Smart-grid Electricity Allocation via Strip Packing with Slicing *

Soroush Alamdari[1], Therese Biedl[1], Timothy M. Chan[1], Elyot Grant[2], Krishnam Raju Jampani[3], S. Keshav[1], Anna Lubiw[1], and Vinayak Pathak[1]

[1] Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada
{s26hosseinialamdari,biedl,tmchan,alubiw,keshav,vpathak}@uwaterloo.ca
[2] Massachusetts Institute of Technology, Cambridge, USA
elyot@mit.edu
[3] University of Guelph, Guelph, Canada
rjampani@uoguelph.ca

Abstract. One advantage of smart grids is that they can reduce the peak load by distributing electricity-demands over multiple short intervals. Finding a schedule that minimizes the peak load corresponds to a variant of a strip packing problem. Normally, for *strip packing problems*, 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. The electricity-allocation application can be modelled as *strip packing with slicing:* each rectangle may be cut vertically into multiple slices and the slices may be packed into the strip as individual pieces. The *stacking constraint* forbids solutions in which a vertical line intersects two slices of the same rectangle.
We give a fully polynomial time approximation scheme for this problem, as well as a practical polynomial time algorithm that slices each rectangle at most once and yields a solution of height at most 5/3 times the optimal height.

1 Introduction

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 [7]. 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 power-hungry appliances simultaneously. However, there is often flexibility in scheduling the use of particular appliances. For example, a water heater requires a certain amount of electricity to heat the water, but can equally well heat the water in one continuous interval

* This work was done as part of an Algorithms Problem Session at the University of Waterloo. Research of TB, TC, SK and AL supported by NSERC.

or in multiple short intervals.[4] It is anticipated that future smart grids would obtain (at each substation) daily "demand schedules" for appliance use from the consumers in its local area, and then automatically re-schedule appliance use to minimize peak load [18].

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. The re-scheduling should cover a given length of time, which corresponds to a strip of given width. The objective is then to pack slices of the rectangles into the strip so as to minimize the maximum power consumption, i.e., the maximum height of the packing. Because appliances cannot be powered at double the usual power, we have the additional *stacking constraint* requiring that no vertical line may intersect two slices from the same rectangle. Slicing with the stacking constraint is new, but strip packing has been well-studied, as we review in the following section.

Strip Packing Problems. 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 [14], and generalizes the *bin packing problem*, which is equivalent to the case in which all rectangles have unit height. The current best approximation algorithm for 2SP has an approximation factor of $5/3 + \varepsilon$ for any $\varepsilon > 0$ [9], and was achieved after a long sequence of successive improvements [1,16,17,19]. This is an *absolute performance bound*, i.e., the height achieved is at most $5/3 + \varepsilon$ times the optimal height. Many other authors have proposed algorithms with *asymptotic* performance guarantees [4,13,11] where an additive term is allowed.

Motivated by the electricity-allocation problem, we study a variant called *two-dimensional strip packing with slicing* (hereafter 2SP-S). In 2SP-S, we are allowed to cut each rectangle vertically into multiple slices, which may be packed into the strip as individual rectangles. Formally, the input consists of a number W and a set of rectangles $r_1, r_2, \ldots, r_n$. Here W is the width of the strip, which consists of two vertical sides at $x = 0$ and $x = W$, and the "base" at $y = 0$. Rectangle r_i has width w_i and height h_i; let $h_{\max} = \max_{i=1}^{n} h_i$ be the maximum height. A solution to 2SP-S consists of a partition of each rectangle r_i into vertical slices and an assignment of positions to the slices so that the interiors of the slices are pairwise disjoint. Slices must not be rotated. 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 with minimal possible height H_{OPT}.

Related Results: Strip packing with 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.

[4] To simplify the modelling we presume that no extra electricity is needed for restarting the appliance.

One may observe that if each rectangle is pre-cut into slices of some small width $\delta = W/m$ (where m is a positive integer), then solving the resulting 2SP problem is precisely equivalent to solving the minimum makespan scheduling problem on m parallel machines (the $P_m||C_{max}$ problem, in three-field notation). Unfortunately, the known approximation schemes for this problem all have a running time that is either exponential in m [15] or in $\frac{1}{\varepsilon}$ [10]. In fact, when the number of machines m is an input to the problem, minimum makespan scheduling on m parallel machines admits no FPTAS unless P=NP [3], so it appears hopeless to find an FPTAS for 2SP-S using such an approach.

A second reason that existing results do not apply to electricity-allocation (at least not as far as we can prove) is that in our application we have the 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.

Results For 2SP-S and 2SP-SSC. The freedom to slice rectangles can be highly beneficial. It is easy to construct an example where slicing reduces the required height by a factor of $2-\varepsilon$. Slicing also makes a difference in the complexity of the problem. Standard 2SP generalizes bin packing and is thus strongly NP-complete. Also, a simple reduction from the *Partition problem* [5] shows that 2SP admits no $(3/2-\varepsilon)$-approximation for any $\varepsilon > 0$ unless P=NP. In contrast, 2SP-S and 2SP-SSC are easily shown to be NP-hard, but not hard to approximate: we will give a fully polynomial-time approximation scheme (FPTAS).

The FPTAS is based on solving a linear program with exponentially many constraints, and hence mostly of theoretical interest. We also develop simpler, more practical algorithms, and also limit the number of times a rectangle may be sliced (which is of interest in the electricity-allocation problem to avoid start-up costs for the appliance).[5] We give two simple 2-approximation algorithms based on the well-known First Fit and Shelf paradigms. In fact, these algorithm achieve height $H_{\mathrm{OPT}} + h_{\max}$, hence they achieve optimal height up to an additive term. Then, building on these algorithms, and splitting the problem into two halves, we give two other polynomial-time algorithms that perform no worse than First Fit and Shelf. One has absolute performance bound 3/2. The other uses at most one cut per rectangle and has absolute performance bound 5/3.

Our paper is organized as follows. The First Fit and Shelf algorithms are in Section 2. Section 3 contains the FPTAS, and Section 4 develops practical algorithms. We conclude in Section 5.

2 Basic Algorithms

This section describes the *First Fit* and *Shelf* heuristics for 2SP-S and 2SP-SSC. Both algorithms achieve an approximation factor of 2, which is noteworthy

[5] More precisely, we want to minimze the times a rectangle is *interrupted*, i.e., one slice ends and no other slice starts. The number of times a rectangle is sliced is certainly an upper bound for this.

given that, for the standard strip packing problem, 2-approximation algorithms are difficult to obtain [16,19]. Both algorithms in fact achieve a height of $H_{OPT}+h_{\max}$, and hence have asymptotically performance bound 1.

First Fit Algorithm Given a list of rectangles $r_1, r_2, \ldots, r_n$, the *First Fit* algorithm processes them in order, repeatedly finds the lowest column in the current solution where a slice of r_i can be placed, and places 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. See Figure 1.

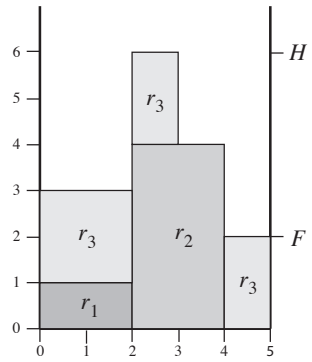

Fig. 1. An execution of the First Fit algorithm on a 2SP-SSC instance. Note that r_3 is sliced twice, and a smaller height would be achieved without the stacking constraint.

It is not hard to show that after placing each rectangle, the difference between the maximum height H and the floor F (the maximum height to which the entire strip is filled) is at most $h_{\max}$. Since by area-consideration $H_{\mathrm{OPT}} \geq F$, First Fit achieves height at most $H_{\mathrm{OPT}} + h_{\max}$ and is a 2-approximation since $h_{\max} \leq H_{\mathrm{OPT}}$.

Unfortunately there are instances where First Fit needs nearly twice the optimal height. A natural improvement to First Fit is to sort the rectangles by decreasing heights first; we call this variant First Fit Decreasing. One can easily construct an instance on which even First Fit Decreasing is a factor of $\frac{4}{3}$ away from the optimum. We do not know whether this is tight, but we can show:

Lemma 1. *First Fit Decreasing is a $\frac{3}{2}$-approximation.*

Proof. (Sketch) We prove this by defining another algorithm that packs (until a certain condition is fulfilled) as much as possible into columns that contain only rectangles of height at most $H_{\mathrm{OPT}}/2$. One can show that the "tall" (in some sense) columns of this algorithm have the same heights as the "tall" columns of First Fit Decreasing. One can also show that this other algorithm is a $\frac{3}{2}$-approximation. Putting the two together shows that First Fit Decreasing is a $\frac{3}{2}$-approximation. □

Shelf Algorithm Given a set of rectangles, the Shelf algorithm 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" (also called "levels"). The first shelf is the base 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 2. Note that the stacking constraint is automatically satisfied, and each rectangle is sliced at most once.

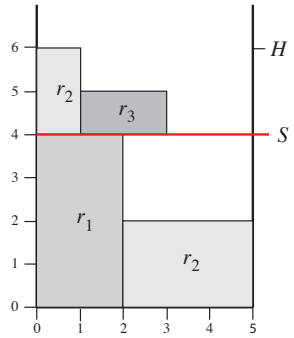

Fig. 2. An execution of the Shelf algorithm on the same instance as Figure 1 (but rectangles have been sorted by height.)

The Shelf algorithm is the same as the Next-Fit-Decreasing-Height algorithm for strip packing [4], except that we fill the entire width of the shelf immediately because we can slice rectangles. It is known that Next-Fit-Decreasing-Height achieves height $2 \cdot H_{\mathrm{OPT}} + h_{\max}$ even for strip packing without slicing [4]. As we will show now, permitting to slice allows to decrease this bound to $H_{\mathrm{OPT}} + h_{\max}$.

Observe that (with $h_{n+1} := 0$) the empty space below height H has area at most $\sum_{i=1}^{n}(h_i - h_{i+1}) \cdot W$. To see this, partition the empty space into rectangles by cutting it horizontally, and assign each empty rectangle to the rectangle r_i that has a slice below it in the same shelf. Therefore, the empty space is at most $h_1 \cdot W = h_{\max} \cdot W$, which proves that the Shelf algorithm achieves height at most $H_{\mathrm{OPT}} + h_{\max}$.

3 Approximation Schemes

In this section, we sketch the 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 input numbers are rationals represented explicitly in binary. Their run-time is polynomial in the input size and $\frac{1}{\varepsilon}$.*

The approach uses a linear programming relaxation and is relatively standard in the literature; in particular it resembles the classic work of Karmarkar and Karp concerning the bin-packing problem [12]. The linear program we solve is similar to the one used to obtain fractional strip packings in [13], though our full algorithm requires different searching and rounding routines since the variables in our linear program must correspond to vertical configurations rather than horizontal ones.

In the remainder of this section, we prove Theorem 1 for the case of 2SP-SSC; we omit the (minor changes) that must be done for 2SP-S.

Step 1: Reducing the general problem to a decision version

Given a guess H_{GUESS} for the optimal height H_{OPT}, the main algorithm that we describe in steps 2 through 5 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_{GUESS}.

Since the optimal height H_{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_{OPT} to within a multiplicative factor of $1 + \varepsilon$ using only $O(\log(\frac{n}{\varepsilon}))$ 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_{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 $\mathcal{S}$ of height H^*. 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_{GUESS}. Conversely, the stacking constraint implies that each vertical line passes through the interior of at most n rectangles in $\mathcal{S}$, so after undoing the rounding, the height of $\mathcal{S}$ increases by at most $\frac{\varepsilon}{2} H_{\text{GUESS}}$. Thus 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

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_{GUESS}. We define a *pattern* to be any subset of $\{r_1, \ldots, r_n\}$ whose total height is at most H_{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. This idea motivates our 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_{GUESS} is equivalent to solving the following linear program:

$$\begin{aligned} \text{minimize:} \quad & \sum_{P \in \mathcal{P}} x_P \\ \text{subject to:} \quad & \sum_{P \in \mathcal{P} | r_i \in P} x_P \geq w_i \text{ for all } 1 \leq i \leq n \\ & x_P \geq 0 \text{ for all } P \in \mathcal{P} \end{aligned} \tag{LP}$$

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.

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):

$$\begin{aligned} \text{maximize:} \quad & \sum_{i=1}^{n} w_i y_i \\ \text{subject to:} \quad & \sum_{i | r_i \in P} y_i \leq 1 \text{ for all } P \in \mathcal{P} \\ & y_i \geq 0 \text{ for all } 1 \leq i \leq n \end{aligned} \tag{LP*}$$

Despite this linear program having exponentially many constraints, we can tackle it using the ellipsoid algorithm. Specifically, since we assumed that the widths of the rectangles in the input are rational numbers represented explicitly in binary, we can find the *exact* optimal objective value of (LP*) in time polynomial in the input size and $\frac{1}{\varepsilon}$, provided that we can solve the corresponding *separation problem* in time polynomial in the input size and $\frac{1}{\varepsilon}$.

The separation problem for this linear program asks 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 essentially asks if there is any set of rectangles of total height less than H_{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_{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 $O(\frac{n^2}{\varepsilon})$ time using standard dynamic programming methods.

If one wishes to achieve a more practical running time, it is feasible to replace the ellipsoid algorithm with the simplex algorithm, using the column generation technique of Gilmore and Gomory [6].

Step 5: Returning the solution

We observe that it is possible to reconstruct an optimum solution to (LP) while solving (LP*) using this technique (see [12] for details). Consequently, we can not only approximate the optimum height of a packing, but can in fact return a packing having that height. We already argued with Step 2 that this satisfies

the approximation bound. 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 sliced at most $n-1$ times.

We also observe that in the solution produced by the FPTAS, the number of cuts per rectangle can be further reduced to a constant that depends only on ε. More precisely, we can show (details are omitted) that any feasible solution can be modified so that each rectangle is sliced at most $(1/\varepsilon)^{O(1/\varepsilon)}$ times, without increasing the height by more than a factor of $1 + O(\varepsilon)$.

4 Algorithms With Few Slices

Although the approximation scheme from the previous section may be more practical if the simplex method is used, it is still unsuitable for electricity-allocation applications both due to its runtime and because it may result in rectangles that have been sliced numerous times. In fact, we can create instances where some rectangle must be sliced $\Omega(n/\log n)$ times in any optimal solution. For practical purposes, it would be worth sacrificing some height if in exchange we can guarantee that rectangles are not sliced too often. We develop such algorithms now.

The approach is to partition the bin vertically into two parts, slice each rectangle once, and pack the two slices in the two parts with Shelf. With a judicious choice of where to partition and slice, this results in a 5/3-approximation that slices each rectangle at most three times. (We note that this result is achieved with Shelf already if $h_{\max} \leq \frac{2}{3}H_{\mathrm{OPT}}$.) With some more work we can align rectangle slices so that each rectangle is sliced at most once.

The algorithm assumes that the value of H_{OPT} is known (we will find H_{OPT} with binary search as explained later). It depends on some parameter $t > H_{\mathrm{OPT}}/2$; using $t = 2H_{\mathrm{OPT}}/3$ gives the best approximation bound.

Step 1. Assuming the rectangles have been sorted in decreasing order of heights, find the largest k such that $w_1 + \cdots + w_k \leq W$. By $t > H_{\mathrm{OPT}}/2$ we have $h_{k+1} \leq t$. Find the largest $j \leq k$ such that $h_j \geq t$. (In case $h_{\max} < t$, we define j to be 0; the algorithm becomes identical to Shelf in this case.) 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 α to be $(w_1 + \cdots + w_j)/W$, so the left side has width αW and the right side has width $(1-\alpha)W$. Note that we may have $\alpha = 0$ or $\alpha = 1$, but $\alpha \leq 1$ since in any optimal packing no two floor rectangles may overlap vertically by $t > H_{\mathrm{OPT}}/2$.

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 αW (resp. $(1-\alpha)W$). Either piece is allowed to be empty. The splitting procedure is described below. In the following, *shifting* a rectangle *rightward* means enlarging the width of its right piece by δ and shrinking the width of its left piece by δ for some amount $\delta > 0$. Shifting a rectangle leftward is similarly defined.

We say that a rectangle is shifted *completely* rightward if the left piece is empty or the right piece has width $(1-\alpha)W$. We say that a rectangle is shifted *completely* leftward if the right piece is empty or the left piece has width αW.

All left floor rectangles are shifted completely leftward and all right floor rectangles are shifted completely rightward. All non-floor rectangles are initially shifted completely rightward. Let A_L^0 and A_R^0 be the total area of all left (resp. right) pieces after this initialization. We now shift rectangles so that A_R (the area of the current right pieces) equals $(1-\alpha)H_{\mathrm{OPT}}W$, if possible, and do this using a greedy procedure:

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

Observe that except for one *critical rectangle*, which we denote by r_x, all rectangles are either shifted completely leftward or completely rightward. We also claim that the above procedure ends with either $A_R = A_R^0$ or $A_R = (1-\alpha)H_{\mathrm{OPT}}W$, whichever is smaller. For assume that all non-floor rectangles have been shifted completely leftward. The left floor rectangles have total area at least $t\alpha W$ and the right floor rectangles have total area less than $t(1-\alpha)W$, so among the floor rectangles, at least an α-fraction of the area has been assigned to the left. Each non-floor rectangle (shifted completely leftward) has at least an α-fraction of the area on the left. Therefore the area to the left is at least α-fraction of the total area, or $A_L \geq \alpha A$, which implies $A_R = A - A_L \leq (1-\alpha)A \leq (1-\alpha)H_{\mathrm{OPT}}W$. So if $A_R^0 > (1-\alpha)H_{\mathrm{OPT}}W$, then at some point during the shifting process we reach a moment when $A_R = (1-\alpha)H_{\mathrm{OPT}}W$ as desired.

Step 3. Pack the left pieces into the left strip and the right pieces into the right strip, using Shelf on both sides.

Theorem 2. *There exists 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.*

Proof. Assume for now that H_{OPT} is known and apply the above algorithm. This gives a packing with at most 3 cuts per rectangle (one to partition it into the left and right piece, and one by each application of Shelf), and the only rectangle that may have 3 cuts is r_x.

We first analyze the height of the left strip. The bottommost shelf has height $h_{\max}$, and (by definition) contains the left floor rectangles whose total area is at least $t\alpha W$. The left pieces of non-floor rectangles hence have total area at most $A_L - t\alpha W =: A_{NFL}$. Let ℓ be the tallest height of a non-floor rectangle whose left piece is non-empty ($\ell = 0$ if there is no such piece.) By the same analysis as for Shelf, the shelves for the left pieces of non-floor rectangles have empty space at most $\ell\alpha W$, hence they contribute height at most $(A_{NFL})/(\alpha W) + \ell = A_L/\alpha W - t + \ell$.

Claim: $\ell + t \leq H_{\mathrm{OPT}}$. Clearly this holds if $\ell = 0$, so assume $\ell > 0$. To prove the claim, consider an optimal packing S^* and assume that its height is less than $\ell + t$. Then no vertical line intersects two left floor rectangles in S^*, since these all have height $\geq t$ and $\geq \ell$. Rearrange S^* so that all vertical lines containing left floor rectangles appear at the left end, and call this part the *left strip of* S^*, which has width αW. Again, since the height of S^* is less than $\ell + t$, no right floor rectangles and no non-floor rectangle of height $\geq \ell$ can appear in the left strip of S^*. Thus the right floor rectangles and non-floor rectangles of height $\geq \ell$ all fit in the right strip of S^*. Also, all such non-floor rectangles have width at most $(1-\alpha)W$. Finally, for any non-floor rectangle of height $< \ell$, at most a slice of width αW can be in the left strip, so if the rectangle has width $> \alpha W$, then its right piece, even when entirely shifted leftward, also fits within the right strip of S^*.

Recall that the greedy procedure for splitting rectangles processes rectangles in increasing height. Since we have a left piece of a non-floor rectangle of height ℓ, all non-floor rectangles of height $< \ell$ must be completely shifted leftward. So by the time the procedure reaches the rectangle of height ℓ, the right pieces consists of the right floor rectangles, the minimum possible right pieces of non-floor rectangles of height $< \ell$, and the entire non-floor rectangles of height $\geq \ell$. By the above discussion, all these pieces fit into the right strip of S^*, which has area $(1-\alpha)WH_{\mathrm{OPT}}$. But then $A_R < (1-\alpha)WH_{\mathrm{OPT}}$ already and the greedy procedure would have stopped and not shifted the rectangle of height ℓ leftward. This is a contradiction, so the optimal height is at least $t + \ell$, which proves the claim.

Putting all of this together and using $h_{\max} \leq H_{\mathrm{OPT}}$, the left strip has height at most

$$h_{\max} + \frac{A_L}{\alpha W} - t + \ell \leq H_{\mathrm{OPT}} + \frac{A_L}{\alpha W} - t + H_{\mathrm{OPT}} - t = 2H_{\mathrm{OPT}} - 2t + \frac{A_L}{\alpha W}.$$

In the right strip, all rectangles have height at most t. Hence the right strip has at most $t(1-\alpha)W$ empty area, and its height is at most

$$\frac{A_R}{(1-\alpha)W} + t \leq \frac{(1-\alpha)H_{\mathrm{OPT}}W}{(1-\alpha)W} + t = H_{\mathrm{OPT}} + t \leq \frac{5}{3}H_{\mathrm{OPT}}$$

by choice of t. We have chosen the partition into left and right pieces carefully to ensure that A_L is "just right". In the first case, $A_R = (1-\alpha)H_{\mathrm{OPT}}W$, which implies that $A_L \leq \alpha H_{\mathrm{OPT}}W$. In this case the left strip has height at most $3H_{\mathrm{OPT}} - 2t \leq \frac{5}{3}H_{\mathrm{OPT}}$ by choice of t. To prove the bound for the left strip in the second case where $A_R = A_R^0 < (1-\alpha)H_{\mathrm{OPT}}W$, we need the following result:

Claim: $\frac{A_L^0}{\alpha W} \leq H_{\mathrm{OPT}}$. To prove this claim, 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 again an optimal solution S^* (with an unbounded number of slices) and rearrange the columns in S^* so that the left floor rectangles form a left strip of width αW. 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 $A_L^0 \leq H_{\mathrm{OPT}} \alpha W$, and hence the claim holds.

With this claim, the left strip has height $2H_{\mathrm{OPT}} - 2t + \frac{A_L^0}{\alpha W} \leq 3H_{\mathrm{OPT}} - 2t \leq \frac{5}{3}H_{\mathrm{OPT}}$ by choice of t, and we have now proved the approximation bound.

Lastly, we remove the assumption that the value of H_{OPT} is given. 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 $H_{\mathrm{GUESS}} \geq H_{\mathrm{OPT}}$, the solution returned has height at most $(5/3)H_{\mathrm{GUESS}}$. Thus, if the solution returned has height at most $(5/3)H_{\mathrm{GUESS}}$, we can conclude that $H_{\mathrm{OPT}} \leq (5/3)H_{\mathrm{GUESS}}$, otherwise $H_{\mathrm{OPT}} > H_{\mathrm{GUESS}}$.

We can apply a binary search to find an approximation to H_{OPT}. Start with $X = \frac{1}{2}H_{\mathrm{S}}$, the height computed by the Shelf algorithm. We know $X \leq H_{\mathrm{OPT}} \leq (5/3)cX$ with $c = 6/5$, so this is a $(5/3)c$-approximation. Now run the above algorithm with $H_{\mathrm{GUESS}} = \sqrt{c}X$, and conclude either that $H_{\mathrm{OPT}} \leq (5/3)\sqrt{c}X$ or $H_{\mathrm{OPT}} > \sqrt{c}X$. In either case, we obtain a $((5/3)\sqrt{c})$-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. □

We can reduce the number of slices even further by packing them into the shelves carefully so that pieces become aligned on the shelf, and (in some cases) apply Steinberg's algorithm for 2SP [19] on one of the sides. Details are omitted.

Theorem 3. *There exists a 5/3-approximation for 2SP-SSC that slices every rectangle at most once and runs in time $O(n \log^2 n \log(nM)/\log\log n)$.*

5 Conclusions

Motivated by an application in electricity allocation, this paper explored variants of the strip packing problem in which rectangles could be sliced vertically as long as no two slices of the same rectangle are stacked atop each other. We provided simple 2-approximation algorithms, an FPTAS 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. For example we conjecture that First Fit Decreasing is actually a 4/3-approximation. Without the stacking constraint, this follows from Graham's 4/3-approximation bounds for multiprocessor scheduling [8], but with the stacking constraint the best bound we can prove is 3/2. Also, can we find a $\frac{3}{2}$-approximation (or even $\frac{4}{3}$-approximation) that slices every rectangle at most once? Finally, is there a simple PTAS for strip packing with slicing (with or without the stacking constraint)?

References

1. B. S. Baker, E. G. Coffman, and R. L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal on Computing*, 9(4):846–855, 1980.
2. M. Bougeret, P.-F. Dutot, K. Jansen, C. Otte, and D. Trystram. Approximating the non-contiguous multiple organization packing problem. In *Theoretical Computer Science - 6th IFIP TCS*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 316–327. Springer, 2010.
3. B. Chen, C. N. Potts, and G. J. Woeginger. A review of machine scheduling: complexity, algorithms and approximability. In *Handbook of combinatorial optimization, Vol. 3*, pages 21–169. Kluwer Acad. Publ., Boston, MA, 1998.
4. E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, and R. E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM J. Comput.*, 9(4):808–826, 1980.
5. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman & Co Ltd, 1979.
6. P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Res.*, 9:849–859, 1961.
7. P. Giudice. Our energy future and smart grid communications. Testimony before the FCC Field Hearing on Energy and Environment. `www.broadband.gov/fieldevents/fh_energy_environment/giudice.pdf`, 2009.
8. R. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math*, 17:416–429, 1969.
9. R. Harren, K. Jansen, L. Prädel, and R. van Stee. A $(5/3 + \varepsilon)$-approximation for strip packing. In *Algorithms and Data Structures Symposium, WADS 2011*, volume 6844 of *Lecture Notes in Computer Science*, pages 475–487. Springer, August 2011.
10. D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *J. Assoc. Comput. Mach.*, 34(1):144–162, 1987.
11. K. Jansen and R. Solis-Oba. New approximability results for 2-dimensional packing problems. In *Mathematical Foundations of Computer Science 2007*, volume 4708 of *Lecture Notes in Computer Science*, pages 103–114. Springer, 2007.
12. N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Symposium on Foundations of Computer Science*, pages 312–320. IEEE, 1982.
13. C. Kenyon and E. Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Math. Oper. Res.*, 25(4):645–656, 2000.
14. A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241 – 252, 2002.
15. S. K. Sahni. Algorithms for scheduling independent tasks. *J. Assoc. Comput. Mach.*, 23(1):116–127, 1976.
16. I. Schiermeyer. Reverse-Fit: A 2-optimal algorithm for packing rectangles. In *European Symp. Algorithms*, volume 855 of *Lecture Notes in Computer Science*, pages 290–299. Springer, 1994.
17. D. D. Sleator. A 2.5 times optimal algorithm for packing in two dimensions. *Information Processing Letters*, 10(1):37–40, Feb. 1980.
18. P. Srikantha, C. Rosenberg, and S. Keshav. An analysis of peak demand reductions due to elasticity of omestic appliances. In *Proc. Energy-Efficient Computing and Networking (e-Energy '12)*, page 28. ACM, 2012.
19. A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 26(2):401–409, 1997.