Improved Approximation Guarantees for Advertisement Placement

Waldo Gálvez
Department of Computer Science, Universidad de Concepción, Chile

Roberto Oliva
Institute of Engineering Sciences, Universidad de O'Higgins, Rancagua, Chile

Victor Verdugo
Institute for Mathematical and Computational Engineering, Department of Industrial and Systems Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile

Abstract

The advertisement placement problem involves selecting and scheduling ads within a timeline that has capacity constraints to maximize profit. Each task is characterized by its height, width, and profit, and must be fully scheduled across multiple time slots. This problem models practical scenarios such as internet advertising and energy management, and it also generalizes classical combinatorial optimization problems like the knapsack and bin packing problems.

We present a simple $(2+\varepsilon)$-approximation algorithm for any $\varepsilon > 0$, which improves upon the state-of-the-art $3+\varepsilon$ factor established by Freund and Naor twenty years ago. Our approach combines rounding techniques with dynamic programming and an efficient extension of list scheduling. Furthermore, we enhance this method with linear programming techniques to provide an almost optimal $(1+\varepsilon)$-approximation algorithm under resource augmentation, which allows for a slight increase in time slot capacities.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Advertisement Placement, Two-dimensional Packing, Geometric Knapsack, Resource Allocation

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2025.10

Category APPROX

Funding *Waldo Gálvez*: Supported by ANID through grants SIA85220118 and Fondecyt de Iniciación 11230663.
Victor Verdugo: Supported by ANID through grant Fondecyt 1241846.

1 Introduction

In the *advertisement placement* problem, the goal is to efficiently select and schedule a set of ads to attract a wider variety of customers, for example, when they visit a specific website or a place. Formally, we are given a set of n tasks, each one characterized by a height, a width and a nonnegative profit, and also a timeline of length T with a fixed capacity C on each time slot. The goal is to select a subset of the tasks with maximum total profit to be scheduled into the timeline while respecting the capacity constraints. Here, scheduling a task means allocating a number of *slices* or *copies* of the task equal to its width into different time slots, and the profit of the task is accounted for only if all its copies are correctly allocated.

This problem was first introduced by Adler, Gibbons, and Matias [1], motivated by space-sharing in internet advertising. In this context, a space exists where ads will be displayed that has capacity C, and each ad i has a size h_i and a *display frequency requirement* w_i. Consequently, a natural goal is to use the space as effectively as possible, which can be modeled by defining profits as $p_i = h_i w_i$; indeed, this is the case considered originally by

Adler et al. and the main focus of this work. Recently, the advertisement placement problem has found applications in energy management; see, e.g., [2, 28, 30]. In this case, the tasks are electric appliances characterized by their execution time and the energy they use to operate, and the goal is to plan the electric consumption of a given journey while optimizing some associated objective function, assuming that tasks can be interrupted and restarted later. From a theoretical standpoint, advertisement placement can be thought of as a natural and interesting two-dimensional generalization of classical combinatorial optimization problems such as knapsack [25], makespan minimization [26], and bin packing [8], among others.

As observed by Adler et al. [1], the advertisement placement problem is NP-hard even in very restricted settings, including the classical framework of *divisible widths* introduced by Coffman et al. [22] in the context of bin packing. Later, Freund, and Naor [11] obtained a $(3+\varepsilon)$-approximation algorithm for the case with arbitrary widths and arbitrary profits. Upon this work, this result is the best-known so far, even for the problem with profits equal to the task areas.

1.1 Our Results

Our main result is the design of improved approximation algorithms for advertisement placement. More specifically, we achieve a $(2+\varepsilon)$-approximation for the problem (Theorem 1), improving the previously mentioned result due to Freund and Naor [11]. We also complement this result with a $(1+\varepsilon)$-approximation algorithm under resource augmentation, meaning that our algorithm might use a slight extra amount of capacity from each time slot (Theorem 7).

To achieve our $(2+\varepsilon)$-approximation, we classify the tasks according to their heights into *tall* and *short*, designing almost optimal algorithms for each case separately. For the case of tall tasks, we show that one can efficiently select a subset of tasks that can be feasibly scheduled and have almost optimal profit via rounding techniques and dynamic programming, which then can be scheduled using known results for the makespan minimization variant of advertisement placement. For the case of short tasks, we show that a simple extension of list scheduling leads to an almost optimal efficient algorithm. Under resource augmentation, we start by scheduling tall tasks precisely as before, but then use a linear programming relaxation to schedule short tasks on top, which can be efficiently rounded to a feasible schedule by adapting the classical scheme due to Lenstra, Shmoys and Tardos [26] while possibly slightly overloading the time slots. This approach requires selecting which short tasks go into the solution, which we do by further distinguishing short tasks into *wide* and *small* according to their widths. For the case of wide tasks, we use rounding techniques as in the $(1+\varepsilon)$-approximation for tall tasks, while in the case of small tasks, we run a greedy procedure.

1.2 Related Work

The advertisement placement problem was first introduced by Adler, Gibbons, and Matias [1]. They considered two different variants of the problem: *makespan minimization*, where the goal is to schedule all the given tasks while minimizing the maximum load, and *profit maximization*, the variant considered here. Their results concern mainly the special case with divisible widths and profits being equal to the area of the tasks, where they provide a 2-approximation. For the case of the Makespan Minimization variant, Dean and Goemans developed a $\frac{4}{3}$-approximation with running time poly(n), and an approximation scheme with running time poly(n, T) [9].

Other two-dimensional generalizations of knapsack have also been studied in the past. One such example is the *two-dimensional geometric knapsack* problem, where the input consists of a rectangular knapsack in the two-dimensional plane, and n rectangular items, characterized by their height, width and profit; the goal is to select and place a subset of items with maximum total profit in such a way that they are axis-parallel and do not overlap (which corresponds to advertisement placement when slices are restricted to go into contiguous time slots and they all must use the same vertical space). In the case of general profits, the best-known results for this problem are a $(17/9+\varepsilon)$-approximation, and a $(1.717+\varepsilon)$-approximation in the case of uniform profits [13]. If the profit of the items is equal to their area, then the problem admits a PTAS [4], and also improved results are known for some special settings, including pseudo-polynomial time algorithms [14], and resource augmentation [21], among others.

Another two-dimensional generalization of knapsack is the *unsplittable flow on a path* problem, which is similar to advertisement placement with the difference that each task is associated with a set of contiguous time slots where its slices must go in case the task is chosen. After a series of improvements for the problem, Grandoni et al. [17] showed that it admits a PTAS even with non-uniform capacities for the time slots. The setting where, instead of specifying precisely into which time slots a task must go, we specify a longer interval where its slices could go for each task, is known as *time windows constrained.* For the case of unsplittable flow on a path with time windows constraints (i.e., where slices still must go into contiguous time slots), the best known polynomial time approximation algorithm has an approximation ratio of $O(\log(n)/\log\log(n))$ [16], and it was recently proved that the problem is APX-hard but admits a $(2+\varepsilon)$-approximation in quasi-polynomial time under resource augmentation [3]. For advertisement placement with time windows constraints, Da Silva et al. [7] provide a PTAS when the timeline length T is bounded by a constant, and a 9-approximation for windows of the form $[r_i, T]$ [27].

Two-dimensional packing problems where all the given tasks must be placed (similar in spirit to makespan minimization) include well-studied problems such as strip packing [18, 12, 19], demand strip packing [10, 20], two-dimensional variants of bin packing [5, 24], among many others. We refer the reader to the survey by Christensen et al. [6] for further details.

Organization of the paper. In Section 2 we provide a formal definition of the problem and some useful notation. Then, in Section 3 we present our $(2+\varepsilon)$-approximation for the problem, and in Section 4 we present our $(1+\varepsilon)$-approximation with resource augmentation.

2 Preliminaries

In the advertisement placement problem, we are given a set of n tasks, where each task $i \in \{1,\dots,n\}$ is characterized by its *height* $h_i \in \mathbb{N}$, its *width* $w_i \in \mathbb{N}$, and its profit $p_i \in \mathbb{N}$; we are also given a timeline defined by $T \in \mathbb{N}$ *time slots*, and a *capacity* $C \in \mathbb{N}$. The goal is to select a subset of tasks of maximum total profit that can be feasibly scheduled into the timeline. In this context, a feasible schedule for a given set of tasks corresponds to an allocation function that assigns, for each task i in the set, w_i different time slots where a *slice* of the task will be scheduled (i.e., a piece of the task that has height h_i and width 1). The allocation function must satisfy that the total *load* of each time slot e, denoted as $\ell(e)$ and corresponding to the sum of the heights of tasks scheduled in the time slot, must be at most C (see Fig. 1 for an example). Scheduling two or more slices of a task in the same time slot is not allowed.

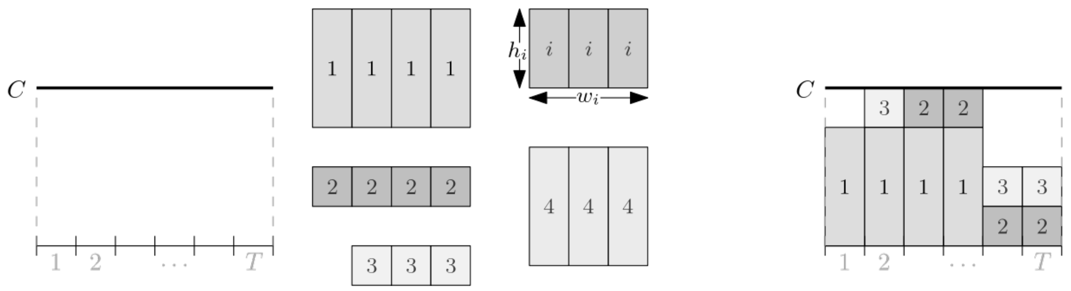

Figure 1 Example of an advertisement placement instance (left) and a corresponding feasible schedule defined by three tasks (right). The load profile of the solution would be $(h_1 + h_3, h_1 + h_2, h_1 + h_2, h_1, h_2 + h_3, h_2 + h_3)$.

Since a feasible schedule requires specifying where each task slice goes, we will assume that T is polynomially bounded by n to ensure that the output can be returned in polynomial time. Furthermore, unless otherwise stated, it is assumed that $p_i = h_i \cdot w_i$, and therefore, the goal is to cover as much area from the region $[0, T] \times [0, C]$ as possible; this objective encodes the original formulation of the problem, as well as other interesting applications (see, e.g., [1]). When referring to the case of general profits, we will denote the problem as *weighted advertisement placement.*

Areas and Configurations. For a given task i, we define $\text{area}(i) = h_i w_i$, which naturally extends to a subset S of tasks as $\text{area}(S) = \sum_{i \in S} \text{area}(i)$; we use an analogous notation for profits. Throughout this work, for a given instance I of the problem, $\text{OPT}(I)$ will denote a fixed optimal schedule for I (dependence on I will be dropped if clear from the context), and $\text{area}(\text{OPT})$ will denote the total area of the tasks included in OPT; when working with general profits, $p(\text{OPT})$ will denote the total profit of the tasks included in OPT. Given a feasible schedule $\mathcal{S}$, we define the *load configuration* of $\mathcal{S}$ as a vector of dimension T, where each entry stores the total load of a time slot, and is sorted non-increasingly by load; see Fig. 1 for an example.

3 A $(2 + \varepsilon)$-approximation for Advertisement Placement

In this section, we prove our first main result.

▶ **Theorem 1.** *For any $\varepsilon > 0$, there exists a $(2 + \varepsilon)$-approximation for advertisement placement.*

To achieve this result, we split our proof into two steps. We start by categorizing the tasks in the input according to their heights into *tall* and *short*, proving that an almost optimal solution can be obtained efficiently for each case. In what follows, $\delta \leq \varepsilon/2$ is a constant, and we say that a task i is tall if $h_i > \delta \cdot C$, and short if $h_i \leq \delta \cdot C$. We denote by $\mathcal{T}$ and $\mathcal{S}$ the sets of tall and short tasks from the instance, respectively; furthermore, we denote by $\text{OPT}_{\mathcal{T}}$ and $\text{OPT}_{\mathcal{S}}$ the optimal solution restricted to tall and short tasks, respectively.

When focusing on tall tasks, we rely on several rounding procedures. First, by adapting the *linear grouping* technique from bin packing [8, 23], we reduce the possible number of heights in the instance to a constant number, and then we round down the total width of each group (classified according to the rounded height) to know which tasks will be included in our solution. After this, it is possible to apply the results from Dean and Goemans [9] for makespan minimization in the context of advertisement placement, yielding a PTAS for the tall tasks.

For the case of short tasks, we show that a slight variation of the *shortest size least full first* (SSLF) algorithm, studied by Freund & Naor [11], yields a PTAS. Since, in any optimal solution, either the tall tasks or the short tasks have total area at least area(OPT)/2, running both algorithms and returning the best of the computed solutions yields a $(2+\varepsilon)$-approximation.

3.1 Linear Grouping for Tall Tasks

We start by devising the algorithm for tall tasks, where our goal is to compute an almost optimal solution efficiently. Interestingly, the solution we construct will exhibit the extra property of having a load configuration upper bounded by the load configuration of $\mathrm{OPT}_{\mathcal{T}}$ (coordinate-wise), which is a structural property of independent interest. To do so, we first reduce the possible number of heights to a constant number by adapting a linear grouping technique; see, e.g., [8, 23].

Our goal is to select a subset of tasks of total area at least $(1-\varepsilon)\mathrm{area}(\mathrm{OPT}_{\mathcal{T}})$ that can be feasibly scheduled. Observe that, if $\mathrm{OPT}_{\mathcal{T}}$ is defined by at most $8/(\varepsilon^2\delta^2)$ tasks, we can guess precisely such a set in polynomial time and conclude this part. Thus, from now on, we assume that $\mathrm{OPT}_{\mathcal{T}}$ has more than $8/(\varepsilon^2\delta^2)$ tall tasks. The outline of our procedure is the following:

(I) Suppose that $\mathcal{T} = \{1, \dots, t\}$ with $h_1 \geq h_2 \geq \cdots \geq h_t$. We guess a set of at most $8/(\varepsilon^2\delta^2)$ tall tasks, that we call the *guessed tasks*; the rest are non-guessed tasks. Out of the guessed tasks, we first guess a subset of at most $2/(\varepsilon\delta)$ tasks, that we call the *rounding tasks*, and a second (disjoint) subset of at most $2/(\varepsilon\delta^2)$ tasks, that we call the *shifted tasks*. We discard both sets, meaning that they will not be included in our solution.

(II) The set of non-guessed tasks that have a smaller index than the rounding task with smallest index is discarded.

(III) For any remaining non-guessed task i, round h_i up to $h_{i'}$, with $i' < i$ being the closest larger height among the rounding tasks (breaking ties in favor of smaller indices).

See Fig. 2 for a depiction of the procedure. Let G_i be the set of non-guessed tasks whose heights were rounded according to the i-th rounding task (sorted non-increasingly by height). As the following lemma shows, there exists a set of guessed tasks with a subset of rounding and shifted tasks that allows to bound the total area of discarded tasks from $\mathrm{OPT}_{\mathcal{T}}$ by $\varepsilon \cdot \mathrm{area}(\mathrm{OPT}_{\mathcal{T}})$, while still having a feasible solution for the rounded tasks from $\mathrm{OPT}_{\mathcal{T}}$ that were not discarded.

▶ **Lemma 2.** *Let I be a set of tall tasks, and I' be the output of the previous rounding procedure. There exists a set $\tilde{I} \subseteq I'$ of rounded tall tasks such that:*

(i) *The number of distinct heights in $\tilde{I}$ is constant.*

(ii) *The total area of $\tilde{I}$ is at least $(1-\varepsilon)\mathrm{area}(\mathrm{OPT}(I))$.*

(iii) *There exists a feasible schedule for $\tilde{I}$.*

Proof. Suppose we place all the tasks in $\mathrm{OPT}(I)$ one next to the other, sorted non-increasingly by height. Since there is a feasible schedule for them and their height is at least δC, the total width of tasks in $\mathrm{OPT}(I)$, say W, is at most T/δ. Starting from the tallest slice in the pile, we partition the slices of the tasks into groups $G_1, \dots, G_g$ of total width exactly $(\varepsilon\delta/2)W$ each, except possibly for the last one, which can have a smaller width. We have at most $2/(\varepsilon\delta)$ groups.

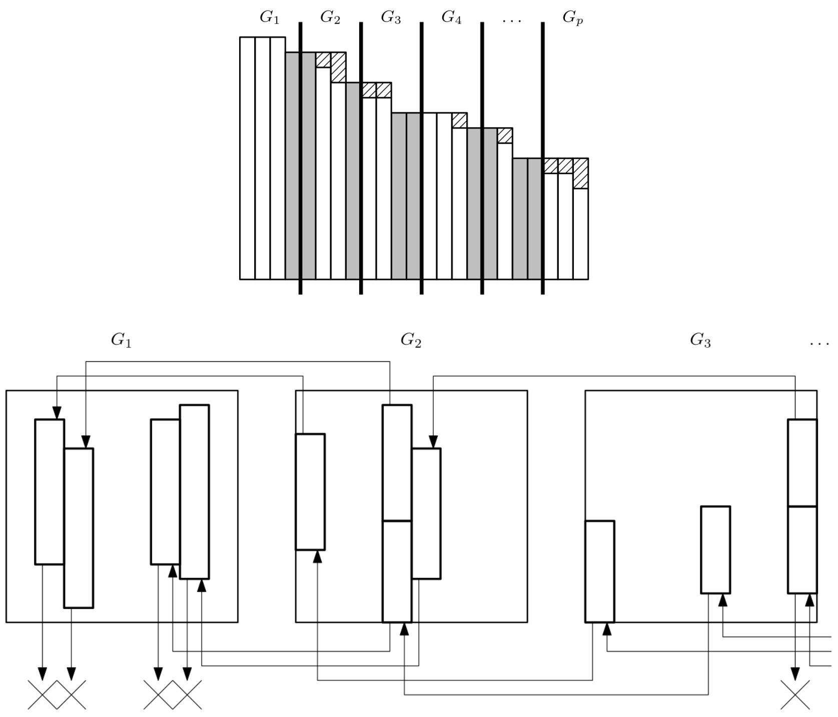

Figure 2 Description of the rounding of tall tasks from Lemma 2. (Top) Grey tasks correspond to rounding tasks, while the other slices are rounded up according to the closest rounding task. (Bottom) Slices from each group are rearranged according to the positions of the previous group. Some tasks are discarded either because they belong to G_1 or because they could not get scheduled in a feasible manner, but their total area is ensured to be negligible.

Our tentative set of rounding tasks from the rounding procedure will initially be defined by taking the task of smallest height from each group G_i, for each $i \in \{1, \ldots, g-1\}$ (this set may contain less than $g-1$ tasks due to repetitions), and our set of shifted tasks will initially be empty. At this point, the set of rounding tasks and guessed tasks is the same, and tasks with smaller index than the rounding task with smallest index are not part of $OPT(I)$, so they can be discarded.

Consider the feasible schedule for OPT(I). Since the first group on the left (i.e., G_1) is discarded, we will use the empty space due to this removal to place the slices from G_2 in the following way: we iteratively place the slices from G_2 into the leftmost time slot that had a slice from G_1 and does not have any slice from the same corresponding task; if some slice cannot be placed, we skip and continue. We claim that the number of tasks whose slices were not placed entirely by this procedure is at most $1/\delta - 1$. Indeed, if there were more, since there must be some time slot that had some slice from G_1 that did not get replaced by a slice from G_2, this means that slices that could not get assigned into that space already have another slice from the same job there. However, since all the heights are at least δC, there can be at most $1/\delta - 1$ other slices in that time slot, meaning that some task could have used that space in our procedure. This leads to a contradiction.

The set of tasks that could not get assigned by this procedure will be added to the set of shifted tasks. Now that the slices from G_2 were either reassigned or removed, we can use that space to place the slices from G_3 via the same procedure, and continue until placing the slices from G_g. We obtain a set of at most $2/(\varepsilon\delta^2)$ shifted tasks that were discarded.

If these sets of rounding and shifted tasks have total area at most $(\varepsilon/2)\text{area}(\text{OPT}(I))$, we end here as all the requirements from the lemma would be fulfilled; otherwise, we remove the sets of rounding and shifted tasks from the initial pile (still considering them as guessed tasks, which will later be scheduled just as they originally were), and redo the procedure with the remaining tasks with respect to the width of the surviving tasks, until finding a set of rounding tasks of total area at most $(\varepsilon/2)\text{area}(\text{OPT}(I))$. Since the sets of rounding and shifted tasks from each iteration are disjoint, the procedure must end after at most $2/\varepsilon$ iterations as the total area of the tasks is at most $\text{area}(\text{OPT}(I))$; consequently, the number of guessed tasks at the end of this procedure is at most $\frac{2}{\varepsilon}\left(\frac{2}{\varepsilon\delta}+\frac{2}{\varepsilon\delta^2}\right)\leq\frac{8}{\varepsilon^2\delta^2}$.

Let $\tilde{I}$ be the outcome of the aforementioned procedure. By construction, the number of possible heights in $\tilde{I}$ is at most $8/(\varepsilon^2\delta^2)$, corresponding to the number of guessed tasks. We can also observe that the total area of G_1 is at most $(\varepsilon/2)\text{area}(\text{OPT}(I))$; to see this, observe that, since every height is at least δC, it holds that $\text{area}(G_1)\leq(1/\delta)\text{area}(G_i)$ for any $i\in\{2,\dots,g-1\}$, and since there are $2/(\varepsilon\delta)$ groups with exactly $(\varepsilon\delta/2)W$ slices each, we can charge the total area of G_1 to $2/\varepsilon$ disjoint sets of $1/\delta$ groups from $G_2,\dots,G_g$. Thus, the total area of discarded tasks, corresponding to tasks in G_1 plus the rounding tasks, is at most $(\varepsilon/2)\text{area}(\text{OPT}(I))+(\varepsilon/2)\text{area}(\text{OPT}(I))\leq\varepsilon\cdot\text{area}(\text{OPT}(I))$. In conclusion, the obtained schedule for $\tilde{I}$ is feasible and has total area at least $(1-\varepsilon)\text{area}(\text{OPT}(I))$. ◀

3.2 Computing a schedule for rounded Tall tasks

Thanks to Lemma 2 applied to $\mathcal{T}$, it is possible to reduce ourselves to instances where tall tasks only have a constant number of possible heights, paying a negligible price on the approximation factor. Now, we need to retrieve a schedule for tall tasks. Suppose we had already selected the tasks included in the solution. In that case, this step can be done using a result due to Dean and Goemans in the context of *advertisement makespan minimization* [9]. In the latter, we are given a set of tasks and a timeline as in our problem, but the goal is to schedule all the given tasks while minimizing the maximum load in the schedule. The authors show that, when tasks have heights at least εM (where M is the optimal makespan) and have constant possible heights, the set of possible schedules can be explored in polynomial time. The following lemma presents guarantees in line with this result but also allows us to check whether a feasible schedule exists respecting the capacity constraints for a given set of tasks, which is required in our context.

▶ **Lemma 3.** *Let I be a set of tasks having a constant number of possible heights that can be scheduled so that each time slot $e\in\{1,\dots,T\}$ has total load at most $C(e)$ and contains a constant number of slices scheduled. Then, there exists a polynomial-time algorithm that computes a polynomial-sized set of solutions containing that solution, even if the values $C(e)$ are not given.*

Proof. Let $K\in O(1)$ be the number of possible heights in the instance, and $t\in O(1)$ be the maximum number of slices scheduled on any time slot. Similar to the classical dynamic program for makespan minimization (see, e.g., [29]), a *configuration* is a vector in $\{0,1,\dots,t\}^K$ specifying, for each possible height, how many slices of that height are scheduled in a given time slot; in particular, the number of possible configurations is at most $(t+1)^K\in O(1)$.

Consider now any feasible schedule for I such that at most t slices are scheduled into each time slot. The number of possible load configurations (i.e., vectors specifying in each coordinate how many time slots follow a given configuration) is at most $T^{(t+1)^K}$, and therefore, we can efficiently enumerate all possible load configurations. To conclude, we provide a way

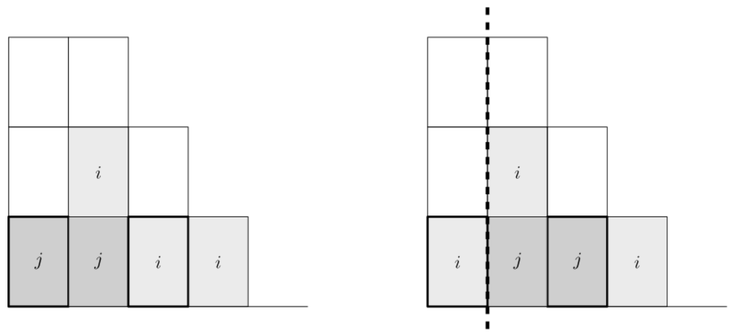

Figure 3 Exchange argument in the proof of Lemma 3. If two tasks i, j with $w_i > w_j$ are scheduled such that j goes to the leftmost time slot and i does not (left), there is a way to swap without losing feasibility. After all such possible exchanges, the inductive hypothesis can be applied to the remaining time slots (right).

to verify whether a given load configuration is feasible, i.e., if the tasks in I can be scheduled such that the solution follows the given load configuration. To see this, we prove that the following greedy algorithm correctly decides whether a load configuration is feasible:

(a) For each possible height, sort the time slots non-decreasingly according to the total load of slices of that height.

(b) For a fixed height, and considering the time slots from left to right, iteratively assign a slice from a compatible task (i.e., that has not been assigned yet to that time slot) having the largest remaining width.

If a feasible schedule exists for the tasks, we prove that this procedure computes such a schedule via a simple induction on the number of time slots in the load configuration. As a base case, if there is one time slot, since there is a feasible schedule, every task must have width 1 and hence the greedy procedure computes the solution. Consider now an instance with $k+1$ time slots, $k \geq 1$, where a feasible schedule exists. Consider the set of tasks whose slices are scheduled into the leftmost time slot. If these tasks have the largest widths in the instance (i.e. they are scheduled exactly as in the greedy procedure), then we can simply apply the inductive hypothesis to the remaining k time slots and the remaining slices. If it is not the case, then there exists some tasks i, j, with $w_i > w_j$, such that a slice of j was scheduled into the leftmost time slot but i was not. Since $w_i > w_j$, there must be some time slot where a slice of i was scheduled into but j was not, and hence we can swap them (see Fig. 3). Consequently, we can again apply the inductive hypothesis, concluding the proof. ◀

To apply Lemma 3, we need to define precisely which tasks we want to include in our solution. The following lemma shows that we can select a set of tasks whose total area is roughly the total area of tall tasks in the optimal solution, and that admits a feasible schedule.

▶ **Lemma 4.** *For any instance of the problem I, and any $\varepsilon \in (0, 1/4)$, there exists a subset of tall tasks $\hat{I}$ that can be selected in polynomial time, such that the following holds:*

(i) $\text{area}(\hat{I}) \geq (1-\varepsilon)\text{area}(\text{OPT}_{\mathcal{T}})$.

(ii) *$\hat{I}$ admits a feasible schedule whose load profile is upper bounded by the load profile of* $\text{OPT}_{\mathcal{T}}$.

Proof. Let $\tilde{I}$ be the set of rounded tall tasks from Lemma 2. Let $h_1, h_2, ..., h_K$ be the possible heights of tall tasks after rounding, and let $\mathcal{C}_1, ..., \mathcal{C}_K$ be the sets of tasks in $\tilde{I}$ of height h_i, for each $i \in \{1, ..., K\}$, respectively. For each $i \in \{1, \ldots, K\}$, let W_i be the total width of tasks from $\mathcal{C}_i$. We will approximately guess W_i, and then select tasks for our solution according to these widths.

We round W_i according to the following procedure. We guess $t \leq 1/(\varepsilon\delta) + 1$ tasks from $\mathcal{C}_i$, say $\{i_1, \ldots, i_t\}$ and consider a value of the form $\sum_{j=1}^{t-1} w_{i_j} + q \cdot w_{i_t}$, with $q \in \{0, 1, \ldots, n\}$. W_i will then be rounded down to the closest value W'_i of that form. As this requires guessing a constant number of tasks and a number that ranges from zero to n, this can be done in polynomial time.

We now prove that there exists a subset of tasks satisfying the requirements of the lemma such that, for each class $\mathcal{C}_i$, their total width is at most W'_i. If $|\mathcal{C}_i| \leq 1/(\varepsilon\delta) + 1$, then $W_i = W'_i$ and the proof follows. If not, consider the widest $1/\varepsilon + 1$ tasks in $\mathcal{C}_i$ (say $i_1, \ldots, i_t$), remove i_t from the set (which is the task of smallest width out of them), and then consider the largest possible value of the form $\sum_{j=1}^{t-1} w_{i_j} + q \cdot w_{i_t}$ that is not larger than W_i. Since i_t is the task of smallest width out of the widest ones, its total area is at most $\varepsilon \cdot \delta \cdot \text{area}(\mathcal{C}_i)$, proving the claim.

Since $\tilde{I}$ is not known, instead of guessing it exactly we do the following: As before, we guess the $1/(\varepsilon\delta) + 1$ widest tasks in $\mathcal{C}_i$, say $\{i_1, \ldots, i_t\}$, discard i_t, and greedily complete the set until reaching total width W'_i using tasks whose width is at most w_{i_t} that are not part of the guessed tasks. Observe that the total width of selected tasks from the group is at least $W'_i - w(i_t) \geq (1 - \varepsilon\delta)W'_i$. The union of the selected tasks will be our set $\hat{I}$.

For each group, suppose now that the guessed tasks are assigned as in the schedule for $\tilde{I}$, and the remaining tasks are iteratively scheduled into the leftmost time slot where they fit (i.e., where there is no slice from the same task already scheduled), using the time slots where original slices of that group were scheduled. Similarly to the proof of Lemma 2, this procedure schedules all the guessed tasks, and every considered task except for at most $1/\delta$ many, and hence the total width of scheduled tasks from the group is at least

$$(1 - \varepsilon)W'_i - \varepsilon W'_i \geq (1 - 2\varepsilon)W'_i \geq (1 - 3\varepsilon)W_i.$$

In conclusion, the total area of scheduled tasks is at least $(1 - 4\varepsilon)\text{area}(\text{OPT}_{\mathcal{T}})$ thanks to the guarantees from Lemma 2, the load profile is upper bounded by the load profile of the optimal solution restricted to tall tasks, and the set $\hat{I}$ can be computed in polynomial time as required. ◀

We can now put the pieces together to obtain our algorithm for tall tasks.

▶ **Lemma 5.** *For every $\varepsilon \in (0, 1/4)$, there exists a polynomial time algorithm that computes a solution for tall tasks with total area at least $(1 - \varepsilon)\text{area}(\text{OPT}_{\mathcal{T}})$.*

Proof. Thanks to Lemma 4, we now have a set of tall tasks with roughly the same total area as $\text{OPT}_{\mathcal{T}}$, that furthermore can be scheduled respecting its load profile. By means of Lemma 3, this solution can actually be computed, proving the result. ◀

3.3 PTAS for Short Tasks

In this section, we show the second component of our approximation algorithm, which is the PTAS for short tasks. We remark that our approximation guarantee for short tasks holds even for the case of general profits, i.e., weighted advertisement placement. We consider two main ideas:

Knapsack relaxation. We can define a relaxation of the problem through an instance of the classical one-dimensional knapsack problem. We define a knapsack instance with capacity $T \cdot C$ and, for each task i, we create an item whose size is $h_i \cdot w_i$ and profit is p_i. If we solve this instance, we obtain a set of tasks, say I^*, that maximizes its total profit while maintaining its total area below $T \cdot C$. The set I^* might not admit a feasible schedule, but it satisfies that $p(I^*) \geq p(\mathrm{OPT})$. If we run the PTAS for knapsack due to Lawler [25] on this relaxation, we obtain in polynomial time a set of tasks I' satisfying that $p(I') \geq (1-\varepsilon)p(\mathrm{OPT})$.

Least-Full (LF) algorithm. We consider an adaptation of Graham's list scheduling algorithm [15] for makespan minimization. The algorithm considers the tasks in any arbitrary but fixed order and iteratively places the slices of the tasks in the time slot with the lowest current load that does not have a copy of that task. The algorithm continues until a task cannot be included, or until scheduling every task in the instance.

Freund and Naor [11] prove that, if tasks obtained from the knapsack relaxation are sorted non-decreasingly by height and then scheduled according to LF, the solution obtained is $(3+\varepsilon)$-approximate even in the presence of general profits. More in detail, the set I^* can be partitioned into three subsets such that each one of them can be feasibly scheduled according to LF; the best of the three corresponds to the desired solution. We prove now that a similar scheme leads to a PTAS for short tasks under general profits. Our procedure works as follows:

1. We compute I^* by approximately solving the knapsack relaxation restricted to short tasks.
2. We sort the tasks in I^* non-increasingly according to $p_i/(h_i \cdot w_i)$.
3. We run the LF algorithm with respect to the previous order, and return the computed solution.

▶ **Lemma 6.** *For every $\varepsilon \in (0, 1/20)$, the previous procedure computes a feasible schedule with profit at least $(1-\varepsilon)p(\mathrm{OPT}_{\mathcal{S}})$ for the weighted advertisement placement problem.*

Proof. Observe first that, if the algorithm schedules the whole set I^*, we directly obtain the desired guarantee since $p(I^*) \geq (1-\varepsilon)p(\mathrm{OPT}_{\mathcal{S}})$. On the other hand, if our algorithm did not schedule all the tasks in I^*, we use the following result due to Freund and Naor: Suppose tasks are placed according to the LF algorithm (regardless of the order in which the tasks are considered), let $h_{\max}$ be the maximum height of a task considered so far. At any given moment during the assignment, it holds that $\ell(e) - \ell(e') \leq h_{\max}$ for any time slots $e, e' \in \{1, \ldots, T\}$. This implies that for each pair of time slots $e, e' \in \{1, \ldots, T\}$, $|\ell(e) - \ell(e')| \leq \delta C$, and therefore the total area of the tasks included in the solution is greater than or equal to $(1-2\delta)CT$, since $\max_{e \in \{1,\ldots,T\}} \ell(e) > (1-\delta)C$ (otherwise, we would have enough space to include the task that does not fit) and $\min_{e \in \{1,\ldots,T\}} \ell(e) \geq (1-2\delta)C$. This already proves the claim for the case of profits being equal to the area (assuming $\delta \leq \varepsilon/2$) because $\mathrm{area}(\mathrm{OPT}) \leq CT$; we will show that this works even for general profits.

Let ALG be the obtained solution. We use the previous facts to show that $p(\mathrm{ALG}) \geq p(I^*) \cdot (1-\varepsilon) \geq (1-2\varepsilon) \cdot p(\mathrm{OPT}_{\mathcal{S}})$. Let $1, \ldots, n^*$ be the tasks of I^* sorted non-increasingly according to $p_i/(w_i h_i)$. Let also k' be the last task that ALG included. We partition ALG into groups of total area at least εCT and at most $(1+\varepsilon)\varepsilon CT$ by iteratively taking tasks until attaining total area at least εCT. The number of groups that we obtain is at least

$$\frac{(1-\delta)CT}{(1+\varepsilon)\varepsilon CT} - 1 = \frac{1-\delta}{\varepsilon(1+\varepsilon)} - 1 = \frac{1-\varepsilon-\varepsilon^2-\delta}{\varepsilon(1+\varepsilon)} \geq \frac{1}{10\varepsilon},$$

where the last inequality holds for every $\varepsilon \leq 0.4$ and $\delta \leq \varepsilon/2$. Let $G_1, ..., G_g$ be the obtained groups, and let G_{g+1} be the tasks in $I^* \setminus \text{ALG}$. For each $i \in \{1, ..., g\}$, we have that $p(G_i) \geq p(G_{g+1})$; indeed, thanks to the order of the tasks, it holds that:

$$\begin{aligned} p(G_i) &= \sum_{j \in G_i} \frac{p_j}{h_j w_j} \cdot h_j w_j \\ &\geq \sum_{j \in G_i} \frac{p_{k'+1}}{h_{k'+1} w_{k'+1}} \cdot h_j w_j \\ &\geq \varepsilon CT \cdot \frac{p_{k'+1}}{h_{k'+1} w_{k'+1}} \geq \sum_{j \in G_{g+1}} \frac{p_{k'+1}}{h_{k'+1} w_{k'+1}} \cdot h_j w_j \geq \sum_{j \in G_{g+1}} p_j = p(G_{g+1}), \end{aligned}$$

where the third inequality holds since the total area of G_{g+1} is at most εCT, and the first and the last inequality follow from the tasks being sorted in non-increasing order according to $p_i/(h_i w_i)$. We now compare the total profit of ALG and $\text{OPT}_{\mathcal{S}}$. First, we have that $p(I^*) - p(\text{ALG}) = p(G_{p+1})$. Since $p(G_i) \geq p(G_{p+1})$ for every $i \in \{1, \ldots, g\}$, we have that $p(G_{g+1}) \leq (p(I^*) - p(G_{g+1}))/g$, which implies that $(g+1)p(G_{g+1}) \leq p(I^*)$, and hence that $p(G_{g+1}) \leq p(I^*)/(g+1) \leq 10\varepsilon \cdot p(I^*)$. In conclusion,

$$p(\text{ALG}) \geq (1 - 10\varepsilon)p(I^*) \geq (1 - 10\varepsilon)(1 - \varepsilon)p(\text{OPT}_{\mathcal{S}}) \geq (1 - 20\varepsilon)p(\text{OPT}_{\mathcal{S}}). \qquad \blacktriangleleft$$

We now have all the ingredients required to prove Theorem 1.

Proof of Theorem 1. We partition the given instance into short and tall tasks as described in the beginning of Section 3.1, using $\delta = \varepsilon/2$. We then run the algorithm for tall tasks from Lemma 5 on the tall tasks, and the algorithm for short tasks from Lemma 6 on the short tasks, and return the best of the two solutions. One of them must be a $(2 + \varepsilon)$-approximate solution for the problem. ◀

4 A PTAS under Resource Augmentation

The results from Section 3 provide an almost optimal way to schedule tall tasks and also a separate way to schedule short tasks; unfortunately, although the schedule for tall tasks leaves enough space for short tasks on top, our methods do not allow to construct a solution that merges both kinds of tasks. The main obstacle comes from the fact that slices of short tasks may be small in some time slots and large in others, requiring to treat parts of the same task in different ways.

To overcome this difficulty, we describe how to obtain almost optimal solutions under *resource augmentation*, meaning that our algorithm may overload the time slots up to capacity $(1+\varepsilon)C$, while the optimal solution cannot. In this case, as we show, it is possible to schedule short tasks on top of tall tasks via linear programming rounding techniques.

▶ **Theorem 7.** *For any $\varepsilon > 0$, there exists a $(1 + \varepsilon)$-approximation for advertisement placement under resource augmentation.*

Given constants $\delta_H, \delta_W > 0$, we classify the tasks as follows (see Fig. 4): A task i is *tall* if $h_i \geq \delta_H C$; a task i is *wide* if $h_i < \delta_H C$ and $w_i \geq \delta_W T$; and a task i is *small* if $h_i < \delta_H C$ and $w_i < \delta_W T$. We denote by $\mathcal{T}$, $\mathcal{W}$, and $\mathcal{S}$ the sets of tall, wide, and small tasks in the instance, respectively. Also, we define $\text{OPT}_{\mathcal{T}}, \text{OPT}_{\mathcal{W}}$, and $\text{OPT}_{\mathcal{S}}$ to be the optimal solution restricted to tall, wide, and small tasks, respectively. As mentioned before, we select and schedule tall tasks using Lemma 5; this ensures that the total area of tall tasks in the solution is at least $(1 - \varepsilon)\text{area}(\text{OPT}_{\mathcal{T}})$, and also that the load profile of the computed solution is upper bounded by the load profile of $\text{OPT}_{\mathcal{T}}$ (in particular, no time slot is overloaded at the moment).

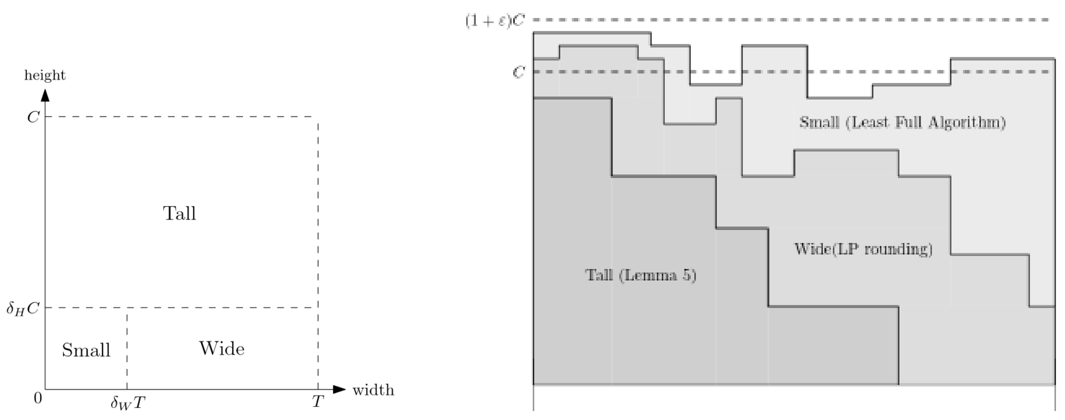

Figure 4 Left: Classification of the tasks for the proof of Theorem 7. Right: Structure of the computed solution from Theorem 7. While the schedule for tall tasks respects the capacity constraints, the placement of wide and small tasks may overload the time slots up to $(1+\varepsilon)C$.

Let $\mathrm{ALG}_{\mathcal{T}}$ be the solution computed so far. To include wide tasks, we slightly adapt the classical LP rounding procedure developed by Lenstra, Shmoys, and Tardos [26] for makespan minimization on unrelated machines. Given a set Q of tasks that can be feasibly scheduled, this procedure computes a feasible solution that includes all the given tasks while possibly overloading the time slots by at most $\max_{i\in Q}\{h_i\} \leq \delta_H C$. Thus, we first need a way to decide what tasks to include in the solution, which will be done again by rounding the coordinates of the tasks using linear grouping. Here, an instance is *horizontally segmented* if each task i of height h_i and width w_i is replaced by h_i tasks of height 1 and width w_i.

▶ **Lemma 8.** *For any instance of the problem I, there exists a subset of wide tasks $\hat{I}$ that can be selected in polynomial time, such that the following holds:*

(i) $\mathrm{area}(\hat{I}) \geq (1-\varepsilon)\mathrm{area}(\mathrm{OPT}_{\mathcal{W}})$.

(ii) *$\hat{I}$ horizontally segmented admits a feasible schedule whose load profile is upper bounded by the load profile of* $\mathrm{OPT}_{\mathcal{W}}$.

In what follows, we show how to prove Lemma 8. Observe first that, if $\mathrm{OPT}_{\mathcal{W}}$ is defined by at most $4/(\varepsilon^2\delta_W)$ tasks, we can guess exactly such a set in polynomial time. Thus, from now on, we assume that $\mathrm{OPT}_{\mathcal{W}}$ has more than $4/(\varepsilon^2\delta_W)$ wide tasks. We will also assume that the tasks are *horizontally segmented*, meaning that each task i of height h_i and width w_i is replaced by h_i tasks of height 1 and width w_i. It is not difficult to see that the optimal profit achieved with the horizontally segmented instance is an upper bound for the original optimal profit. We round up the widths of the wide tasks in the instance as follows:

(I) Suppose that $\mathcal{W} = \{1,\dots,t\}$ with $w_1 \geq w_2 \geq \cdots \geq w_t$. We guess a set of at most $4/(\varepsilon^2\delta_W)$ wide tasks, denoted as *guessed tasks* (the rest are non-guessed tasks). Out of the guessed tasks, we guess at most $2/(\varepsilon\delta_W)$ tasks, denoted as *rounding tasks*. We discard the latter.

(II) The set of non-guessed tasks with a smaller index than the rounding task with smallest index is discarded.

(III) For any remaining non-guessed task i, round w_i up to $w_{i'}$, $i' < i$, the closest width among the rounding tasks (breaking ties in favor of smaller indices).

Let G_i be the set of non-guessed tasks whose widths were rounded according to the i-th rounding task (sorted non-increasingly by width). As the following proposition shows, there exists a set of guessed tasks with a subset of rounding tasks that allow to bound the total area of discarded tasks by $\varepsilon \cdot \mathrm{area}(\mathrm{OPT}_{\mathcal{W}})$, while still having a feasible solution for the rounded tasks that were not discarded.

▶ **Proposition 9.** *Let I be a set of wide tasks, and I' be the output of the previous rounding procedure. There exists a set $\tilde{I} \subseteq I'$ of rounded wide tasks such that::*

(i) *The number of distinct widths in $\tilde{I}$ is constant.*

(ii) *The total area of $\tilde{I}$ is at least $(1-\varepsilon)\mathrm{area}(\mathrm{OPT}(I))$.*

(iii) *There exists a feasible schedule for $\tilde{I}$ horizontally segmented.*

Proof. Suppose we place all the tasks in $\mathrm{OPT}(I)$ one on top of the other, sorted non-increasingly by width. Since there is a feasible schedule for them and their width is at least $\delta_W T$, the total height of tasks in $\mathrm{OPT}(I)$, say H, is at most C/δ_W. From now on, we work with $\mathrm{OPT}(I)$ horizontally segmented. Starting from the widest horizontal segment in the pile, we partition them into groups $G_1, \ldots, G_g$ of total height exactly $(\varepsilon\delta_W/2)H$ each, except possibly for the last one, which can have a smaller height. Observe that we have at most $2/(\varepsilon\delta_W)$ groups. Our tentative set of rounding tall tasks from the rounding procedure will initially be defined by taking the task of smallest width from each group G_i, with $i \in \{1, \ldots, g-1\}$ (this set may contain less than $g-1$ tasks due to repetitions). Observe that, at this point, the set of rounding tasks and guessed tasks is the same, and tasks with smaller index than the rounding task with smallest index are not part of $\mathrm{OPT}(I)$, so they can be safely discarded.

We now prove that there is a feasible schedule for these (horizontally segmented) rounded tasks. Consider a feasible schedule for $\mathrm{OPT}(I)$. Since the first group, G_1, is discarded, we use the space due to this removal to place the horizontal segments from G_2 by simply replacing them one by one. Since the original schedule was feasible for G_1, it must be as well for the horizontal segments in G_2. Now that the slices from G_2 were reassigned, we can use that space to place the slices from G_3 in the same way, and continue until placing the slices from G_g. If this set of rounding tasks has total area at most $(\varepsilon/2)\mathrm{area}(\mathrm{OPT}(I))$, we end the argument here as all the requirements from the lemma would be fulfilled; otherwise, we remove the set of rounding tasks from the initial pile (still considering these tasks as guessed tasks, which will later be scheduled just as they originally were), and redo the procedure with the remaining tasks with respect to the width of the surviving tasks, until finding a set of rounding tasks of total area at most $(\varepsilon/2)\mathrm{area}(\mathrm{OPT}(I))$. Since the sets of rounding tasks from each iteration are disjoint, the procedure must end after at most $2/\varepsilon$ iterations as the total area of the tasks is at most $\mathrm{area}(\mathrm{OPT}(I))$; consequently, the number of guessed tasks at the end of this procedure is at most $4/(\varepsilon^2\delta_W)$.

Let $\tilde{I}$ be the outcome of the previous procedure. By construction, the number of possible heights in $\tilde{I}$ is at most $4/(\varepsilon^2\delta_W)$, corresponding to the number of guessed tasks. We can also observe that the total area of G_1 is at most $(\varepsilon/2)\mathrm{area}(\mathrm{OPT}(I))$; to see this, observe that, since every width is at least $\delta_W T$, it holds that $\mathrm{area}(G_1) \leq \mathrm{area}(G_i)/\delta_W$ for any $i \in \{2, \ldots, g-1\}$, and since there are $2/(\varepsilon\delta_W)$ groups with exactly $(\varepsilon\delta_W/2)H$ slices each, we can charge the total area of G_1 to $2/\varepsilon$ disjoint sets of $1/\delta_W$ groups from $G_2, \ldots, G_g$. Thus, the total area of discarded tasks, corresponding to tasks in G_1 plus the rounding tasks, is at most $(\varepsilon/2)\mathrm{area}(\mathrm{OPT}(I)) + (\varepsilon/2)\mathrm{area}(\mathrm{OPT}(I)) \leq \varepsilon \cdot \mathrm{area}(\mathrm{OPT}(I))$. As argued before, the obtained schedule for $\tilde{I}$ horizontally segmented is feasible and has total area at least $(1-\varepsilon)\mathrm{area}(\mathrm{OPT}(I))$, which concludes the proof. ◀

We now have the ingredients to prove Lemma 8.

Proof of Lemma 8. Let $\tilde{I}$ be the set of rounded wide tasks from Proposition 9. Let $h_1, h_2, ..., h_K$ be the possible widths of wide tasks after rounding. Let $\mathcal{C}_1, ..., \mathcal{C}_K$ be the sets of tasks in $\tilde{I}$ of width w_i, for each $i \in \{1, \dots, K\}$ respectively. For each $i \in \{1, \dots, K\}$, let H_i be the total height of tasks from $\mathcal{C}_i$. We will approximately guess H_i, and then select tasks to include in the desired solution according to these rounded total widths.

We round H_i according to the following procedure. We guess $t \leq 1/(\varepsilon\delta_W) + 1$ tasks from $\mathcal{C}_i$, say $\{i_1, \dots, i_t\}$ and consider a value of the form $\sum_{j=1}^{t-1} h_{i_j} + q \cdot h_{i_t}$, with $q \in \{0, 1, \dots, n\}$. H_i will then be rounded down to the closest value H'_i of that form. As this requires guessing a constant number of tasks and a number that ranges from zero to n, this can be done in polynomial time. We now prove that there exists a subset of tasks satisfying the requirements of the lemma such that, for each class $\mathcal{C}_i$, their total height is at most H'_i, by simply running the known PTAS for the one-dimensional knapsack problem [25]. For each task, we define an item of size and profit equal to h_i, and consider a knapsack of capacity H'_i. Since $\mathcal{C}_i \setminus \{i_t\}$ has total height at most H'_i, the PTAS on the created knapsack instance will find a set of tasks with total height at least $(1-\varepsilon)H'_i$, which can be feasibly scheduled (horizontally segmented) where $\mathcal{C}_i$ originally was, defining our set $\hat{I}$. ◀

Now that we have our set of wide tasks, we show how to efficiently schedule them, but possibly overloading the time slots by at most $\delta_H C$.

▶ **Lemma 10.** *Let I' be a set of horizontally segmented tasks that can be feasibly scheduled so that each time slot e has total load $\ell(e)$. There exists a polynomial time algorithm that computes a schedule for I' without horizontally segmenting such that, for each time slot e, the total load of tasks scheduled in it is at most $\ell(e) + \max_{i \in I'} h_i$.*

Proof. We consider the following feasibility program for scheduling I':

$$\begin{aligned}
\textstyle\sum_{e=1}^{T} x_{ie} &= w_i && \text{for every } i \in I', \\
\textstyle\sum_{i \in I'} x_{ie} h_i &\leq \ell(e) && \text{for every } e \in \{1, \dots, T\}, \\
x_{ie} &\in \{0, 1\} && \text{for every } i \in I' \text{ and } e \in \{1, \dots, T\}.
\end{aligned}$$

Here, x_{ie} is a binary variable encoding whether a slice of task i is scheduled into time slot e or not. Since I' horizontally segmented can be scheduled in such a way that each time slot e has total load $\ell(e)$, the linear relaxation of this program is feasible.

Let y be a feasible solution for the LP formulation. We can now round this solution by adapting the classical rounding scheme by Lenstra, Shmoys, and Tardos [26] (see also [9]). We construct a bipartite graph with nodes representing tasks on one side, and time slots on the other, and we have an edge from a task i to a time slot e if $0 < y_{ie} < 1$. We can assume that these edges form a forest; if there were alternating cycles, we can augment the flow in the cycle appropriately, preserving the amount assigned to any time slot, maintaining feasibility, until one of the values becomes either zero or one, breaking the cycle. The outdegree of each task must be at least 2 since the widths are integral, implying that we can always find an alternating path with endpoints at two different time slots e and e', such that both of them have indegree 1. We can again augment the flow along this path until every pair (i, e) involved becomes integral, which only increases the flow entering the endpoints of the path, by at most $h_i \leq \max_{i \in I'} h_i$. In conclusion, this rounding procedure returns a feasible integral schedule that increases the load of each time slot by at most $\max_{i \in I'} h_i$. ◀

We now have all the ingredients required to prove Theorem 7.

Proof of Theorem 7. Consider $\delta_H, \delta_W \leq \varepsilon/2$. By applying Lemmas 8 and 10, we can schedule wide tasks on top of $\mathrm{ALG}_{\mathcal{T}}$ in a way that the load of each time slot is at most $C + \delta_H C \leq (1+\varepsilon/2)\,C$. Let $\mathrm{ALG}_{\mathcal{T}\cup\mathcal{W}}$ be the solution constructed so far. Consider the small tasks in any arbitrary but fixed order, which will be scheduled according to the LF algorithm, i.e., scheduling each slice into the least loaded time slot that does not contain slices of the same task, until the first time the load of a time slot becomes larger than $(1+\varepsilon)C$ (or until scheduling all the small tasks). If all the small tasks are scheduled, the proof is finished; if not, this means that at least $(1-\delta_W)T$ time slots have total load at least $(1+\varepsilon/2)C$, and thus the total area of tasks scheduled so far would be at least $(1+\varepsilon/2)(1-\delta_W)TC \geq (1-\varepsilon)TC \geq (1-\varepsilon)\mathrm{area}(\mathrm{OPT})$, concluding the proof of Theorem 7. ◀

References

1 Micah Adler, Phillip B. Gibbons, and Yossi Matias. Scheduling space-sharing for internet advertising. *Journal of Scheduling*, 5(2):103–119, 2002. `doi:10.1002/jos.74`.

2 Soroush Alamdari, Therese Biedl, Timothy M. Chan, Elyot Grant, Krishnam Raju Jampani, Srinivasan Keshav, Anna Lubiw, and Vinayak Pathak. Smart-grid electricity allocation via strip packing with slicing. In *WADS*, volume 8037, pages 25–36, 2013. `doi:10.1007/978-3-642-40104-6_3`.

3 Alexander Armbruster, Fabrizio Grandoni, Edin Husic, Antoine Tinguely, and Andreas Wiese. On the approximability of unsplittable flow on a path with time windows. In *IPCO*, volume 15620, pages 29–42, 2025. `doi:10.1007/978-3-031-93112-3_3`.

4 Nikhil Bansal, Alberto Caprara, Klaus Jansen, Lars Prädel, and Maxim Sviridenko. A structural lemma in 2-dimensional packing, and its implications on approximability. In *ISAAC*, volume 5878, pages 77–86, 2009. `doi:10.1007/978-3-642-10631-6_10`.

5 Nikhil Bansal and Arindam Khan. Improved approximation algorithm for two-dimensional bin packing. In *SODA*, pages 13–25, 2014. `doi:10.1137/1.9781611973402.2`.

6 Henrik I. Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017. `doi:10.1016/J.COSREV.2016.12.001`.

7 Mauro R. C. da Silva, Rafael C. S. Schouery, and Lehilton L. C. Pedrosa. A polynomial-time approximation scheme for the MAXSPACE advertisement problem. In *LAGOS*, volume 346, pages 699–710, 2019. `doi:10.1016/J.ENTCS.2019.08.061`.

8 Wenceslas Fernandez de la Vega and George S. Lueker. Bin packing can be solved within 1+epsilon in linear time. *Combinatorica*, 1(4):349–355, 1981. `doi:10.1007/BF02579456`.

9 Brian C. Dean and Michel X. Goemans. Improved approximation algorithms for minimum-space advertisement scheduling. In *ICALP*, volume 2719, pages 1138–1152, 2003. `doi:10.1007/3-540-45061-0_87`.

10 Franziska Eberle, Felix Hommelsheim, Malin Rau, and Stefan Walzer. A tight $(3/2+\epsilon)$-approximation algorithm for demand strip packing. In *SODA*, pages 641–699, 2025. `doi:10.1137/1.9781611978322.20`.

11 Ari Freund and Joseph Naor. Approximating the advertisement placement problem. *Journal of Scheduling*, 7(5):365–374, 2004. `doi:10.1023/B:JOSH.0000036860.90818.5F`.

12 Waldo Gálvez, Fabrizio Grandoni, Afrouz Jabal Ameli, Klaus Jansen, Arindam Khan, and Malin Rau. A tight $(3/2+\epsilon)$-approximation for skewed strip packing. *Algorithmica*, 85(10):3088–3109, 2023. `doi:10.1007/S00453-023-01130-2`.

13 Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, Sandy Heydrich, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via L-packings. *ACM Transactions on Algorithms*, 17(4):33:1–33:67, 2021. `doi:10.1145/3473713`.

14 Waldo Gálvez, Fabrizio Grandoni, Arindam Khan, Diego Ramírez-Romero, and Andreas Wiese. Improved approximation algorithms for 2-dimensional knapsack: Packing into multiple l-shapes, spirals, and more. In *SOCG*, volume 189, pages 39:1–39:17, 2021. `doi:10.4230/LIPICS.SOCG.2021.39`.

15 Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17(2):416–429, 1969.

16 Fabrizio Grandoni, Salvatore Ingala, and Sumedha Uniyal. Improved approximation algorithms for unsplittable flow on a path with time windows. In *WAOA*, volume 9499, pages 13–24, 2015. doi:10.1007/978-3-319-28684-6_2.

17 Fabrizio Grandoni, Tobias Mömke, and Andreas Wiese. A PTAS for unsplittable flow on a path. In *STOC*, pages 289–302, 2022. doi:10.1145/3519935.3519959.

18 Rolf Harren, Klaus Jansen, Lars Prädel, and Rob van Stee. A $(5/3 + \epsilon)$-approximation for strip packing. *Computational Geometry*, 47(2):248–267, 2014. doi:10.1016/J.COMGEO.2013.08.008.

19 Klaus Jansen and Malin Rau. Closing the gap for pseudo-polynomial strip packing. In *ESA*, volume 144, pages 62:1–62:14, 2019. doi:10.4230/LIPICS.ESA.2019.62.

20 Klaus Jansen, Malin Rau, and Malte Tutas. Hardness and tight approximations of demand strip packing. In *SPAA*, pages 479–489, 2024. doi:10.1145/3626183.3659971.

21 Klaus Jansen and Roberto Solis-Oba. Rectangle packing with one-dimensional resource augmentation. *Discrete Optimization*, 6(3):310–323, 2009. doi:10.1016/J.DISOPT.2009.04.001.

22 Edward G. Coffman Jr., M. R. Garey, and David S. Johnson. Bin packing with divisible item sizes. *Journal of Complexity*, 3(4):406–428, 1987. doi:10.1016/0885-064X(87)90009-4.

23 Claire Kenyon and Eric Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25(4):645–656, 2000. doi:10.1287/MOOR.25.4.645.12118.

24 Ariel Kulik, Matthias Mnich, and Hadas Shachnai. Improved approximations for vector bin packing via iterative randomized rounding. In *FOCS*, pages 1366–1376, 2023. doi:10.1109/FOCS57990.2023.00084.

25 Eugene L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979. doi:10.1287/MOOR.4.4.339.

26 Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990. doi:10.1007/BF01585745.

27 Lehilton L. C. Pedrosa, Mauro R. C. da Silva, and Rafael C. S. Schouery. Approximation algorithms for the MAXSPACE advertisement problem. *Theory of Computing Systems*, 68(3):571–590, 2024. doi:10.1007/S00224-024-10170-2.

28 Shaojie Tang, Qiuyuan Huang, Xiang-Yang Li, and Dapeng Wu. Smoothing the energy consumption: Peak demand reduction in smart grid. In *INFOCOM*, pages 1133–1141, 2013. doi:10.1109/INFCOM.2013.6566904.

29 Vijay V. Vazirani. *Approximation algorithms.* Springer, 2001.

30 Sean Yaw, Brendan Mumey, Erin McDonald, and Jennifer Lemke. Peak demand scheduling in the smart grid. In *IEEE SmartGridComm*, pages 770–775, 2014. doi:10.1109/SMARTGRIDCOMM.2014.7007741.