Approximation algorithms for the MAXSPACE advertisement problem

Mauro R. C. da Silva[1], Lehilton L. C. Pedrosa[1], and Rafael C. S. Schouery[1]

[1]Institute of Computing, University of Campinas
maurorcsc@gmail.com, {lehilton, rafael}@ic.unicamp.br

May 10, 2023

Abstract

In MAXSPACE, given a set of ads $\mathcal{A}$, one wants to schedule a subset $\mathcal{A}' \subseteq \mathcal{A}$ into K slots $B_1, \ldots, B_K$ of size L. Each ad $A_i \in \mathcal{A}$ has a *size* s_i and a *frequency* w_i. A schedule is feasible if the total size of ads in any slot is at most L, and each ad $A_i \in \mathcal{A}'$ appears in exactly w_i slots and at most once per slot. The goal is to find a feasible schedule that maximizes the sum of the space occupied by all slots. We consider a generalization called MAXSPACE-R for which an ad A_i also has a release date r_i and may only appear in a slot B_j if $j \geq r_i$. For this variant, we give a 1/9-approximation algorithm. Furthermore, we consider MAXSPACE-RDV for which an ad A_i also has a deadline d_i (and may only appear in a slot B_j with $r_i \leq j \leq d_i$), and a value v_i that is the gain of each assigned copy of A_i (which can be unrelated to s_i). We present a polynomial-time approximation scheme for this problem when K is bounded by a constant. This is the best factor one can expect since MAXSPACE is strongly NP-hard, even if $K = 2$.

keywords: Approximation Algorithm, PTAS, Scheduling of Advertisements, MAXSPACE.

1 Introduction

Many websites (such as Google, Yahoo!, Facebook, and others) offer free services while displaying advertisements (or ads) to users. Each website often has a single strip of fixed height, which is reserved for scheduling ads, and the set of displayed ads changes on a time basis. For such websites, advertisement is the primary source of revenue. Thus, it is essential to find the best way to dispose the ads in the available time and space while maximizing the revenue [15].

The revenue from web advertising grew considerably in the 21st century. In 2022, the total revenue was US$209.7 billion, an increase of 10.8% from the previous year. It is estimated that web advertising comprised 52% of all advertising spending, overtaking television advertising. In 2022, banners and search engine ads comprised 70.5% of internet advertising, representing a revenue of US$147.9 billion [10]. Web advertising has created a multi-billionaire industry where algorithms for scheduling advertisements play an important role.

Websites like Facebook and Mercado Livre (a large Latin American marketplace) use banners to display advertisements while users browse. Google displays ads sold through Google Ad Words in its search results within a limited area, in which ads are in text format and have sizes that vary according to the price (see Figure 1).

Figure 1: Example of search engine homepage with ads with variable sizes displayed as results in a limited space, represented by the dotted area.

We consider the class of Scheduling of Advertisements problems introduced by Adler et al. [1], where, given a set $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ of advertisements, the goal is to schedule a subset $\mathcal{A}' \subseteq \mathcal{A}$ into a banner in K equal time-intervals. The set of ads scheduled to a particular time interval j, $1 \leq j \leq K$, is represented by a set of ads $B_j \subseteq \mathcal{A}'$, which is called a *slot*. Each ad A_i has a *size* s_i and a *frequency* w_i associated with it. The size s_i represents the amount of space A_i occupies in a slot, and the frequency $w_i \leq K$ represents the number of slots that should contain a copy of A_i. An ad A_i can be displayed at most once in a slot, and A_i is said to be *scheduled* if w_i copies of A_i appear in slots with at most one copy per slot [1, 5].

The main problems in this class are MINSPACE and MAXSPACE. In MINSPACE, all ads have to be scheduled, and the goal is to minimize the fullness of the fullest slot. In MAXSPACE, an upper bound L is specified, representing each slot's size. A feasible solution for this problem is a schedule of a subset $\mathcal{A}' \subseteq \mathcal{A}$ into slots $B_1, B_2, \ldots, B_K$, such that each $A_i \in \mathcal{A}'$ is scheduled and the fullness of any slot does not exceed the upper bound L, that is, for each slot B_j, $\sum_{A_i \in B_j} s_i \leq L$. MAXSPACE aims to maximize the slots' fullness, defined by $\sum_{A_i \in \mathcal{A}'} s_i w_i$. Both problems are strongly NP-hard [1, 5].

Even though these problems where introduced with advertisement in mind, since MAXSPACE and MINSPACE are packing problems, they can be applied to pack several kinds of items into bins or slots. For example, a solution for this problem can populate the columns of the social photo-sharing network Pinterest [16] and other sites with the same kind of layout (called *grid layout*).

1.1 Previous Works

In the literature, there are works regarding approximation and exact algorithms for MINSPACE and MAXSPACE. Also, some special cases of these problems where defined by Dawande et al. [5]. In MAX_w, every ad has the same frequency w. In $\text{MAX}_{K|w}$, every ad has the same frequency w, and the number of slots K is a multiple of w. Moreover, in MAX_s, every ad has the same size s. Analogously, they define three special cases of MINSPACE: MIN_w, $\text{MIN}_{K|w}$ and MIN_s.

Regarding approximation algorithms for MAXSPACE, Adler et al. [1] present a $\frac{1}{2}$-approximation when the

ad sizes form a sequence $s_1 > s_2 > \cdots > s_n$, such that for all i, s_i is a multiple of s_{i+1}. Dawande et al. [5] present three approximation algorithms: a $(\frac{1}{4} + \frac{1}{4K})$-approximation for MAXSPACE, a $\frac{1}{3}$-approximation for MAX_w and a $\frac{1}{2}$-approximation for $\text{MAX}_{K|w}$. Freund and Naor [7] proposed a $(\frac{1}{3} - \varepsilon)$-approximation for MAXSPACE and a $(\frac{1}{2} - \varepsilon)$-approximation for the special case in which the size of the ads are in the interval $[L/2, L]$.

For MINSPACE, Adler et al. [1] present a 2-approximation called *Largest-Size Least-Full* (LSLF) which is also a $(\frac{4}{3} - \frac{w}{3K})$-approximation to $\text{MIN}_{K|w}$ [5]. Dawande et al. [5] present a 2-approximation for MINSPACE using *LP Rounding*, and Dean and Goemans [6] present a $\frac{4}{3}$-approximation for MINSPACE using Graham's algorithm for schedule [9].

From the exact-algorithm standpoint, Kaul et al. [12] present an integer programming model for placing advertisements optimally in a two-dimensional banner. Kim and Moon [14] present a variant of MAXSPACE with a new objective function that includes factors that influence advertising effectiveness in terms of click-through rate. They provide an integer programming model and two meta-heuristics for this problem.

1.2 Our results

In practice, the time interval relative to each slot in scheduling advertising can represent minutes, seconds, or long periods, such as days and weeks. One often considers the idea of *release dates* and *deadlines*. An ad's release date indicates the beginning of its advertising campaign. Analogously, the deadline of an ad indicates the end of its advertising campaign. For example, ads for Christmas must be scheduled before December 25th.

With this in mind, we consider a MAXSPACE generalization called MAXSPACE-R in which each ad A_i has one additional parameter, a release date $r_i \geq 1$. The release date of ad A_i represents the first slot where a copy of A_i can be scheduled; that is, a copy of A_i cannot be scheduled in a slot B_j with $j < r_i$. In MAXSPACE-R, we assume that the frequency of each ad A_i is compatible with its release date, that is, $K - r_i + 1 \geq w_i$.

Notice that in the original MAXSPACE and in MAXSPACE-R, the value of an ad corresponds to the space it occupies multiplied by the number of times it appears. In practice, the value of an ad can be influenced by other factors, such as the expected number of clicks it generates for the advertiser [2]. The number of times the ad appears can also be influenced by other factors, such as the budget provided by the advertiser.

In order to consider that the value of the ad is not necessarily related to its size and to consider deadlines, we also consider a MAXSPACE-R generalization called MAXSPACE-RDV in which each ad A_i has a deadline $d_i \leq K$ and a value v_i. Similarly to the release date, the deadline of an ad A_i represents the last slot where we can schedule a copy of A_i; thus a_i cannot be scheduled in a slot B_j with $j > d_i$. We assume that the frequency of each ad A_i is compatible with its release date and deadline, that is, $w_i \leq d_i - r_i + 1$. In MAXSPACE-RDV, each assigned copy of an ad A_i also has a value v_i, and the value of a solution is the sum of $v_i w_i$ for each scheduled ad A_i. Note that v_i can be unrelated to the size s_i of A_i.

Let Π be a maximization problem. A family of algorithms $\{H_\varepsilon\}$ is a *Polynomial-Time Approximation Scheme* (PTAS) for Π if, for every constant $\varepsilon > 0$, H_ε is a $(1-\varepsilon)$-approximation for Π [17]. A *Fully Polynomial-Time Approximation Scheme* (FPTAS) is a PTAS whose running time is also polynomial in $1/\varepsilon$. Notice that MAXSPACE does not admit an FPTAS even for $K = 2$, since it generalizes the *Multiple Subset Sum Problem* with identical capacities (MSSP-I), which does not admit an FPTAS even for $K = 2$ [13].

In a previously published conference paper [4], we proposed a PTAS for MAXSPACE-RD with bounded K, which is the particular case of MAXSPACE-RDV where $v_i = s_i$ for every ad A_i. We improve this result, using a different technique, by presenting a PTAS for MAXSPACE-RDV. We also present a 1/9-approximation algorithm for MAXSPACE-R (where K is not necessarily bounded by a constant).

In the 1/9-approximation to MAXSPACE-R, we divide the ads into large, medium, and small. We create an optimal and polynomial dynamic programming algorithm for large ads based on the classic dynamic programming for the Knapsack Problem [13], and we use algorithms based on the best-fit heuristic to allocate medium and small ads. We also execute a step based on a local search for small ads to relocate them between slots when possible. As our problem has release dates, we cannot use the area bounds as in previous works for MAXSPACE since an optimal solution does not necessarily have a good slot fullness. Thus, it is necessary to compare the algorithm solution with the ads' allocation in an optimal solution to show the approximation factors.

In the PTAS for MAXSPACE-RDV, since the number of slots is bounded by a constant, we enumerate the most valuable ads on the solution and all possible solutions involving these ads in polynomial time. To schedule the other ads, we use a linear program algorithm to obtain a relaxed allocation of ads to slots. We give an algorithm that rounds off the fractional assignment, showing that the losses are small. This algorithm is inspired by the approximation scheme presented by Frieze et al. [8] for the m-dimensional knapsack problem. However,

in MAXSPACE, the ads have copies; thus, assigning and rounding the less valuable advertisements requires different and more elaborate techniques. We hope these strategies can be adapted to similar packing problems, specifically with release dates and/or deadlines.

In Section 2, we present a 1/9-approximation algorithm for MAXSPACE-R, and in Section 3, we present a PTAS for MAXSPACE-RDV with a constant number of slots. In Section 4, we discuss the results and future works.

2 A 1/9-approximation for MAXSPACE-R

This section presents a 1/9-approximation algorithm for MAXSPACE-R. For this problem, we assume that K is polynomially bounded, as otherwise, the size of the solution is not polynomially bounded. We also assume that $L = 1$ and $0 < s_i \leq 1$ for each $A_i \in \mathcal{A}$. We partition the ads into three sets: the set $G = \{A_i \in \mathcal{A} \mid s_i > 1/2\}$ of large ads, the set $M = \{A_i \in \mathcal{A} \mid 1/4 < s_i \leq 1/2\}$ of medium ads, and the set $P = \{A_i \in \mathcal{A} \mid s_i \leq 1/4\}$ of small ads.

Let S denote a feasible solution $\mathcal{A}' \subseteq \mathcal{A}$ scheduled into slots $B_1, B_2, \ldots, B_K$. Then the *fullness* of a slot B_j is defined as $f(B_j) = \sum_{A_i \in B_j} s_i$. Also, the fullness of solution S is $f(S) = \sum_{j=1}^{K} f(B_j)$.

In Section 2.1, we present an exact algorithm for large ads; in Section 2.2, we present a 1/4-approximation for medium ads; and, in Section 2.3, we present a 1/4-approximation for small ads. Moreover, in Section 2.4, we combine these algorithms to obtain a 1/9-approximation for the whole set of ads $\mathcal{A}$.

2.1 An exact algorithm for large ads

We present an exact polynomial-time algorithm for large ads based on the dynamic programming algorithm for the Binary Knapsack Problem [13].

An instance of the Binary Knapsack Problem consists of a container with capacity W and a set I of items. Each item $i \in I$ has a profit v_i and a weight p_i. The goal is to find a subset $I' \subseteq I$ that maximizes the total profit and such that the sum of the weights does not exceed the container's capacity, that is, $\sum_{i \in I'} p_i \leq W$.

We say that an ad A_i appears *sequentially* in a schedule S if, for each pair of slots B_j and B_k that have copies of A_i in S, there is a copy of A_i in each slot B_ℓ of S, with $j \leq \ell \leq k$. Notice that an ad that has only one copy is always sequentially scheduled.

Lemma 1. *Let S be a feasible schedule with ads of G in K slots and let $\mathcal{A}' \subseteq G$ be the set of ads scheduled in S. There is a feasible schedule S' in which all ads of $\mathcal{A}'$ appear sequentially and in non-decreasing order of release dates, that is, if some ad A_i appears before an ad A_j then $r_i \leq r_j$.*

Proof. Note that it is not possible to add more than one copy of any ad per slot since $s_i > 1/2$ for all $A_i \in G$. Let S' be a schedule of $\mathcal{A}'$ in which the number of ads that appear sequentially is maximum. Assume by contradiction that there is an ad A_i in S' that does not appear sequentially. Let X be the set of ads with at least one copy between the first and last copies of A_i in S'. Figure 2 shows the schedule of S'.

Figure 2: Schedule of ads of S'. In blue is the ad A_i and in red the ads of set X. The rest of colors represent the other ads in this schedule.

Exchange the last copies of A_i with copies of ads in X such that A_i appears sequentially in the solution maintaining the order of X. Since the scheduling of copies of ads in X is delayed, their release dates are respected. Moreover, each copy of A_i is moved to some slot after the first copy of A_i. Thus, the release date of A_i is also respected. Therefore, the modified schedule is feasible, as seen in Figure 3. However, the number of ads sequentially scheduled increases, which contradicts the assertion that S' has a maximum number of ads sequentially scheduled.

Now, assume there are copies of ads in S', which are not ordered by release dates. Then, let A_i be an ad whose copies are scheduled immediately after the copies of an ad A_j with $r_j > r_i$. Let B_k be the slot in which the first copy of A_j appears in S', then $r_j \leq k$, and thus $r_i \leq k$. We exchange the order of the ads A_i and A_j, that is, we move each copy of A_i scheduled in a slot B_z to a slot B_{z-w_j}, and each copy of A_j scheduled in a

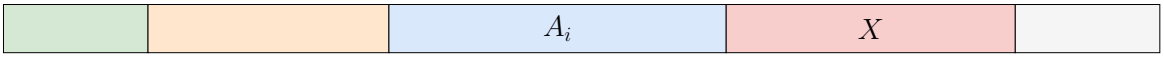

Figure 3: Schedule of ads of S' after move copies of A_i to appears sequentially.

slot B_z to a slot B_{z+w_i}. Since the first copy of A_i is scheduled in slot B_k, and copies of A_j are only delayed, the modified schedule is feasible. By repeating this process, we obtain a schedule in which all ads appear in non-decreasing order of release dates. □

In Lemma 1, we show that, given a feasible schedule S, it is possible to construct a feasible schedule S' with the same set of ads in S, but so that all ads appear sequentially in slots and ordered by release dates. Thus, given an instance for MAXSPACE-R with large ads G, we built an instance of the Knapsack Problem, in which each ad A_i of G is an item j with weight $p_j = w_i$ and profit $v_j = s_i w_i$. Since it is not possible to add more than one ad of G per slot, we can ignore the height dimension of slots, given by L, and use only the width dimension, given by the number of slots K. We define the knapsack capacity as $W = K$. In Algorithm 1, we present a dynamic programming algorithm for the binary knapsack problem with release date restrictions. In this algorithm, $m[i, j]$ corresponds to the optimal value of scheduling a set of i ads with the smallest release dates to the first j slots.

Algorithm 1 Dynamic programming algorithm for large ads.

```
1: procedure DP(G)
2:     creates a matrix m[0 ... |G|][0 ... K]
3:     for j ← 1, ..., K do
4:         m[0, j] ← 0
5:     for i ← 1, ..., |G| do
6:         m[i, 0] ← 0
7:     for each A_i ∈ G in non-decreasing order of r_i do
8:         x ← r_i + w_i − 1
9:         for j ← 1, 2, ..., x − 1 do
10:            m[i, j] ← m[i − 1, j]
11:        for j ← x, x + 1, ..., K do
12:            m[i, j] ← max{m[i − 1, j], m[i − 1, j − w_i] + s_i w_i}
13:    backtrack in matrix m and return the solution
14: end procedure
```

Lemma 2. *Algorithm 1 runs in polynomial time in the instance size, returns a feasible solution, and is optimal for G.*

Proof. The time complexity of the algorithm is $O(|G|K + |G| \lg |G|)$, which is polynomial since K is polynomially bounded.

For each ad A_i, the algorithm considers only feasible sequential schedules. If A_i is in the computed solution, it has exactly w_i copies in distinct compatible slots. Thus, the algorithm always returns a feasible solution.

By Lemma 1, any feasible schedule for G can be modified to obtain a new schedule in which the same ads appear sequentially in the slots. Then, let Opt be an optimal schedule for G. There is a schedule Opt' in which all ads of Opt appears sequentially and $f(Opt) = f(Opt')$. The schedule Opt' induces a feasible solution for the knapsack problem defined since every ad appears sequentially as an item of the knapsack problem with weight w_i and the capacity of the knapsack is not violated since the algorithm adds at most a copy per slot and $W = K$.

A similar argument to the one used by Kellerer et al. [13] to prove the optimality of the dynamic programming algorithm for the binary knapsack problem can be used to prove that Algorithm 1 finds an optimal solution for the knapsack problem with release dates and, thus, for MAXSPACE-R with large ads. □

2.2 A 1/4-approximation algorithm for medium ads

In this section, we present an algorithm for ads with medium size. We show this algorithm is a 1/4-approximation for ads of M (Lemma 7).

Algorithm 2 Algorithm for medium ads.

```
1: procedure ALG_MEDIUM(M)
2:     for j ← 1, ..., K do
3:         B_j ← ∅
4:     for each A_i ∈ M in non-decreasing order of r_i do
5:         X ← ∅
6:         for k ← 1, ..., w_i do
7:             if there exists j ∉ X with j ≥ r_i and B_j is empty then
8:                 j ← min{j' | j' ∉ X, j' ≥ r_i and B_j' is empty}
9:                 X ← X ∪ {j}
10:            else if there exists j ∉ X with j ≥ r_i and f(B_j) ≤ 1 − s_i then
11:                j ← arg min{f(B_j') | j' ∉ X, j' ≥ r_i}
12:                X ← X ∪ {j}
13:            else
14:                discard A_i and continue at Line 4
15:        add a copy of A_i to B_j for each j ∈ X
16:    return {B_1, B_2, ..., B_K}
17: end procedure
```

The idea behind Algorithm 2 is to try to add the ads to the least full compatible slots. It receives as input a set of medium ads M and iterates over them in order of release date (from smallest to highest). For each copy of an ad A_i, the algorithm finds the first empty slot compatible with r_i. If such a slot exists, the algorithm adds a copy of A_i to it. Otherwise, the algorithm finds the least full slot compatible with r_i. A set X is used to maintain the slots to which A_i was assigned. If it is possible to assign all w_i copies of A_i, the slots in X are updated. Otherwise, the ad A_i is discarded, and the algorithm goes to the next ad. The algorithm returns a schedule of ads to the slots.

Consider the output of Algorithm 2 and let $Opt = \{B_1^*, B_2^*, \ldots, B_K^*\}$ be an optimal schedule. Also, let H be the set of ads not scheduled by the algorithm and let H^* be the subset of ads in H that are in Opt. In Lemma 3, we show that Algorithm 2 runs in polynomial time in the instance size and returns a feasible solution. In Lemma 4, we show that if there is an ad A_i in an optimal schedule that was not scheduled by the algorithm, that is, $A_i \in H^*$, then each slot B_j such that $j \geq r_i$ has fullness $f(B_j) \geq 1/4$ in the solution returned by the algorithm. The Lemma 4 is used to prove that this algorithm is a 1/4-approximation for medium ads (Lemma 7).

Lemma 3. *Algorithm 2 runs in polynomial time in the instance size and returns a feasible solution.*

Proof. Sorting the ads by release date in the loop of Line 4 can be done in polynomial time. Finding the slots to assign the ads takes time $O(K)$. Therefore, the complexity of this algorithm is $O(K \sum_{A_i \in M} w_i + |M| \lg |M|)$, which is polynomial since $w_i \leq K$.

The algorithm adds a copy of an ad A_i only to compatible slots. Moreover, A_i is added only if exactly w_i copies of A_i can be added to compatible slots. Therefore, the algorithm returns a feasible solution. □

Lemma 4. *Let $A_i \in H^*$ and let Z be the set of slot indices j such that $j \geq r_i$. For each $j \in Z$, $f(B_j) \geq 1/4$.*

Proof. Consider $B_1, \ldots, B_K$ at the moment in which the algorithm tries to add an ad $A_i \in H^*$. As A_i was not added, there exists at least a slot B_j of Z whose fullness is greater than 1/2, since $s_i \leq 1/2$. Then, it follows that B_j has at least 2 ads. Let $A_{i'}$ be the last ad assigned to B_j until this moment. Then, at the moment that $A_{i'}$ was assigned, the fullness of B_j was at least 1/4 since it had at least one medium ad. Thus, the copy of $A_{i'}$ assigned to B_j corresponds to the case of Line 10 of the algorithm. Note that $r_{i'} \leq r_i$, by the order in which the algorithm considered the ads. Therefore, it follows that all slots $B_{j'}$, with $j' \geq r_i$, were considered in the case of Line 7 to assign $A_{i'}$, and no such a slot satisfied this case's criteria, so each of these slots had fullness at least 1/4. □

2.3 A 1/4-approximation algorithm for small ads

In this section, we present an algorithm which, later on, we prove that it is a 1/4-approximation for small ads (Lemma 7).

Algorithm 3 Algorithm for small ads.

```
1: procedure ALG_SMALL(P)
2:     for j ← 1, ..., K do
3:         B_j ← ∅
4:     for each A_i ∈ P in non-decreasing order of r_i do
5:         X ← ∅
6:         for k ← 1, ..., w_i do
7:             if there exists j ∉ X with j ≥ r_i and f(B_j) < 1/4 then
8:                 j ← min{j' | j' ∉ X, j' ≥ r_i and f(B_j') < 1/4}
9:                 X ← X ∪ {j}
10:            else if there exists j_1 ∈ X, j_2 ∉ X with j_1, j_2 ≥ r_i, f(B_j1) < 1/4, and f(B_j2) ≥ 3/4 then
11:                j_1 ← min{j' | j' ∈ X, j' ≥ r_i and f(B_j') < 1/4}
12:                j_2 ← min{j' | j' ∉ X, j' ≥ r_i and f(B_j') ≥ 3/4}
13:                find T ⊂ B_j2 such that 1/4 ≤ f(T) ≤ 1/2 and T ∩ B_j1 = ∅
14:                move T to B_j1
15:                X ← X ∪ {j_2}
16:            else if there exists j ∉ X with j ≥ r_i and f(B_j) ≤ 1 − s_i then
17:                j ← arg min{f(B_j') | j' ∉ X, j' ≥ r_i}
18:                X ← X ∪ {j}
19:            else
20:                discard A_i and continue at Line 4
21:        add a copy of A_i to B_j for each j ∈ X
22:    return {B_1, B_2, ..., B_K}
23: end procedure
```

The idea behind Algorithm 3 is to try to add the ads to the least full compatible slots and, when it is not possible, try to move ads from a slot with high fullness to a slot with low fullness. It receives as input a set of small ads P and iterates over it in order of release dates, from the smallest to the highest release date.

For each copy of an ad A_i, the algorithm looks up for the first slot with fullness smaller than $1/4$ compatible with the release date r_i of A_i. If such a slot exists, the algorithm adds a copy to it. Otherwise, it tries to find two slots B_{j_1} and B_{j_2} which are compatible with A_i and such that B_{j_1} has a copy of A_i and fullness smaller than $1/4$, and B_{j_2} has no copy of A_i and has fullness at least $3/4$. If such slots B_{j_1} and B_{j_2} are found, the algorithm moves a subset of ads T from B_{j_2} to B_{j_1}. The set T must have no intersection with B_{j_1} (that is, the ads of T do not appear in B_{j_1}) and have fullness of at least $1/4$ and at most $1/2$. Note that it is always possible to find such a subset of B_{j_2}. To see this, observe that at least $1/2$ of the fullness of B_{j_2} is composed by ads that are not in B_{j_1} since the fullness of B_{j_1} is at most $1/4$. From this subset with fullness at least $1/2$, it is possible to find a subset of fullness of at least $1/4$ and at most $1/2$, since the ads are small. The algorithm then moves T from B_{j_2} to B_{j_1} and add a copy of A_i to B_{j_2}. Note that this movement does not violate any restriction of release dates since the ads of T have release dates at most r_i. When no such a pair B_{j_1} and B_{j_2} is found, the algorithm tries to add a copy of A_i to the first slot where it fits.

A set X is used to maintain the slots to which A_i was assigned. If it is possible to assign all w_i copies of A_i, the slots in X are updated. Otherwise, X is ignored, and the algorithm goes to the next ad. The algorithm returns a schedule of ads to the slots.

Consider the output of Algorithm 3 and let $Opt = \{B_1^*, B_2^*, \ldots, B_K^*\}$ be an optimal schedule. Also, let H be the set of ads not scheduled by the algorithm and let H^* be the subset of ads in H that are in Opt. In Lemma 5, we show that Algorithm 3 runs in polynomial time in the instance size and returns a feasible solution. In Lemma 6, we show that if there exists some ad A_i that is in an optimal schedule but was not scheduled by the algorithm, that is, $A_i \in H^*$, then each slot B_j such that $j \geq r_i$ has fullness $f(B_j) \geq 1/4$ in the solution returned by the algorithm. The Lemma 6 is used to prove that this algorithm is a $1/4$-approximation for small ads (Lemma 7).

Lemma 5. *Algorithm 3 runs in polynomial time in the instance size and returns a feasible solution.*

Proof. Sorting the ads by release date in the loop of Line 4 can be done in polynomial time. Finding the slots to assign the ads takes time $O(K)$, and changing the ads from slots takes time $O(K|P|)$. Therefore, the complexity of this algorithm is $O(K|P| \sum_{A_i \in P} w_i + |P| \lg |P|)$, which is polynomial since $\sum_{A_i \in P} w_i \in O(K|P|)$.

The algorithm adds a copy of an add A_i only to compatible slots. Besides that, A_i is scheduled only if exactly w_i copies of A_i can be added to compatible slots. When the algorithm moves a set of ads from a slot B_{j_2} to a slot B_{j_1}, it does not violate any restriction of release dates since the ads in T have release dates smaller than or equal to the release date of the ad considered by the iteration, by the order in which the ads are considered. Also, the algorithm does not violate the fullness of any slot since $f(B_{j_1}) < 1/4$ and $f(T) \leq 1/2$. The set of ads $T \subset B_{j_2}$ is not in B_{j_1}, then the restriction of each ad has at most a copy per slot is also not violated when the algorithm moves T from B_{j_2} to B_{j_1}. Therefore, the algorithm returns a feasible solution. □

Lemma 6. *Let $A_i \in H^*$ and let Z be the set of indices j such that $j \geq r_i$. For each $j \in Z$, $f(B_j) \geq 1/4$.*

Proof. Consider the slots $B_1, \ldots, B_K$ at the moment the algorithm tries to assign A_i. Consider the moment in which ad A_i was discarded. Since the case of Line 7 fails, all slots $Z \setminus X$ have fullness at least $1/4$. And, as the case of Line 16 fails, there exists at least one $j \in Z \setminus X$ with $f(B_j) > 3/4$ (since $s_i < 1/4$). Finally, since the case of Line 10 fails, all slots in X, at this moment, had fullness at least $1/4$. Note that the fullness of j_1 remains at least $1/4$ after the ads of T are removed from B_{j_1}. We conclude that, at this moment, all slots in $(Z \setminus X) \cup X = Z$ had fullness at least $1/4$. Therefore, the result follows. □

2.4 A 1/9-approximation algorithm for the general case

Now, we present an algorithm for the general case, showing that it is a $1/9$-approximation. First, we show on Lemma 7 that Algorithms 2 and 3 are $1/4$-approximations for, respectively, medium and small ads. Then, in Algorithm 4, we present a pseudocode for the whole set of ads $\mathcal{A}$.

Lemma 7. *Algorithms 2 and 3 are $1/4$-approximations for medium and small ads, respectively.*

Proof. Consider the execution of one of these algorithms. Let W be the set of copies of ads (considering w_i copies of each ad A_i) scheduled by the algorithm. Let W^* be the ads scheduled in an optimal solution *Opt* (also considering w_i copies of each ad A_i), such that $f(Opt) = f(W^*)$. Also, let $B_j^* \subseteq W^*$ be the ads scheduled in a slot B_j in *Opt*. We partition W^* into $E^* = W^* \cap W$ and $N^* = W^* \setminus W$. The set E^* corresponds to the ads in W^* scheduled by the algorithm, and the set N^* to those not scheduled. Let ℓ be smallest index of slot such that $B_\ell^* \cap N^* \neq \emptyset$. Let m be the smallest index of slot such that $f(B_j) \geq 1/4$ for all $j \geq m$. Note that $m \leq \ell$, since each slot B_j with $j \geq \ell$ has fullness $f(B_j) \geq 1/4$, by Lemmas 4 and 6. This implies that each ad $A_i \in B_j^*$ with $j \leq m$ is in E^* by the minimality of ℓ.

Let $Z = \{1, 2, \ldots, m-1\}$ and $\overline{Z} = \{m, m+1, \ldots, K\}$. Let $H = \bigcup_{j \in Z} B_j^*$. Let $\tilde{w}_i$ be the number of copies of an ad A_i scheduled by the algorithm in slots with index in Z, and let $\tilde{w}_i^*$ be the number of copies of A_i scheduled in slots with index in Z in the optimal solution. Let F be the set of ads which have been scheduled by the algorithm in slots of Z at least as many times as the optimal solution does, and let R be the set of ads that have been scheduled by the algorithm in slots of Z fewer times than the optimal solution does, that is, $F = \{A_i \mid \tilde{w}_i \geq \tilde{w}_i^*\}$ and $R = H \setminus F$. Let Q be the set of indices j of Z with $f(B_j) \geq 1/4$ in the solution computed by the algorithm.

Let $A_i \in R$ and $j \in Z \setminus Q$ with $j \geq r_i$. We will prove that $A_i \in B_j$ in the solution computed by the algorithm. Since $A_i \in R$, there is a copy of A_i in $\overline{Z}$. Assume that $A_i \notin B_j$, then the algorithm did not add a copy of A_i to B_j because $f(B_j) \geq 1/4$, then $j \in Q$, which is a contradiction. We conclude that $A_i \in B_j$ in the algorithm's solution.

We are going to prove that $f(R) < 1/4$. First, note that $m - 1 \notin Q$ by the minimality of m. Now, observe that $R \subseteq B_{m-1}$. In fact, if $A_i \in R$, then $r_i \leq m - 1$ and by the previous paragraph we know that $A_i \in B_{m-1}$. It follows that $R \subseteq B_{m-1}$, and then $f(R) \leq f(B_{m-1}) < 1/4$.

To derive the lemma, it suffices to show that $\sum_{j \in Z} f(B_j) \geq 1/4 \sum_{j \in Z} f(B_j^*)$. We can rewrite these sums as follows:

$$\sum_{j \in Z} f(B_j^*) = \sum_{A_i \in R} s_i \tilde{w}_i^* + \sum_{A_i \in F} s_i \tilde{w}_i^* \qquad \text{and} \qquad \sum_{j \in Z} f(B_j) \geq \sum_{A_i \in R} s_i \tilde{w}_i + \sum_{A_i \in F} s_i \tilde{w}_i.$$

The equality follows because the ads scheduled by the optimal solution in slots of Z correspond to H, partitioned by R, F. The inequality follows from the definition of w_i.

Note that $\sum_{A_i \in F} s_i \tilde{w}_i \geq \sum_{A_i \in F} s_i \tilde{w}_i^*$. Thus, if $\sum_{A_i \in F} s_i \tilde{w}_i^* \geq 1/4 \sum_{j \in Z} f(B_j^*)$, the statement follows. Then, in the following we assume that $\sum_{A_i \in F} s_i \tilde{w}_i^* < 1/4 \sum_{j \in Z} f(B_j^*)$, which implies that $\sum_{A_i \in R} s_i \tilde{w}_i^* \geq 3/4 \sum_{j \in Z} f(B_j^*)$.

The fullness of slots in Z in the solution found by the algorithm can be rewritten as:

$$\begin{aligned}\sum_{j\in Z} f(B_j) &= \sum_{j\in Q} f(B_j) + \sum_{j\in Z\setminus Q} f(B_j)\\ &\geq \sum_{j\in Q} \frac{1}{4} + \sum_{j\in Z\setminus Q} f(B_j)\\ &> \sum_{j\in Q} f(R) + \sum_{j\in Z\setminus Q} f(B_j \cap R)\\ &\geq \sum_{A_i\in R} (m - r_i)s_i\\ &\geq \sum_{A_i\in R} \tilde{w}_i^* s_i \geq \frac{3}{4}\sum_{j\in Z} f(B_j^*).\end{aligned}$$

The first inequality holds by the definition of Q. The second inequality holds because $f(R) < 1/4$. For the third one, consider the sums on the left side of the inequality and notice that each ad $A_i \in R$ appears in all terms of the first sum and in all terms of the second sum of indices j with $j \geq r_i$; thus, A_i appears in at least $(m - r_i)$ terms. The penultimate inequality holds because an ad cannot be displayed before the release date. Thus, also in this case we conclude that $\sum_{j\in Z} f(B_j) \geq 1/4 \sum_{j\in Z} f(B_j^*)$.

Finally, we bound the value of the solution W:

$$\begin{aligned}f(W) = \sum_j f(B_j) &= \sum_{j\in Z} f(B_j) + \sum_{j\in \overline{Z}} f(B_j)\\ &\geq \sum_{j\in Z} \frac{1}{4} f(B_j^*) + \sum_{j\in \overline{Z}} \frac{1}{4}\\ &\geq \sum_{j\in Z} \frac{1}{4} f(B_j^*) + \sum_{j\in \overline{Z}} \frac{1}{4} f(B_j^*)\\ &= \frac{1}{4}\left(\sum_{j\in Z} f(B_j^*) + \sum_{j\in \overline{Z}} f(B_j^*)\right)\\ &= \frac{1}{4} f(\mathit{Opt}).\end{aligned}$$

The first inequality holds by the definition of m and the statement of the previous paragraph. The second inequality holds by the fact that $\sum_{j\in Z} f(B_j^*) \leq 1$. □

Algorithm 4 Algorithm for general case

```
1: procedure ALG_GENERAL(A)
2:     G = {A_i ∈ A | s_i > 1/2}
3:     M = {A_i ∈ A | 1/4 < s_i ≤ 1/2}
4:     P = {A_i ∈ A | s_i ≤ 1/4}
5:     S_1 ← DP(G)
6:     S_2 ← ALG_MEDIUM(M)
7:     S_3 ← ALG_SMALL(P)
8:     return max{S_1, S_2, S_3}
9: end procedure
```

The Algorithm 4 divides the ads into large G, medium M and small P and executes the Algorithms 1, 2 and 3, respectively, for G, M and P. Finally, the algorithm returns the best of the solutions of the executed algorithms. In Theorem 1, we show that this algorithm is a 1/9-approximation for MAXSPACE-R.

Theorem 1. *Algorithm 4 is a* 1/9*-approximation for MAXSPACE-R problem.*

Proof. Algorithm 4 only partitions the ads and executes Algorithms 1, 2 and 3. By Lemmas 2, 3 and 5, these algorithms run in polynomial time in the instance size and return feasible solutions. Then, Algorithm 4 runs in polynomial time in the instance size and returns a feasible solution.

Let W^* be the copies of ads scheduled in an optimal solution (considering w_i copies of each ad A_i) and let $f(Opt) = f(W^*)$. If $f(W^* \cap G) \geq \frac{1}{9} f(Opt)$, it is possible to obtain a solution with fullness at least $\frac{1}{9} f(Opt)$, since Algorithm 1 is exact for large ads (Lemma 2). Otherwise, we know that $f(W^* \cap M) \geq \frac{4}{9} f(Opt)$ or $f(W^* \cap P) \geq \frac{4}{9} f(Opt)$. If $f(W^* \cap M) \geq \frac{4}{9} f(Opt)$, then a solution for ads of M has fullness at least $\frac{1}{4}(\frac{4}{9} f(W^*)) = \frac{1}{9} f(Opt)$, since Algorithm 2 is a 1/4-approximation for medium ads (by Lemma 7). If $f(W^* \cap P) \geq \frac{4}{9} f(Opt)$, then a solution for ads of P has fullness at least $\frac{1}{4}(\frac{4}{9} f(W^*)) = \frac{1}{9} f(Opt)$, since Algorithm 3 is a 1/4-approximation for small ads (by Lemma 7). □

3 A PTAS for MAXSPACE-RDV with a constant number of slots

In what follows, assume that the number of slots K is a constant, $L = 1$, and $0 < s_i \leq 1$ for each $A_i \in \mathcal{A}$. In MAXSPACE-RDV, we define $f(B_j) = \sum_{A_i \in B_j} v_i$ as the value of a slot B_j and $f(S) = \sum_{B_j \in S} f(B_j)$ as the value of a solution S.

Let S denote a feasible solution $\mathcal{A}' \subseteq \mathcal{A}$ scheduled into slots $B_1, B_2, \ldots, B_K$. The *type* t of an ad $A_i \in \mathcal{A}'$ with respect to S is the subset of slots to which A_i is assigned, that is, $A_i \in B_j$ if and only if $j \in t$. Let $\mathcal{T}$ be a set of all the subsets of slots, then $\mathcal{T}$ contains every possible type and $|\mathcal{T}| = 2^K$. Observe that two ads with the same type have the same frequency, and thus one can think of all ads in $\mathcal{A}'$ with the same type as a single ad.

Let $\varepsilon > 0$ be a constant such that $1/\varepsilon$ is an integer, and let $q = \min\{|\mathcal{A}|, 2^{2^K} 2^K/\varepsilon\}$. Our algorithm guesses a set V with at most q ads with the largest values in an optimal solution. For each $V \subseteq \mathcal{A}$ such that $|V| \leq q$, we define U as the set of every ad $A_i \in \mathcal{A} \setminus V$ such that $v_i w_i \leq v_{\min}$, where $v_{\min} = \min\{v_i w_i : A_i \in V\}$. Then, for each feasible scheduling of $V \subseteq \mathcal{A}$, we fill the remaining spaces in the slots with ads in U using a linear program.

A *configuration* for a subset of ads $\mathcal{A}' \subseteq \mathcal{A}$ is a feasible solution which schedules every ad in $\mathcal{A}'$. Lemma 8 states that if K is constant, then the number of possible configurations containing only ads in V is polynomial in the number of ads in V and can be enumerated by a brute-force algorithm.

Lemma 8. *If K is constant, then the configurations for all subsets of V can be listed in polynomial time.*

Proof. There are $O(q|\mathcal{A}|^q)$ possible choices for V, and there exist $O(q^{2^K})$ possible solutions for each set since the number of types is 2^K. Thus, we can enumerate V and all of its configurations in time $O(q^{2^K+1}|\mathcal{A}|^q)$, which is polynomial since K and q are constants. □

Since all candidate configurations can be listed in polynomial time by Lemma 8, we may assume that we guessed the configuration S_V of the most valuable ads induced by Opt_V. We are left with the residual problem of placing ads of U. For each slot j in S_V, $1 \leq j \leq K$, the space which is unused by the most valuable ads V is

$$u_j = 1 - \sum_{A_i \in B_j} s_i.$$

We define RESIDUAL-MAXSPACE-RDV as the problem of, given a set of ads U, where $v_i w_i \leq v_{\min}$ for all $A_i \in U$, finding a subset $\mathcal{A}'_t \subseteq U$ for each $t \in \mathcal{T}$ such that the occupation of each slot j is at most u_j, and which maximizes the value

$$\sum_{t \in \mathcal{T}} \sum_{A_i \in \mathcal{A}'_t} |t| v_i.$$

Let $\mathcal{T}(A_i)$ be the subset of types *compatible* with an ad A_i, i.e., the set of types $t \in \mathcal{T}$ with $|t| = w_i$, and such that the slots in t are compatible with the release date r_i and the deadline d_i. We solve the linear program (P) to assign ads of U to types.

$$(P) \quad \text{Maximize} \quad \sum_{A_i \in U} \sum_{t \in \mathcal{T}(A_i)} v_i w_i X_{A_i,t} \tag{1}$$

$$\text{Subject to:} \quad \sum_{t \in \mathcal{T}(A_i)} X_{A_i,t} \leq 1 \qquad \forall A_i \in U \tag{2}$$

$$\sum_{\substack{t \in \mathcal{T}: \\ j \in t}} \sum_{A_i \in U} s_i X_{A_i,t} \leq u_j \qquad j = 1, 2, \ldots, K \tag{3}$$

$$X_{A_i,t} \geq 0 \qquad \forall A_i \in U, \forall t \in \mathcal{T}(A_i) \tag{4}$$

Algorithm 5 Algorithm for reassigning fractional solution.

```
1: procedure REASSIGN(W, U_W)
2:     for each A_i ∈ U_W and t ∈ W do
3:         X'_{A_i,t} ← 0
4:     for each t ∈ W do
5:         z_t ← Σ_{A_i ∈ U_W} s_i X_{A_i,t}
6:     U'_W ← ∅
7:     z_W ← Σ_{t ∈ W} z_t                          ▷ Total area of items with support W
8:     for each A_i ∈ U_W in non-increasing order of v_i/s_i do
9:         if Σ_{A_j ∈ U'_W} s_j < z_W then          ▷ The last ad may not fit entirely
10:            U'_W ← U'_W ∪ {A_i}
11:    Let {t_1, t_2, ..., t_{|W|}} be the types in W
12:    k ← 1
13:    for each A_i ∈ U'_W do
14:        s'_i ← s_i
15:        while s'_i > 0 do
16:            m ← min{s'_i, z_{t_k} − s'_i}
17:            X'_{A_i,t_k} ← m/s_i
18:            s'_i ← s'_i − m
19:            z_{t_k} ← z_{t_k} − m
20:            if z_{t_k} = 0 then
21:                k ← k + 1
22:    return X', U'_W
23: end procedure
```

The variables $X_{A_i,t}$ indicate if ad A_i is assigned to type t, constraints (2) ensure that an ad cannot be assigned more than once, and constraints (3) guarantee that the capacity of any slot will not be violated.

Consider a solution X for (P), which can be obtained in polynomial time [11], and notice that X induces an assignment of ads to types. In this assignment, if the solution is such that $X_{A_i,t} < 1$ units from ad A_i are assigned to type t, then we say that ad A_i is fractionally assigned to t by an amount of $X_{A_i,t}$. The set of all types t for which $X_{A_i,t} > 0$ is called the *support* of A_i and is denoted by $Sup(A_i)$.

To eliminate fractional assignments, we group ads with the same support. Let W be a subset of types, and U_W be the set of ads A_i with $Sup(A_i) = W$. In particular, each ad $A_i \in U_W$ is compatible with any type $t \in W$. For each type $t \in W$, we define the total fullness received by t from U_W as

$$z_t = \sum_{A_i \in U_W} s_i X_{A_i,t}.$$

By the fact that each ad in U_W is fractionally assigned to types in W, we know that

$$\sum_{A_i \in U_W} s_i \geq \sum_{A_i \in U_W} \sum_{t \in W} s_i X_{A_i,t} = \sum_{t \in W} z_t.$$

In other words, the total size of U_W given by $\sum_{A_i \in U_W} s_i$ is not smaller than the size received by types W from U_W. Therefore, we remove the fractional assignment of all ads in U_W and integrally reassign each ad in U_W to types in W, discarding any remaining ad.

The process of rounding the fractional assignment is summarized in Algorithm 6, which receives as input a fractional assignment X of ads to types W and returns an integer assignment X'.

As part of the rounding, we use a procedure called REASSIGN, which takes a support W and the ads scheduled to that support U_W and returns a new allocation of these ads in W. This procedure removes all ads from U_W from the fractional solution and greedily fills their space with ads from U_W in order of efficiency (v_i/s_i). Note that this new schedule does not have a worse value since all the space is filled, the advertisements are chosen in order of efficiency, and this new solution is fractional. Let $U'_W \subseteq U_W$ be the newly scheduled ads; note that all U'_W ads, except perhaps the last one, are fully scheduled to W types. The REASSIGN pseudocode is presented in Algorithm 5.

In Lemma 9, we observe that Algorithm 5 is polynomial in the instance size. Lemma 10 shows that REASSIGN does not worsen the solution.

Lemma 9. *Algorithm 5 runs in polynomial time.*

Proof. The size of U_W is $O(n)$ and the size of W is $O(2^K)$; thus, Algorithm 5 running time is $O(n2^K)$, which is polynomial since K is constant. □

Lemma 10. *Algorithm 5 returns a solution with the same value of linear programming (P) assignment.*

Proof. The algorithm fills the space for the U_W items in the W types with the best efficiency items in U_W. This way, the algorithm obtains an optimal value for the fractional allocation of the U_W items in the considered space. The linear programming algorithm (P) also obtains a fractional optimal solution for the same items and considers the same space. Therefore, both solutions have the same value. □

Algorithm 6 Algorithm for rounding ad assignment.

```
1: procedure ROUNDING(X)
2:     for each A_i ∈ U and t ∈ T do
3:         X'_{A_i,t} ← 0
4:     for each W ⊆ T do
5:         U_W ← all ads A_i with Sup(A_i) = W
6:         X', U'_W ← REASSIGN(W, U_W)
7:         discard from U'_W any ad that is not integrally assigned to the same type in W
8:     return X'
9: end procedure
```

Lemma 11 shows that Algorithm 6 is polynomial in the instance size. Corollary 1 is obtained from Lemma 12, and bounds the total value of ads discarded by Algorithm 6 in each execution of Line 7. And Corollary 2 is obtained from Lemma 12 and Corollary 1.

Lemma 11. *Algorithm 6 runs in polynomial time in the instance size.*

Proof. The loop of Line 4 executes a constant number of iterations, since $|\mathcal{T}| = 2^K$ and the number of subsets of $\mathcal{T}$ is 2^{2^K}. The REASSIGN algorithm is also polynomial, by Lemma 9. Then, the algorithm runs in polynomial time. □

Lemma 12. *Let $W \subseteq \mathcal{T}$ and let U'_W be the set of ads with support W after the reassign algorithm. The number of discarded ads from U'_W in Line 7 of the* ROUNDING *is at most $|W|$.*

Proof. The REASSIGN algorithm adds an advertisement fractionally to a type in two ways: starting at a type t and continuing at a type $t+1$, and completing the fullness in the last type of W. In the first case, the algorithm can add a maximum of $|W| - 1$ fractional ads, and in the second case, it is possible to add at most one ad fractionally to a type. Thus, at most $|W|$ advertisements are added fractionally to types of W, and the result follows. □

Corollary 1. *Let $W \subseteq \mathcal{T}$ and let U'_W be the set of ads scheduled to W after reassigning the algorithm. Then the total value of selected ads in U'_W after the execution of rounding is*

$$\sum_{A_i \in U'_W} \sum_{t \in W} v_i w_i X'_{A_i,t} \geq \sum_{A_i \in U'_W} \sum_{t \in W} v_i w_i X_{A_i,t} - |W| v_{\min}.$$

Proof. Let U''_W be the set of discarded advertisements of U'_W,

$$\begin{aligned}\sum_{A_i \in U'_W} \sum_{t \in W} v_i w_i X'_{A_i,t} &\geq \sum_{A_i \in U'_W} \sum_{t \in W} v_i w_i X_{A_i,t} - \sum_{A_i \in U''_W} v_i w_i \\ &\geq \sum_{A_i \in U'_W} \sum_{t \in W} v_i w_i X_{A_i,t} - |W| v_{\min}.\end{aligned}$$

The second inequality holds because the number of ads discarded in U_W is at most $|W|$ in Line 7 (Lemma 12), and all advertisements $A_i \in U'_W$ has value $v_i w_i \leq v_{\min}$, by the definition of U. □

Corollary 2. *The difference between the maximum fractional and modified solution values is not larger than $2^{2^K} 2^K v_{\min}$. That is,*

$$\sum_{A_i \in \mathcal{A}} \sum_{t \in \mathcal{T}} v_i w_i X'_{A_i,t} \geq \sum_{A_i \in \mathcal{A}} \sum_{t \in \mathcal{T}} v_i w_i X_{A_i,t} - 2^{2^K} 2^K v_{\min}.$$

Proof. Consider the value of variables W and U'_W of Algorithm 6. Using Corollary 1, we have that

$$\begin{aligned}
\sum_{A_i \in \mathcal{A}} \sum_{t \in \mathcal{T}} v_i w_i X'_{A_i,t} &= \sum_{W \subseteq \mathcal{T}} \sum_{A_i \in U'_W} \sum_{t \in W} v_i w_i X'_{A_i,t} \\
&\geq \sum_{W \subseteq \mathcal{T}} \left(\sum_{A_i \in U'_W} \sum_{t \in W} v_i w_i X_{A_i,t} - |W| v_{\min} \right) \\
&\geq \sum_{A_i \in \mathcal{A}} \sum_{t \in \mathcal{T}} v_i w_i X_{A_i,t} - \sum_{W \subseteq \mathcal{T}} 2^K v_{\min} \\
&= \sum_{A_i \in \mathcal{A}} \sum_{t \in \mathcal{T}} v_i w_i X_{A_i,t} - 2^{2^K} 2^K v_{\min},
\end{aligned}$$

where the last inequality holds because $|W| \leq 2^K$, and the last equality holds because there are $2^{|\mathcal{T}|} = 2^{2^K}$ distinct choices for W. □

The complete algorithm for MAXSPACE-RDV is presented in Algorithm 7. Given parameter $\varepsilon > 0$, this algorithm receives a set of ads $\mathcal{A}$ as input. The algorithm tries to guess which q ads are most valuable for an optimal solution. It explores all possible combinations of subsets $V \subseteq \mathcal{A}$ with at most q ads, and for each feasible scheduling for V, it tries to fill the remaining spaces with ads less valuable than the ones in V, called U. In this step, the algorithm associates ads of U to types using the linear program (P). The Algorithm ROUNDING transforms the fractional assignment X into an integer assignment X'. Note that this assignment can be easily converted into a schedule of ads U into solution S'. The algorithm returns the solution of maximum value among those considered.

Algorithm 7 Algorithm for MAXSPACE-RDV with K constant.

```
1: procedure ALGRDV_ε(A)
2:     q ← min{|A|, 2^{2^K} 2^K/ε}
3:     S ← ∅
4:     for each V ⊆ A such that |V| ≤ q do
5:         for each feasible assignment S_V of V do
6:             v_min ← min{v_i w_i : A_i ∈ V}
7:             U ← {A_i ∈ A \ V | v_i w_i ≤ v_min}
8:             X ← solve LP (P) with ads in U
9:             X' ← ROUDING(X)
10:            Add ads of U to S_U according to integral assignment X'
11:            S' ← S_V ∪ S_U
12:            if f(S') ≥ f(S) then
13:                f(S) ← f(S')
14:    return S
15: end procedure
```

In Lemma 13 and 14, we prove that Algorithm 7 is polynomial in the instance size and returns a feasible solution. In Theorem 2, we prove that Algorithm 7 is a PTAS for MAXSPACE-RDV.

Lemma 13. *Algorithm 7 executes in polynomial time.*

Proof. The linear program is solved in polynomial time in the size of the model [11], and the model is polynomial in the size of the instance since it has $O(|U| + K)$ restrictions and $O(|U|2^{2^K})$ variables. The ROUDING algorithm is also polynomial, by Lemma 11. The loops on Lines 4 and 5 are polynomial, by Lemma 8. Then, Algorithm 7 is polynomial in the instance size. □

Lemma 14. *Algorithm 7 returns a feasible solution.*

Proof. Since each ad configuration in V is feasible, S_V respects release date and deadline restrictions. Solution S_U also respects the release date and deadline restrictions, and constraints (3) guarantee that this solution respects the slots' capacities. Thus, the algorithm returns a feasible solution. □

Theorem 2. *Algorithm 7 is a PTAS for MAXSPACE-RDV.*

Proof. We try every schedule for V with $|V| \leq q$. Thus, consider the moment when the schedule of V is the same as the $|V|$ most valuable ads in an optimal solution Opt. Let S_V be the schedule of ads of V in the returned solution S. Thus, $f(S_V) = f(Opt_V)$, where Opt_V is the schedule of V in Opt. Note that, if $q = |\mathcal{A}|$ or $|Opt| \leq q$, then $f(S) = f(S_V) = f(Opt_V) = f(Opt)$ and the result follows. Now, consider that $q = 2^{2^K}2^K/\varepsilon < |\mathcal{A}|$ and $|Opt| > q$.

Let X be the linear program solution and X' be the output of ROUNDING. Define

$$f(X) = \sum_{A_i \in \mathcal{A}} \sum_{t \in \mathcal{T}} v_i w_i X_{A_i,t} \quad \text{and} \quad f(X') = \sum_{A_i \in \mathcal{A}} \sum_{t \in \mathcal{T}} v_i w_i X'_{A_i,t}.$$

Let Opt_U be an optimal solution for ads in U in the remaining spaces of Opt_V. Observe that Opt_U induces a feasible solution with value $f(Opt_U)$. This implies that $f(X) \geq f(Opt_U)$, as X is an optimal fractionally solution in the remaining spaces of S_V, which has the same fullness of Opt_V. Also, note that $f(S) = f(X') + f(S_V)$, then using Corollary 2 we have

$$\begin{aligned}
f(S) &= f(X') + f(S_V) \\
&= f(X') + f(Opt_V) \\
&\geq f(X) - 2^{2^K}2^K v_{\min} + f(Opt_V) \\
&\geq f(Opt_U) - 2^{2^K}2^K \frac{f(S_V)}{q} + f(Opt_V) \\
&= f(Opt_U) - 2^{2^K}2^K \frac{f(S_V)}{\frac{2^{2^K}2^K}{\varepsilon}} + f(Opt_V) \\
&= f(Opt_U) - \varepsilon f(S_V) + f(Opt_V) \\
&\geq f(Opt) - \varepsilon f(Opt).
\end{aligned}$$

Where the first inequality holds by Corollary 2, the second inequality holds since $v_{\min} \leq f(S_V)/q$ and the last inequality holds since $f(Opt) \geq f(S_V)$.

Since the algorithm returns the best solution and considers solution S, the result follows. □

4 Final remarks

This paper consider two generalizations for the MAXSPACE problem, called MAXSPACE-R and MAXSPACE-RDV. We present a 1/9-approximation algorithm for MAXSPACE-R and a PTAS for MAXSPACE-RDV for the case that the number of slots is bounded by a constant. These are the first approximation algorithm and approximation schemes to these MAXSPACE variants.

A PTAS is the best approximation ratio for MAXSPACE-RDV one can expect since it does not admit an FPTAS even for $K = 2$ [13]. This variant is a generalization of the Multiple Knapsack Problem [3].

In future works, we will also consider MAXSPACE-RDV with the number of slots given in the instance, for which the ideas used in this work are not sufficient.

Funding This project was supported by São Paulo Research Foundation (FAPESP) grants #2015/11937-9, #2016/23552-7, #2017/21297-2, and #2020/13162-2, and National Council for Scientific and Technological Development (CNPq) grants #425340/2016-3, #312186/2020-7, and #311039/2020-0.

References

[1] Micah Adler, Phillip B. Gibbons, and Yossi Matias. Scheduling space-sharing for internet advertising. *J. Sched.*, 5(2):103–119, 2002. ISSN 1094-6136, 1099-1425. doi: 10.1002/jos.74.

[2] Rex Briggs and Nigel Hollis. Advertising on the web: Is there response before click-through? *Journal of Advertising research*, 37(2):33–46, 1997.

[3] Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J. Comput.*, 35(3):713–728, January 2005. ISSN 0097-5397, 1095-7111. doi: 10.1137/s0097539700382820.

[4] Mauro RC Da Silva, Rafael CS Schouery, and Lehilton LC Pedrosa. A polynomial-time approximation scheme for the maxspace advertisement problem. *Electronic Notes in Theoretical Computer Science*, 346: 699–710, 2019.

[5] Milind Dawande, Subodha Kumar, and Chelliah Sriskandarajah. Performance bounds of algorithms for scheduling advertisements on a web page. *Journal of Scheduling*, 6(4):373–394, 2003.

[6] Brian C Dean and Michel X Goemans. Improved approximation algorithms for minimum-space advertisement scheduling. In *In Proceedings of International Colloquium on Automata, Languages, and Programming*, pages 1138–1152, 2003.

[7] Ari Freund and Joseph Seffi Naor. Approximating the advertisement placement problem. In *Proceedings of International Conference on Integer Programming and Combinatorial Optimization*, pages 415–424, 2002.

[8] Alan M Frieze, Michael RB Clarke, et al. Approximation algorithms for the m-dimensional 0-1 knapsack problem: worst-case and probabilistic analyses. *European Journal of Operational Research*, 15(1):100–109, 1984.

[9] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. In *Annals of Discrete Mathematics*, pages 287–326. Elsevier, 1979.

[10] IAB. Internet advertising revenue report: Full year 2022, 2022. URL `https://www.iab.com/wp-content/uploads/2023/04/IAB_PwC_Internet_Advertising_Revenue_Report_2022.pdf`. [Online; Accessed on: 2023-05-03].

[11] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[12] Arshia Kaul, Sugandha Aggarwal, Anshu Gupta, Niraj Dayama, Mohan Krishnamoorthy, and PC Jha. Optimal advertising on a two-dimensional web banner. *International Journal of System Assurance Engineering and Management*, 9(1):306–311, 2018.

[13] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Introduction to NP-Completeness of knapsack problems*, chapter Introduction to NP-Completeness of Knapsack Problems, pages 483–493. Springer Berlin Heidelberg, 2004. ISBN 9783642073113, 9783540247777. doi: 10.1007/978-3-540-24777-7_16.

[14] Gwang Kim and Ilkyeong Moon. Online banner advertisement scheduling for advertising effectiveness. *Computers & Industrial Engineering*, 140:106226, 2020.

[15] Subodha Kumar. *Optimization Issues in Web and Mobile Advertising*. Springer International Publishing, 2016. ISBN 9783319186443, 9783319186450. doi: 10.1007/978-3-319-18645-0.

[16] Pinterest. Pinterest's homepage. `https://www.pinterest.com`, 2021. Accessed: 2021-03-17.

[17] Vijay V. Vazirani. *Approximation Algorithms*. Springer Berlin Heidelberg, 2003. ISBN 9783642084690, 9783662045657. doi: 10.1007/978-3-662-04565-7.