Abstract. In the strip packing problem, a given set of axis-aligned rectangles must be packed into a fixed-width strip, and the goal is to minimize the height of the strip. In this paper, we examine a variant in which each rectangle may be cut vertically into multiple slices and the slices may be packed into the strip as individual pieces. Our results are: (1) analysis of the approximation ratio of several simple heuristics based on first fit and shelf techniques; (2) a simple polynomial algorithm with approximation factor $\frac{5}{3}$ for the case where each rectangle can only be sliced once; (3) a fully polynomial time approximation scheme (FPTAS).

1 Introduction

In the two-dimensional strip packing problem (abbreviated 2SP), a set of axis-aligned rectangles of specified dimensions must be packed, without rotation, into a rectangular strip of fixed width, with the goal of minimizing the height of the strip. The 2SP problem is very well-studied [17], and generalizes the bin packing problem, which is equivalent to the case in which all rectangles have unit height.

In this paper, we study a variant called two-dimensional strip packing with slicing (hereafter 2SP-S). In 2SP-S, we are given the freedom to cut each rectangle vertically into multiple slices, which may be packed into the strip as individual rectangles. The ability to slice is motivated by an application in electricity allocation (see Section 1.2 for details) where the height of a rectangle represents power consumption of a household appliance and the width represents a request for time, which need not be allocated consecutively. In this application, we must obey an additional stacking constraint requiring that no vertical line may intersect two slices from the same rectangle. The version of 2SP-S with the stacking constraint is denoted by 2SP-SSC. Many of our results in this paper hold for both 2SP-S and 2SP-SSC; we shall note situations in which there are differences.

The freedom to slice rectangles can be highly beneficial in strip packing. The required height can sometimes be reduced by a factor of $2 - \varepsilon$, and, rather surprisingly, the height can even be reduced in the case of 2SP-SSC for unit-width rectangles in an integer-width strip. Slicing also makes a difference in the
complexity of the problem. Standard 2SP is strongly NP-complete, and the best approximation factor achievable by a polynomial time algorithm (assuming P ≠ NP) is between 3/2 and 5/3 + ε [10]. However, we show that 2SP-S and 2SP-SSC both admit an FPTAS, placing them among the “easiest” NP-hard problems.

Our FPTAS is similar to that developed in the classic work of Karmarkar and Karp concerning the bin-packing problem [14]. As in their paper, we formulate our problem as a linear program and solve its dual program using a technique of Grötschel, Lovász, and Schrijver involving the ellipsoid method [9]. Of course, the running time of the ellipsoid algorithm likely prevents this from being useful in practice; however, we also outline how the column generation technique of Gilmore and Gomory [7] can be applied to solve this problem using the simplex method instead, yielding a (1 + ε)-approximation that, despite not being guaranteed to terminate in polynomial time, may be practical to implement.

An additional problem with our FPTAS is that it may yield solutions in which rectangles are sliced into many very small slivers. Such highly fragmented solutions may be poor in practice, so we develop simpler algorithms that limit the number of times a rectangle may be sliced. We obtain a polynomial algorithm that, using at most one cut per rectangle, produces a solution whose height is at most 5/3 times the optimal height when arbitrary slicing is permitted. This algorithm, unrelated to the (5/3 + ε)-approximation of [10], invokes simple subroutines based on first fit and shelf heuristics. We also establish some upper and lower bounds on the performance ratios of these heuristics themselves.

Our paper is organized as follows. The current section contains further background information, and Section 2 contains preliminary results. The First Fit and Shelf algorithms are in Section 3. Section 4 contains the FPTAS, Section 5 develops practical algorithms, and Section 6 contains a conclusion.

1.1 Background

Strip packing generalizes bin packing, so it is simple to show, via a reduction from the Partition problem [6], that 2SP admits no (3/2 − ε)-approximation for any ε > 0 unless P=NP. The current best approximation algorithm for 2SP has an approximation factor of 5/3 + ε for any ε > 0 [10], and was achieved after a long sequence of successive improvements [1, 3, 18, 20, 21]. 2SP also admits a polynomial algorithm yielding a solution of height at most (1 + ε)OPT(I) + h_{max} on an instance I, where h_{max} is the height of the tallest rectangle in I [12]. Such an algorithm behaves effectively like a PTAS when OPT(I) ≫ h_{max}.

Strip packing is sometimes encountered in problems related to scheduling tasks on processors [4, 19, 5, 2]. However, to the best of our knowledge, the exact problems 2SP-S and 2SP-SSC are new. Slicing has been studied for a variant in which the width of each rectangle represents a demand for a number of concurrently running processors [2]. However, this problem differs substantially from 2SP-S because the slices must have integer widths and must be horizontally aligned due to concurrency, and results for it do not carry over.

Several other variants of the strip packing problem have previously been studied, including online versions [11], versions where rotation is allowed [13], and
versions having multiple available strips [22]. A general survey of two-dimensional rectangle packing problems is given by Lodi et al. [17].

1.2 Applications to electricity allocation

The conventional approach to generating and distributing electricity relies on sizing infrastructure to support the peak load, when demand for electricity is highest. However, this peak is rarely reached, so much of the expensive infrastructure is idle most of the time. For example, in 2009, 15% of the generation capacity in Massachusetts was used less than 88 hours per year [8]. Reducing the infrastructure size is not practical since unsupported demand can cause blackouts. Therefore, there is considerable benefit to reducing the peak load itself.

Peak load occurs when many consumers use heavy appliances simultaneously. However, there is often flexibility in scheduling the use of particular appliances. In the future smart grid, it is anticipated that a substation would be able to obtain daily “demand schedules” for appliance use from the houses in its local area, and then automatically re-schedule appliance use to minimize peak load [15]. The demand schedule can be modelled as a set of rectangles, one for each appliance, with power consumption as height, and desired running time as width. Assuming demands can be arbitrarily scheduled, and appliances can be paused and restarted, this is an instance of 2SP-SSC. Slicing represents pausing and restarting an appliance, and the stacking constraint is essential.

2 Preliminaries

The input to a 2SP-S or 2SP-SSC instance consists of a number \( W \)—the width of the strip—and a set of rectangles \( r_1, r_2, \ldots, r_n \), where \( r_i \) has width \( w_i \) and height \( h_i \). The strip is bounded by two vertical sides at \( x = 0 \) and \( x = W \), and a “base” at \( y = 0 \). We assume that \( w_i \leq W \) for all \( i \) so that a solution always exists. Denote the maximum rectangle height by \( h_{\text{max}} = \max_{i=1}^{n} h_i \).

A solution to 2SP-S consists of a partition of each rectangle \( r_i \) into vertical “slices”, and a packing of the slices into the strip. We denote the slices of \( r_i \) by \( r_{i,1}, r_{i,2}, \ldots, r_{i,\ell_i} \), where \( \ell_i \geq 1 \). Slice \( r_{i,j} \) has height \( h_i \) and width \( w_{i,j} \) with \( \sum_{j=1}^{\ell_i} w_{i,j} = w_i \). In a feasible packing, each slice must be assigned a position inside the strip so that the interiors of the slices are pairwise disjoint. Slices may not be rotated. In 2SP-SSC, the additional stacking constraint demands that no vertical line intersects the interior of two slices of \( r_i \), for any \( i \in \{1 \ldots n\} \).

The height of a solution, denoted by \( H \), is the minimum \( y \)-coordinate above which the strip is empty. The objective is to find a solution that minimizes the value of \( H \). The floor of a solution, denoted by \( F \), is the maximum \( y \)-coordinate below which the strip is completely filled. Figure 1 provides an example.

If \( \mathcal{A} \) is an algorithm for strip packing with slicing, we use \( H_{\mathcal{A}} \) and \( F_{\mathcal{A}} \) to denote the height and floor of a solution produced by algorithm \( \mathcal{A} \). We use \( H_{\text{OPT}} \) to denote the height of an optimum solution. Algorithm \( \mathcal{A} \) is called a \( c \)-approximation algorithm if \( H_{\mathcal{A}} \leq c \cdot H_{\text{OPT}} \) for all instances.
**NP-hardness.** We observe that 2SP-S and 2SP-SSC are both weakly NP-hard, even when all rectangles have width 1 and $W = 2$. Complete proofs are provided in Appendix A. For 2SP-SSC, it suffices to perform a straightforward reduction from the Partition problem, but 2SP-S needs a more sophisticated reduction.

**Bounds on $H_{\text{OPT}}$.** The following lower bounds are immediate, but will be useful later.

Lemma 1.

1. In both 2SP-SSC and 2SP-S, $H_{\text{OPT}} \geq A/W \geq F$, where $A = \sum_{i=1}^{n} h_i \cdot w_i$ is the total area of all rectangles, and $F$ is the floor achieved by any packing.
2. In both 2SP-SSC and 2SP-S, $H_{\text{OPT}} \geq h_{\text{max}}$.
3. In 2SP-SSC, for any $I \subset \{1, \ldots, n\}$ with $\sum_{i \in I} w_i > W$, we have $H_{\text{OPT}} \geq h_{i_1} + h_{i_2}$, where $h_{i_1}$ and $h_{i_2}$ are the two smallest heights among items in $I$.

**Proof.** (1) follows from considering the area of the $H_{\text{OPT}} \times W$ and the $F \times W$ rectangles. (2) is obvious. For (3), we observe that some column in the solution must contain (slices of) two of the items in $I$, and cannot contain two slices of the same item by the stacking constraint. The result follows.

**Slicing Helps.** It is clear that slicing can reduce the height of a strip packing by a factor of 2 asymptotically. For example, take $2k$ rectangles of width $1/2 + 1/2k$ and height 1 in a strip of width $W = 1$. The optimum strip packing without slicing requires height $2k$, but slicing reduces the height to $k + 1$, even with the stacking constraint, and even with just one cut per rectangle.

What is more surprising is that slicing makes a difference even for unit-width rectangles in an integer-width strip, even when the stacking constraint is present. In this situation, basic algorithms such as those of Section 3 would not slice. The following lemma is proved in Appendix B:

**Lemma 2.** There exists an instance of 2SP-SSC involving 18 unit-width rectangles in a strip of width 6 such that the optimum solution without slicing is a factor of $105/104$ worse than the optimum solution with slicing.

We also show in Appendix C that there is an instance of 2SP-SSC with $n$ rectangles such that in any optimum solution, one of the rectangles must be sliced $\Omega(n/\log n)$ times.

### 3 Basic Algorithms

In this section, we discuss simple algorithms for 2SP-S and 2SP-SSC based on first fit and shelf heuristics. We show that both algorithms achieve an approximation factor of 2. This is in contrast with the standard strip packing problem, where 2-approximation algorithms are difficult to obtain [18, 21].
3.1 First Fit Algorithm

Given an ordered list of rectangles \( r_1, r_2, \ldots, r_n \), the First Fit algorithm works as follows: Processing the rectangles in order, repeatedly find the lowest point in the current solution where a slice of \( r_i \) can be placed, and place the widest possible slice of \( r_i \) there, breaking ties arbitrarily. Repeat with the remainder of \( r_i \), and continue until all rectangles have been processed. In the case of 2SP-SSC, the stacking constraint must be respected when placing slices. Figure 1 illustrates an execution of the First Fit algorithm on a 2SP-SSC instance.

By induction, we can easily show that the First Fit algorithm slices rectangle \( r_i \) into at most \( i \) pieces and leaves at most \( i + 1 \) distinct slices exposed after placing \( r_i \). The First Fit algorithm can be implemented to run in time \( O(n^2) \) by keeping a list of the slices touching the top boundary in sorted \( y \)-order.

We use \( H_{FF} \) and \( F_{FF} \) to denote the height and the floor of a solution produced by the First Fit algorithm. Figure 2 shows that First Fit sometimes uses a height that is almost twice the optimum, regardless of whether the stacking constraint is present. On the other hand, we show that it is never worse than that, i.e., the approximation ratio for First Fit is at most 2 for both 2SP-S and 2SP-SSC.

The key step in proving this is the following structural result:

**Lemma 3.** For both 2SP-S and 2SP-SSC instances, \( H_{FF} - F_{FF} \leq h_{max} \).

**Proof.** This can be proven rather straightforwardly by induction on the number of rectangles. A full proof is given in Appendix D.

**Lemma 4.** \( H_{FF} \leq A/W + h_{max} \leq H_{OPT} + h_{max} \leq 2H_{OPT} \).

**Proof.** We have \( H_{FF} = (H_{FF} - F_{FF}) + F_{FF} \leq h_{max} + F_{FF} \), by Lemma 3. Now \( F_{FF} \leq A/W \leq H_{OPT} \) by Lemma 1, and \( h_{max} \leq H_{OPT} \), which proves the claim.

Observe that the bad example in Figure 2 can be avoided by applying the First Fit algorithm after rearranging the rectangles in a better order, such as in decreasing order of height. We call the resulting algorithm \textit{FF-decreasing}. We provide a lower bound on its performance, with a proof in Appendix E:

**Lemma 5.** For both 2SP-S and 2SP-SSC, FF-decreasing achieves a factor no better than \( \frac{4}{3} \), even if all rectangles have the same width and no slicing occurs.
3.2 Shelf Algorithm

We now describe the Shelf algorithm for 2SP-S, which is even simpler than the First Fit algorithm. While it may seem naive, it achieves a 2-approximation just like the First Fit algorithm, can be implemented in linear time after sorting the input, automatically respects the stacking constraint and is thus valid for 2SP-SSC, and uses only one cut per rectangle.

Given a set of rectangles, the Shelf algorithm for strip packing with slicing works as follows. Sort the rectangles by decreasing height so that \( h_1 \geq h_2 \geq \ldots \geq h_n \). Pack the rectangles in this order on “shelves”. The first shelf is the floor of the strip. Place rectangles on the current shelf from left to right. When we reach a rectangle \( r_i \) that is too wide for the remaining space, we pack the widest possible slice of \( r_i \). The rest of \( r_i \) goes back in the list of remaining rectangles. Then we place a horizontal line across the strip to form a new shelf at the current maximum height of the packing, and continue on the new shelf with the remaining rectangles. See Figure 3. Note that the stacking constraint is automatically satisfied, because if a rectangle is sliced, then one piece is placed at the right side of the strip and the other piece is placed at the left side of the strip, and they cannot overlap since no rectangle is wider than the strip.

The Shelf algorithm takes linear time, not counting the time to sort. Note that each rectangle is sliced at most once.
We use $H_S$ to denote the height of a solution produced by the Shelf algorithm. The example from Figure 2 shows that the Shelf algorithm does not perform better than a factor of 2 in general. However, it does yield a 2-approximation for both 2SP-S and 2SP-SSC:

**Lemma 6.** $H_S \leq A/W + h_{\text{max}} \leq H_{\text{OPT}} + h_{\text{max}} \leq 2H_{\text{OPT}}$.

**Proof.** Let $E$ be the empty space below height $H_S$ in the solution found by the Shelf algorithm. Then $W \cdot H_S = A + E$, so $H_S = A/W + E/W$. We know that $A/W \leq H_{\text{OPT}}$, so it remains to analyze $E$. Observe that $E \leq \sum_{i=1}^{n} (h_i - h_{i+1})W = h_1W$ (where we define $h_{n+1}$ to be 0). Therefore $E/W \leq h_1 = h_{\text{max}}$.

### 4 Approximation Schemes

In this section, we give a fully polynomial time approximation scheme (FPTAS) for 2SP-S and 2SP-SSC:

**Theorem 1.** For any $\varepsilon > 0$, there exist $(1 + \varepsilon)$-approximation algorithms for 2SP-S and 2SP-SSC, assuming all inputs are rational. Moreover, these algorithms run in time polynomial in the input size and $1/\varepsilon$.

Our strategy is similar to that employed in the classic work of Karmarkar and Karp concerning the bin-packing problem [14]. Additional background can be found in a standard text such as [16]. The main steps are as follows:

1. Guess the height $H_{\text{OPT}}$ of an optimal solution to the problem (using, for example, binary search).
2. Round the height of each rectangle down to the nearest multiple of some small value (dependent on $\varepsilon$, $n$, and our guess for $H_{\text{OPT}}$).
3. Formulate the rounded problem as a linear program, with one variable for each possible arrangement of rectangles in a vertical slice of the packing.
4. Solve this linear problem exactly by solving its dual program with the ellipsoid method, and reconstruct a packing for the original problem.

As mentioned before, a more practical running time can be achieved by replacing the ellipsoid algorithm with the simplex algorithm, using the column generation technique of Gilmore and Gomory [7].

In the remainder of this section, we prove Theorem 1 for the case of 2SP-SSC. In Appendix F, we outline the changes necessary to make the algorithm work for 2SP-S. The main practical difference is that in the case of 2SP-S, a finer rounding scheme is necessary, resulting in a running time blow up of $O(\frac{1}{\varepsilon})$.

**Step 1: Reducing the general problem to a decision version**

Given a guess $H_{\text{GUESS}}$ for the optimal height $H_{\text{OPT}}$, the main algorithm that we describe in steps 2 through 4 is capable of establishing one of the following:

- (YES) There is a solution of value at most $H_{\text{GUESS}}(1 + \frac{\varepsilon}{2})$.
- (NO) There is no solution of value less than or equal to $H_{\text{GUESS}}$. 
Since the optimal height $H_{\text{OPT}}$ is at most $\sum_{i=1}^{n} h_i$ and at least $\frac{1}{n} \sum_{i=1}^{n} h_i$, it is possible, via binary search, to establish $H_{\text{OPT}}$ to within a multiplicative factor of $1 + \varepsilon$ using only $O(\log(\frac{1}{\varepsilon}))$ queries to our main algorithm. This then yields a $(1 + \varepsilon)$-approximation for the problem. The remaining steps describe how such queries can be answered constructively in polynomial time.

**Step 2: Rounding the heights**

Our linear programming method will require us to solve an instance of the knapsack problem to obtain a solution to the separation problem for the dual linear program. To render these knapsack instances tractable, we must round the heights of the rectangles in the input to multiples of an appropriate value $h_0$.

For 2SP-S, given a value of $H_{\text{GUESS}}$, we round all of the heights of the input rectangles down to the nearest multiple of $h_0 = \frac{\varepsilon}{2n} H_{\text{GUESS}}$. We will subsequently solve the resulting instance exactly using linear programming, obtaining a solution $S$ of height $H^*$. Since we rounded all of the heights down, it is immediate that $H^* \leq H_{\text{OPT}}$, and thus if $H^* \geq H_{\text{GUESS}}$, then there is no solution of value less than or equal to $H_{\text{GUESS}}$. Additionally, the stacking constraint implies that each vertical line passes through the interior of at most $n$ rectangles in $S$, so after undoing the rounding, the height of $S$ increases by at most $\frac{\varepsilon}{2} H_{\text{GUESS}}$. From this, it follows that if $H^* \leq H_{\text{GUESS}}$, then there exists a solution to the original (unrounded) problem of value at most $H_{\text{GUESS}} (1 + \frac{\varepsilon}{2})$. Consequently, we can answer (YES) or (NO) depending on whether or not $H^* \leq H_{\text{GUESS}}$.

**Step 3: Linear programming formulation**

We now show how to exactly formulate the rounded problem as a linear program. After rounding, each rectangle’s height is a multiple of $h_0$, and we attempt to pack all rectangles into a strip of height at most $H_{\text{GUESS}}$.

Given a feasible packing of height at most $H_{\text{GUESS}}$, we shall think of a pattern as the unordered set of rectangles intersected by some vertical line drawn through the packing. Formally, we define a pattern to be any subset of $\{r_1, \ldots, r_n\}$ whose total height is at most $H_{\text{GUESS}}$, and let $\mathcal{P}$ denote the set of all patterns. We observe that if arbitrary vertical slicing is permitted, then a solution to the strip packing problem can be exhibited by specifying, for each pattern $P \in \mathcal{P}$, the total width of pattern $P$ used in the arrangement—any reordering of the patterns themselves or the slices within the patterns preserves feasibility of the solution. This idea forms the motivation for our linear programming formulation.

For each pattern $P$, we define the variable $x_P$ to represent the total width of pattern $P$ used in a solution. It follows that determining the minimum strip width required to pack all of the rectangles into a strip of height $H_{\text{GUESS}}$ is equivalent to solving the following linear program:

\[
\begin{align*}
\text{minimize:} & \quad \sum_{P \in \mathcal{P}} x_P \\
\text{subject to:} & \quad \sum_{P \in \mathcal{P} : r_i \in P} x_P \geq w_i \quad \text{for all } 1 \leq i \leq n \\
& \quad x_P \geq 0 \text{ for all } P \in \mathcal{P}
\end{align*}
\]  

(LP)
It is immediate that upon solving this exactly, we may answer (YES) if and only if the optimal objective value \( W^* \) is at most \( W \). We note that no constraints involving heights are required, as these are accounted for when defining patterns.

**Step 4: Solving the linear program**

We provide a polynomial algorithm for finding the optimal objective value \( W^* \) to our linear program. To do this, we examine the following dual of (LP):

\[
\begin{align*}
\text{maximize} & \quad \sum_{i=1}^{n} w_i y_i \\
\text{subject to:} & \quad \sum_{i: r_i \in P} y_i \leq 1 \text{ for all } P \in \mathcal{P} \\
& \quad y_i \geq 0 \text{ for all } 1 \leq i \leq n \\
\end{align*}
\]

\((LP^*)\)

Despite this linear program having exponentially many constraints, we can tackle it using the ellipsoid algorithm or column generation method, provided we can solve the corresponding separation problem. Moreover, since we assumed that the widths are rational, we can find the exact optimal objective value of \((LP^*)\) in polynomial time if we can solve the separation problem in polynomial time.

The separation problem for this linear program asks us the following: Given values of \( y_i \), either find a pattern \( P \) such that \( \sum_{i: r_i \in P} y_i > 1 \), or determine that no such pattern exists. If we regard each rectangle as having height \( h_i \) and value \( y_i \), then this is essentially asking us to determine if there is any set of rectangles of total height less than \( H_{\text{GUESS}} \) having total value greater than 1, and to return such a pattern if one exists. This can be answered by solving a knapsack instance having weight-value pairs \((h_i, y_i)\) and maximum weight \( H_{\text{GUESS}} \). Since each height in the rounded problem is a multiple of \( h_0 \) and \( H_{\text{GUESS}} = \frac{2n}{\varepsilon} h_0 \), this can be done in \( O\left(\frac{n^2}{\varepsilon} \right) \) time using standard dynamic programming methods.

As a final note, we observe that it is possible to reconstruct an optimum solution to (LP) while solving \((LP^*)\) using this technique (see [14] for details). Consequently, we can not only approximate the optimum height of a packing, but can in fact return a packing having that height. Moreover, since a basic solution to (LP) is obtained, there are at most \( n \) patterns \( P \) for which the primal variables \( x_P \) are non-zero in the solution, implying that our algorithm returns a solution in which each rectangle is cut at most \( n - 1 \) times.

### 4.1 Reducing the number of cuts

We also observe that in the solution produced by the PTAS, the number of cuts per rectangle can be further reduced to a constant that depends only on \( \varepsilon \). More precisely, we can show that any feasible solution can be modified so that each rectangle is cut at most \((1/\varepsilon)^{O(1/\varepsilon)} \) times, without increasing the height by more than a factor of \( 1 + O(\varepsilon) \). Details are provided in Appendix G.
Although the approximation scheme from the previous section may be practical if the simplex method is used, it is likely unsuitable for resource allocation applications in which a lot of slicing is undesirable. Our goal in this section is to develop practical algorithms that are simple to implement, run quickly, have decent approximation guarantees, and do not slice rectangles too often.

By judiciously splitting the problem into two parts and applying our basic first fit and shelf algorithms in each part, we obtain: (1) a practical polynomial-time algorithm with approximation factor 3/2; (2) a practical polynomial-time algorithm with approximation factor 5/3 that slices each rectangle at most once.

We note that these algorithms are only relevant if $h_{\text{max}} > \frac{1}{2} H_{\text{OPT}}$ (resp. $h_{\text{max}} > \frac{2}{3} H_{\text{OPT}}$) otherwise the first fit (resp. shelf) algorithms already give the results.

All the algorithms in this section will adopt the following approach:

- Step 1. Divide the strip into two parts: the left side and the right side.
- Step 2. Split each rectangle into a left piece and a right piece.
- Step 3. Apply one of the basic algorithms in Section 3 to pack the left (resp. right) pieces in the left (resp. right) side of the strip.

We now provide details of Steps 1 and 2, assuming that the value of $H_{\text{OPT}}$ is known. (We remove this assumption in Appendix H.5.) Let $t$ be a value to be chosen later.

**Step 1.** Assuming the rectangles have been sorted in decreasing order of heights, find the largest $k$ such that $w_1 + \cdots + w_k \leq W$. Find the largest $j \leq k$ such that $h_j \geq t$. (In case $h_{\text{max}} < t$, we define $j$ to be 0.) Call $r_1, \ldots, r_j$ the left floor rectangles (of heights $\geq t$) and $r_{j+1}, \ldots, r_k$ the right floor rectangles (of heights $< t$). We divide the strip into two parts, where the left side has width $w_1 + \cdots + w_j$. Define $\alpha$ to be $(w_1 + \cdots + w_j)/W$ so the left side has width $\alpha W$ and the right side has width $(1 - \alpha)W$. Note that we may have $\alpha = 0$ or $\alpha = 1$.

**Step 2.** We split each rectangle into left and right pieces, subject to the constraint that the width of the left (resp. right) piece is at most $\alpha W$ (resp. $(1 - \alpha)W$). Either piece is allowed to be empty. The splitting procedure is described below.

Assume we are given an initial splitting of the rectangles. Let $A_L$ (resp. $A_R$) denote the total area of all left (resp. right) pieces. In the following, shifting a rectangle rightward means enlarging the width of its right piece by $\delta$ and shrinking the width of its left piece by $\delta$ for some amount $\delta > 0$. Shifting a rectangle leftward is similarly defined.

We say that a left (resp. right) piece is full if it has width exactly $\alpha W$ (resp. $(1 - \alpha)W$). We say that a rectangle is shifted completely rightward if the left piece is empty or the right piece is full. Shifting a rectangle completely leftward is similarly defined (with the right piece empty or left piece full).

All left floor rectangles are shifted completely leftward and all right floor rectangles are shifted completely rightward. For the non-floor rectangles, we attempt to split so that $A_R$ is equal to $(1 - \alpha)H_{\text{OPT}}W$, using a greedy procedure:

5 Good Approximations for 2SP-SSC with Limited Cuts
Start by shifting all non-floor rectangles completely rightward. Let $A_L^0$ and $A^0_R$ be the initial value of $A_L$ and $A_R$.

- If $A^0_R \leq (1 - \alpha)H_{OPT}W$, then stop.
- Otherwise, for each non-floor rectangle from minimum to maximum height while $A_R > (1 - \alpha)H_{OPT}W$, decrease $A_R$ by shifting the rectangle leftward either completely or until $A_R = (1 - \alpha)H_{OPT}W$.

Observe that the above procedure ends with either $A_R = A^0_R$ or $A_R = (1 - \alpha)H_{OPT}W$, whichever is smaller. If not, then at the end, $A_R > (1 - \alpha)H_{OPT}W$ and all non-floor rectangles have been shifted completely leftward. So, at most a fraction $1 - \alpha$ of the area of each non-floor rectangle is on the right side. Considering the floor rectangles, at most a fraction $1 - \alpha$ of the area of the floor rectangles is on the right side, since the left floor rectangles have total area at least $t_0W$ and the right floor rectangles have total area at most $t(1 - \alpha)W$. Combining floor and non-floor areas gives $A_R \leq (1 - \alpha)A \leq (1 - \alpha)H_{OPT}W$: a contradiction. Observe that except for one critical rectangle, which we denote by $r_X$, all rectangles are either shifted completely leftward or completely rightward.

We develop the following three algorithms based on this approach (further details are in Appendix H).

1. By choosing $t = H_{OPT}/2$ and applying the First Fit algorithm on both sides, we obtain a 3/2-approximation for 2SP-SSC that runs in $O(n^2)$ time.
2. By choosing $t = 2H_{OPT}/3$ and applying the Shelf algorithm on both sides, we obtain a 5/3-approximation for 2SP-SSC that slices every rectangle at most three times and runs in time $O(n \log(nM))$, where $M$ is an upper bound on the integer heights of the rectangles.
3. By modifying the above algorithm and carefully rearranging the shelves, we obtain a 5/3-approximation for 2SP-SSC that slices every rectangle at most once and runs in time $O(n \log^2 n \log(nM)/ \log \log n)$.

6 Conclusions

Motivated by an application in electricity allocation, this paper explored variants of the strip packing problem in which rectangles could be sliced vertically. We provided simple 2-approximation algorithms, a PTAS of mostly theoretical interest, and practical approximation algorithms that slice rectangles only a few times. The main remaining open problem is to find practical algorithms with better approximation factors, and ideally find a simple PTAS for this problem.

References


A Proof that 2SP-S and 2SP-SSC are NP-Hard

For 2SP-SSC, we reduce from the Partition problem:

Given $n$ integers $a_1, \ldots, a_n$ that sum to $2B$, is there a subset $S$ of those integers with sum equal to $B$?

Such a problem can be encoded via a strip packing instance in which $W = 2$ and the $n$ rectangles have width 1 and height $a_i$ for $i = 1, \ldots, n$. We claim that an instance of 2SP-SSC with those inputs has optimum value $B$ if and only if a partition exists.

If the Partition instance has a solution $S$, then we can pack the strip with height $B$ in the obvious way: pack all rectangles whose heights are in $S$ atop each other in one column of width 1, and all remaining rectangles (whose heights also sum to $B$) in the other column of width 1.

For the other direction, suppose we have a strip packing of height at most $B$. Since the total area is $2B$, the packing then fills a $2 \times B$ rectangle entirely. Fix one $x$-coordinate $X$ that is not on the left or right boundary of any slice, and let $S$ be the heights of all the slices that are intersected by the vertical line at $X$. Then clearly the items in $S$ sum to $B$. Furthermore, no $a_i$ can appear twice in the list by the stacking constraint. Thus a packing of height $B$ yields a solution to the Partition instance, completing the proof.

For 2SP-S, more care is required, because the absence of the stacking constraint causes the above construction to fail. Our strategy of avoiding this problem is to effectively encode an NP-hard variation of the Partition problem that is carefully designed so that the corresponding strip packing instance generated does not admit any optimal solutions in which the stacking constraint is violated. In doing so, we establish a reduction to 2SP-S that effectively mimics the NP-hardness proof of the Partition problem, originally due to Karp.\(^1\) We reduce from the NP-hard Exact Cover problem [6]:

Given positive integers $n$, $t$, and $k$, and a family $\mathcal{S} = \{S_1, \ldots, S_n\}$ of $n$ subsets of the set $U = \{0, \ldots, t-1\}$, does there exist a subfamily $\mathcal{C} \subseteq \mathcal{S}$ of $k$ disjoint sets whose union is exactly $U$?

To encode an instance of Exact Cover as a 2SP-S instance, we set $W = 2$ and introduce $n + 2$ rectangles having unit width and the following heights:

- $h_i = (n + 1)^i + \sum_{j \in S_i} (n + 1)^j$, for all $1 \leq i \leq n$
- $h_{n+1} = k(n + 1)^t + \sum_{j=0}^{t-1} (n + 1)^j + (n + 1)^{t+1}$
- $h_{n+2} = \sum_{i=1}^{n} h_i + 2(n + 1)^{t+1} - h_{n+1}$

We claim that an instance of 2SP-S with these inputs has optimum value $B = \sum_{i=1}^{n} h_i + (n + 1)t + 1$ if and only if our Exact Cover instance admits an exact cover. The backward direction is straightforward: if an exact cover $C = \{S_{c_1}, \ldots, S_{c_k}\}$ exists, then it suffices to stack rectangles $\{r_{c_1}, \ldots, r_{c_k}\}$ and rectangle $r_{n+2}$ atop each other in one column of width 1, and the remaining rectangles in the remaining column of width 1. The total height of both sets of rectangles can easily be verified to equal $B$.

For the forward direction, we suppose we have a strip packing of height at most $B$. We observe that the total area of all input rectangles is $2B$, and thus the packing must fill a $2 \times B$ rectangle entirely. Again, we fix an $x$-coordinate $X$ that is not on the left or right boundary of any slice, and select $X$ so that a vertical line at $X$ intersects at least one slice of $h_{n+2}$. Such a vertical line cannot intersect two slices of $h_{n+2}$ or a slice of $h_{n+1}$ because we necessarily have $B < 2h_{n+2}$ and $B < h_{n+1} + h_{n+2}$ by our construction. So a vertical line must pass through $h_{n+2}$ and several other slices having total height $k(n+1) + \sum_{j=0}^{t-1}(n+1)^j$. One can then check that the $k(n+1)^t$ term in this total height necessitates that these other slices must consist of exactly $k$ slices from $\{r_1, \ldots, r_n\}$, and the $\sum_{j=0}^{t-1}(n+1)^j$ term in this total height necessitates that these other slices correspond to sets that form an exact cover of $U$. Verifying this completes the proof.

\section*{B Proof of Lemma 2}

\textit{Proof.} Here, we show that there exists an instance of 2SP-SSC involving 18 unit-width rectangles in a strip of width 6 such that the optimum solution without slicing is a factor of $\frac{105}{104}$ worse than the optimum solution with slicing.

Consider the following 18 rectangles $A \cup B \cup C$ that all have unit width, and whose height is as listed below:

\begin{align*}
A &= \{1, 2, 3, 4, 5, 6\} \\
B &= \{26, 27, 35, 37, 46, 47\} \\
C &= \{52, 53, 63, 66, 75, 76\}
\end{align*}

We aim to pack these into a strip of width 6. Figure 4 shows a packing where some of these rectangles use two slices and that achieves height 104, which is optimal since the floor equals the height. It also shows a packing of height 105 that does not slice rectangles.

We now claim that no packing of height 104 is possible if rectangles are not sliced. Assume to the contrary that there is such a packing. Note that in any column of the strip, we cannot have two rectangles from group $C$ (they sum to at least 105), so every column must contain one rectangle from $C$ by the pigeon-hole principle. We also cannot have one rectangle from $C$ and two from $B$ (they sum to at least 105), so every column must contain one rectangle from $C$ and one rectangle from $B$. But we also cannot have one rectangle from $C$ and one from $B$ and nothing else, for no two values in $B$ and $C$ sum to 104 (and we
cannot afford to have less than 104 in any column since there must not be any empty space.)

Therefore, any column in the packing must contain one rectangle each from A, B and C. We will now argue that this is not possible. For if no rectangles are split, then we can consider the strip as being split into 6 columns of unit width, and each of them contains three rectangles (one each from A, B and C) in its entirety, and the height of the three rectangles sum to 104. In other words, we can view the 18 rectangles as being split into 6 triplets whose heights sum to 104 each. We now show that this is not possible.

Consider first rectangle 75 (we will in the following identify the rectangle with its height.) The only way for it to sum to 104 with two others is \(75 + 27 + 2\) or \(75 + 26 + 3\). The only way for rectangle 76 to sum to 104 with two others is
76 + 26 + 2 or 76 + 27 + 1. But any rectangle can belong to only one column, so
this shows that we must have the triplets 75 + 26 + 3 and 76 + 27 + 1. In a similar
way arguing about 52 and 53 shows that we must have the triples 52 + 46 + 6
and 53 + 47 + 4. But now 63 cannot be in a triplet, for it would either have to
be 63 + 35 + 6 (but rectangle 6 is used already) or 63 + 37 + 4 (but rectangle 4
is used already.) So no split into 6 triplets that sum to 104 exists, and therefore
no strip packing without splitting of height 104 exists either.

C Many slices may be required

To motivate why algorithms that achieve few slices are interesting, even if they
achieve a worse approximation ratio, we prove here that sometimes many slices
are required in an optimal solution.

Lemma 7. There exists an instance of 2SP-SSC with \(n\) rectangles, such that in
any optimal solution, there exists a rectangle that gets sliced \(\Omega(n/\log n)\) times.

Proof. For \(i = 1, \ldots, m\), let \(h_i = 1/2^{i+1}\). We define an instance of 2SP-SSC
with strip width \(W = 2^m - 1\) and the following rectangles:

1. For \(i \in \{1, \ldots, m\}\), we define a long rectangle \(r_i\) with height \(h_i\) and width \(2^{m-1}\).
2. For every non-empty set \(S \subseteq \{1, \ldots, m\}\), we define a short rectangle \(r_S\) with
   height \(1 - \sum_{i \in S} h_i\) and width 1.

Thus the total number of rectangles in the instance is \(n = m + 2^m - 1\). We
claim that in any optimal solution of this instance, there is a long rectangle that
gets sliced at least \((2^m - 1)/m = \Omega(n/\log n)\) times.

We first note that an optimal solution has height 1 and can be constructed
by placing the short rectangles on the floor. Then for each short rectangle \(r_S\)
and each \(i \in S\), place a slice of width 1 of the long rectangle \(r_i\) on top of \(r_S\).
By definition of the height of \(r_S\), this gives height 1, and since long rectangles
have width \(2^{m-1}\) and occur in \(2^{m-1}\) subsets, this places all long rectangles. This
solution has height \(A/W\) and hence is optimal.

Now consider any optimal solution of height 1. Since the height of any short
rectangle is greater than 1/2, no two slices of short rectangles can appear in the
the same vertical column. There are \(2^m - 1\) small rectangles of unit width and
\(W = 2^m - 1\), so any vertical column contains a slice of a small rectangle. In
consequence there are at least \(2^m - 2\) x-coordinates \(X \neq 0, W\) where a slice of
a small rectangle \(S\) ends and one of a small rectangle \(T\) begins.

Since the heights \(h_i\) are distinct powers of 2, we have \(\sum_{i \in S} h_i \neq \sum_{i \in T} h_i\).
Thus, at least one slice of a long rectangle must begin or end at \(X\). Put differently,
one slice of a long rectangle must have a vertical end segment at \(X\). Since there
are at least \(2^m - 2\) such \(x\)-coordinates, the total number of vertical end segments
of all the long rectangle slices is at least \(2^m - 2\). Thus there exists a long rectangle
\(r_i\) with at least \((2^m - 2)/m\) vertical end segments. Since each slice has two vertical
end segments, \(r_i\) needs to be sliced at least \((2^m - 1)/m = \Omega(n/\log n)\) times.
D Proof of Lemma 3

Proof. Here, we prove Lemma 3, which states that $H_{FF} - F_{FF} \leq h_{\text{max}}$ for all 2SP-S and 2SP-SSC instances, where $H_{FF}$ and $F_{FF}$ denote the height and the floor of a solution produced by the First Fit algorithm.

Our proof is by induction on the number rectangles, with the base case of 0 rectangles. Suppose by induction that the result holds for $k - 1$ rectangles, and consider the addition of rectangle $r = r_k$ with height $h$. We will drop the subscript FF, and let $H$ and $F$ denote the height and floor before packing rectangle $r$, and $H'$ and $F'$ denote the height and floor after packing rectangle $r$. So we have $H - F \leq h_{\text{max}}$, $H' \geq H$, $F' \geq F$, and we wish to show that $H' - F' \leq h_{\text{max}}$.

If packing the slices of $r$ does not increase the height of the solution (i.e., $H' = H$) then the result holds. So suppose a slice of $r$ has its top boundary at y-coordinate $H'$. Let $b$ denote the y-coordinate of the bottom of this slice. If $F' \geq b$ we are done, because $H' - F' \leq H' - b = h \leq h_{\text{max}}$.

So we may suppose that $F' < b$. The First Fit algorithm packs slices of $r$ starting with the lowest possible position. Therefore, every position below $b$ must already have a slice of $r$. Thus $F' \geq F + h$. Now $H' - F' \leq H + h - F' \leq H + h - F - h = H - F \leq h_{\text{max}}$.

E Proof of Lemma 5

In this section, we give the proof of Lemma 5: There exists an example where FF-decreasing does not perform better than a factor of $\frac{4}{3}$, even if all rectangles have the same width and no slicing occurs.

Our proof is inspired by the work of Pinedo. He gave (on p.115) an example of a strip packing problem where the First-Fit Decreasing solution is a factor $\frac{5}{4}$ worse than the optimum. We generalize his example of 9 rectangles to $4M + 1$ rectangles and hence achieve an even worse ratio.

Let $M > 0$ be an integer. We define $4M + 1$ rectangles, all of unit width, with the following heights:

$$4M - 1, 4M - 2, 4M - 3, 4M - 4, \ldots, 3M + 1, 3M + 2, 3M + 3, 3M + 4, 2M + 1, 2M + 2, 2M + 3, 2M + 4, 2M + 5, \ldots, 2M, 2M.$$

In other words, every integral height between $2M + 1$ and $4M - 1$ occurs exactly twice, and height $2M$ occurs three times. We wish to pack these rectangles into a strip of width $W = 2M$. Figure 5 shows a packing of these rectangles that has height $6M = A/W$, which is optimal. Figure 6 shows the packing with First-Fit Decreasing, which has height $8M - 1$. Thus, for sufficiently large $M$ the ratio between the First-Fit Decreasing solution and the optimal solution is arbitrarily close to $\frac{4}{3}$.

---

F  Proof of Theorem 1 for 2SP-S

Here, we outline all the changes required to make the FPTAS presented in Section 4 work when the stacking constraint is absent. The basic idea of the algorithm follows the same four steps as that provided for 2SP-SSC in Section 4:

**Step 1: Reducing the general problem to a decision version**
No changes are necessary from what is provided in Section 4.

**Step 2: Rounding the heights**
Here, we require some additional work because if a naive approach is used, then the number of rectangles in a single column of the solution may be large, presenting potential problems when undoing the rounding. We define a rectangle to be short if its height is less than $\frac{\varepsilon}{4}H_{\text{GUESS}}$. We shall proceed by eliminating all short rectangles from consideration without rounding their heights, and then adding them back into the solution after solving the linear program. We note that by taking a solution to the resulting problem and simply stacking the short rectangles naively above the solution, we can reintroduce the short rectangles while increasing the height of the solution by at most $\frac{\varepsilon}{4}H_{\text{GUESS}}$ (since we are adding back at most $n$ short rectangles of height at most $\frac{\varepsilon}{4}H_{\text{GUESS}}$). Note that despite not being dependent on the stacking constraint, this nonetheless requires the assumption that no rectangle has width wider than $W$.  

Fig. 5. An instance and its optimal packing.
Fig. 6. The instance of Figure 5 packed with First-Fit decreasing, which is a factor of almost $\frac{4}{3}$ worse than the optimal packing.

After eliminating the short rectangles, we round all the remaining rectangles (each of height at least $\epsilon \frac{H_{\text{GUESS}}}{n}$) down to the nearest multiple of $h_0 = \epsilon \frac{\epsilon^2}{16n} H_{\text{GUESS}}$. We then solve the resulting instance exactly via linear programming, obtaining a solution $S$ to the subproblem with short rectangles removed. Again, let $H^*$ denote the height of $S$, and note that if $H^* > H_{\text{GUESS}}$, then there is again no solution of value less than or equal to $H_{\text{GUESS}}$ and we may answer (NO). Otherwise, we have $H^* \leq H_{\text{GUESS}}$ and thus no vertical line may pass through the interior of more than $\frac{4n}{\epsilon H_{\text{GUESS}}}$ slices in $S$ (since each slice has height $\frac{\epsilon^2}{16n} H_{\text{GUESS}}$). This implies that after undoing the rounding, the height of $S$ increases by at most $(\frac{4n}{\epsilon}) \left(\frac{\epsilon^2}{16n}\right) H_{\text{GUESS}} = \frac{\epsilon}{4} H_{\text{GUESS}}$. With the additional $\frac{\epsilon}{4} H_{\text{GUESS}}$ possibly added by reintroducing the short rectangles, the total height of the solution reconstructed from $S$ is at most $H_{\text{GUESS}}(1 + \frac{\epsilon}{4})$, and we may thus answer (YES) whenever $H^* \leq H_{\text{GUESS}}$. 

19
Step 3: Linear programming formulation

For 2SP-S, a similar formulation to that given in Section 4 will work; the only difference is that patterns must now consist of multisets of \( \{r_1, \ldots, r_n\} \) whose total height is at most \( H_{\text{GUESS}} \), and the constraints in (LP) must be replaced by those of the form

\[
\sum_{P : r_i \in P} k_{i,P} x_P = w_i \text{ for all } 1 \leq i \leq n
\]

where \( k_{i,P} \) is the number of times rectangle \( i \) appears in pattern \( P \). We note that \( k_{i,P} \) can be at most \( \frac{16n}{\varepsilon} \) if rounding is performed as described in step 2.

Step 4: Solving the linear program

The only important difference here is that the separation problem for the linear program is an instance of the knapsack problem in which items can be taken multiple times; nevertheless, the same dynamic programming approach suffices to obtain a solution in polynomial time.

Unfortunately, the finer rounding scheme used in Step 2 results in an increase in the running time of the solver. Since each rectangle’s height is rounded to a multiple of \( \frac{\varepsilon}{16n} H_{\text{GUESS}} \), there are \( O\left(\frac{n^2}{\varepsilon}\right) \) possible distinct heights in subproblems encountered while performing dynamic programming. Consequently, the running time for each call to the separation oracle is worsened from \( O\left(\frac{n^2}{\varepsilon}\right) \) to \( O\left(\frac{n^2}{\varepsilon^2}\right) \).

G Reducing the number of cuts

Here, we show that any feasible solution to a 2SP-S or 2SP-SSC instance can be modified so that each rectangle is cut at most \( (1/\varepsilon)^{O(1/\varepsilon)} \) times, without increasing the height by more than a factor of \( 1 + O(\varepsilon) \).

The approach involves a different rounding scheme to that used in Section 4. Let \( H \) be the height of the given solution. We define a rectangle as short if its height is less than \( \varepsilon H \). We round the height of each non-short rectangle up to a power of \( 1 + \varepsilon \) (i.e., a number of the form \( (1 + \varepsilon)^j \) for some integer \( j \)). As a result, there are now at most \( O\left(\log_{1+\varepsilon}(1/\varepsilon)\right) = O\left((1/\varepsilon) \log(1/\varepsilon)\right) \) possible height values for the non-short rectangles, and the modified solution has height at most \( (1 + \varepsilon)H \).

The pattern \( P \) at a vertical line \( \ell \) refers to the unordered set of non-short rectangles intersecting \( \ell \); this set has cardinality at most \( 1/\varepsilon \). The total number of different patterns is thus bounded by \( C_{\varepsilon} := \left[ O\left(1/\varepsilon\right) \log(1/\varepsilon) \right]^{1/\varepsilon} \leq \left(1/\varepsilon\right)^{O(1/\varepsilon)} \).

We first rearrange the solution so that vertical lines with the same pattern \( P \) appear consecutively in a sub-strip \( \sigma_P \). This rearrangement may increase the number of cuts per rectangle, but each non-short rectangle is cut at most \( C_{\varepsilon} \) times. Inside each sub-strip \( \sigma_P \), we move the short rectangles so that they fit in a region \( \sigma'_P \) of width \( x_P \) and height \( y_P \), while the non-short rectangles fit in a region of width \( x_P \) and height \( (1 + \varepsilon)H - y_P \). We finally re-pack the short rectangles in \( \sigma'_P \) by using the shelf algorithm from Section 3. Since the maximum height of the short rectangles is at most \( \varepsilon H \), this re-packing requires height at

20
most \( y_p + \varepsilon H \) by Lemma 6, and each short rectangle is cut at most once in \( \sigma'_p \).
Thus, the new packing has an overall height of at most \((1 + 2\varepsilon)H\), and each short rectangle is cut a total of at most \(2C_\varepsilon\) times.

**H Approximation Algorithms of Section 5**

We first observe that after applying the greedy procedure for shifting, with at most one exception, all rectangles are either shifted completely leftward or completely rightward by the procedure described in Section 5. The exception, if it exists, is called the **critical** rectangle and denoted by \( r_X \).

**H.1 Analysis Preliminaries**

Let \( H_L \) (resp. \( H_R \)) be the height of the left (resp. right) side in the solution produced by the algorithm. Let \( f \) denote the height of the tallest non-floor rectangle, i.e., \( f = w_{k+1} \). Let \( \ell \) denote the height of the tallest non-floor (non-empty) left piece. In the subsequent analyses, we use three more lower bounds on \( H_{\text{OPT}} \):

**Lemma 8.**

3. \( H_{\text{OPT}} \geq 2f \)
4. \( H_{\text{OPT}} \geq A_L^0 / (\alpha W) \)
5. If \( \alpha > 0 \), then \( H_{\text{OPT}} \geq t + \ell \)

**Proof.** (3) Apply Lemma 1(3) to rectangles 1, \ldots, \( k + 1 \).

(4) Let \( R_L^0 \) denote the set of left pieces (including the left floor rectangles \( r_1, \ldots, r_j \)) when all non-floor rectangles are shifted completely rightward. Consider an optimal solution \( S^* \) (with an unbounded number of cuts). Divide \( S^* \) into columns by making a vertical cut at every left/right boundary of a slice of a rectangle in \( S^* \) and rearrange the columns in \( S^* \) so that the rectangles \( r_1, \ldots, r_j \) appear on the left side of the strip of width \( \alpha W = w_1 + \cdots + w_j \). Then the left side of the strip in \( S^* \) must contain at least the pieces in \( R_L^0 \) and thus must contain a total area of at least \( A_L^0 \). It follows that \( A_L^0 \leq H_{\text{OPT}} \alpha W \), implying (4).

(5) We may assume that the rectangles \( r_1, \ldots, r_j \), which all have height \( \geq t \) and \( \geq \ell \), are vertically separated (i.e., no two slices are stacked) in the left side of the strip in \( S^* \), for otherwise \( H_{\text{OPT}} \geq t + \ell \) by Lemma 1(3). It suffices to show that some additional rectangle of height \( \geq \ell \) also appears on the left side of the strip in \( S^* \). Assume the contrary. Then the right side in \( S^* \) must contain all the right floor rectangles and all non-floor rectangles of height \( \geq \ell \) (in particular, all these rectangles must have width at most \((1 - \alpha)W\)). The total area of the rectangles on the right side in \( S^* \) is at most \((1 - \alpha)H_{\text{OPT}} W\), and this remains true even if all non-floor rectangles of height \( < \ell \) are shifted completely leftward. But then during the greedy procedure, by the time a rectangle of height \( \ell \) is considered, this rectangle would not be shifted leftward and would have an empty left piece: a contradiction.
H.2 3/2-Approximation with An Unbounded Number of Cuts

As a warm-up, we analyze a version of the algorithm, where we set \( t = \frac{H_{\text{OPT}}}{2} \) and in Step 3 we use the First Fit algorithm on both the left and right sides. The resulting algorithm uses \( O(n) \) cuts per rectangle, and runs in time \( O(n^2) \), assuming \( H_{\text{OPT}} \) is known.

We analyze the right and left sides separately, but skip an empty side. Let \( h_R \) be the maximum height of a right piece. Then \( h_R \leq \max\{t, f\} = \frac{1}{2} H_{\text{OPT}} \).

Also \( A_R \leq \min\{A_R^0, (1 - \alpha) H_{\text{OPT}} W\} \). By Lemma 4,

\[
H_R \leq \frac{A_R}{(1 - \alpha)W} + h_R \leq H_{\text{OPT}} + \frac{1}{2} H_{\text{OPT}} \leq \frac{3}{2} H_{\text{OPT}}.
\]

On the left side, after the floor rectangles have been placed, the left side is filled to level \( t \), the difference between the highest and lowest point is at most \( h_{\text{max}} - t \), and the remaining left pieces have maximum height \( \ell \). The First Fit algorithm does the same as if it started at level \( t \) with rectangles of maximum height \( \max\{\ell, h_{\text{max}} - t\} \). Also, \( A_L = A - A_R = \max\{A_L^0, A - (1 - \alpha) H_{\text{OPT}} W\} \leq \max\{A_L^0, H_{\text{OPT}} W - (1 - \alpha) H_{\text{OPT}} W\} \leq \alpha H_{\text{OPT}} W \). By Lemma 4,

\[
H_L \leq \frac{A_L}{\alpha W} + \max\{\ell, h_{\text{max}} - t\}
\leq H_{\text{OPT}} + (H_{\text{OPT}} - t) = \frac{3}{2} H_{\text{OPT}}.
\]

Thus, the algorithm achieves approximation factor at most 3/2. Although this is worse than the FPTAS results in Section 4, this algorithm is simple and fast.

H.3 5/3-Approximation with 3 Cuts

Next we analyze a version of the algorithm where we set \( t = 2H_{\text{OPT}}/3 \) and in Step 3 we use the Shelf algorithm on both sides. The resulting algorithm runs in linear time after the initial sort, assuming \( H_{\text{OPT}} \) is known. The algorithm uses at most 3 cuts per rectangle—at most 1 cut to divide into left and right pieces, and at most 1 more cut per piece due to the Shelf algorithm.

Using the same approach as in Section H.2, and applying Lemma 6, we have

\[
H_R \leq \frac{A_R}{(1 - \alpha)W} + h_R \leq H_{\text{OPT}} + \frac{2}{3} H_{\text{OPT}} = \frac{5}{3} H_{\text{OPT}}.
\]

On the left side, the floor shelf has height \( h_{\text{max}} \), and the left floor rectangles have total area at least \( t\alpha W \); the left pieces not on the floor shelf have area at
most $A_L - \alpha W$ and maximum height $\ell$. By Lemma 6,

$$H_L \leq h_{\text{max}} + \frac{A_L - \alpha W}{\alpha W} + \ell \quad (1)$$

$$\leq h_{\text{max}} + \frac{\text{max} \{ A^0_L, H_{\text{OPT}} - (1 - \alpha)H_{\text{OPT}} \} - \alpha W}{\alpha W} + \ell$$

$$\leq h_{\text{max}} + (H_{\text{OPT}} - t) + \ell$$

$$\leq H_{\text{OPT}} + 2(H_{\text{OPT}} - t) = \frac{5}{3} H_{\text{OPT}}.$$

Thus, the algorithm achieves approximation factor at most $5/3$.

**H.4 5/3-Approximation with 1 Cut**

We now describe a modification to the algorithm in Section H.3 that reduces the number of cuts per rectangle by gluing together some of the slices. In Step 3, on both sides, we first remove the full pieces before applying the Shelf algorithm; we can put each full piece in a separate shelf. The upper bound on $H_R$ remains unaffected: if the full pieces have total height $H'$, then their removal decreases $A_R/((1 - \alpha)W)$ by $H'$, which compensates for the height $H'$ of the extra separate shelves. The upper bound on $H_L$ is similarly unaffected.

We make further modifications: First, we reverse the order of the rectangles in each shelf on the left side, so that the tallest piece in each shelf touches the right boundary of the left side. Second, whenever we find two matching shelves, one on the left side and another on the right, such that their tallest pieces come from a common rectangle $r$, we move these two shelves to the bottom of the strip and place them side by side. This rearrangement of shelves does not change the heights $H_L$ and $H_R$. As a result, we can glue two pieces of $r$ together, in effect, reducing the number of cuts for $r$ by 1.

Consider a rectangle $r$ that is shifted completely leftward or rightward. If $r$ is split into two nonempty pieces, the full piece is not cut, and if the other piece is cut by the Shelf algorithm, its top sub-piece is glued to the full piece. We conclude that every rectangle is cut at most once, with one exception: the critical rectangle $r_X$.

For $r_X$, if the Shelf algorithm cuts both its left and the right piece, then the top left sub-piece is glued to the top right sub-piece. Thus, $r_X$ is cut at most twice.

We now show that with some modifications, we can achieve that $r_X$ is also cut at most once. We now show how to guarantee at most 1 cut even for the exceptional rectangle $r_X$. This requires more substantial changes to the algorithm. We divide into two cases.

The $\alpha \leq 1/2$ case. We add an extra step between Steps 2 and 3 to ensure that $r_X$ is the rectangle with the tallest non-full (and non-empty) left piece:

- Suppose the tallest non-full (and non-empty) left piece is defined by a rectangle $r$ different from $r_X$. Note that $r$ is currently shifted completely rightward.
Simultaneously shift $r$ leftward and $r_X$ rightward, while keeping $A_R$ constant, until (i) $r$ has been shifted completely leftward or (ii) $r_X$ has been shifted completely rightward.

When (i) happens, $r$’s left piece must be full, since $\alpha \leq 1/2$, and we can repeat with a new $r$. When (ii) happens, $r_X$ is no longer critical and we can reset $r_X$ to $r$ and stop.

Note that the procedure does not change the value of $\ell$, so our analysis on the approximation factor still holds.

It remains to check that $r_X$ is cut at most once. Since $r_X$ forms the tallest non-full left piece, this left piece is the first to be placed by the Shelf algorithm after the floor shelf and so is not cut. Furthermore, if the Shelf algorithm cuts the right piece of $r_X$, the top right sub-piece is glued to the left piece.

The $\alpha > 1/2$ case. Again, we add an extra step between Steps 2 and 3, this time, to ensure that $r_X$ is the rectangle with the shortest non-full (and non-empty) right piece.

Suppose the shortest non-full (and non-empty) right piece is defined by a rectangle $r$ different from $r_X$. Note that $r$ is currently shifted completely leftward.

Simultaneously shift $r$ rightward and $r_X$ leftward, while keeping $A_R$ constant, until (i) $r$ has been shifted completely rightward or (ii) $r_X$ has been shifted completely leftward.

When (i) happens, $r$’s right piece must become full, since $\alpha > 1/2$, and we can repeat with a new $r$. When (ii) happens, $r_X$ is no longer critical and we can reset $r_X$ to $r$ and stop.

Note that the procedure does not increase the value of $\ell$, so our analysis on the approximation factor still holds.

We make one final modification for the $\alpha > 1/2$ case: in Step 3, on the left side, we switch to Steinberg’s strip packing algorithm, which achieves the following bound (see Equation (2.3) of [21]):

**Lemma 9.** There is an algorithm that finds a strip packing of height at most $\max\{A/W + h_{\text{max}}, 2A/W\}$ without making any cuts. The run time is $O(n \log^2 n / \log \log n)$, where $n$ is the number of rectangles.

The chain of inequality for $H_L$ in (1) needs to be modified, but fortunately the result is the same:

\[
H_L \leq h_{\text{max}} + \max \left\{ \frac{A_L - t\alpha W}{\alpha W} + \ell, \frac{2A_L - t\alpha W}{\alpha W} \right\}
\]

\[
\leq h_{\text{max}} + \max \{ (H_{\text{OPT}} - t) + \ell, 2(H_{\text{OPT}} - t) \}
\]

\[
\leq H_{\text{OPT}} + 2(H_{\text{OPT}} - t) = \frac{5}{3} H_{\text{OPT}}.
\]

Consider a rectangle $r$ that is shifted completely leftward. If $r$ is split into two nonempty pieces, and the right piece is cut by the Shelf algorithm, its top sub-piece is glued to the left full piece. So, $r$ is cut at most once.
Consider a rectangle $r$ that is shifted completely rightward. If $r$ is split into two nonempty pieces, the left piece is not cut by Steinberg’s algorithm, and the right full piece is not cut either. So, $r$ is cut at most once.

It remains to check that $r_X$ is cut at most once. Its left piece is not cut by Steinberg’s algorithm. If its right piece is cut by the Shelf algorithm, since $r_X$ forms the shortest non-full right piece, this right piece is the last to be placed by the Shelf algorithm. Hence, we can transfer the bottom right sub-piece to the shelf holding the (standalone) top right sub-piece, and avoid a cut, without changing the height $H_R$.

**H.5 Guessing $H_{OPT}$**

Lastly, we remove the assumption that the value of $H_{OPT}$ is given. This value is used in setting $t$ and in the greedy procedure in Step 2 of the algorithm. By replacing $H_{OPT}$ with a user-supplied value $H_{GUESS}$ in the algorithm, it is easy to check that the algorithm has the following behavior: if $H_{GUESS} \geq H_{OPT}$, the solution returned has height at most $(5/3)H_{GUESS}$. Thus, if the solution returned has height at most $(5/3)H_{GUESS}$, we can conclude that $H_{OPT} \leq (5/3)H_{GUESS}$, otherwise $H_{OPT} > H_{GUESS}$.

We can apply a binary search to find an approximation to $H_{OPT}$. Start with $X = \frac{1}{2}H_S$, the height computed by the Shelf algorithm. We know $X \leq H_{OPT} \leq (5/3)cX$ with $c = 6/5$, so this is a $(5/3)c$-approximation. Now run the above algorithm with $H_{GUESS} = \sqrt{c}X$, and conclude either that $H_{OPT} \leq (5/3)\sqrt{c}X$ or $H_{OPT} > \sqrt{c}X$. In either case, we obtain an improved $((5/3)\sqrt{c})$-approximation. Repeating for $O(\log(1/\varepsilon))$ iterations, we then obtain a $(5/3 + \varepsilon)$-approximation. Assuming that all rectangle heights are integers bounded by $M$, we can set $\varepsilon = 1/(4nM)$, for example, and a $(5/3 + \varepsilon)$-approximation becomes a $5/3$-approximation; the running time increases by an $O(\log(nM))$ factor only.

**Theorem 2.** There is a polynomial-time algorithm for 2SP-SSC that finds a strip packing of height at most $\frac{5}{3}H_{OPT}$ where each rectangle is cut at most once. The run time is $O(n \log^2 n \log(nM)/\log \log n)$, where $M$ is an upper bound on the (integer) heights of the rectangles.

**Remark.** For the algorithm in Section H.2, the binary search can be avoided. We can just set $H_{GUESS} = A$, and it can be checked that the $3/2$-factor analysis still goes through (we can use a simpler inequality $2\ell \leq H_{OPT}$ in place of (5)). The algorithm’s total run time is then $O(n^2)$. 