each column $k_c \in S_2$. As $\sum_{j=1}^{3} F_{j,c} > 4/3$ for each column in S_2, then $F_{\beta,c} > 1/3$ for each column k_c in S_2. Thus, $\sum_{j=1}^{3} F_{j,c} - F_{\beta,c} < 1$ for each column in S_2. □

Lemma 3.4.5 *Assume segments S_2 and S_3 are not empty and S_2 contains no columns k_c for which* $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$. *We can rename and reorder the configurations so that:*

1. *Lemma 3.4.1 holds,*
2. *at most one rectangle with fractional height crosses the boundary between S_2 and S_3, and this rectangle is in configuration C_1, and*
3. *C_β is either configuration C_2 or configuration C_3.*

Proof According to Lemma 3.4.3, two rectangles r_1 and r_2 with fractional height cross the vertical cutting line at P_σ for this case. Recall from Subsection 3.4.1 and Definition 3.2 that P_{min} is the leftmost position between the right sides of r_1 and r_2, and that if S_3 is not empty P_{min} is at the boundary between S_2 and S_3. As stated in Definition 3.4, C_β is the configuration that does not contain neither r_1 nor r_2.

We first rename and reorder configurations according to Lemma 3.4.1. By Definition 3.2, only one of r_1 and r_2 can cross the boundary at P_{min}; if that rectangle is in C_3 and C_β is C_2 then we rename and reorder configurations so that C_3 becomes C_1 and vice versa.

If C_β is C_1, we rename and reorder configurations so that C_3 becomes C_1 and vice versa. Then if there is a rectangle with fractional height that crosses the boundary at P_{min} that rectangle must be in C_1 because by the definition of P_{min}, P_{min} must be the right boundary of C_2. □

Recall from Definition 3.3 that if columns exists for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$, then C_α is the configuration with the largest fractional value $F_{\alpha,c}$ in the leftmost column k_c for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$.

Lemma 3.4.6 *Assume S_2 is not empty and contains columns k_c for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$. Then $F_{\alpha,c} > 1/3$ and $\sum_{j=1}^{3} F_{j,c} - F_{\alpha,c} < 1$ for each column in S_2.*

Proof In this case, if at most one rectangle with fractional height crosses the vertical line at P_σ, then as described in Subsection 3.4.1 those columns for which $\sum_{j=1}^{3} F_{j,c} > 4/3$ are all in S_3 so that P_σ is at the boundary between S_2 and S_3. There must be in S_2 at least one rectangle type T_i in C_α such that $f_{i,\alpha} > 1/3$ and $F_{\alpha,c} > 1/3$ because $\sum_{j=1}^{3} F_{j,c} > 1$ for each column $k_c \in S_2$. Thus, $\sum_{j=1}^{3} F_{j,c} - F_{\alpha,c} < 1$ for each column k_c in S_2 because $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$ for each column in S_2.

If two rectangles r_1 and r_2 with fractional height cross the vertical cutting line at P_σ, we need to consider two additional cases. Let r_1 and r_2 belong to configurations C_k and $C_{k'}$ respectively.

1. If r_1 or r_2 is in C_α, then as described in Subsection 3.4.1 all those columns for which $\sum_{j=1}^{3} F_{j,c} > 4/3$ are in S_3 so that P_σ is at the boundary between S_2 and S_3. By the same above reasoning, $F_{\alpha,c} > 1/3$ and $\sum_{j=1}^{3} F_{j,c} - F_{\alpha,c} < 1$ for each column in S_2.

2. If neither r_1 nor r_2 are in C_α then, as stated in Subsection 3.4.1, S_2 contains columns k_c between P_σ and P_{min} where $\sum_{j=1}^{3} F_{j,c} > 4/3$; recall from Definition 3.2 that P_{min} is the leftmost position between the right sides of r_1 and r_2. The columns between P_σ and P_{min} must have part of r_1 and part of r_2 because these rectangles intersect or touch P_σ and P_{min}.

   There must be in S_2 at least one rectangle type T_i in C_α such that $f_{i,\alpha} > 1/3$ and $F_{\alpha,c} > 1/3$ because $\sum_{j=1}^{3} F_{j,c} > 1$ for each column $k_c \in S_2$. Since r_1 and r_2 cross the vertical cutting line at P_σ and columns for which $\sum_{j=1}^{3} F_{j,c} \leq 4/3$ are to the left of P_σ, part of r_1 and part of r_2 must be in a column for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$. However, since r_1 and r_2 are not in C_α, then $F_{k,c} + F_{k',c} < 1$ for each column in S_2 that contains parts of r_1 and r_2. Thus, $\sum_{j=1}^{3} F_{j,c} - F_{\alpha,c} < 1$ for each column k_c in S_2. □

Lemma 3.4.7 *Assume segments S_2 and S_3 are not empty and S_2 contains columns k_c for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$. We can rename and reorder configurations so that:*

1. *Lemma 3.4.1 holds, and*

2. *any rectangle of fractional height not in configuration C_α that crosses the boundary between S_2 and S_3 must be in configuration C_1.*

Proof According to Defintion 3.3, C_α is the configuration with the largest fractional value $F_{\alpha,c}$ in the leftmost column k_c for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$. In this case, if at most one rectangle with fractional height crosses the vertical cutting line at P_σ, then as described in Lemma 3.4.6 P_σ is at the boundary between S_2 and S_3. After renaming and reordering configurations according to Lemma 3.4.1, if C_3 has a rectangle with fractional height that crosses the vertical line at P_σ and C_3 is not C_α, then we rename and reorder configurations so that C_3 becomes C_1 and vice versa.

If two rectangles r_1 and r_2 with fractional height cross the vertical line at P_σ, we need to consider two cases.

1. If r_1 or r_2 is in C_α, then as described in Lemma 3.4.6 P_σ is at the boundary between S_2 and S_3. After renaming and reordering configurations according to Lemma 3.4.1, by definition r_1 and r_2 cannot be in C_2. If C_3 is not C_α, we rename and reorder configurations so that C_3 becomes C_1 and vice versa.

2. If neither r_1 nor r_2 are in C_α then, if S_3 is not empty, P_{min} is at the boundary between S_2 and S_3; recall from Definition 3.2 that P_{min} is the leftmost position between the right sides of r_1 and r_2. We first rename and reorder the configurations according to Lemma 3.4.1. By Definition 3.2, only one of r_1 and r_2 can cross the boundary at P_{min}; if that rectangle is in C_3 and C_α is C_2, then we rename and reorder configurations so that C_3 becomes C_1 and vice versa.

   If C_α is C_1, we rename and reorder configurations so that C_3 becomes C_1 and vice versa. Then if there is a rectangle with fractional height that crosses the boundary at P_{min} that rectangle must be in C_1 because by the definition of P_{min}, P_{min} must be the right boundary of C_2. □

After renaming and reordering configurations according to Lemmas 3.4.2, 3.4.5, and 3.4.7, we invert configuration C_1 so that rectangles with fractional height are at the bottom of C_1; for the rest of this thesis, we assume that the configuration at the top of the packing is inverted (for example, see Figure 3.14). Pseudocode for the algorithm that partitions S into segments S_1, S_2, and S_3 is given below.

Algorithm 3.2 partitionPartS(T, C)

```
1: In: A set $T$ of three rectangle types and a set $C$ of three configurations in an optimal
       fractional packing
2: colS1, colS2, colS3 ← empty sets // sets of positions for the columns of $S_1$, $S_2$, and $S_3$
3: positions ← empty set // used to partition $S$ into columns
4: if $S'$ is empty then
5:     positions ← positions ∪ {1} // strip's rightmost border is at position 1
6: else
7:     positions ← positions ∪ {horizontal position of boundary between $S$ and $S'$}
8: for each configuration $C_j$ do
9:     Calculate the fraction $f_{i,j}$ as defined in Definition 2.1 for each rectangle type $T_i$ in part
       $S$ of $C_j$.
10:    Sort these fractions in nondecreasing order.
11:    Sort rectangles in part $S$ of $C_j$ by rectangle type so that rectangle types are in nonde-
       creasing order of $f_{i,j}$.
12:    positions ← positions ∪ {positions in part $S$ where $C_j$ switches from a rectangle type
                  to a different rectangle type}
13:    positions ← positions ∪ {1 - total width of $C_j$} // position where $C_j$ ends
14: Sort positions in nondecreasing order and remove duplicates.
15: spotS2 ← -1 // change value if we encounter a column $k_c$ for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$.
16: for each value P in positions do
17:    if P is the last value in positions then
18:        Invert configuration $C_1$.
19:        return
20:    P2 ← the first value after P in positions
21:    Let $k_c$ be the column created by positions P and P2
22:    if $\sum_{j=1}^{3} F_{j,c} \leq 1$ then
23:        if $k_c$ is the leftmost column of packing then
24:            colS1 ← colS1 ∪ {P}
25:        colS1 ← colS1 ∪ {P2}
26:    else if $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$ then
27:        if spotS2 = -1 then
28:            spotS2 ← configuration with the largest fractional value $F_{j,c}$ in column $k_c$
29:            colS2 ← colS2 ∪ {P}
30:        colS2 ← colS2 ∪ {P2}
31:    else
32:        Break out of for loop. // column $k_c$ could be in $S_2$ or $S_3$ so handle separately
```

```
33: $P_{min} \leftarrow$ -1 // changes value if $S_2$ contains column(s) for which $\sum_{j=1}^{3} F_{j,c} > 4/3$
34: $P_{\sigma} \leftarrow$ -1 // if $S_3$ has an adjacent segment, then changes value to the position of the left
35:             border of the leftmost column $k_c$ for which $\sum_{j=1}^{3} F_{j,c} > 4/3$
36: **if** colS1 is not empty or colS2 is not empty **then**
37:     $P_{\sigma} \leftarrow$ max{largest value from colS1, largest value from colS2}
38:     **if** two rectangles of fractional height cross the vertical cutting line at $P_{\sigma}$ **then**
39:         Let $r_1$ and $r_2$ be the rectangles of fractional height that cross the vertical cutting
            line at $P_{\sigma}$ in configurations $C_k$ and $C_{k'}$ respectively.
40:         **if** colS2 is empty **then**
41:             $P_{min} \leftarrow$ leftmost position between the right sides of $r_1$ and $r_2$.
42:             **if** $P_{min}$ is not in *positions* **then**
43:                 $positions \leftarrow positions \cup \{P_{min}\}$ // add $P_{min}$ in its sorted position
44:         **else**
45:             **if** neither $r_1$ nor $r_2$ are in $C_{spotS2}$ **then**
46:                 $P_{min} \leftarrow$ leftmost position between the right sides of $r_1$ and $r_2$.
47:                 **if** $P_{min}$ is not in *positions* **then**
48:                     $positions \leftarrow positions \cup \{P_{min}\}$ // add $P_{min}$ in its sorted position
49: **for** each value P in *positions* that is greater than or equal to $P_{\sigma}$ **do**
50:     **if** P is the last value in *positions* **then**
51:         Break out of for loop
52:     P2 $\leftarrow$ the first value after P in *positions*
53:     Let $k_c$ be the column created by positions P and P2
54:     **if** $S_2$ is empty and P = $P_{\sigma}$ and $P_{min} \neq$ -1 **then**
55:         colS2 $\leftarrow$ colS2 $\cup$ {P}
56:     **if** P = $P_{min}$ or (P = $P_{\sigma}$ and $P_{min}$ = -1) or $k_c$ is the leftmost column of packing **then**
57:         colS3 $\leftarrow$ colS3 $\cup$ {P}
58:     **if** P2 $\leq P_{min}$ **then**
59:         colS2 $\leftarrow$ colS2 $\cup$ {P2}
60:     **else**
61:         colS3 $\leftarrow$ colS3 $\cup$ {P2}
62: **if** $S_1$ and $S_3$ are not empty and $S_2$ is empty **then**
63:     Rename and reorder configurations according to Lemma 3.4.2.
64: **else if** $S_2$ and $S_3$ are not empty **then**
65:     Rename and reorder configurations according to Lemmas 3.4.5 and 3.4.7.
66: Invert configuration $C_1$.
```

3.5 Packing Segments S_1, S_2, and S_3

3.5.1 Packing Rectangles in S_1

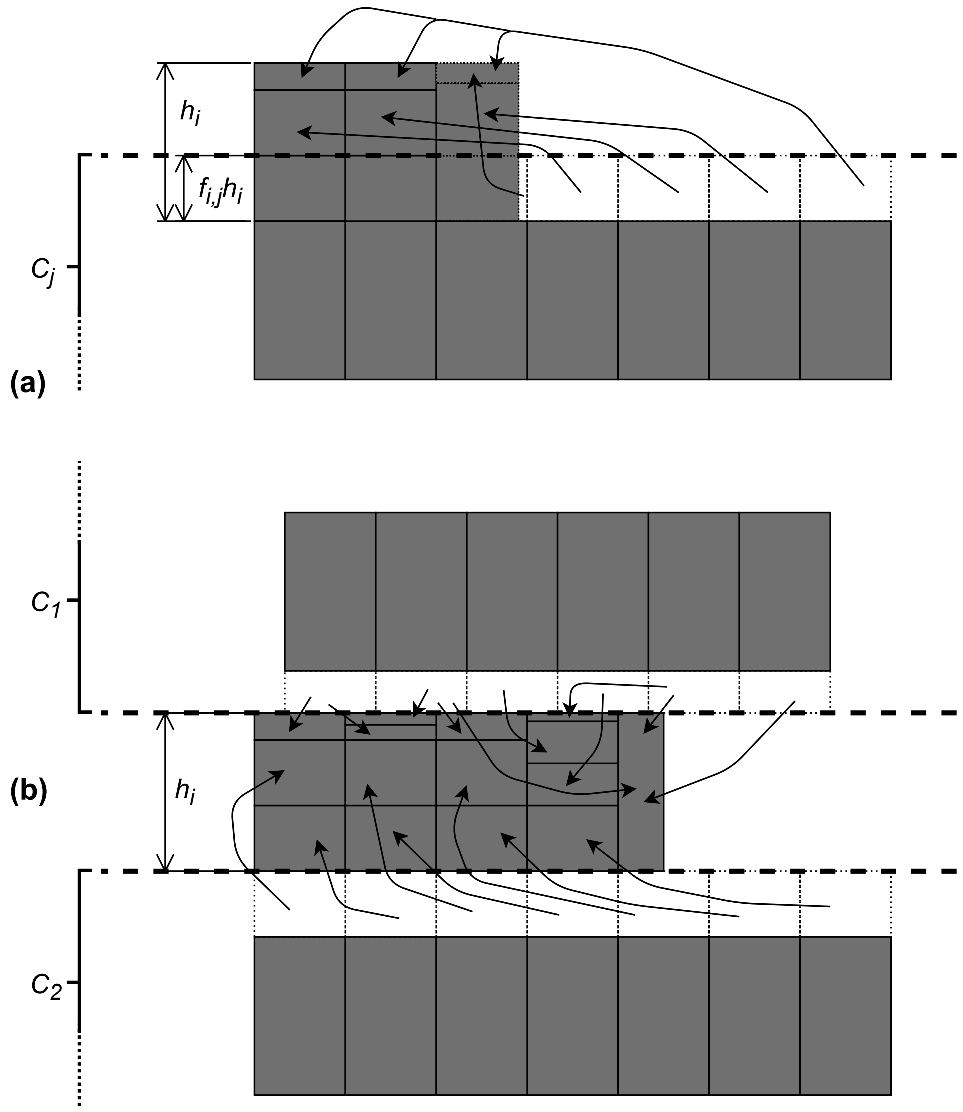

FIGURE 3.9: (a) Arrows show how rectangles of fractional height of type T_i in C_j are stacked up forming whole rectangles and at most one rectangular piece of height h_i. (b) Arrows show how rectangles of fractional height of type T_i from C_1 and C_2 are stacked up in a region between C_1 and C_2 to form whole rectangles and at most one rectangular piece of height h_i. Note how C_1 is inverted.

Consider the set of rectangles with fractional height of type T_i in S_1. We take all these rectangles and stack them on top of each other to form whole rectangles and at most one rectangular piece of height h_i as shown in Figure 3.9 (a). We do the same with rectangles of fractional height from the other configurations in S_1 (see Figure 3.9 (b)).

Lemma 3.5.1 *The rectangles of fractional height of each configuration in segment S_1 can be reshaped to form rectangular pieces of full height that can be packed side-by-side into a region of width w_1, where w_1 is the total width of S_1.*

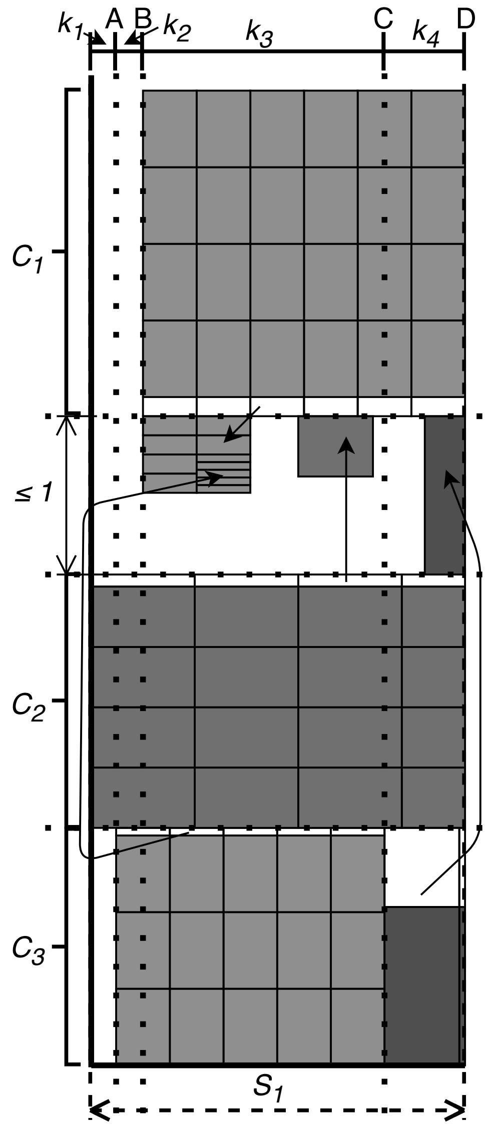

Figure 3.10: Packing rectangles of fractional height in S_1. Arrows show how rectangles of fractional height located in each configuration are reshaped into rectangular pieces of full height and this increases the height of S_1 by at most 1 unit. Note how rectangles of fractional height are stacked on top of each other as described in Figure 3.9.

Proof $\sum_{k_c \in S_1} \sum_{j=1}^{3} F_{j,c} W_c H_{j,c}$ is the total area of all rectangles with fractional height in S_1, where $k_c \in S_1$ represents a column in S_1, W_c is the width of column k_c, and $F_{j,c}H_{j,c}$ is the height of the rectangles with fractional height in column k_c of configuration C_j; note that in a configuration each column has at most one rectangle type. If we reshape all rectangles with fractional height of type T_i, including those rectangles of fractional height that were vertically cut at the right boundary of S_1, into rectangular pieces of full height and lay them side-by-side the width of the resulting packing is at most $\sum_{k_c \in S_1} \sum_{j=1}^{3} \frac{F_{j,c} W_c H_{j,c}}{H_{j,c}}$ because we do not change the total area of these fractional rectangles. Note that

$$\sum_{k_c \in S_1} \sum_{j=1}^{3} \frac{F_{j,c} W_c H_{j,c}}{H_{j,c}} = \sum_{k_c \in S_1} \sum_{j=1}^{3} F_{j,c} W_c = \sum_{k_c \in S_1} W_c \sum_{j=1}^{3} F_{j,c} \leq \sum_{k_c \in S_1} W_c = w_1$$

The last inequality holds because in each column k_c of segment S_1, $\sum_{j=1}^{3} F_{j,c} \leq 1$. □

According to Lemma 3.5.1, we can reshape all rectangles of fractional height that are in segment S_1 to form rectangular pieces of full height; these pieces, when placed side by side, fit into a region of the same width as S_1. We place these rectangular pieces between C_1 and C_2, thus shifting down C_2 and C_3; this increases the height of S_1 by at most 1 (see Figure 3.10).

3.5.2 Packing Rectangles in S_2

By the way in which segment S_2 was defined, $\sum_{j=1}^{3} F_{j,c} > 1$ for each column $k_c \in S_2$.

Lemma 3.5.2 *There is a configuration C_γ in segment S_2 such that $F_{\gamma,c} > 1/3$ and $\sum_{j=1}^{3} F_{j,c} - F_{\gamma,c} < 1$ for each column $k_c \in S_2$.*

Proof Consider first when S_2 only contains columns for which $\sum_{j=1}^{3} F_{j,c} \leq 4/3$. As stated in Definition 3.3, let C_α be the configuration with the largest fractional value $F_{\alpha,c}$ in the leftmost column k_c for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$. In this case C_α must exist because by the definition of S_2, $\sum_{j=1}^{3} F_{j,c} > 1$ for each column $k_c \in S_2$, so if S_2 is not empty then there must be at least one column k_c for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$. If we choose C_γ to be C_α, then according to

Lemma 3.4.6, $F_{\gamma,c} > 1/3$ and $\sum_{j=1}^{3} F_{j,c} - F_{\gamma,c} < 1$ for each column $k_c \in S_2$. Now consider that S_2 contains columns for which $\sum_{j=1}^{3} F_{j,c} > 4/3$. We show below that $\sum_{j=1}^{3} F_{j,c} - F_{\gamma,c} < 1$ still holds:

1. If S_2 is not empty and contains no columns for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$, then rectangles r_1 and r_2 with fractional height cross the vertical cutting line at P_σ according to Lemma 3.4.3. Recall from Definition 3.1 that P_σ is the position of the left border of the leftmost column for which $\sum_{j=1}^{3} F_{j,c} > 4/3$. As stated in Definition 3.4, let C_β be the configuration that does not contain r_1 nor r_2. If we choose C_γ to be C_β, then according to Lemma 3.4.4, $F_{\gamma,c} > 1/3$ and $\sum_{j=1}^{3} F_{j,c} - F_{\gamma,c} < 1$ for each column $k_c \in S_2$.

2. If S_2 contains column(s) for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$, then by the same reasoning above C_α must exist so choose C_γ to be C_α. According to Lemma 3.4.6, $F_{\gamma,c} > 1/3$ and $\sum_{j=1}^{3} F_{j,c} - F_{\gamma,c} < 1$ for each column $k_c \in S_2$. □

We round up all rectangles of fractional height in C_γ that are in S_2 so that they are of full height (see Figure 3.11). As described in Section 3.3 all these fractional rectangles are *tall*, so this rounding increases the height of S_2 by at most 2/3. By Lemmas 3.5.1 and 3.5.2, we can reshape the remaining fractional rectangles in S_2 to form rectangular pieces of full height and pack them side by side into a region with the same width as S_2. Similar to Subsection 3.5.1, we pack these rectangular pieces between configurations C_1 and C_2 in S_2; this packing further increases the height of S_2 by at most 1. We now need to consider two cases if S_1 is not empty:

1. Suppose first that C_γ is C_3. We add empty space between C_2 and C_3 in S_1 so that if we draw a horizontal line at the bottom of C_2 in S_2, that horizontal line will also be at the bottom of C_2 in S_1. We say that the bottom of C_2 is at the same level in S_1 and S_2. In Figure 3.11, the height of the empty space between C_2 and C_3 in S_1 is equal to the height of the tallest rounded part in S_2. This empty space increases the height of S_1 by at most 2/3. Also note how we add empty space between C_1 and C_2 in S_2 so that the top of C_1 is at the same level in S_1 and S_2. The reason why we need these empty spaces is given in Section 3.6.

2. Suppose now that C_γ is either C_1 or C_2. Similar to the previous case, we add empty space between C_1 and C_2 of height equal to the height of the tallest rounded part in S_2 (see Figure 3.12). This empty space increases the height of S_1 by at most 2/3.

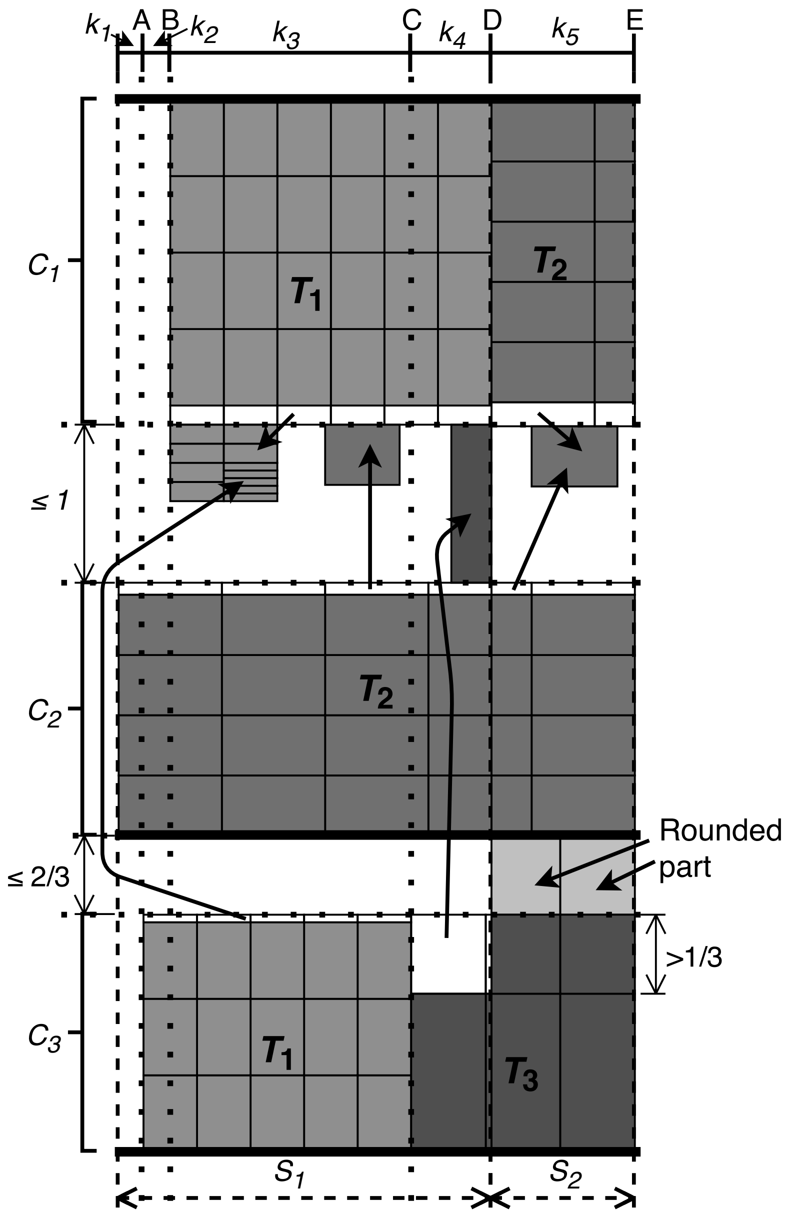

Figure 3.11: Packing where we round up rectangles of fractional height in C_3. We add empty space between C_2 and C_3 in S_1 so that the bottom of C_2 appears at the same level in S_1 and S_2. Arrows show how rectangles with fractional height in C_1 and C_2 are reshaped into rectangular pieces of full height; this increases the height of S_1 and S_2 by at most 1. Note how we add empty space between C_1 and C_2 in S_2 so that the top of C_1 appears at the same level in S_1 and S_2. We round up the rectangles with fractional height of C_3 that are in S_2; this further increases the height of S_2 by at most 2/3. Rounded parts are in light grey.

In Figure 3.12, we add empty space between C_1 and C_2 in S_2 so that the tops of the reshaped rectangular pieces are at the same level in S_1 and S_2. The added empty spaces are such that the top of C_1 is at the same level in S_1 and S_2.

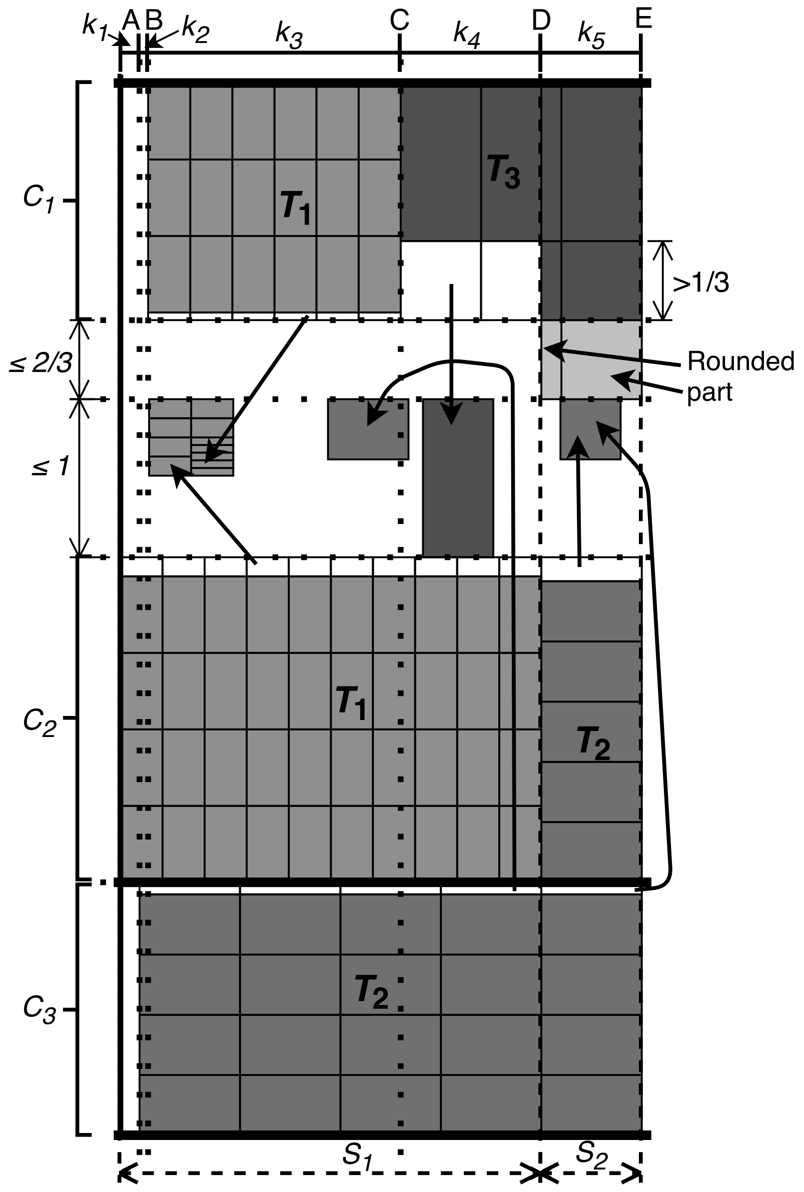

FIGURE 3.12: Packing where we round up rectangles of fractional height in C_1. We add empty space between C_1 and C_2 in S_1 of height equal to the height of the tallest rounded part in S_2. We also add empty space between C_1 and C_2 in S_2 so that the top of C_1 is at the same level in S_1 and S_2.

All together, the algorithm described above increases the height of S_1 and S_2 by at most 5/3 in total.

3.5.3 Packing Rectangles in S_3

We round up all rectangles with fractional height in S_3 so that they are of full height; this rounding increases the height of S_3 by at most 5/3 (see Figure 3.13) because $\sum_{j=1}^{3} F_{j,c} > 4/3$ for each column $k_c \in S_3$.

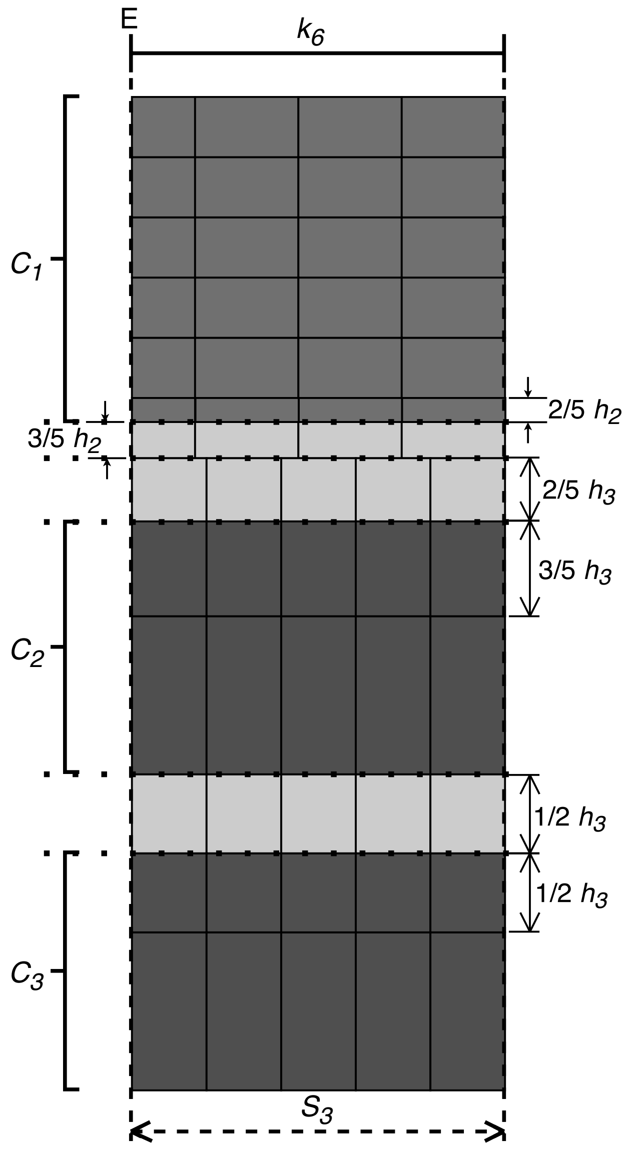

Figure 3.13: Rounding up rectangles with fractional height in S_3. This rounding increases the height of S_3 by at most 5/3 units. Rounded parts are in light grey.

3.6 Aligning Configurations

The final packing must be integral—this means that we cannot have fractional rectangles in the final packing. Recall from Subsection 3.4.2 that we vertically cut any rectangle with fractional height that crosses the border between two segments.

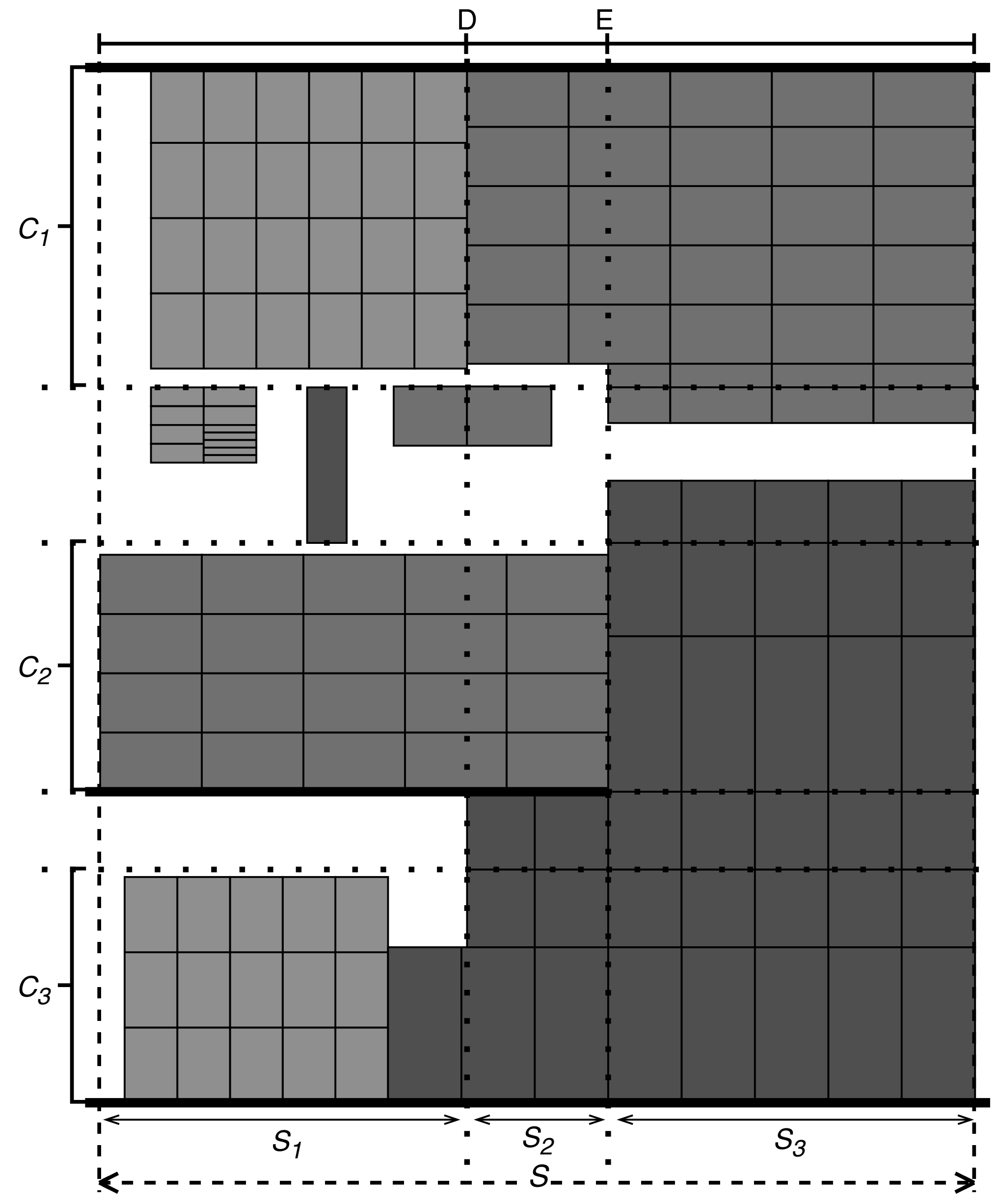

FIGURE 3.14: Aligning configurations so that whole rectangles that cross the border between two segments are not vertically cut. The bottom of configuration C_2 is at the same level in segments S_1 and S_2. The top of configuration C_1 is at the same level in all segments; the bottom of configuration C_3 is also at the same level in all segments.

To ensure that the final packing has no fractional rectangles, we show how to align configurations so that whole rectangles that cross the border between two segments are not vertically cut. For the following lemmas, let H be the height of an optimal fractional packing.

Lemma 3.6.1 *Assume segments S_1 and S_2 are not empty. We can rearrange the rectangles of configuration C_2 so that whole rectangles in C_2 that cross the border between S_1 and S_2 are not vertically cut. If a rectangle r of fractional height is vertically cut between S_1 and S_2 such that only the part in S_2 is rounded up, then there is enough empty space next to the rounded-up part to pack a rectangular piece of the size needed to form exactly one whole rectangle of the same type as r.*

Proof As described in Subsection 3.5.2, we round up the *tall* fractional rectangles of one configuration C_γ in S_2. If we choose C_γ according to Lemma 3.5.2 so that $F_{\gamma,c} > 1/3$ and $\sum_{j=1}^{3} F_{j,c} - F_{\gamma,c} < 1$ for each column $k_c \in S_2$, we now need to consider two cases.

1. If C_γ is C_3, recall from Subsection 3.5.2 that we add empty space between C_2 and C_3 of height equal to the height of the tallest rounded part in S_2 (see Figure 3.11). We shift the rectangles of C_2 so that the bottom of C_2 is at the same level in S_1 and S_2 as the top of the tallest rounded part in C_3 of S_2. In Figure 3.14 note how the horizontal line at the bottom of C_2 in S_1 is also at the bottom of C_2 in S_2. Hence, the whole rectangles in C_2 that cross the boundary between S_1 and S_2 (see point **D** in Figure 3.14) are not vertically cut.

2. If C_γ is either C_1 or C_2, recall from Subsection 3.5.2 that we add empty space between C_1 and C_2 of height equal to the height of the tallest rounded part in S_2 (see Figure 3.12). We shift the rectangles of C_2 so that the bottom of C_2 is at the same level in S_1 and S_2 as the top of C_3. Hence, the whole rectangles in C_2 that cross the boundary between S_1 and S_2 are not vertically cut.

If a rectangle r with fractional height of type T_i in C_γ is vertically cut such that only the part of r in S_2 is rounded up, recall from Lemma 3.5.1 that we take the part of r in S_1, reshape it into a rectangular piece of full height, and pack it into a region between C_1 and C_2. Note how in S_1 this leaves an empty space of width (w_i - width of the rounded-up part of r) and height $f_{i,\gamma}h_i$ next to the rounded-up part of r. Since we add empty space to S_1 of height equal to the height of the tallest rounded part in S_2 as described above, then we must have enough empty space in S_1 to pack a rectangular piece of height h_i and width (w_i - width of the rounded-up part of r) next to the rounded-up part of r to form exactly one whole rectangle of type T_i. □

Lemma 3.6.2 *Assume segments S_1 and S_2 are not empty. We can rearrange the rectangles so that:*

1. *Lemma 3.6.1 holds,*
2. *the whole rectangles in C_1 and C_3 that cross the border between S_1 and S_2 are not vertically cut, and*
3. *the height of the packing for S_1 and S_2 is at most $H + 5/3$.*

Proof Recall from Subsection 3.5.2 that we only round up the *tall* fractional rectangles of C_γ that are in S_2 and this increases the height of the packing for S_2 by at most 2/3. After shifting rectangles according to Lemma 3.6.1, because the height of the tallest rounded part in S_2 is at most 2/3 the empty space that we add in S_1 will only increase the height of the packing for S_1 by at most 2/3.

If the heights of S_1 and S_2 differ we add empty space to the segment with the shorter height so that the top of C_1 is at the same level and the bottom of C_3 is at the same level in both segments. Hence, the whole rectangles in C_1 and C_3 that cross the boundary between S_1 and S_2 are not vertically cut. For example, in Figure 3.11 the whole rectangles of C_3 that cross the vertical line at point **D** are not vertically cut. In Figure 3.12, the whole rectangles of C_1 that cross the vertical line at point **D** are not vertically cut.

For the rectangles of fractional height in S_1 and those in S_2 that were not rounded, we reshape these rectangles to form rectangular pieces of full height and pack them side by side into a region between C_1 and C_2 with the same total width as S_1 and S_2 as described in Lemmas 3.5.1 and 3.5.2. We then ensure that the configurations are at the same level so that the height of the packing for S_1 and S_2 will further increase by at most 1. Thus, the height of the packing for S_1 and S_2 is at most $H + 5/3$. □

After rearranging rectangles according to Lemma 3.6.2, note how at most one fractional rectangle is vertically cut between S_1 and S_2 and for which only the part in S_2 is rounded up.

Lemma 3.6.3 *Assume segment S_3 and another segment are not empty. We can rearrange the rectangles so that:*

1. *Lemma 3.6.2 holds,*
2. *the whole rectangles that cross the border between any two segments are not vertically cut, and*
3. *the height of the packing for part S is at most $H + 5/3$.*

Proof As described in Lemma 3.4.1 we rename and reorder the configurations so that C_2 does not have rectangles that cross the boundary between S_3 and its adjacent segment. We shift the rectangles of C_2 so that the bottoms of the bottom-most rectangles of C_2 that are in S_3 are all at the same level as the top of the tallest rounded part of C_3 (see Figure 3.15). If we also rearrange the rectangles of C_2 according to Lemma 3.6.2, then all whole rectangles in C_2 that cross the border between any two segments are not vertically cut. In Figure 3.15 note how the bottoms of the bottom-most rectangles of C_2 in S_3 are not at the same level as the bottoms of the bottom-most rectangles of C_2 that are in the other segments. We can have these rectangles at different levels because the rectangles of C_2 are not vertically cut at the boundary of S_3.

As described in Subsection 3.5.3, all rectangles with fractional height in S_3 are rounded up so that they are of full height. After rearranging the rectangles as described above, the algorithm increases the height of the packing for S_3 by at most 5/3 because $\sum_{j=1}^{3} F_{j,c} > 4/3$ for all columns k_c in S_3.

If the height of segment S_3 is less than the height of the other segments, then we add empty space to S_3 so that the top of C_1 is at the same level in all segments and the bottom of C_3 is also at the same level in all segments. Hence, the whole rectangles in C_1 and C_3 are not vertically cut at the boundary between S_3 and its adjacent segment. For example, in Figure 3.14 the whole rectangles of C_1 that cross the vertical line at point **E** are not vertically cut. The height of the packing for S is at most $H + 5/3$.

Similarly, if the height of S_3 is greater than the height of the other segments, then we add empty space to the other segments so that the top of C_1 is at the same level in all segments and the bottom of C_3 is at the same level in all segments. For example, in Figure 3.15 we add empty space between C_1 and C_2 so that the whole rectangles of C_1 and C_3 that cross the boundary between S_2 and S_3 are not vertically cut. The height of the packing for S is again at most $H + 5/3$. □

Pseudocode for the algorithm described in Sections 3.5 and 3.6 is given below. If any segment S_1, S_2, or S_3 is empty, the problem is simpler as only rectangles in the remaining segments need to be packed.

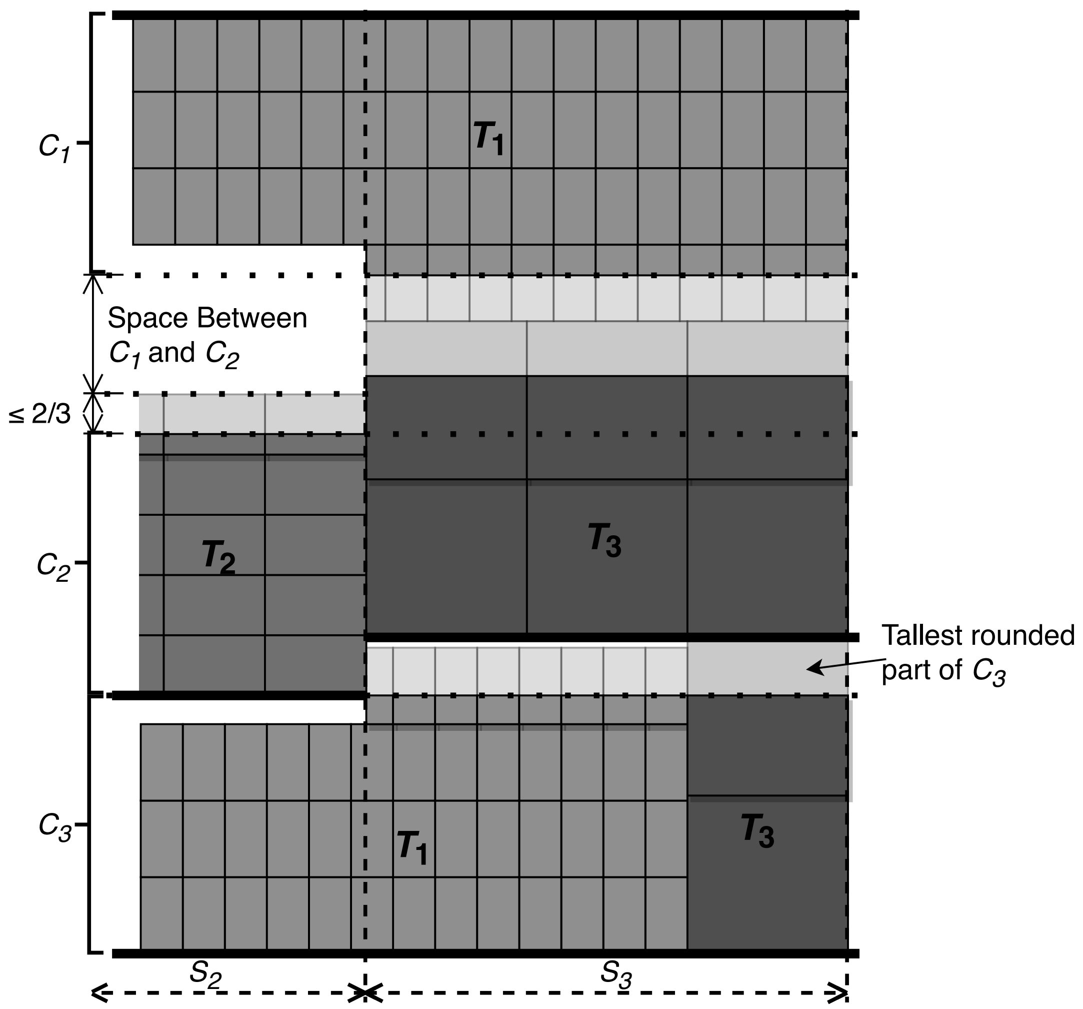

Figure 3.15: Packing rectangles of configuration C_2 in S_3. We shift the rectangles of C_2 in S_3 so that if we draw a horizontal line at the bottoms of the bottom-most rectangles of C_2 in S_3, that horizontal line touches the tops of the tallest rounded rectangles of C_3. Rectangles packed according to Section 3.5 are in very light grey. Note how the bottom of C_2 in S_2 is at a different level than the bottom of C_2 in S_3. The heights of S_2 and S_3 are the same because of empty space added between C_1 and C_2 in S_2.

Algorithm 3.3 packSegments(T, C)

```
1: In: A set $T$ of three rectangle types and a set $C$ of three configurations in part $S$
2: if $S_1$ is not empty then
3:     for all rectangle types $T_i \in S_1$ do
4:         Reshape rectangles of fractional height of type $T_i$ in $S_1$ into rectangular pieces of
           full height of type $T_i$, and pack them side by side into a region between $C_1$ and $C_2$
           in $S_1$, as described in Subsection 3.5.1.
5: if $S_2$ is not empty then
6:     if no columns $k_c$ exist in $S_2$ for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$ then
7:         $C_\gamma \leftarrow C_\beta$ according to Lemma 3.4.5.
8:     else
9:         $C_\gamma \leftarrow C_\alpha$ according to Lemma 3.4.7.
10:    for all rectangle types $T_i \in S_2$ do
11:        Round up rectangles of fractional height of type $T_i$ in $C_\gamma$ that are in $S_2$, as described
           in Subsection 3.5.2.
12:        Reshape the remaining rectangles of fractional height of type $T_i$ in $S_2$ into rect-
           angular pieces of full height of type $T_i$ and pack them side by side into a region
           between $C_1$ and $C_2$ in $S_2$, as described in Subsection 3.5.2.
13:    if $S_1$ is not empty then
14:        if $C_\gamma = C_3$ then
15:            Add empty space between $C_2$ and $C_3$ in $S_1$ so that the bottom of $C_2$ is at the
               same level in $S_1$ and $S_2$ as the top of the tallest rounded part in $C_3$ of $S_2$.
16:        else
17:            Add empty space between $C_1$ and $C_2$ in $S_1$ so that the bottom of $C_2$ is at the
               same level in $S_1$ and $S_2$ as the top of $C_3$.
18:        Add empty space to $S_1$ or $S_2$ so that the top of $C_1$ is at the same level in both
           segments, and the bottom of $C_3$ is also at the same level in both segments.
19: if $S_3$ is not empty then
20:    for all rectangle types $T_i \in S_3$ do
21:        Round up rectangles of fractional height of type $T_i$ in $S_3$, as described in Subsec-
           tion 3.5.3.
22:    Rearrange rectangles of $C_2$ so that the bottoms of the bottom-most rectangles of $C_2$
       that are in $S_3$ are all at the same level as the top of the tallest rounded part of $C_3$ in $S_3$.
23:    if other segments are not empty and $S_3$ has greater height than these segments then
24:        Add empty space to the other segments so that the top of $C_1$ is at the same level
           and the bottom of $C_3$ is at the same level in all segments.
25:    else if other segments are not empty and $S_3$ has less height than these segments then
26:        Add empty space to $S_3$ so that the top of $C_1$ is at the same level and the bottom of
           $C_3$ is at the same level in all segments.
```

3.7 Rounding Fractional Rectangles to Produce an Integral Solution

Recall that n_i is the total number of rectangles of type T_i in the packing. Also recall that H is the height of an optimal fractional packing. We consider two scenarios:

- **Scenario 1.** $S = S_1$. We reshape all rectangles with fractional height in part S into rectangular pieces of full height and pack them side by side in a region between C_1 and C_2 according to Lemma 3.5.1 (see Steps 3–4 in algorithm PACKSEGMENTS). All rectangles with fractional height of each type T_i in the packing must add up to an integer number of rectangles because n_i is integer for each type T_i. If for any rectangle type T_i the rectangular pieces of full height in S add up to a non-integer number n_i', then part S' is not empty and fractional pieces of total size n_i' - $\lfloor n_i' \rfloor$ can be discarded as these pieces must have already been packed when the rectangles with fractional height in S' were rounded up to their full height. Because segments S_2 and S_3 are empty, the height of the packing in this scenario is at most $H + 1$.

- **Scenario 2.** $S \neq S_1$. As described in Subsections 3.5.2 and 3.5.3, we round up the *tall* fractional rectangles of one configuration in S_2 and all rectangles with fractional height in S_3 so that they are of full height. The remaining rectangles of fractional height in S_1 and those in S_2 that were not rounded are reshaped into rectangular pieces of full height and packed according to Lemmas 3.5.1 and 3.5.2. We shift these rectangular pieces so that pieces of the same type are beside each other forming either whole rectangles or larger rectangular pieces. We now have two additional cases:

  (a) No rectangles are cut vertically such that one part of a rectangle is rounded up and the other part is packed according to Lemma 3.5.1. If for any rectangle type T_i the rectangular pieces of full height in part S add to a non-integer number n_i', then fractional pieces of total size n_i' - $\lfloor n_i' \rfloor$ can be discarded as these pieces must have

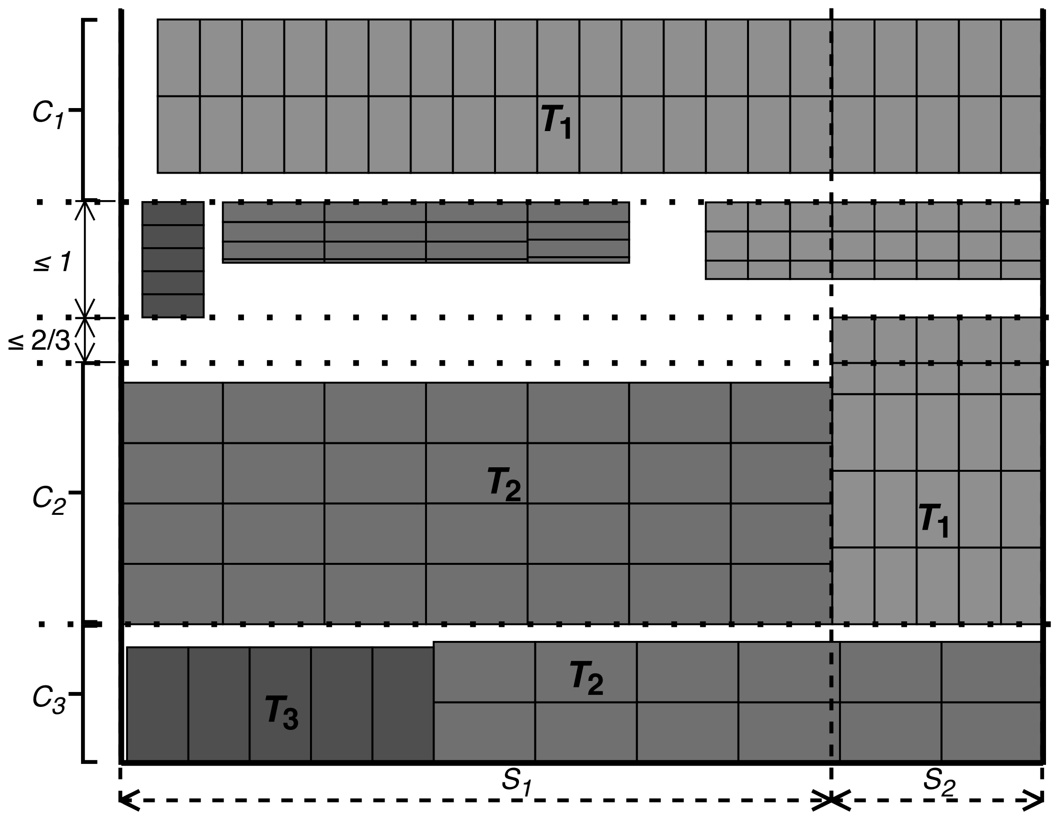

Figure 3.16: Packing rectangles in S for Scenario 2(a) where no pieces are discarded. All rectangles with fractional height in C_1 and C_3, and rectangles with fractional height in configuration C_2 of S_1, are reshaped into whole rectangles and packed between C_1 and C_2. Part S' is empty in this example.

already been packed when the rectangles with fractional height in S_2, S_3, and/or S' were rounded up to their full height. In Figure 3.16 no fractional rectangles exist because part S' is empty, the rectangles with fractional height of type T_1 in C_1 form whole rectangles, the rectangles with fractional height of type T_3 in C_3 form a whole rectangle, the rectangles with fractional height of type T_2 in C_2 and C_3 also form whole rectangles, and the rectangles with fractional height of type T_1 in C_2 are rounded up so that they are of whole height. According to Lemmas 3.6.2 and 3.6.3, the height of the packing is at most $H + 5/3$.

(b) Some rectangles are cut vertically so that one part of a rectangle is rounded up and the other part is packed according to Lemma 3.5.1. In this case, we could end up with fractional rectangles because pieces from the same rectangle could be packed in different parts of the bin. The following lemma bounds the maxmium number of fractional rectangular pieces in the packing that we have produced so far.

Lemma 3.7.1 *At most two rectangles with fractional height in part S are vertically cut between adjacent segments such that:*

1. *For each one of these rectangles, one part of the rectangle is rounded up and the other part is packed as in Lemma 3.5.1, and*

2. *any of these rectangles that are vertically cut between S_3 and its adjacent segment are in configuration C_1.*

Proof Recall from Subsections 3.5.3 and 3.5.2 that we only round up rectangles with fractional height in segment S_3 and *tall* fractional rectangles in S_2. We need to consider rectangles of fractional height that are vertically cut between S_1 and S_2, between S_2 and S_3, and between S_1 and S_3:

1. Boundary between S_1 and S_2. As described in Subsection 3.5.2, we only round up the *tall* fractional rectangles that are in S_2. Thus, at most one rectangle r of fractional height is vertically cut between S_1 and S_2 such that only the part of r in S_2 is rounded up.

2. Boundary between S_2 and S_3. According to Lemma 3.4.1, configuration C_2 does not have any rectangles of fractional height that cross the boundary between S_2 and S_3; therefore, there cannot be three rectangles with fractional height that are vertically cut between S_2 and S_3. Recall from Subsection 3.5.2 that we round up all the *tall* fractional rectangles of one configuration that are in S_2 so that they are of full height. We now need to consider two additional cases:

   (a) If S_2 contains no columns k_c for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$, then as described in Lemma 3.4.5, at most one rectangle with fractional height crosses the boundary between S_2 and S_3 and this fractional rectangle is in C_1. Thus, at most one fractional rectangle r is vertically cut between S_2 and S_3 such that only the part of r in S_3 is rounded up and r is in C_1.

(b) If S_2 contains columns k_c for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$, then as described in Lemma 3.4.7, any rectangle of fractional height not in configuration C_α that crosses the boundary between S_2 and S_3 is in C_1. Recall from Lemma 3.5.2 that rectangles with fractional height in C_α are rounded up. Thus, at most one fractional rectangle r is vertically cut between S_2 and S_3 such that only the part of r in S_3 is rounded up and r is in C_1.

3. Boundary between S_1 and S_3. According to Lemma 3.4.2, at most one rectangle r with fractional height is vertically cut between S_1 and S_3 such that only the part of r in S_3 is rounded up and r is in C_1.

If there is a boundary between S_1 and S_3, then S_2 is empty. Otherwise, there can be a boundary between S_1 and S_2, and one between S_2 and S_3. Therefore, at most two rectangles with fractional height in part S can be vertically cut between adjacent segments such that one part of each rectangle is rounded up and the other part is packed as in Lemma 3.5.1. $\square$

For the last scenario 2(b), Lemma 3.7.1 states that at most two rectangles with fractional height in part S are vertically cut between adjacent segments such that one part of a rectangle is rounded up and the other part is packed as in Lemma 3.5.1. In Subsection 3.7.1, we show how to produce a packing that is integral for the case when only one fractional rectangle is vertically cut. In Subsection 3.7.2, we show how to produce a packing that is integral for the case when two fractional rectangles of different types are vertically cut. Finally, in Subsection 3.7.3, we show how to produce a packing that is integral for the case when two fractional rectangles of the same type are vertically cut.

3.7.1 One Fractional Rectangle is Vertically Cut between Two Adjacent Segments

Consider a fractional solution where a rectangle r of type T_i with fractional height is vertically cut between two adjacent segments (see Figure 3.17) so that the algorithm described in Section 3.5 packs one part according to Lemma 3.5.1 and rounds up the other part.

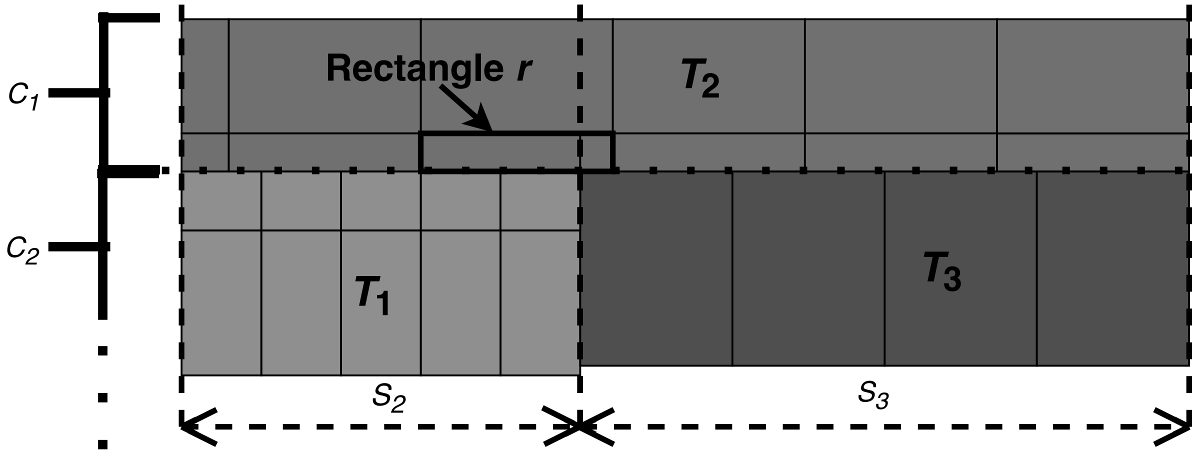

Figure 3.17: Rectangle r of type T_2 with fractional height in C_1 is vertically cut between S_2 and S_3. We pack the part of r in S_2 according to Lemma 3.5.1 and we round up the part of r in S_3 so that it is of full height.

After processing the rectangles as described in algorithm packSegments, we can shift the reshaped rectangular pieces of full height that were packed as in Lemma 3.5.1 so that they form either whole rectangles or larger rectangular pieces as explained in Lemma 3.7.2 below:

Lemma 3.7.2 *We can rearrange the rectangular pieces of full height packed as in Lemma 3.5.1 so that for each rectangle type T_i, there is at most one rectangular piece of type T_i in the region between configurations C_1 and C_2 in segments S_1 and S_2.*

Proof Recall that we pack rectangles with fractional height according to Lemma 3.5.1 in segments S_1 and S_2 in a region of height at most 1 between C_1 and C_2. We can shift the rectangular pieces of full height within this region so that pieces of the same type are adjacent to each other to form whole rectangles or larger rectangular pieces. Thus, for each rectangle type T_i at most one rectangular piece of full height of type T_i is in this region. □

In this section we assume only one rectangle r with fractional height is vertically cut between two adjacent segments such that only one part of r is rounded up. Thus, there are at most two rectangular pieces with full height of the same type T_i as r: the rounded-up part of r and the remaining piece described in Lemma 3.7.2. According to Lemma 3.7.2, there is at most one rectangular piece of full height for each of the other types, T_j, T_k; hence we discard the fractional rectangles of types T_j and T_k because the numbers n_j, n_k of rectangles of these types are integer, so these fractional rectangles must have already been packed when the rectangles in S_2, S_3, and S' were rounded up.

Definition 3.5. Let r be a rectangle with fractional height that is vertically cut between two adjacent segments such that one part is rounded up and the other part is packed according to Lemma 3.5.1. We denote with r_f the piece that is packed according to Lemma 3.5.1 and we denote with r_r the rounded-up part of full height.

If rectangular pieces r_f and r_r do not add up to a whole rectangle, then we discard these fractional pieces as the number n_i of rectangles of type T_i is integer. However, if these rectangular pieces add up to at least one whole rectangle, these fractional pieces cannot be simply discarded. Lemma 3.7.3 below shows that we can always move a rectangular piece of width (w_i - width of r_r) next to r_r to form a whole rectangle of type T_i.

Lemma 3.7.3 *If r_f and r_r add up to at least one whole rectangle, we can move (part of) r_f next to r_r to form a whole rectangle without overlapping other rectangles or increasing the height of the packing.*

Proof Recall from Subsections 3.5.2 and 3.5.3 that we only round up rectangles of fractional height in segments S_2 and S_3. Thus, we need to consider two different cases:

1. If r_r is in S_3, then according to Lemma 3.7.1 r, and thus r_r, are in C_1 (see Figure 3.17). As described in Lemma 3.7.2, r_f is in the region between C_1 and C_2; the height of this region is at least the height h_i of a rectangle of the same type as r. Thus we can shift r_f to

the right within this region until it is adjacent to r_r and then we can shift (part of) r_f up to form a whole rectangle with r_r. For example, in Figure 3.18 we can move a piece of r_f of width (w_i - width of r_r) right beside r_r to form one whole rectangle of type T_i. Note that the part of r_f needed for r_r to form a whole rectangle can be moved up towards r_r because this is the space that r occupied in the optimal fractional solution and thus it is empty.

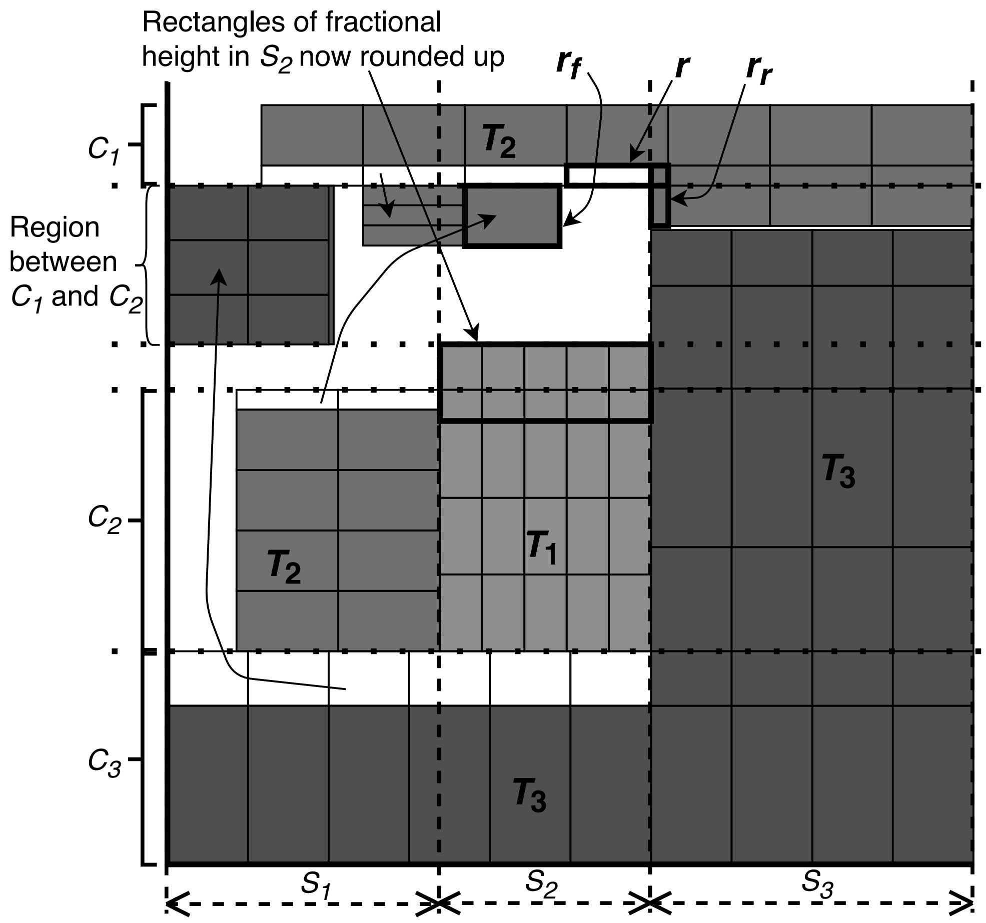
Rectangles of fractional height in S_2 now rounded up
r_f
r
r_r
C_1
Region between C_1 and C_2
C_2
C_3
T_2
T_1
T_3
T_2
T_3
S_1
S_2
S_3

FIGURE 3.18: Rectangle r with fractional height in C_1 is vertically cut between segments S_2 and S_3 such that only the part of r in S_3 is rounded up. Arrows show how rectangles with fractional height in S_1 and S_2 are reshaped into rectangular pieces of full height and then packed in a region between C_1 and C_2. Rectangular pieces r_f and r_r add up to at least one whole rectangle. There is enough empty space to move (part of) r_f next to r_r to form a whole rectangle of type T_2.

2. If r_r is in S_2, then according to Lemma 3.6.1, there is enough empty space in S_1 next to r_r to move (part of) r_f next to r_r to form a whole rectangle. For example, in Figure 3.19, we can move a piece of r_f of width (w_i - width of r_r) right beside r_r to form one whole

rectangle of type T_i. □

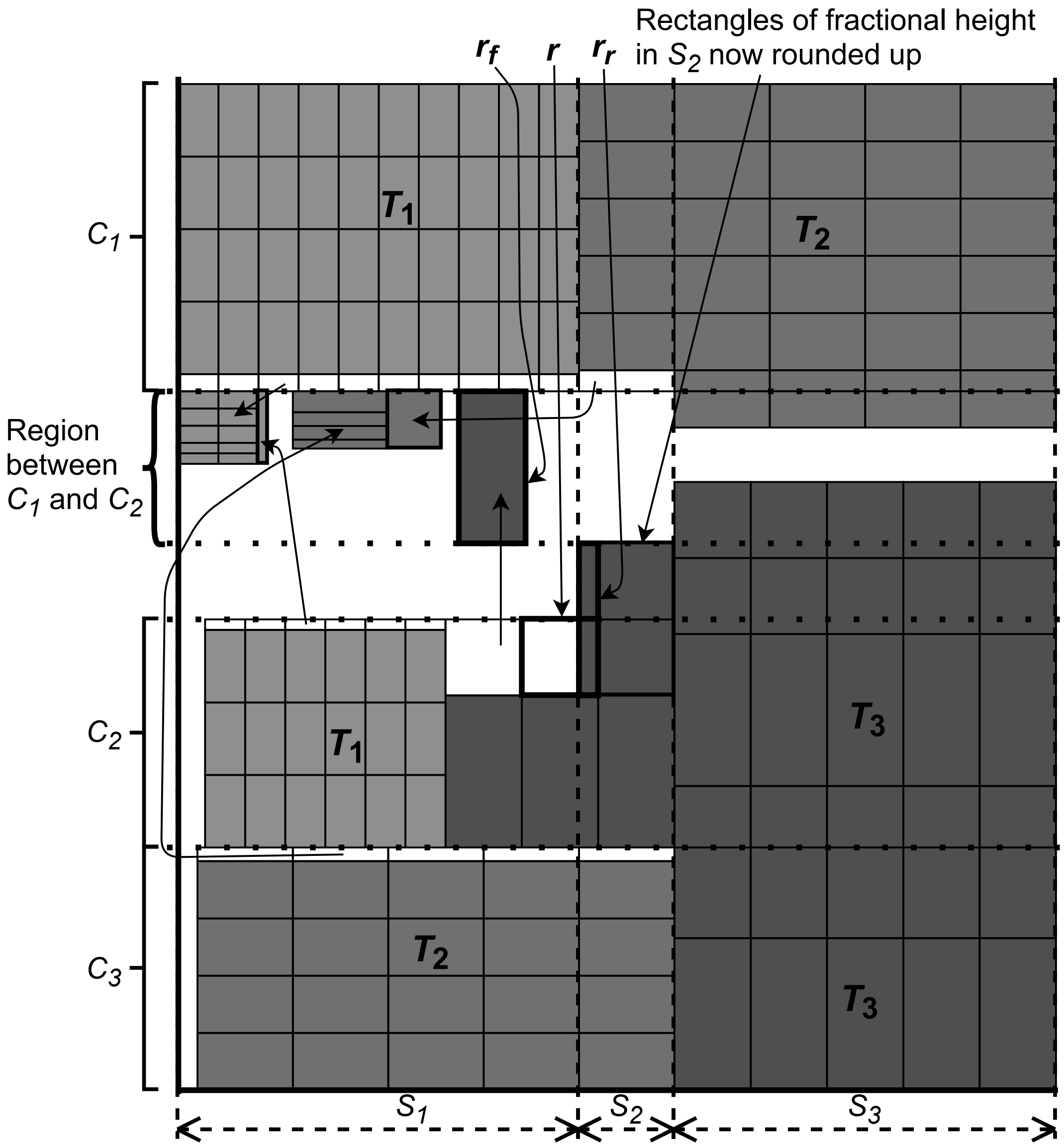
r_f
r
r_r
Rectangles of fractional height in S_2 now rounded up
C_1
T_1
T_2
Region between C_1 and C_2
C_2
T_1
T_3
C_3
T_2
T_3
S_1
S_2
S_3

FIGURE 3.19: A rectangle r with fractional height in C_2 is vertically cut between segments S_1 and S_2 such that only the part of r in S_2 is rounded up. Rectangular pieces r_f and r_r add up to at least one whole rectangle. There is enough empty space to move (part of) r_f next to r_r to form one whole rectangle of type T_3. We then discard all remaining fractional pieces to make this packing integral.

After packing r_f and r_r according to Lemma 3.7.3, we discard the remaining piece of r_f and any other remaining fractional pieces because the number n_i of rectangles of type T_i is integer. Figure 3.20 shows the integral packing obtained from the fractional packing in Figure 3.18.

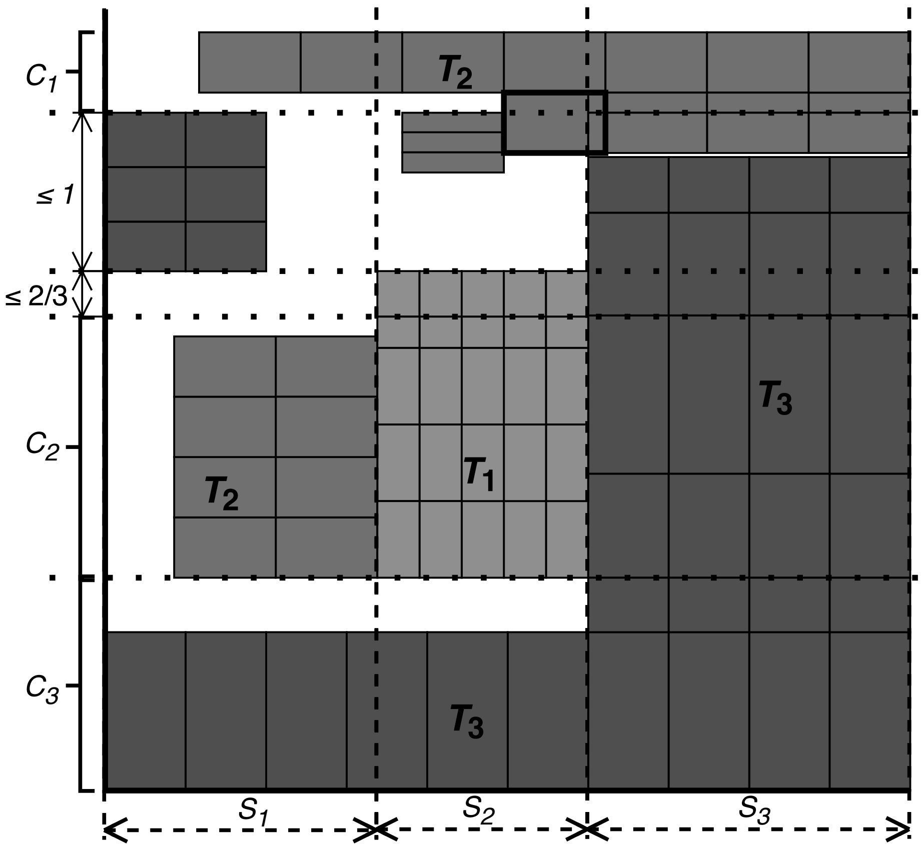

FIGURE 3.20: Moving a piece of r_f right beside r_r to form one whole rectangle of type T_2. Since r_f and r_r cannot add up to two whole rectangles, we move (part of) r_f next to r_r as described in Figure 3.18 to form one whole rectangle of type T_2. Note that we discard all remaining fractional pieces to make this packing integral.

3.7.2 Two Fractional Rectangles of Different Types are Vertically Cut between S_1 and S_2 and between S_2 and S_3

Consider that two rectangles r_1 and r_2 with fractional height and of different types T_i and T_j are vertically cut between segments S_1 and S_2, and between segments S_2 and S_3, respectively, so that the algorithm described in Section 3.5 rounds up the part of r_1 in S_2, rounds up the part of r_2 in S_3, and packs the other parts according to Lemma 3.5.1.

After processing the rectangles as described in algorithm PACKSEGMENTS, we rearrange according to Lemma 3.7.2 the reshaped rectangular pieces of full height that were packed as in Lemma 3.5.1. Thus, there are at most two rectangular pieces of full height of the same type T_i as r_1: the rounded-up part of r_1 and the remaining piece described in Lemma 3.7.2. Similarly, there are at most two rectangular pieces of full height of the same type T_j as r_2: the rounded-up part of r_2 and the remaining piece described in Lemma 3.7.2.

Definition 3.6. We denote with r_{r1} and r_{r2} the rounded-up parts of r_1 and r_2, and we denote with r_{f1} and r_{f2} the remaining pieces of r_1 and r_2 as described in Lemma 3.7.2.

According to Lemma 3.7.2, there is at most one rectangular piece of full height of type $T_k \neq T_i$, T_j. We discard the fractional rectangle of type T_k because the number n_k of rectangles of this type is integer, so all rectangles of type T_k must have already been packed when the rectangles in S_2, S_3, and S' were rounded up.

If r_{r1} and r_{f1} add up to at least a whole rectangle, but r_{r2} and r_{f2} do not add up to a whole rectangle, we discard r_{r2} and r_{f2}, pack r_{r1} and r_{f1} according to Lemma 3.7.3, and then discard any remaining fractional pieces. Similarly, if r_{r2} and r_{f2} add up to at least a whole rectangle, but r_{r1} and r_{f1} do not add up to a whole rectangle we discard r_{r1} and r_{f1}, pack r_{r2} and r_{f2} according to Lemma 3.7.3 and then discard any remaining fractional pieces. If r_{r1} and r_{f1} add up to at least a whole rectangle, and r_{r2} and r_{f2} also add up to at least a whole rectangle, then we proceed as follows:

First, we pack r_{f1} and r_{r1}, and then pack r_{f2} and r_{r2}, according to Lemma 3.7.3. We then discard any remaining fractional pieces because the numbers n_i, n_j of rectangles of these types are integer. There is enough empty space next to the rounded-up part of each fractional rectangle r_1, r_2 to complete it into a whole rectangle. Thus, because r_{r1} and r_{r2} are in different parts of the packing, we simply use the algorithms mentioned in the proof of Lemma 3.7.3 to make r_{r1} and r_{r2} whole. For example, in Figure 3.21, we move a piece of r_{f1} of width (w_3 - width of r_{r1}) right beside r_{r1} to form one whole rectangle of type T_3. We then move a piece of r_{f2} of width (w_2 - width of r_{r2}) right beside r_{r2} to form one whole rectangle of type T_2. Figure 3.22 shows the integral packing for Figure 3.21.

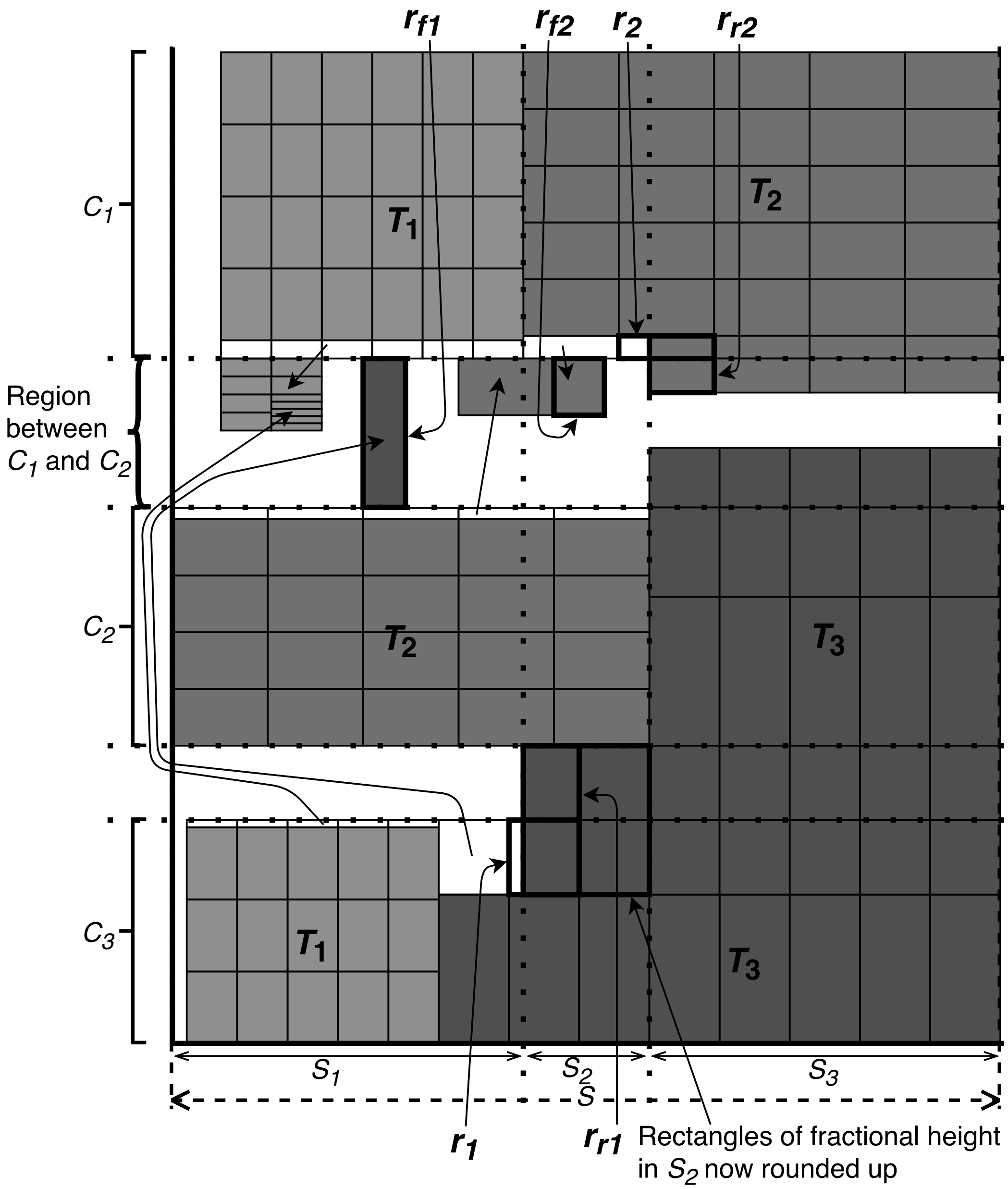

Figure 3.21: A rectangle r_1 with fractional height of type T_3 in C_3 is vertically cut between S_1 and S_2; another rectangle r_2 with fractional height of type T_2 in C_1 is vertically cut between S_2 and S_3. Rectangular pieces r_{f2} and r_{r2} add up to at least one whole rectangle. There is enough empty space to move (part of) r_{f2} next to r_{r2} to form a whole rectangle of type T_2. Similarly, r_{f1} and r_{r1} add up to at least one whole rectangle. There is enough empty space to move (part of) r_{f1} next to r_{r1} to form a whole rectangle of type T_3.

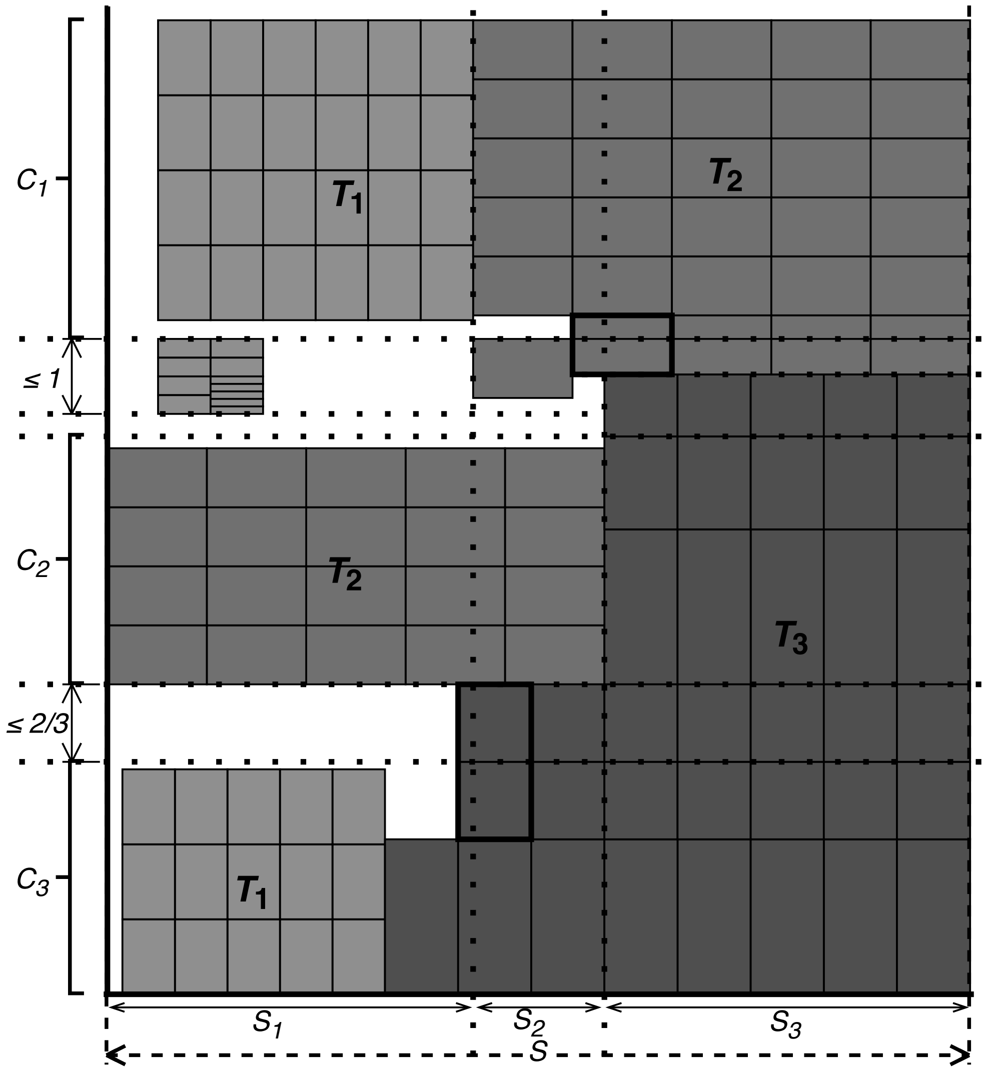

Figure 3.22: Moving pieces r_{f1} and r_{f2} right beside r_{r1} and r_{r2} as described in Figure 3.21 to form one whole rectangle of type T_2 and one whole rectangle of type T_3. Note that we discard all remaining fractional rectangular pieces to make this packing integral.

3.7.3 Two Fractional Rectangles of the Same Type are Vertically Cut between S_1 and S_2 and between S_2 and S_3

Consider now that two rectangles r_1 and r_2 with fractional height and of the same type T_i are vertically cut between segments S_1 and S_2, and between segments S_2 and S_3, respectively, so that the algorithm described in Section 3.5 rounds up the part of r_1 in S_2, rounds up the part of r_2 in S_3, and packs the other parts according to Lemma 3.5.1.

After processing the rectangles as described in algorithm packSegments, we rearrange the reshaped rectangular pieces of full height that were packed as in Lemma 3.5.1 according to Lemma 3.7.2. Thus, there are at most three rectangular pieces with full height of type T_i: r_f as specified in Lemma 3.7.2, and the rounded-up parts r_{r1} of r_1 and r_{r2} of r_2. Note that now r_2 and r_{r2} are of type T_i instead of type T_j. According to Lemma 3.7.2, there is at most one rectangular piece of full height for each of the other types, T_j, T_k. We discard the fractional rectangles of types T_j and T_k because the numbers n_j, n_k of rectangles of these types are integer, so these fractional rectangles must have already been packed when the rectangles in S_2, S_3, and S' were rounded up.

If rectangular pieces r_{r1}, r_{r2}, and r_f do not add up to a whole rectangle, then we discard these fractional pieces because the number n_i of rectangles of type T_i is integer. However, if these rectangular pieces add up to at least one whole rectangle, then these fractional pieces cannot be simply discarded. We need to consider three different cases:

1. **Rectangular pieces r_f and r_{r1} do not add up to at least one whole rectangle**. In this case, r_f, r_{r1}, and r_{r2} do not add up to at least two whole rectangles because the width of r_{r2} is less than w_i. Lemma 3.7.4 below shows how we can always move r_f and a rectangular piece of width (w_i - width of r_f - width of r_{r1}) next to r_{r1} to form one whole rectangle.

   Lemma 3.7.4 *Assume r_f and r_{r1} do not add up to at least one whole rectangle. If r_f, r_{r1}, and r_{r2} add up to at least one whole rectangle, we can move r_f and (part of) r_{r2} next to r_{r1} to form a whole rectangle without overlapping other rectangles or increasing the height of the packing.*

   Proof According to Lemma 3.6.1, there is enough empty space in S_1 next to r_{r1} to pack a rectangular piece of the size needed to form exactly one whole rectangle. If r_f, r_{r1}, and r_{r2} add up to at least one whole rectangle, we move r_f and (part of) r_{r2} next to r_{r1} to form one whole rectangle. For example, in Figure 3.23, we move r_f and a piece of r_{r2} of width (w_1 - width of r_f - width of r_{r1}) right beside r_{r1} to form one whole rectangle of type T_1. □

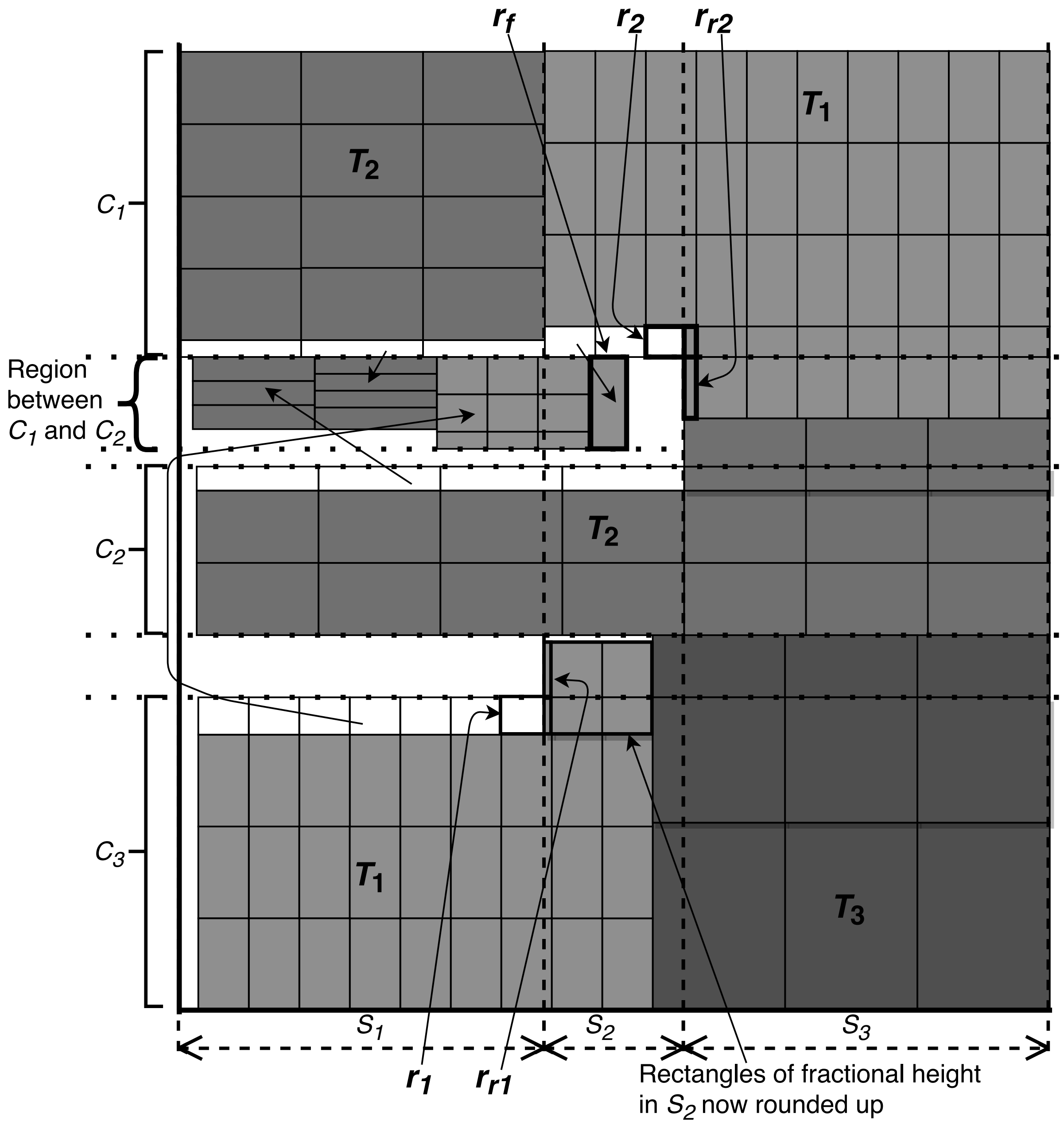

Figure 3.23: A rectangle r_1 with fractional height of type T_1 in C_3 is vertically cut between S_1 and S_2; another rectangle r_2 with fractional height of type T_1 in C_1 is vertically cut between S_2 and S_3. Note how r_{r1} and r_{r2} are in different configurations. Rectangular pieces r_f and r_{r1} do not add up to a whole rectangle. There is enough empty space to move r_f and (part of) r_{r2} next to r_{r1} to form a whole rectangle of type T_1.

2. **Rectangular pieces of type T_i add up to less than two whole rectangles, but r_f and r_{r1} add up to at least one whole rectangle**. We simply pack r_f and r_{r1} according to Lemma 3.7.3.

3. **Rectangular pieces of type T_i add up to at least two whole rectangles**. Note that r_f and r_{r1} must add up to at least one whole rectangle because we would otherwise pack r_f, r_{r1}, and r_{r2} according to Lemma 3.7.4. First, we pack r_f and r_{r1} according to Lemma 3.7.3. We do not yet discard the remaining piece from r_f; let r_f' denote this piece. If r_f' and r_{r2} add up to at least one whole rectangle, we pack r_f' and r_{r2} according to Lemma 3.7.3.

After packing the rectangular pieces of type T_i as described above, we discard any remaining fractional piece(s) because the number n_i of rectangles of type T_i is integer. Pseudocode for our algorithm for producing a packing with no fractional rectangles is given below.

Algorithm 3.4 removeFractionalRectangles(T, C)

```
1: In: A set $T$ of three rectangle types and a set $C$ of three configurations in part $S$
2: if $S_1$ and $S_2$ are not empty then
3:     for all rectangle types $T_i$ do
4:         Shift the reshaped rectangular pieces of full height that are in the region between
           $C_1$ and $C_2$ in $S_1$ and $S_2$ to place rectangular pieces of type $T_i$ beside each other to
           form either whole rectangles or larger rectangular pieces of type $T_i$.
5: if one rectangle $r$ with fractional height and of type $T_i$ is vertically cut between two adjacent
   segments so that only one part $r_r$ of $r$ is rounded up then
6:     Let $r_f$ be the remaining piece of $r$ as described in Lemma 3.7.2.
7:     resolveOneVerticalCut($T_i$, $r_f$, $r_r$)
```

```
 8: else if two rectangles $r_1$ and $r_2$ with fractional height and of types $T_i$ and $T_j$ are vertically
    cut between $S_1$ and $S_2$, and between $S_2$ and $S_3$ respectively, so that only one part $r_{r1}$ of $r_1$
    and one part $r_{r2}$ of $r_2$ are rounded up then
 9:     if $T_i \neq T_j$ then
10:         Let $r_{f1}$ and $r_{f2}$ be the remaining pieces of $r_1$ and $r_2$ as described in Lemma 3.7.2.
11:         if $r_{f1}$ and $r_{r1}$ add up to at least one whole rectangle of type $T_i$, and $r_{f2}$ and $r_{r2}$ do
12:         not add up to a whole rectangle of type $T_j$ then
13:             resolveOneVerticalCut($T_i$, $r_{f1}$, $r_{r1}$)
14:         else if $r_{f2}$ and $r_{r2}$ add up to at least one whole rectangle of type $T_j$, and $r_{f1}$ and $r_{r1}$
15:         do not add up to a whole rectangle of type $T_i$ then
16:             resolveOneVerticalCut($T_j$, $r_{f2}$, $r_{r2}$)
17:         else if $r_{f1}$ and $r_{r1}$ add up to at least one whole rectangle of type $T_i$, and $r_{f2}$ and $r_{r2}$
18:         also add up to at least one whole rectangle of type $T_j$ then
19:             resolveOneVerticalCut($T_i$, $r_{f1}$, $r_{r1}$)
20:             resolveOneVerticalCut($T_j$, $r_{f2}$, $r_{r2}$)
21:     else
22:         Let $r_f$ be the remaining piece of type $T_i$ as described in Lemma 3.7.2.
23:         resolveTwoVerticalCuts($T_i$, $r_f$, $r_{r1}$, $r_{r2}$)
24: for all rectangle types $T_i$ do
25:     Discard remaining fractional pieces of type $T_i$.
```

Algorithm 3.5 resolveOneVerticalCut(T_i, r_f, r_r)

```
1: In: Rectangle type $T_i$, a rectangular piece $r_f$ of type $T_i$ as described in Lemma 3.7.2, and a
       rounded-up part $r_r$ with full height of type $T_i$
2: if $r_f$ and $r_r$ add up to at least one whole rectangle of type $T_i$ then
3:     Move piece of $r_f$ of width ($w_i$ - width of $r_r$) right beside $r_r$ to form one whole rectangle
       of type $T_i$.
```

Algorithm 3.6 resolveTwoVerticalCuts(T_i, r_f, r_{r1}, r_{r2})

```
1: In: Rectangle type $T_i$, a rectangular piece $r_f$ of type $T_i$ as described in Lemma 3.7.2, a
       rounded-up part $r_{r1}$ with full height of type $T_i$, and a rounded-up part $r_{r2}$ with full
       height of type $T_i$
2: if $r_f$, $r_{r1}$, and $r_{r2}$ add up to at least one whole rectangle of type $T_i$ then
3:     if $r_f$ and $r_{r1}$ add up to less than one whole rectangle of type $T_i$ then
4:         Move $r_f$ and piece of $r_{r2}$ of width ($w_i$ - width of $r_f$ - width of $r_{r1}$) right beside $r_{r1}$
           to form one whole rectangle of type $T_i$.
5:     else
6:         Move a piece of $r_f$ of width ($w_i$ - width of $r_{r1}$) right beside $r_{r1}$ to form a whole
           rectangle of type $T_i$.
7:         Let $r_f'$ be the remaining rectangular piece from $r_f$ after Step 6.
8:         if $r_f'$ and $r_{r2}$ add up to at least one whole rectangle of type $T_i$ then
9:             Move a piece of $r_f'$ of width ($w_i$ - width of $r_{r2}$) right beside $r_{r2}$ to form one
               whole rectangle of type $T_i$.
```

Chapter 4

Analysis of the Algorithm

4.1 Correctness

Lemma 4.1.1 *Algorithm* roundThreeTypes *produces a packing that packs at least as many rectangles as the optimal fractional solution.*

Proof For each rectangle type T_i, all rectangles with fractional height of type T_i in an optimal fractional solution must add up to an integer number of rectangles because n_i, the total number of rectangles of type T_i is integer. Algorithm roundThreeTypes rounds up the *tall* fractional rectangles of one configuration in segment S_2 (see Steps 10–12 of algorithm packSegments) and all rectangles with fractional height in segment S_3 (see Steps 20–21 of algorithm packSegments) and part S' (see Step 8 of algorithm roundThreeTypes) so that they are of full height. If we do not discard any rectangular pieces after processing the rectangles as described in algorithm packSegments, then algorithm roundThreeTypes produces a packing that packs at least as many rectangles as the optimal fractional solution because we never round down fractional rectangles. However, in Steps 24–25 of algorithm removeFractionalRectangles we discard some fractional pieces; to show that algorithm roundThreeTypes still produces a packing that packs all rectangles, we need to consider the two scenarios from Section 3.7:

– **Scenario 1.** $S = S_1$. We reshape all rectangles of fractional height in S_1 and then pack them according to Lemma 3.5.1 to form as many whole rectangles as possible. As described in Scenario 1 of Section 3.7, if we have any remaining fractional pieces, then part S' is not empty. We can discard these pieces because they must have already been packed when the fractional rectangles in S' were rounded up to their full height. Thus, the final packing will pack all the rectangles.

– **Scenario 2.** $S \neq S_1$. According to Lemma 3.7.2, we form as many whole rectangles as possible from the rectangles of fractional height in part S that were not rounded up to their full height. As described in Scenario 2 of Section 3.7, we now have two additional cases:

  (a) No rectangles are cut vertically such that one part of a rectangle is rounded up and the other part is packed according to Lemma 3.5.1. As described in Scenario 2(a) of Section 3.7, we can discard any remaining fractional pieces because these pieces must have already been packed when the fractional rectangles in S_2, S_3, and/or S' were rounded up to their full height. Thus, the final packing will include all the rectangles.

  (b) Some rectangles (at most two according to Lemma 3.7.1) are cut vertically such that one part of a rectangle is rounded up and the other part is packed according to Lemma 3.5.1.

    – If only one fractional rectangle r is vertically cut (see Steps 5–7 of algorithm REMOVEFRACTIONALRECTANGLES), then there must be at most two fractional rectangular pieces of the same type as r according to Subsection 3.7.1. Since these two pieces cannot add up to two whole rectangles, we pack them according to Lemma 3.7.3 to form at most one whole rectangle (see algorithm RESOLVEONEVERTICALCUT).

    – If two fractional rectangles r_1 and r_2 of different types are vertically cut (see Steps 8–20 of algorithm REMOVEFRACTIONALRECTANGLES), there are at most two fractional rectangular pieces of the same type as r_1, and at most two fractional rectangular pieces of the same type as r_2, according to Subsection 3.7.2. We pack these pieces according to Lemma 3.7.3 to form at most one whole rectangle for each type.

– If two fractional rectangles r_1 and r_2 of the same type are vertically cut (see Steps 21–23 of algorithm REMOVEFRACTIONALRECTANGLES), there are at most three fractional rectangular pieces of the same type as r_1 and r_2 according to Subsection 3.7.3. Since these three pieces cannot add up to three whole rectangles, we pack these pieces according to Lemmas 3.7.4 (see Steps 2–4 of algorithm RESOLVETWOVERTICALCUTS) and 3.7.3 (see Steps 5–9 of algorithm RESOLVETWOVERTICALCUTS) to form at most two whole rectangles.

We then discard any remaining fractional pieces because the number n_i for any rectangle type T_i is integer, so these pieces must have already been packed when the fractional rectangles in S_2, S_3, and/or S' were rounded up to their full height. Thus, the final packing will pack all the rectangles.

Therefore, algorithm ROUNDTHREETYPES produces a packing that packs at least as many rectangles as the optimal fractional solution. □

Theorem 4.1.2 *Algorithm* ROUNDTHREETYPES *produces a packing of height at most $H + 5/3$, where H is the height of the optimal fractional solution. This packing is integral with no fractional parts.*

Proof As described in Section 3.1, we rearrange the rectangles within each configuration such that no vertical cuts occur between parts S' and S. Note that the height increase in S' does not affect the height increase of S. Thus, we analyze the height increase of each part separately:

Height increase at S'. In Step 8 of algorithm ROUNDTHREETYPES, we round up all rectangles with fractional height at the top of S' so that they are whole. Thus, part S' has no fractional parts and the maximum increase in height for S' is 1.

Height increase at *S*. As described in Section 3.4 we partition part *S* into at most three segments: S_1, S_2, and S_3. We need to consider four different cases:

- **Only S_1 is not empty.** In Steps 3–4 of algorithm packSegments, we reshape all rectangles of fractional height in *S* and then pack them according to Lemma 3.5.1. Thus, the maximum increase in height for this case is 1.

- **Only S_2 is not empty.** In Steps 10–12 of algorithm packSegments, we round up the *tall* fractional rectangles of one configuration C_γ as described in Lemma 3.5.2 to increase the height of S_2 by at most 2/3. We reshape the remaining rectangles of fractional height in *S* and then pack them according to Lemmas 3.5.1 and 3.5.2 to further increase the height of S_2 by at most 1. Thus, the maximum increase in height for this case is 5/3.

- **Only S_3 is not empty.** In Steps 20–21 of algorithm packSegments, we round up all rectangles of fractional height in *S*. Thus, the maximum increase in height for this case is 5/3 as described in Subsection 3.5.3. Note that part *S* has no fractional parts because we rounded up all rectangles with fractional height.

- **At least two segments are not empty.** After processing the rectangles as described in algorithm packSegments, according to Lemmas 3.6.2 and 3.6.3 all whole rectangles that cross the border between any two segments remain whole as they are not vertically cut. Lemmas 3.6.2 and 3.6.3 also state that the maximum increase in height for this case is 5/3. However, according to Lemma 3.7.1 at most two fractional rectangles cross the border between two adjacent segments and are vertically cut such that one part is rounded up and the other part is packed as in Lemma 3.5.1. We now have three additional cases:

  – If only one fractional rectangle *r* is vertically cut, then there must be at most two fractional rectangular pieces of the same type as *r* as explained in Subsection 3.7.1. We process these pieces according to Lemma 3.7.3 to form at most one whole rectangle without increasing the height of the packing.

- If two fractional rectangles r_1 and r_2 of different types T_i and T_j are vertically cut, there are at most two fractional rectangular pieces of the same type as r_1, and at most two fractional rectangular pieces of the same type as r_2, as explained in Subsection 3.7.2. We process these pieces according to Lemma 3.7.3 to form at most one whole rectangle for each type T_i, T_j without increasing the height of the packing.
- If two fractional rectangles r_1 and r_2 of the same type are vertically cut, there are at most three fractional rectangular pieces of the same type as r_1 and r_2 as explained in Subsection 3.7.3. We process these pieces according to Lemmas 3.7.4 and 3.7.3 to form at most two whole rectangles without increasing the height of the packing.

Thus, the maximum increase in height when at least two segments are not empty is still 5/3.

In Steps 24–25 of algorithm REMOVEFRACTIONALRECTANGLES we discard any remaining fractional pieces. Since by Lemma 4.1.1 algorithm ROUNDTHREETYPES produces a final packing that packs at least as many rectangles as the optimal fractional solution, the algorithm produces a valid integral packing of height at most $H + 5/3$. □

4.2 Time Complexity

4.2.1 Input and Output of the Algorithm

The input to algorithm ROUNDTHREETYPES is an optimal fractional solution F with three configurations that we obtain by computing a basic feasible solution for the linear program (2.1). Recall that $K = 3$ is both the number of distinct rectangle types and the maximum number of configurations in an optimum basic feasible solution for the linear program (2.1). We can compactly represent F with a set of 21 numbers—first, we use 9 numbers to represent the set of rectangles, where for each rectangle type T_i there is a number for the multiplicity n_i (or number of rectangles of type T_i), and two numbers w_i and h_i for the width and height of each rectangle of type T_i. Second, we use 12 numbers to represent a set $C = \{C_1, C_2, C_3\}$ of configurations, where for each configuration C_j we use the following 4 numbers:

- One number x_j specifying that rectangles of the same type in configuration C_j are stacked up to height x_j in F.
- Three numbers $t_{i,j}$, for $i = 1, 2, 3$, specifying the number of rectangles of type T_i packed side-by-side in configuration C_j for F.

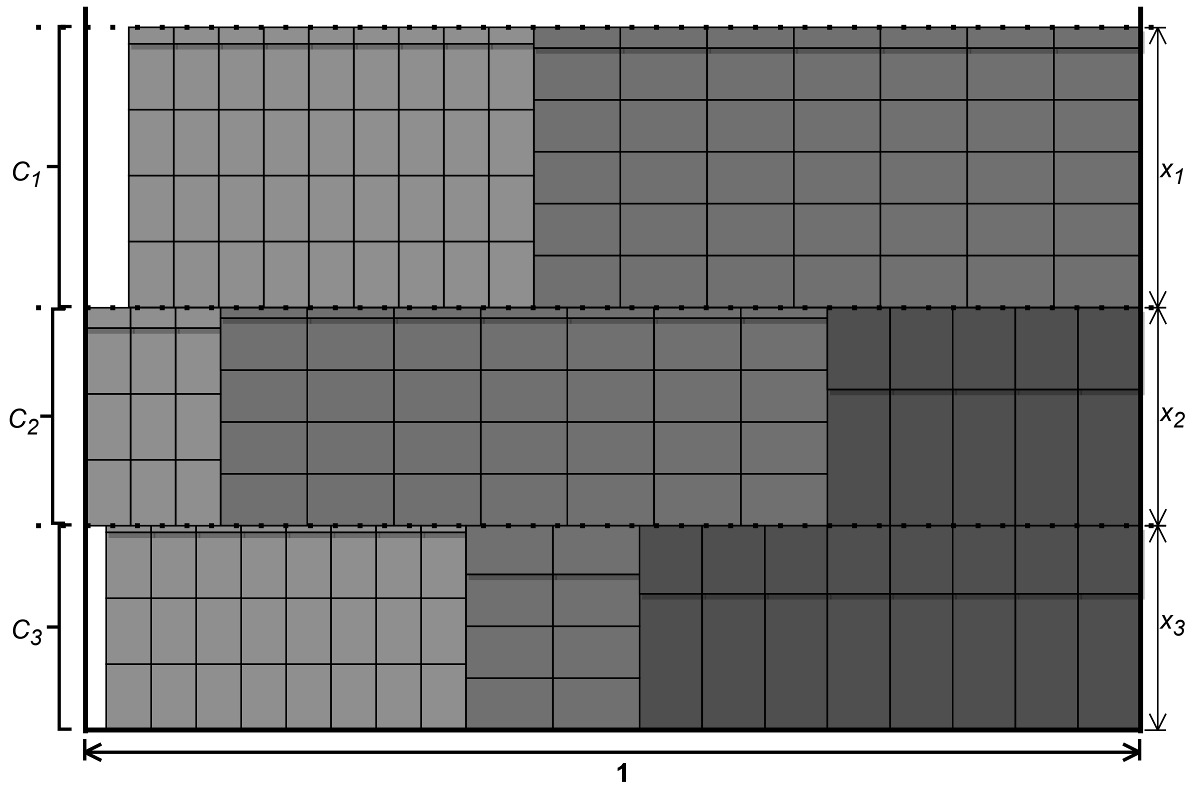

Figure 4.1: An optimal fractional solution for an instance of 3T-SPP with three configurations. Rectangles at the top of each configuration are cut horizontally. No rectangles are vertically cut.

Note that we do not need to specify the dimensions nor the position of each individual rectangle in F. We can compute the number of rectangles of type T_i that are stacked one on top of the other in configuration C_j for F by calculating x_j/h_i for each $i, j = 1, 2, 3$. For example, in Figure 4.1 $x_3/h_3 = 3/2$, so one and a half rectangles of type T_3 are stacked one on top of the other in C_3. Note that x_j/h_i can be a fractional number because F allows rectangles to be horizontally cut, but each $t_{i,j}$ is always integer because F does not allow rectangles to be vertically cut.

The output of algorithm ROUNDTHREETYPES is an integral packing (i.e., a solution without fractional rectangles) that packs all the rectangles. We can represent this packing with a set of 45 numbers—first, we use 6 numbers to represent the part S' for the integral packing as follows:

- Three numbers x_i', for $i = 1, 2, 3$, specifying the number of rectangles of type T_i that are stacked one on top of the other in part S' for the integral packing.

- Three numbers m_i, for $i = 1, 2, 3$, specifying the number of rectangles of type T_i packed side-by-side in part S' for the integral packing.

Second, we use 3 numbers η_i, for $i = 1, 2, 3$, to denote the number of rectangles of type T_i in part S packed according to Lemma 3.5.1. Finally, we use 36 numbers to represent the set $C = \{C_1, C_2, C_3\}$ of configurations in part S for the integral packing, where for each configuration C_j we use the following 12 numbers:

- Three numbers $x_{i,j}$, for $i = 1, 2, 3$, specifying the number of rectangles of type T_i in configuration C_j that are stacked one on top of the other in part S.

- Three numbers $\tau_{i,j}$, for $i = 1, 2, 3$, specifying the number of rectangles of type T_i packed side-by-side in configuration C_j that are in part S; for each of these rectangles, we stack on top of it more rectangles of the same type T_i until the number of rectangles in each stack is $x_{i,j}$.

- Three numbers $\Psi_{i,j}$, for $i = 1, 2, 3$, specifying the number of rectangles of type T_i packed side-by-side in configuration C_j that are in part S; for each of these rectangles, we stack on top of it more rectangles of the same type T_i until the number of rectangles in each stack is $x_{i,j} + 1$.

- Three rectangle type numbers to specify the order in which rectangle types are packed from left to right in configuration C_j for part S.

The output of 45 numbers is compact; we do not specify the dimensions nor the position of each rectangle in the integral packing. All 45 numbers are integer as we only pack whole rectangles; we use these numbers to determine how many whole rectangles of each type to pack in parts S' and S. For example, suppose we have the following output:

- $x_1' = 11$, $x_2' = 14$, $x_3' = 0$, $m_1 = 3$, $m_2 = 2$, $m_3 = 0$
- $\eta_1 = 2$, $\eta_2 = 1$, $\eta_3 = 0$
- $x_{1,1} = 4$, $x_{2,1} = 5$, $x_{3,1} = 0$, $\tau_{1,1} = 6$, $\tau_{2,1} = 1$, $\tau_{3,1} = 0$, $\Psi_{1,1} = 0$, $\Psi_{2,1} = 4$, $\Psi_{3,1} = 0$, 3, 1, 2 for C_1
- $x_{1,2} = 0$, $x_{2,2} = 4$, $x_{3,2} = 1$, $\tau_{1,2} = 0$, $\tau_{2,2} = 5$, $\tau_{3,2} = 0$, $\Psi_{1,2} = 0$, $\Psi_{2,2} = 0$, $\Psi_{3,2} = 5$, 2, 1, 3 for C_2
- $x_{1,3} = 3$, $x_{2,3} = 0$, $x_{3,3} = 1$, $\tau_{1,3} = 5$, $\tau_{2,3} = 0$, $\tau_{3,3} = 1$, $\Psi_{1,3} = 0$, $\Psi_{2,3} = 0$, $\Psi_{3,3} = 7$, 1, 3, 2 for C_3

This output means that in part S', we pack rectangle type T_1 three across and 11 up, and rectangle type T_2 two across and 14 up (see part S' in Figure 4.2). Note that we can pack rectangle types from left to right in any order for part S'.

In part S, we first pack rectangles of C_3 at the bottom of the strip, then rectangles of C_2 on top of C_3, then rectangles packed according to Lemma 3.5.1, and finally C_1 at the top. For each configuration, we pack rectangle types from left to right as specified in the input. First, we pack at the bottom of the strip rectangle type T_1 five across and 3 up, one rectangle of type T_3, and rectangle type T_3 seven across and 2 up. On top of these rectangles, we pack rectangle type T_2 five across and 4 up, and rectangle type T_3 five across and 2 up. On top of these rectangles, we pack rectangle type T_1 two across and rectangle type T_2 one across. Finally, we pack on top rectangle type T_1 six across and 4 up, rectangle type T_2 one across and 5 up, and rectangle type

T_2 four across and 6 up. Note that in a configuration C_j we pack the $\tau_{i,j}$ rectangles of type T_i to the left of the $\Psi_{i,j}$ rectangles of the same type T_i so that the segments of part S as described in Section 3.4 are in the correct order (see part S in Figure 4.2).

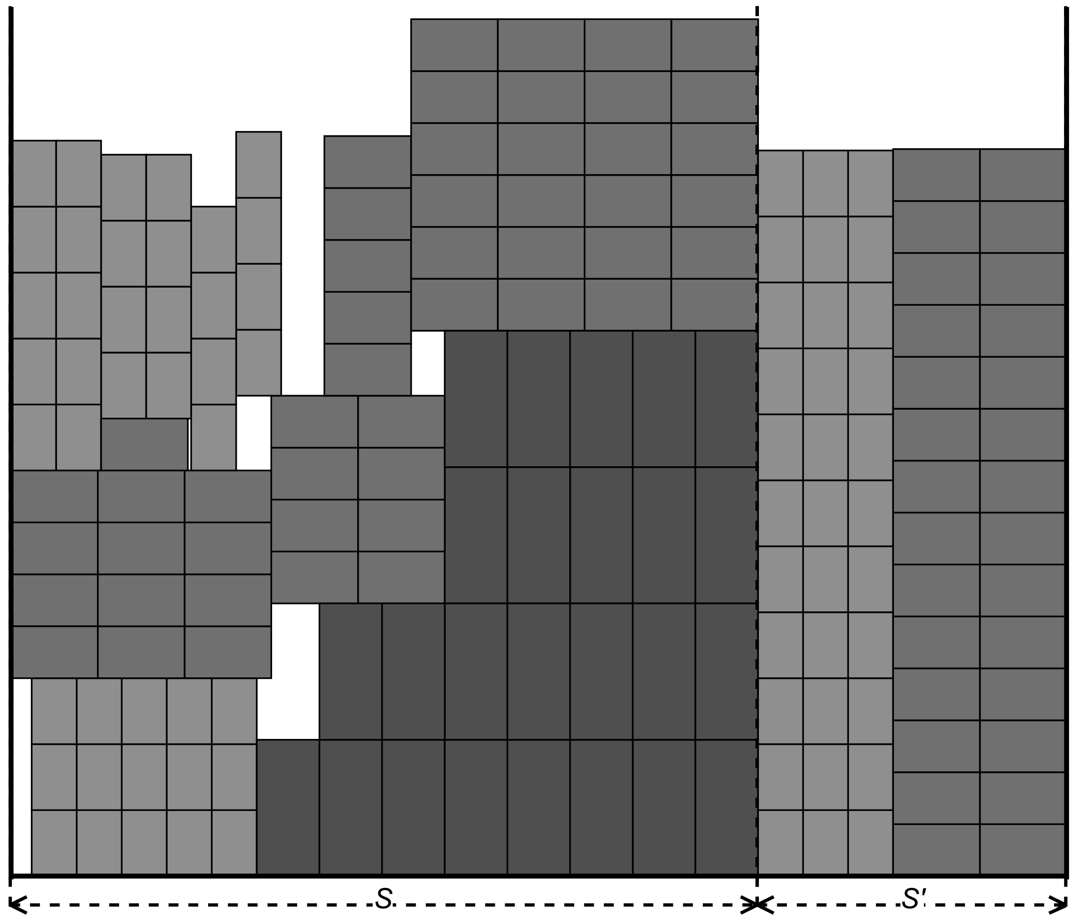

Figure 4.2: Using an output of 45 numbers to create an integral packing with only whole rectangles. This packing has more rectangles than the corresponding optimal fractional solution in Figure 4.1.

Because the algorithm rounds up some fractional rectangles, the output can contain more rectangles than in the input. Thus, when constructing the final integral packing, we simply stop packing rectangles of type T_i once we have packed n_i rectangles of type T_i.

4.2.2 Running Time

In Steps 3–9 of algorithm roundThreeTypes we calculate the numbers x_i' and m_i for $i = 1, 2, 3$. We also compute $\Phi_{i,j}$, the number of rectangles of type T_i packed side-by-side in configuration

C_j that are in part S, for each $i, j = 1, 2, 3$.

Lemma 4.2.1 *Steps 3–9 of algorithm* roundThreeTypes *run in constant time.*

Proof Steps 3–8 compute the following numbers for all rectangle types T_i:

- Three numbers $m_i = \min\left\{t_{i,j} \mid j = 1, 2, 3\right\}$.
- Three numbers x_i'. If $m_i > 0$ we set $x_i' = \lceil(\sum_{j=1}^{3} x_j)/h_i\rceil$; otherwise we set $x_i' = 0$.
- Three numbers $\Phi_{i,j} = t_{i,j}$ - m_i for each configuration C_j.

The algorithm requires $O(K^2)$ operations to compute the above numbers because we have K distinct rectangle types and K configurations; the algorithm performs at most K^2 times a constant number of comparisons, ceiling operations, additions, divisions, and subtractions. Step 7 is for the proof of correctness, i.e., the algorithm does not actually rearrange individual rectangles because the runtime of the algorithm would be superpolynomial on the input if the number of individual rectangles is much larger than the input size. Similarly, Step 3 does not actually stack configurations on top of each other. For Step 9, if any $\Phi_{i,j} > 0$ then part S is not empty. Therefore, Step 9 requires at most $O(K^2)$ operations because we have K distinct rectangle types and K configurations; the algorithm performs at most K^2 comparisons. Thus, Steps 3–9 of algorithm roundThreeTypes run in constant time. □

Lemma 4.2.2 *Assume part S is empty. Algorithm* roundThreeTypes *runs in constant time.*

Proof According to Lemma 4.2.1, Steps 3–9 of algorithm roundThreeTypes run in constant time. If Step 9 determines that part S is empty, the algorithm does not perform Steps 10–12; instead, the algorithm returns the integral packing in Step 13. This step requires $O(K^2)$ operations because we have K distinct rectangle types and K configurations; the algorithm performs a constant number of operations K^2 times to set all output numbers that represent part S to 0. Thus, algorithm roundThreeTypes runs in constant time if part S is empty. □

If S is not empty, the algorithm performs Steps 10–12. Step 10 uses algorithm PARTITIONPARTS to calculate rectangle type numbers that specify the order in which rectangles types are packed in S for the output. The input to algorithm PARTITIONPARTS is the set of rectangles, and the set $C = \{C_1, C_2, C_3\}$ of configurations. As explained in Subsection 4.2.1, we use 9 numbers to represent the set of rectangles. For set C, we use 12 numbers, where for each configuration C_j we use the following 4 numbers:

- One number x_j specifying that rectangles of the same type in configuration C_j are stacked up to height x_j in the optimal fractional solution F.
- Three numbers $\Phi_{i,j}$, for $i = 1, 2, 3$, specifying the number of rectangles of type T_i packed side-by-side in configuration C_j that are in part S.

We also use 3 numbers m_i, for $i = 1, 2, 3$ to specify the number of rectangles of type T_i packed side-by-side in part S'. Note that these three numbers are not explicitly mentioned in the pseudocode of algorithm PARTITIONPARTS.

The output of algorithm PARTITIONPARTS is a set of 31 numbers. We use 3 numbers ω_1, ω_2, and ω_3 to denote the widths of segments S_1, S_2, and S_3. We use one number γ to denote configuration C_γ as described in Lemma 3.5.2. Finally, we use 27 numbers to represent the set $C = \{C_1, C_2, C_3\}$ of configurations, where for each configuration C_j we use the following 9 numbers:

- Three numbers $f_{i,j}$, for $i = 1, 2, 3$, specifying the fraction of each rectangle of type T_i that lies just below the top of configuration C_j in the optimal fractional solution F.
- Three numbers $w_{i,j}$, for $i = 1, 2, 3$, specifying the width of rectangles of type T_i packed side-by-side in configuration C_j that are in part S.
- Three rectangle type numbers to specify the order in which rectangle types are packed from left to right in configuration C_j for part S.

Lemma 4.2.3 *Algorithm* PARTITIONPARTS *runs in constant time.*

Proof Steps 3–7 compute $1 - \sum_{i=1}^{3} m_i w_i$, the position of the right border of part S, where $\sum_{i=1}^{3} m_i w_i$ is the width of part S'. Thus, Steps 3–7 require $O(K)$ operations because we have K distinct rectangle types; the algorithm performs a constant number of multiplications and additions K times and then one subtraction. In Steps 8–13, the algorithm requires $O(K^2)$ operations to compute the following numbers for all configurations C_j because we have K distinct rectangle types and K configurations; the algorithm performs a constant number of multiplications, divisions, floor operations, and subtractions K^2 times:

- Three numbers $f_{i,j} = x_j/h_i - \lfloor x_j/h_i \rfloor$, for each rectangle type T_i.
- Three numbers $w_{i,j} = \Phi_{i,j} \cdot w_i$, for each rectangle type T_i.

Step 10 uses selection sort to sort $f_{i,j}$ for each configuration C_j in nondecreasing order and then list rectangle type numbers $i = 1, 2, 3$ in that order. Thus, Step 10 requires at most $O(K^3)$ operations because we have K distinct rectangle types and K configurations; running selection sort for a configuration requires at most $O(K^2)$ operations and the algorithm performs selection sort K times. Step 11 is for the proof of correctness, i.e., the algorithm does not actually sort individual rectangles because the runtime of the algorithm would be superpolynomial on the input if the number of individual rectangles is much larger than the input size.

Steps 12–13 calculate at most K positions for each configuration C_j because each C_j has one position where it ends and at most $K - 1$ positions where C_j switches from some rectangle type to a different rectangle type in part S. The algorithm requires at most $O(K^2)$ operations to compute at most K^2 positions because part S has at most K distinct rectangle types as well as K configurations; the algorithm performs at most K subtractions K times. For example, if configuration C_2 contains 3 rectangle types in part S and 1 is the rectangle type number listed first, the algorithm requires K subtractions to compute $(1 - \sum_{i=1}^{3} m_i w_i) - w_{3,2}$, $(1 - \sum_{i=1}^{3} m_i w_i) - w_{3,2} - w_{2,2}$, and $(1 - \sum_{i=1}^{3} m_i w_i) - w_{3,2} - w_{2,2} - w_{1,2}$ because Steps 3–7 previously computed $(1 - \sum_{i=1}^{3} m_i w_i)$. The last computed number in the above example is the position where C_2 ends.

Step 14 uses selection sort to sort the set of at most K^2 positions in nondecreasing order while also removing any duplicate positions; Step 14 requires at most $O(K^4)$ operations because selection sort has quadratic worst-case time complexity. The algorithm then adds the position of the right border of part S to the end of this set, which requires one comparison with the last element of the set. Thus, this set has at most $K^2 + 1$ unique values; Steps 16–21 create at most K^2 columns which are used to partition part S into at most 3 segments: S_1, S_2, and S_3.

Recall from Section 3.4 that $F_{j,c}$ is the fraction of each rectangle that lies just below the cutting line in configuration C_j for column k_c, and that within a column each configuration has at most one rectangle type. The algorithm uses the position of the right border of column k_c to determine this rectangle type. For example, if the right border of column k_c is at the position where C_j switches from rectangle type T_i to a different rectangle type in part S, the rectangle type is T_i. If the right border of column k_c is at the same position as the right border of part S, the rectangle type is the rectangle type number listed last for C_j that has $w_{i,j} > 0$. For any column k_c, the algorithm performs K additions to compute $\sum_{j=1}^{3} F_{j,c}$ because we have K configurations, so the algorithm performs $O(K)$ operations.

For Steps 15–66, we need to consider three different cases:

1. If $\sum_{j=1}^{3} F_{j,c} \leq 4/3$ for each column $k_c \in S$, then Steps 15–32 require at most $O(K^3)$ operations to scan at most K^2 columns and determine if they belong to either S_1 or S_2. The algorithm performs $O(K)$ operations at most K^2 times to compute and compare each $\sum_{j=1}^{3} F_{j,c}$ to at most 2 other numbers. If $\sum_{j=1}^{3} F_{j,c} > 1$ for any column k_c, Steps 27–28 set γ as the configuration with the largest value $F_{j,c}$ in the leftmost column for which $\sum_{j=1}^{3} F_{j,c} > 1$; otherwise, the algorithm sets $\gamma = 0$. Step 19 terminates the algorithm so it never performs Steps 33–66. Note that the width of any segment is the difference between the positions of the right border of the rightmost column and the left border of the leftmost column for that segment. Thus, Step 19 uses a constant number of subtractions to calculate ω_1, ω_2, and ω_3.

2. If $\sum_{j=1}^{3} F_{j,c} > 4/3$ for each column $k_c \in S$, then Steps 15–61 require at most $O(K^3)$ operations to scan at most K^2 columns and determine if they belong to S_3: the algorithm

performs $O(K)$ operations at most K^2 times to compute and compare each $\sum_{j=1}^{3} F_{j,c}$ to at most 2 other numbers. As we previously calculated $1 - \sum_{i=1}^{3} m_i w_i$, the algorithm uses a constant number of operations to set $\omega_1 = 0$, $\omega_2 = 0$, $\omega_3 = 1 - \sum_{i=1}^{3} m_i w_i$, and $\gamma = 0$.

3. If a column exists in S for which $\sum_{j=1}^{3} F_{j,c} \leq 4/3$ and another column exists in S for which $\sum_{j=1}^{3} F_{j,c} > 4/3$, then Steps 33–48 determine whether to add an additional column. Recall from Definition 3.1 that P_σ is the position of the left border of the leftmost column for which $\sum_{j=1}^{3} F_{j,c} > 4/3$. For any configuration C_j, after determining the rectangle type T_i that is in the leftmost column of C_j for which $\sum_{j=1}^{3} F_{j,c} > 4/3$, the algorithm computes the difference between the position of the right border of the rightmost rectangle of type T_i in C_j that is in part S and P_σ. If the resulting number is not divisible by w_i, i.e., the remainder ϱ is not equal to 0, then rectangles of type T_i cross P_σ. Note that the sum of P_σ and ϱ is the position of the right border of any rectangles of C_j that cross P_σ. Thus, the algorithm requires at most $O(K)$ operations to determine if rectangles from two configurations cross P_σ because we have K configurations; the algorithm performs at most K times a constant number of comparisons, subtractions, and modulo operations.

   If rectangles from 2 configurations cross P_σ, recall from Definition 3.2 that P_{min} is the leftmost position between the right borders of rectangles from the 2 configurations that cross P_σ. Steps 40–48 now have two additional cases:

   (a) If no columns exist for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$, then the algorithm adds P_{min} to the set of positions. The algorithm first uses binary search to find P_σ within the set of positions, and if P_{min} is not in the set already, then adds P_{min} in its sorted position. Thus, the algorithm requires at most $O(K^2)$ operations because the set initially had at most K^2 positions. The algorithm sets γ as the configuration that does not have rectangles that cross P_σ.

   (b) If columns exist for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$, note that Steps 27–28 set γ as the configuration with the largest value $F_{j,c}$ in the leftmost column for which $\sum_{j=1}^{3} F_{j,c} >$

1. If the two configurations with rectangles that cross P_σ are both different from C_γ, and if P_{min} is not in the set of positions already, then the algorithm adds P_{min} in its sorted position. This process performs at most $O(K^2)$ operations.

If rectangles from two configurations do not cross P_σ, the algorithm does not add an additional column. If no columns exist for which $4/3 \geq \sum_{j=1}^{3} F_{j,c} > 1$, the algorithm sets $\gamma = 0$; otherwise, Steps 27–28 set γ as the configuration with the largest value $F_{j,c}$ in the leftmost column for which $\sum_{j=1}^{3} F_{j,c} > 1$. Therefore, Steps 15–61 require at most $O(K^3)$ operations to scan at most $K^2 + 1$ columns and determine if they belong to either S_1, S_2, or S_3. The algorithm performs $O(K)$ operations at most $K^2 + 1$ times to compute and compare each $\sum_{j=1}^{3} F_{j,c}$ to at most 2 other numbers. Note that the width of any segment is the difference between the positions of the right border of the rightmost column and the left border of the leftmost column for that segment. Thus, the algorithm uses a constant number of subtractions to calculate ω_1, ω_2, and ω_3.

In Steps 62–65, if S_3 and its adjacent segment are not empty, the algorithm requires at most $O(K)$ operations to determine the configurations with rectangles that cross the boundary between S_3 and its adjacent segment because the algorithm performs a constant number of comparisons, subtractions, and modulo operations at most K times. The position of the boundary between S_3 and its adjacent segment is either P_σ or $P_{\min}$. As we previously determined C_γ, the algorithm requires at most $O(K^2)$ operations to rename configurations according to Lemmas 3.4.2, 3.4.5, and 3.4.7 because there are K distinct rectangle types and K configurations.

Steps 18 and 66 are strictly for the proof of correctness, i.e., the algorithm does not actually invert configuration C_1. Similarly, Steps 62–65 do not actually reorder configurations. Therefore, algorithm PARTITIONPARTS runs in constant time. □

Step 11 of algorithm ROUNDTHREETYPES uses algorithm PACKSEGMENTS to calculate the numbers $x_{i,j}$. The algorithm also computes the numbers $\tau_{i,j}$, $\Psi_{i,j}$, and η_i for Step 12 of algorithm roundThreeTypes. The input to algorithm packSegments is the set of rectangles, and the set $C = \{C_1, C_2, C_3\}$ of configurations in part S. As explained in Subsection 4.2.1, we use 9 numbers to represent the set of rectangles. For set C we use 39 numbers, where for each configuration C_j we use the following 13 numbers:

- One number x_j specifying that rectangles of the same type in configuration C_j are stacked up to height x_j in the optimal fractional solution F.
- Three numbers $f_{i,j}$, for $i = 1, 2, 3$, specifying the fraction of each rectangle of type T_i that lies just below the top of configuration C_j in the optimal fractional solution F.
- Three numbers $\Phi_{i,j}$, for $i = 1, 2, 3$, specifying the number of rectangles of type T_i packed side-by-side in configuration C_j that are in part S.
- Three numbers $w_{i,j}$, for $i = 1, 2, 3$, specifying the width of rectangles of type T_i packed side-by-side in configuration C_j that are in part S.
- Three rectangle type numbers to specify the order in which rectangle types are packed from left to right in configuration C_j for part S.

We also use one number γ to specify configuration C_γ as described in Lemma 3.5.2, and three numbers ω_1, ω_2, and ω_3 specifying the width of segments S_1, S_2, and S_3. Note that these four numbers are not explicitly stated in the pseudocode of algorithm packSegments.

The output of algorithm packSegments is a set of 30 numbers. We use 3 numbers η_i, for $i = 1, 2, 3$, to denote the number of rectangles of type T_i in part S packed according to Lemma 3.5.1. We use 27 numbers to represent the set $C = \{C_1, C_2, C_3\}$ of configurations in part S, where for each configuration C_j we use the following 9 numbers:

- Three numbers $x_{i,j}$, for $i = 1, 2, 3$, specifying the number of rectangles of type T_i in configuration C_j that are stacked one on top of the other in part S.
- Three numbers $\tau_{i,j}$, for $i = 1, 2, 3$, specifying the number of rectangles of type T_i packed side-by-side in configuration C_j that are in part S; for each of these rectangles, we stack

on top of it more rectangles of the same type T_i until the number of rectangles in each stack is $x_{i,j}$.

- Three numbers $\Psi_{i,j}$, for $i = 1, 2, 3$, specifying the number of rectangles of type T_i packed side-by-side in configuration C_j that are in part S; for each of these rectangles, we stack on top of it more rectangles of the same type T_i until the number of rectangles in each stack is $x_{i,j} + 1$.

Lemma 4.2.4 *Algorithm* packSegments *computes $\tau_{i,j}$ for any configuration C_j and rectangle type T_i by performing at most $O(K)$ operations.*

Proof Recall from the description of the input for algorithm packSegments that a configuration C_j uses these 6 numbers:

- Three numbers $w_{i,j}$, for $i = 1, 2, 3$.
- Three rectangle type numbers to specify the order in which rectangle types are packed from left to right in configuration C_j for part S.

The input also uses one number γ to specify configuration C_γ as described in Lemma 3.5.2 and three numbers ω_1, ω_2, and ω_3 to denote the widths of segments S_1, S_2, and S_3. To compute ω_s, the total width of part S, we add up ω_1, ω_2, and ω_3 by performing a constant number of additions. Having computed $w_{i,j}$ and ω_s, the algorithm performs at most K subtractions to compute the positions of the leftmost and rightmost rectangles of type T_i in C_j that are in part S because S contains at most K distinct rectangle types, so the algorithm requires at most $O(K)$ operations. For example, if 1 is the rectangle type number listed first in the input, the position of the left side of the leftmost rectangle of type T_1 in C_j that is in S is ω_s - $w_{3,j}$ - $w_{2,j}$ - $w_{1,j}$. After computing ω_1, ω_2, ω_3, and ω_s, as part S has at most 3 segments, we perform a constant number of additions to compute the position of any segment border. For example, if S_2 and S_3 are not empty, the position of the boundary between S_2 and S_3 is $\omega_1 + \omega_2$.

If configuration C_j is the same as C_γ, then $\tau_{i,j}$ is the number of rectangles of type T_i packed side-by-side in C_j that are in S_1. Let $\omega_{i,A}$ be the width of the rectangles of type T_i packed side-by-side in C_j that are in S_1. Using the positions of the leftmost and rightmost rectangles of type T_i in C_j that are in S and the position of the right border of S_1, we can compute $\tau_{i,j}$ by performing a constant number of comparisons, one subtraction, and one division. For example, if the boundary between S_1 and S_2 is to the left of the rightmost rectangle of type T_i in C_j that is in S and to the right of the leftmost rectangle of type T_i in C_j, then $\omega_{i,A}$ is the difference between the position of the boundary between S_1 and S_2 and the position of the left side of the leftmost rectangle of type T_i. We then divide $\omega_{i,A}$ by w_i to obtain $\tau_{i,j}$.

If configuration C_j is different from C_γ, then $\tau_{i,j}$ is the number of rectangles of type T_i packed side-by-side in C_j that are in S_1 and S_2. Let $\omega_{i,B}$ be the width of the rectangles of type T_i packed side-by-side in C_j that are in S_1 and S_2. Using the positions of the leftmost and rightmost rectangles of type T_i that are in S and the positions of the right borders of S_1 and S_2, we can compute $\tau_{i,j}$ by performing a constant number of comparisons, one subtraction, and one division. For example, if the rightmost rectangle of type T_i in C_j that is in S is to the left of the boundary between S_2 and S_3, then $\omega_{i,B}$ is the difference between the position of the right side of the rightmost rectangle of type T_i and the position of the left side of the leftmost rectangle of type T_i. We then divide $\omega_{i,B}$ by w_i to obtain $\tau_{i,j}$. Thus, algorithm PACKSEGMENTS computes $\tau_{i,j}$ for any configuration C_j and rectangle type T_i by performing at most $O(K)$ operations. □

Lemma 4.2.5 *Algorithm* PACKSEGMENTS *runs in constant time.*

Proof The algorithm computes the following numbers for all rectangle types T_i:

- Three numbers $\tau_{i,j}$ for each configuration C_j.
- Three numbers $x_{i,j}$ for each configuration C_j. If $\Phi_{i,j} > 0$, we set $x_{i,j} = \lfloor x_j/h_i \rfloor$; otherwise, we set $x_{i,j} = 0$.
- Three numbers $\Psi_{i,j} = \Phi_{i,j}$ - $\tau_{i,j}$ for each configuration C_j.

- Three numbers $\eta_i = \sum_{j=1}^{3} f_{i,j}\tau_{i,j}$.

According to Lemma 4.2.4, the algorithm requires at most $O(K)$ operations to compute $\tau_{i,j}$ for any configuration C_j and rectangle type T_i. Thus, the algorithm requires at most $O(K^3)$ operations to compute the above numbers because we have K distinct rectangle types and K configurations; the algorithm performs $O(K)$ subtractions and a constant number of floor operations, comparisons, and divisions K^2 times, and a constant number of additions and multiplications K times.

Steps 4, 11, 12, and 21 do not actually pack or round up individual rectangles because the runtime of the algorithm would be superpolynomial on the input if the number of individual rectangles is much larger than the input size. Similarly, Steps 13–18 and 22–26 are strictly for the proof of the correctness, i.e., the algorithm does not actually rearrange individual rectangles or add empty space to the packing. Note that for Steps 6–9, C_γ is given as part of the input of the algorithm. Therefore, algorithm packSegments runs in constant time. □

Step 12 of algorithm roundThreeTypes uses algorithm removeFractionalRectangles to calculate the numbers η_i, $\tau_{i,j}$, and $\Psi_{i,j}$. The input to algorithm removeFractionalRectangles is the set of rectangles, and the set $C = \{C_1, C_2, C_3\}$ of configurations in part S. As explained in Subsection 4.2.1, we use 9 numbers to represent the set of rectangles. For set C we use 27 numbers, where for each configuration C_j we use the following 9 numbers:

- Three numbers $\tau_{i,j}$, for $i = 1, 2, 3$, specifying the number of rectangles of type T_i packed side-by-side in configuration C_j that are in part S; for each of these rectangles, we stack on top of it more rectangles of the same type T_i until the number of rectangles in each stack is $x_{i,j}$.

- Three numbers $\Psi_{i,j}$, for $i = 1, 2, 3$, specifying the number of rectangles of type T_i packed side-by-side in configuration C_j that are in part S; for each of these rectangles, we stack on top of it more rectangles of the same type T_i until the number of rectangles in each stack is $x_{i,j} + 1$.

- Three rectangle type numbers to specify the order in which rectangle types are packed from left to right in configuration C_j for part S.

We also associate with set C three numbers η_i, for $i = 1, 2, 3$, specifying the number of rectangles of type T_i in part S packed according to Lemma 3.5.1. Thus, the input to algorithm removeFractionalRectangles is a set of 39 numbers. Note that two values $\tau_{i,j}$ and two values $\Psi_{i,j}$ can be fractional because as explained in Lemma 3.7.1 at most two rectangles with fractional height in part S are vertically cut between adjacent segments such that one part of a rectangle is rounded up and the other part is packed as in Lemma 3.5.1. All three values η_i can also be fractional because $f_{i,j}$ can be a fractional number.

The output of algorithm removeFractionalRectangles is a set of 21 numbers, where each number must be integer because we only pack whole rectangles in the integral packing. We use 3 numbers η_i, for $i = 1, 2, 3$ to denote rectangles in part S packed according to Lemma 3.5.1. We then use 18 numbers to represent the set $C = \{C_1, C_2, C_3\}$ of configurations in part S, where for each configuration C_j we use the following 6 numbers:

- Three numbers $\tau_{i,j}$ for $i = 1, 2, 3$

- Three numbers $\Psi_{i,j}$ for $i = 1, 2, 3$

For algorithm resolveOneVerticalCut, the input is 9 numbers denoting the set of rectangles, 2 numbers denoting the rectangle type and configuration of a rectangle r with fractional height that is vertically cut between two adjacent segments such that only one part of r is rounded up, and 3 numbers $\tau_{i,j}$, $\Psi_{i,j}$, and η_i. For algorithm resolveTwoVerticalCuts, the input is 9 numbers denoting the set of rectangles, 2 numbers denoting the rectangle type T_i and configuration C_j of a rectangle r_1 with fractional height that is vertically cut between S_1 and S_2 such that only one part of r_1 is rounded up, 1 number denoting the configuration C_v of a rectangle r_2 with fractional height and of type T_i that is vertically cut between S_2 and S_3 such that only one part of r_2 is rounded up, and 5 numbers $\tau_{i,j}$, $\Psi_{i,j}$, $\tau_{i,v}$, $\Psi_{i,v}$, and η_i. Note that the values $\tau_{i,j}$, $\Psi_{i,j}$,

$\tau_{i,v}$, $\Psi_{i,v}$, and η_i are not explicitly mentioned in the pseudocode of algorithms RESOLVEONEVERTICALCUT and RESOLVETWOVERTICALCUTS.

Lemma 4.2.6 *Algorithm* REMOVEFRACTIONALRECTANGLES *runs in constant time.*

Proof Steps 2–4 are strictly for the proof of correctness, i.e., algorithm PACKSEGMENTS already calculated the values η_i. Step 4 does not actually rearrange individual rectangular pieces because the runtime of the algorithm would be superpolynomial on the input if the number of individual rectangular pieces is much larger than the input size.

Recall from Lemma 4.2.5 that $\Psi_{i,j} = \Phi_{i,j} - \tau_{i,j}$. As $\Phi_{i,j}$ is integer, if any value $\tau_{i,j}$ is fractional, the corresponding $\Psi_{i,j}$ must also be fractional and vice versa. Therefore, Step 5 computes the total number of values $\Psi_{i,j}$ that are fractional to determine if any rectangles of fractional height are vertically cut between adjacent segments such that one part of a rectangle is rounded up and the other part is packed as in Lemma 3.5.1. This step requires at most $O(K^2)$ operations because there are K distinct rectangle types and K configurations; the algorithm performs at most K^2 comparisons.

If only one value $\Psi_{i,j}$ is fractional, Step 7 uses algorithm RESOLVEONEVERTICALCUT to calculate values $\tau_{i,j}$, $\Psi_{i,j}$, and η_i for rectangle type T_i and configuration C_j. Let $part(x) = x - \lfloor x \rfloor$. If $part(\eta_i) + part(\Psi_{i,j}) \geq 1$, then $\eta_i = \eta_i - (w_i - part(\Psi_{i,j}) \cdot w_i)$, $\tau_{i,j} = \lfloor \tau_{i,j} \rfloor$, and $\Psi_{i,j} = \lceil \Psi_{i,j} \rceil$. The algorithm requires a constant number of subtractions, floor operations, ceiling operations, and multiplications to calculate these numbers.

If two values $\Psi_{i,j}$ and $\Psi_{g,v}$ are fractional, then two rectangles r_1 and r_2 with fractional height and of types T_i, T_g in configurations C_j and C_v are vertically cut between segments S_1 and S_2, and between segments S_2 and S_3, respectively, so that we round up the part of r_1 in S_2, round up the part of r_2 in S_3, and pack the other parts according to Lemma 3.5.1. We need to consider two different cases:

1. If types T_i and T_g are different, then in Steps 9–20 if $part(\eta_i) + part(\Psi_{i,j}) \geq 1$ and $part(\eta_g) + part(\Psi_{g,v}) < 1$, the algorithm calls algorithm RESOLVEONEVERTICALCUT to

calculate new values $\tau_{i,j}$, $\Psi_{i,j}$, and η_i for rectangle type T_i and configuration C_j. If $part(\eta_g) + part(\Psi_{g,v}) \geq 1$ and $part(\eta_i) + part(\Psi_{i,j}) < 1$, the algorithm calls algorithm resolveOneVerticalCut to calculate new values $\tau_{g,v}$, $\Psi_{g,v}$, and η_g for rectangle type T_g and configuration C_v. Finally, if $part(\eta_i) + part(\Psi_{i,j}) \geq 1$ and $part(\eta_g) + part(\Psi_{g,v}) \geq 1$, the algorithm calls algorithm resolveOneVerticalCut to calculate new values $\tau_{i,j}$, $\Psi_{i,j}$, and η_i for rectangle type T_i and configuration C_j, and then calls algorithm resolveOneVerticalCut to calculate new values $\tau_{g,v}$, $\Psi_{g,v}$, and η_g for rectangle type T_g and configuration C_v. As we only call algorithm resolveOneVerticalCut at most 2 times, Steps 9–20 require a constant number of subtractions, floor operations, ceiling operations, and multiplications to calculate these numbers.

2. If types T_i and T_g are equal, recall from Lemma 3.7.1 that rectangle r_2 must be in configuration C_1 so that $C_v = C_1$. As T_i and T_g are equal, C_j cannot be C_1 because otherwise $\Psi_{i,j}$ and $\Psi_{i,1}$ would have the same value. Therefore, Step 23 uses algorithm resolveTwoVerticalCuts to calculate new values $\tau_{i,j}$, $\Psi_{i,j}$, $\tau_{i,1}$, $\Psi_{i,1}$, η_i for rectangle type T_i and configurations C_j of r_1 and C_1 of r_2 where $C_j \neq C_1$.

   In algorithm resolveTwoVerticalCuts, if $part(\eta_i) + part(\Psi_{i,j}) + part(\Psi_{i,1}) \geq 1$, then we proceed as follows. If $part(\eta_i) + part(\Psi_{i,j}) < 1$, then $\tau_{i,1} = \tau_{i,1} + (w_i - part(\eta_i) \cdot w_i - part(\Psi_{i,j}) \cdot w_i)$, $\Psi_{i,1} = \Psi_{i,1} - (w_i - part(\eta_i) \cdot w_i - part(\Psi_{i,j}) \cdot w_i)$, $\tau_{i,j} = \lfloor \tau_{i,j} \rfloor$, $\Psi_{i,j} = \lceil \Psi_{i,j} \rceil$, $\eta_i = \lfloor \eta_i \rfloor$. Otherwise, if $part(\eta_i) + part(\Psi_{i,j}) \geq 1$, then $\eta_i = \eta_i - (w_i - part(\Psi_{i,j}) \cdot w_i)$, $\tau_{i,j} = \lfloor \tau_{i,j} \rfloor$, $\Psi_{i,j} = \lceil \Psi_{i,j} \rceil$. If $part(\eta_i) + part(\Psi_{i,1}) \geq 1$, then $\eta_i = \eta_i - (w_i - part(\Psi_{i,1}) \cdot w_i)$, $\tau_{i,1} = \lfloor \tau_{i,1} \rfloor$, $\Psi_{i,1} = \lceil \Psi_{i,1} \rceil$.

   The algorithm requires a constant number of subtractions, floor operations, ceiling operations, and multiplications to calculate these numbers.

In Steps 24–25, the algorithm rounds down any fractional values $\tau_{i,j}$, $\Psi_{i,j}$, and η_i so that they are integer. Thus, Steps 24–25 require $O(K^2)$ operations because we have K distinct rectangle types and K configurations; the algorithm performs at most a constant number of floor operations K^2 times. Step 3 of algorithm resolveOneVerticalCut and Steps 4, 6, 9 of algorithm resolveTwoVerticalCuts do not actually move rectangular pieces to form whole rectangles as this is strictly for the proof of correctness. Therefore, algorithm removeFractionalRectangles runs in constant time. □

Theorem 4.2.7 *Algorithm* roundThreeTypes *runs in constant time.*

Proof According to Lemma 4.2.1, Steps 3–9 of algorithm roundThreeTypes run in constant time. After checking whether part S is empty, we have two cases:

1. If part S is empty, then algorithm roundThreeTypes runs in constant time according to Lemma 4.2.2.
2. If part S is not empty, then algorithm roundThreeTypes runs in constant time according to Lemmas 4.2.3, 4.2.5, and 4.2.6.

Therefore, algorithm roundThreeTypes runs in constant time. □

Theorem 4.2.8 *There exists a polynomial-time approximation algorithm A for 3T-SPP that produces an integral packing of height A(I) for any instance I of 3T-SPP. If OPT(I) is the minimum possible height within which the rectangles in I can be packed, then* $A(I) \leq OPT(I) + 5/3$.

Proof Recall from Section 2.2 that our algorithm first uses an algorithm in [7] as a subroutine to solve the fractional 3T-SPP in polynomial time. Price [7] proved that the time complexity of this algorithm is dominated by the running time of the Karmarkar and Karp algorithm [36]. The runtime for this algorithm in the worst case is $O\left(K^{10} \log K \log^2\left(\frac{Kn}{a}\right) + K^3 \log K \log n\right)$, where K is the number of rectangle types and a is the width of the thinnest rectangle. As 3T-SPP has three rectangle types, we substitute $K = 3$ to get $O\left(\log^2\left(\frac{n}{a}\right) + \log n\right)$. This algorithm runs in polynomial time because the number of bits used to represent the input is at least $\log n$ as $\log(n_1) + \log(n_2) + \log(n_3) \geq \log(\frac{n}{3}) = \log n - \log 3$ and the number of bits used to represent the widths and heights of the rectangles is $\sum_{i=1}^{3}\log(w_i) + \sum_{i=1}^{3}\log(h_i)$. Note that the number of bits used to represent a is part of the number of bits used to represent the widths of the rectangles.

The fractional solution F obtained by computing a basic feasible solution for the linear program (2.1) consists of at most 3 configurations. If F uses one configuration, then we simply round up the rectangles with fractional height at the top of the configuration and discard any extra rectangles. If F uses two configurations, then we use the algorithm for HM-SPP in [7] to transform F into an integral solution for 3T-SPP. Both of these algorithms produce a solution to 3T-SPP of height at most $OPT(I) + 1$ as described in Section 2.2. Price [7] proved that the time complexity of the algorithm to compute a basic feasible solution for the linear program (2.1) dominates the worst-case runtime of these algorithms.

If F uses three configurations, we use algorithm roundThreeTypes to transform F into an integral solution for 3T-SPP. According to Theorem 4.1.2, the algorithm produces a correct solution to 3T-SPP of height at most $OPT(I) + 5/3$ because the height of the optimal fractional solution is a lower bound for $OPT(I)$ as described in Section 2.1. The time complexity of the algorithm in [7] to compute a basic feasible solution for the linear program (2.1) dominates the worst-case runtime of algorithm roundThreeTypes because algorithm roundThreeTypes runs in constant time according to Theorem 4.2.7. Therefore, we have a polynomial-time approximation algorithm that produces a solution of height at most $OPT(I) + 5/3$ for any instance of 3T-SPP. $\square$

Chapter 5

Conclusions

This thesis considers the high multiplicity version of the two-dimensional strip packing problem (HM-SPP) in which the number of distinct rectangle types in the set $R = \{r_1, ..., r_n\}$ of n rectangles is a fixed, positive integer K. In particular, we focused on the special case where $K = 3$. We represent the input of an instance I of 3T-SPP as a set of rectangle types $T = \{T_1, T_2, T_3\}$; each rectangle type T_i has multiplicity n_i, width w_i, and height h_i. We present a new polynomial-time approximation algorithm for 3T-SPP that produces a solution of height at most $OPT(I) + 5/3$ and runtime $O\left(\log^2\left(\frac{n}{a}\right) + \log n\right)$, where $OPT(I)$ is the minimum possible height within which the rectangles in I can be packed and a is the width of the thinnest rectangle.

5.1 Remarks

In this thesis we assume without loss of generality that the height h_{max} of the tallest rectangle in an instance I of 3T-SPP is normalized to 1. If we have an instance I of 3T-SPP where $h_{max} > 1$, we would multiply all heights by $1/h_{max}$ to create a new instance I', use our algorithm on I', and then scale this packing vertically by h_{max} to obtain a packing of I. However, this scaling changes the additive constant from 5/3 to $5/3 \cdot h_{max}$.

The question posed in [7] is still open: what is the complexity class for 3T-SPP and HM-

SPP? In [33], Goemans and Rothvoß proved that HM-BPP is in the class **P**. Perhaps we can build upon their work to prove that 3T-SPP and HM-SPP are also in the class **P**? Note that a proof for HM-SPP being in the class **P** would likely be more complex than the proof in [33] because HM-BPP involves only uni-dimensional items.

Currently, 3T-SPP is not known to be polynomially solvable: is $OPT(I) + 5/3$ the best possible performance guarantee for 3T-SPP? Our algorithm rounds up only the *tall* fractional rectangles of one configuration in segment S_2 and all rectangles of fractional height in segment S_3 so that they are of full height. Recall from Section 3.4 that we partition part S so that $\sum_{j=1}^{3} F_{j,c} > 4/3$ for each column $k_c \in S_3$. If we partition S differently so that $\sum_{j=1}^{3} F_{j,c} \gg 4/3$ for each column $k_c \in S_3$, our algorithm would then increase the height of S_3 by less than 5/3. Perhaps we can improve our algorithm if we find a different way of packing the rectangles that previously would have been in S_3? Similarly, if we only round up fractional rectangles in S_2 that are taller than the shortest *tall* fractional rectangle, perhaps we can find another way to pack the rectangles in S_2 that previously would have been rounded up?

In Chapter 3, we assumed the input of algorithm roundThreeTypes (that converts a fractional packing with 3 configurations into an integral packing) is an optimal fractional packing for 3T-SPP. As explained in linear program (2.1), a fractional packing for 3T-SPP obtained from a basic feasible solution for (2.1) has at most three rectangle types. However, suppose now that the input of algorithm roundThreeTypes is an optimal fractional packing for an instance of HM-SPP with three configurations but K rectangle types. Algorithm roundThreeTypes would still produce a solution of height at most $OPT(I) + 5/3$ because the increase in height does not depend on the number of rectangle types. For example, partitioning the fractional packing into parts S and S' as described in Section 3.1 creates a common part S' where rectangles of the same type line up between configurations. Thus, when our algorithm determines how many rectangles to pack in S', the increase in height does not depend on the number of rectangle types. Similarly, we partition part S into segments as described in Section 3.4 based on the fractional values of columns, where each configuration in a column has at most one rectangle type. Thus, when our algorithm determines how many rectangles to pack in each segment, the increase in height does not depend on the number of rectangle types.

In Section 2.1, we formulated the fractional 3T-SPP as linear program (2.1). As HM-SPP is a general version of 3T-SPP where the number of distinct rectangle types is K, we can express the fractional HM-SPP as a linear program obtained by changing the number of constraints in linear program (2.1) to K. Thus, an optimal basic feasible solution for this new linear program uses at most K configurations. If an optimal fractional packing for HM-SPP uses K configurations, we can convert this solution into an integral one by first using algorithm ROUNDTHREETYPES on 3 adjacent configurations to increase the height of the packing by at most 5/3. In the remaining configurations, we round up the rectangles with fractional height so that they are whole to increase the height of the packing by at most $K - 3$. Thus, we can generalize our algorithm to produce a solution for HM-SPP of height at most $OPT(I) + 5/3 + (K - 3) = OPT(I) + K - 4/3$. Note how this result is better than the bound $OPT(I) + K - 1$ in [7].

Although this thesis answers an open question in [7] by designing an algorithm that approximates HM-SPP with a better performance bound than $OPT(I) + K - 1$, if we assume HM-SPP is not in the class **P**, how closely can we can approximate the optimum solution of HM-SPP? In our algorithm we partition the non-common part of the fractional packing into segments and then determine how many rectangles to pack in each segment by applying different techniques for each segment as explained in Section 3.5. Perhaps we can use some of these techniques to design better approximation algorithms for HM-SPP and other special cases of HM-SPP? For example, Bloch-Hansen [37] recently designed a polynomial-time approximation algorithm for the special case of HM-SPP where $K = 4$. His algorithm for the four-type strip packing problem uses techniques similar to our own algorithm and produces a solution of height at most $OPT(I) + 5/2$. We can also generalize the algorithm in [37] similar to how we generalize our own algorithm to produce a solution for HM-SPP of height at most $OPT(I)+K-3/2$; note how this result is better than our bound of $OPT(I) + K - 4/3$. Perhaps designing an algorithm for the five-type strip packing problem and beyond could better approximate the optimum solution of HM-SPP?

Another open question is whether we can improve the running time of the approximation algorithm for 3T-SPP. In Theorem 4.2.8, we state that solving the linear program (2.1) dominates the worst-case runtime of our algorithm. However, the underlying algorithm that dominates the runtime of our algorithm is the Karmarkar and Karp algorithm [36], which has not seen improvement for many decades. In [38], Rothvoß designed an approximation algorithm that produces a solution closer to the optimum solution than the Karmarkar and Karp algorithm, but this algorithm has a higher running time.

To the best of our knowledge, no work exists that have applied the high multiplicity concept to other variants of the strip packing problem. For example, consider the multiple strip packing problem (M-SPP) where the objective is to pack rectangles into a constant number M of strips. We can represent an instance of the high multiplicity version of M-SPP (HMM-SPP) as an instance of HM-SPP with the constraint that rectangles are packed into M strips. Note that we cannot formulate the fractional HMM-SPP as linear program (2.1) because this linear program assumes rectangles are packed into only one strip. However, Bougeret et al. [27] formulated a linear program for the fractional M-SPP and then designed an algorithm that solves this linear program in polynomial time. According to Lemmas 5.2 and 5.3 in [27] the fractional solution to M-SPP uses at most $2M$ configurations which are then distributed to M strips so that each strip has at most 2 different configurations. If we can formulate a linear program for the fractional HMM-SPP and then solve this linear program in polynomial time, perhaps we can approximate the optimum solution of HMM-SPP by using a generalized version of our algorithm to convert a fractional packing into an integral one.

Bibliography

[1] B. Baker, E. Coffman, Jr., and R. Rivest, "Orthogonal packings in two dimensions," *SIAM Journal on Computing*, vol. 9, no. 4, pp. 846–855, 1980.

[2] S. Martello, M. Monaci, and D. Vigo, "An exact approach to the strip-packing problem," *INFORMS Journal on Computing*, vol. 15, no. 3, pp. 310–319, 2003.

[3] W. Fernandez de la Vega and G. S. Lueker, "Bin packing can be solved within $1 + \epsilon$ in linear time," *Combinatorica*, vol. 1, no. 4, pp. 349–355, 1981.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[5] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151–158, ACM, 1971.

[6] D. S. Hochbaum and R. Shamir, "Strongly polynomial algorithms for the high multiplicity scheduling problem," *Operations Research*, vol. 39, no. 4, pp. 648–653, 1991.

[7] D. Price, "High multiplicity strip packing," Master's thesis, The University of Western Ontario, London, ON, 2014.

[8] A. Ranjan, P. Khargonekar, and S. Sahni, "Offline first-fit decreasing height scheduling of power loads," *Journal of Scheduling*, vol. 20, no. 5, pp. 527–542, 2017.

[9] E. Coffman, Jr., M. Garey, D. Johnson, and R. Tarjan, “Performance bounds for level-oriented two-dimensional packing algorithms,” *SIAM Journal on Computing*, vol. 9, no. 4, pp. 808–826, 1980.

[10] D. D. Sleator, “A 2.5 times optimal algorithm for packing in two dimensions,” *Information Processing Letters*, vol. 10, no. 1, pp. 37–40, 1980.

[11] I. Schiermeyer, “Reverse-fit: A 2-optimal algorithm for packing rectangles,” in *Algorithms — ESA ’94* (J. van Leeuwen, ed.), vol. 855 of *LNCS*, pp. 290–299, Springer, 1994.

[12] A. Steinberg, “A strip-packing algorithm with absolute performance bound 2,” *SIAM Journal on Computing*, vol. 26, no. 2, pp. 401–409, 1997.

[13] R. Harren and R. van Stee, “Improved absolute approximation ratios for two-dimensional packing problems,” in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques* (I. Dinur, K. Jansen, J. Naor, and J. Rolim, eds.), vol. 5687 of *LNCS*, pp. 177–189, Springer, 2009.

[14] R. Harren, K. Jansen, L. Prädel, and R. van Stee, “A $(5/3+\epsilon)$-approximation for strip packing,” *Computational Geometry*, vol. 47, no. 2, Part B, pp. 248–267, 2014.

[15] I. Golan, “Performance bounds for orthogonal oriented two-dimensional packing algorithms,” *SIAM Journal on Computing*, vol. 10, no. 3, pp. 571–582, 1981.

[16] B. S. Baker, D. J. Brown, and H. P. Katseff, “A 5/4 algorithm for two-dimensional packing,” *Journal of Algorithms*, vol. 2, no. 4, pp. 348–368, 1981.

[17] C. Kenyon and E. Rémila, “A near-optimal solution to a two-dimensional cutting stock problem,” *Mathematics of Operations Research*, vol. 25, no. 4, pp. 645–656, 2000.

[18] K. Jansen and R. Solis-Oba, “Rectangle packing with one-dimensional resource augmentation,” *Discrete Optimization*, vol. 6, no. 3, pp. 310–323, 2009.

[19] M. Sviridenko, “A note on the Kenyon–Remila strip-packing algorithm,” *Information Processing Letters*, vol. 112, no. 1–2, pp. 10–12, 2012.

[20] K. Jansen and R. Thöle, “Approximation algorithms for scheduling parallel jobs,” *SIAM Journal on Computing*, vol. 39, no. 8, pp. 3571–3615, 2010.

[21] G. Nadiradze and A. Wiese, “On approximating strip packing with a better ratio than 3/2,” in *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1491–1510, SIAM, Dec. 2015.

[22] W. Gálvez, F. Grandoni, S. Ingala, and A. Khan, “Improved Pseudo-Polynomial-Time Approximation for Strip Packing,” in *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2016)* (A. Lal, S. Akshay, S. Saurabh, and S. Sen, eds.), vol. 65 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 9:1–9:14, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

[23] K. Jansen and M. Rau, “Improved approximation for two dimensional strip packing with polynomial bounded width,” in *WALCOM: Algorithms and Computation* (S.-H. Poon, M. S. Rahman, and H.-C. Yen, eds.), vol. 10167 of *LNCS*, pp. 409–420, Springer, 2017.

[24] K. Jansen and M. Rau, “Closing the gap for pseudo-polynomial strip packing,” *CoRR*, vol. abs/1712.04922, 2017.

[25] U. Schwiegelshohn, A. Tchernykh, and R. Yahyapour, “Online scheduling in grids,” in *2008 IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–10, IEEE, Apr. 2008.

[26] S. Zhuk, “Approximate algorithms to pack rectangles into several strips,” *Discrete Mathematics and Applications*, vol. 16, no. 1, pp. 73–85, 2006.

[27] M. Bougeret, P. F. Dutot, K. Jansen, C. Otte, and D. Trystram, “Approximation algorithms for multiple strip packing and scheduling parallel jobs in platforms,” *Discrete Mathematics, Algorithms and Applications*, vol. 3, no. 4, pp. 553–586, 2011.

[28] D. Ye, X. Han, and G. Zhang, “Online multiple-strip packing,” *Theoretical Computer Science*, vol. 412, no. 3, pp. 233–239, 2011.

[29] S. T. McCormick, S. R. Smallwood, and F. C. R. Spieksma, “A polynomial algorithm for multiprocessor scheduling with two job lengths,” *Mathematics of Operations Research*, vol. 26, no. 1, pp. 31–49, 2001.

[30] C. Filippi and A. Agnetis, “An asymptotically exact algorithm for the high-multiplicity bin packing problem,” *Mathematical Programming*, vol. 104, no. 1, pp. 21–37, 2005.

[31] C. Filippi, “On the bin packing problem with a fixed number of object weights,” *European Journal of Operational Research*, vol. 181, no. 1, pp. 117–126, 2007.

[32] K. Jansen and R. Solis-Oba, “An $OPT + 1$ algorithm for the cutting stock problem with constant number of object lengths,” in *Integer Programming and Combinatorial Optimization* (F. Eisenbrand and F. B. Shepherd, eds.), vol. 6080 of *LNCS*, pp. 438–449, Springer, 2010.

[33] M. X. 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*, pp. 830–839, SIAM, 2014.

[34] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[35] M. Grötschel, L. Lovász, and A. Schrijver, “The ellipsoid method and its consequences in combinatorial optimization,” *Combinatorica*, vol. 1, no. 2, pp. 169–197, 1981.

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

[37] A. D. Bloch-Hansen, "High multiplicity strip packing," Master's thesis, The University of Western Ontario, London, ON, 2019.

[38] T. Rothvoß, "Approximating bin packing within $O(\log OPT \cdot \log\log OPT)$ bins," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pp. 20–29, IEEE, Oct. 2013.

Curriculum Vitae

Name:	Andy Yu
Post-Secondary Education and Degrees:	The University of Western Ontario London, ON 2012–2016 B.Sc.
	The University of Western Ontario London, ON 2016–2019 M.Sc.
Related Work Experience:	Teaching Assistant The University of Western Ontario 2016–2017