Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 346 (2019) 699–710

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

A Polynomial-time Approximation Scheme for the MAXSPACE Advertisement Problem

Mauro R. C. da Silva[1,2] Rafael C. S. Schouery[1,3]
Lehilton L. C. Pedrosa[1,4]

Institute of Computing
University of Campinas
Campinas, Brazil

Abstract

In the MAXSPACE problem, given a set of ads $\mathcal{A}$, one wants to place 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. The goal is to find a feasible schedule which maximizes the sum of the space occupied by all slots. We introduce a generalization, called MAXSPACE-RD, in which each ad A_i also has a release date $r_i \geq 1$ and a deadline $d_i \leq K$, and may only appear in a slot B_j with $r_i \leq j \leq d_i$. These parameters model situations where a subset of ads corresponds to a commercial campaign with an announcement date that may expire after some defined period. We present a polynomial-time approximation scheme for MAXSPACE-RD when K is bounded by a constant, i.e., for any $\varepsilon > 0$, we give a polynomial-time algorithm which returns a solution with value at least $(1-\varepsilon)Opt$, where Opt is the optimal value. This is the best factor one can expect, since MAXSPACE is 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 simply ads, to users. Often, each website 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, the advertisement is the main source of revenue. Thus, it is important to find the best way to dispose the ads in the available time and space while maximizing the revenue [7].

[1] This project was supported by São Paulo Research Foundation (FAPESP) grants #2015/11937-9, #2016/23552-7 and #2017/21297-2, and National Council for Scientific and Technological Development (CNPq) grants #425340/2016-3, #313026/2017-3, #308689/2017-8 and #425806/2018-9.

[2] Email: maurorcsc@gmail.com

[3] Email: rafael@ic.unicamp.br

[4] Email: lehilton@ic.unicamp.br

https://doi.org/10.1016/j.entcs.2019.08.061
1571-0661/

The revenue from web advertising grew considerably in the 21st century. In 2013, the total revenue was US$42.78 billion, an increase of 17% from the previous year. It is estimated that the U.S. web advertising reached US$77 billion in 2016, and comprised 35% of all advertising spending, overtaking television advertising [7]. In 2016, ads in banners comprised 31,4% of internet advertising (considering banners and mobile platforms), which represents a revenue of US$22.7 billion [8]. Web advertising has created a multi-billionaire industry where algorithms for scheduling advertisements play an important role.

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 which 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, 4].

The main problems in that class are MINSPACE and MAXSPACE. In MINSPACE, all the ads are to be scheduled in the slots, and the goal is to minimize the height of the highest slot. In MAXSPACE, an upper bound L is specified which represents the size of each slot. 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, $\sum_{A_i \in B_j} s_i \leq L$. The goal of MAXSPACE is to maximize the fullness of the slots, defined by $\sum_{A_i \in \mathcal{A}'} s_i w_i$. Both of these problems are strongly NP-hard [1, 4].

Dawande et al. [4] define three special cases of MAXSPACE: MAX_w, $\text{MAX}_{K|w}$ and MAX_s. 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. And, in MAX_s, every ad has the same size s. In an analogous way, they define three special cases of MINSPACE: MIN_w, $\text{MIN}_{K|w}$ and MIN_s.

Adler et al. [1] present a $\frac{1}{2}$-approximation called SUBSET-LSLF for MAXSPACE when the ad sizes form a sequence $s_1 > s_2 > s_3 > \ldots$, such that for all i, s_i is a multiple of s_{i+1}. Dawande et al. [4] 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 [6] 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 ads are in the interval $[L/2, L]$.

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

In practice, the time interval relative to each slot in scheduling advertising can represent minutes, seconds or long periods, such as days and weeks. Often, one considers the idea of *release dates* and *deadlines*. An ad has a release date that 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.

We introduce a MAXSPACE generalization called MAXSPACE-RD in which each ad A_i has two additional parameters, a release date $r_i \geq 1$ and a deadline $d_i \leq K$. 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 $j < r_i$. Similarly, 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, $d_i - r_i + 1 \geq w_i$.

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 [9]. A *Fully Polynomial-Time Approximation Scheme* (FPTAS) is a PTAS whose running time is also polynomial in $1/\varepsilon$. In this work, we present a PTAS to MAXSPACE-RD when the number of slots is a constant. This approximation is the best one can expect, since MAXSPACE is NP-hard even when the number of slots is 2 [1, 4].

In Section 2 we define the notation and concepts used in this work. In Section 3 we present an algorithm to schedule small ads and in Section 4 we present a PTAS to the whole set of ads. In Section 5 we discuss the results and future works.

2 Preliminaries

In what follows, assume that the number of slots K is a constant, that $L = 1$ and $0 < s_i \leq 1$ for each $A_i \in \mathcal{A}$.

We partition the ads into two groups: *large ads* and *small ads*. For a constant $\varepsilon > 0$, we say that A_i is a large ad if $s_i \geq \varepsilon/(2^{2^K}2^K K)$, otherwise we say it is a small ad. Then, let $G = \{A_i \in \mathcal{A} \mid s_i \geq \varepsilon/(2^{2^K}2^K K)\}$ be the set of large ads and $P = \mathcal{A} \setminus G$ be the set 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 $f(S) = \sum_{j=1}^{K} f(B_j)$.

The *type* t of an ad $A_i \in \mathcal{A}'$ with respect to a solution 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 that contains 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 with the same type as a single ad. For each $t \in \mathcal{T}$, the *occupation* o_t of type t is the space the ads with this type occupy in each of its slots, i.e., let $\mathcal{A}'_t$ be subset of $\mathcal{A}'$ composed of ads of type t, then $o_t = \sum_{A_i \in \mathcal{A}'_t} s_i$.

A *configuration* for a subset of ads $\mathcal{A}'$ is a feasible solution which schedules every ad in $\mathcal{A}'$. Lemma 2.1 states that if K is constant, then the number of possible configurations containing only large ads is polynomial in the number of large ads, and can be enumerated by a brute-force algorithm.

Lemma 2.1 *If K is constant, then the set of configurations for all subsets of G can be listed in polynomial time.*

Proof. Since each container has height 1, the number of large ads which can be scheduled into a single slot is at most $1/(\varepsilon/(2^{2^K}2^K K)) = (2^{2^K}2^K K)/\varepsilon$. Then, a feasible solution can contain at most $R = K(2^{2^K}2^K K)/\varepsilon$ large ads. Select each subset of $G' \subseteq G$ with $|G'| \leq R$. Observe that the number of such subsets is at most $O\left(\binom{|G|}{R}\right)$, which is polynomial in $|G|$ since R is constant.

For each G', create a multiset M in which each ad $A_i \in G'$ appears w_i times, and list every partition of M into sets $B_1, B_2, \ldots, B_K$. Since $|M| \leq KR$ is bounded by a constant, the number of such partitions for each G' is bounded by a constant. Now note that any feasible solution of G', if any, is a partition of M. This completes the lemma. □

The so-called *first-fit heuristic* is the algorithm which iteratively schedules each copy of an ad in the first compatible slot (respecting release date and deadline restrictions and not exceeding the slot size) and stops as soon as an ad cannot be scheduled. Lemma 2.2 states that first-fit heuristic is optimal if the optimal value is less then 1/2. This result is used in Section 4.

Lemma 2.2 *Let Opt be an optimal solution to $\mathcal{A}$ with $f(Opt) < 1/2$. The first-fit heuristic schedules the whole set $\mathcal{A}$.*

Proof. First, note that for each $A_i \in \mathcal{A}$, $s_i < 1/2$, since otherwise we could schedule only A_i to obtain a solution S' such that $f(S') \geq 1/2 > f(Opt)$, which is a contradiction.

For the sake of contradiction assume that there is an optimal solution Opt to $\mathcal{A}$ with $f(Opt) < 1/2$ and the first-fit heuristic does not schedule the whole set $\mathcal{A}$. Now, let S be the solution returned by the first-fit heuristic and let $\mathcal{A}'$ be the set of ads scheduled by S. We claim that $\mathcal{A}' = \mathcal{A}$. If not, then the algorithm must have stopped when trying to schedule an ad A_i. Since the algorithm respects the release date and the deadline, it must be the case that A_i could not be added to some slot B_j without exceeding capacity. Since in this case $f(B_j) + s_i > 1$, and $s_i < 1/2$, it follows that $f(B_j) > 1/2$. But then again $f(S) > 1/2 > f(Opt)$, which is a contradiction. □

3 An algorithm for small ads

Lemma 2.1 states that all configurations of large items can be listed in polynomial time, and thus one can guess the configuration of large ads induced by an optimal solution. This suggests that the hard part of MAXSPACE is obtaining a solution for small ads. In this section, we consider a variant of MAXSPACE which contains only small ads and present an almost optimal algorithm for this problem.

Formally, let $\varepsilon > 0$ be a constant such that $1/\varepsilon$ is an integer. We

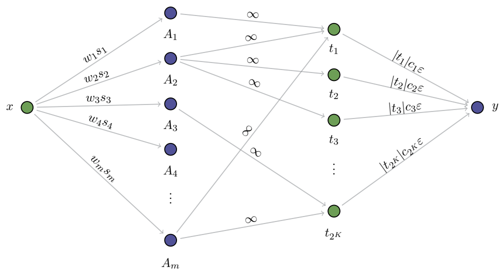

Fig. 1. Example of flow graph to assign ads to types. In blue we have the set U and in green V.

define SMALL-MAXSPACE-RD as the problem of, given a set of ads P, where for each $A_i \in P$, $s_i < \varepsilon/(2^{2^K} 2^K K)$, and for each type $t \in \mathcal{T}$, and integer $c_t \in \{0, 1, \ldots, 1/\varepsilon\}$, one wants to find a subset $\mathcal{A}'_t \subseteq P$ for each $t \in \mathcal{T}$ such that the occupation $\sum_{A_i \in \mathcal{A}'_t} s_i \leq c_t \varepsilon$, and which maximizes the fullness

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

The reasoning behind this problem is that we try to infer the space occupied by each type in any of its slots. Thus, for each type, we guess an integer multiple of ε as its capacity. Since a container's height is $L = 1$, there are exactly $1/\varepsilon$ possible guesses for each type capacity. Also, since there are 2^K types, if C is the set of capacity combinations, then the size of C is

$$|C| = \left(\frac{1}{\varepsilon}\right)^{2^K}. \tag{1}$$

For each combination c in C, we create an assignment of ads to types using a maximum flow algorithm. For this step, we create a graph H as follows. The vertices of H are a source vertex x, a sink vertex y, and sets U and V, where U contains a vertex for each ad, and V contains a vertex for each type. For each ad $A_i \in P$ and each type $t \in T$ we add an edge (A_i, t) with flow capacity ∞ if the slots in type t correspond to a valid schedule to ad A_i, that is, t contains exactly w_i slots, and each slot of t respects the release date and the deadline of A_i. Furthermore, we add an edge (x, A_i) with flow capacity $w_i s_i$ for each $A_i \in U$, and an edge (t, y) with capacity $|t| c_t$ for each $t \in \mathcal{T}$. Figure 1 illustrates this graph.

Consider then a maximum xy-flow F in H, which can be obtained in polynomial time [2] and notice that F induces an assignment from ads to types. In this assignment, if the maximum flow is such that $F_{A_i,t}$ units flow from ad A_i to type t, then we say that ad A_i is fractionally assigned to t by an amount of $F_{A_i,t}/(w_i s_i)$. Observe that this ratio is at most one. The set of all types t for which $F_{A_i,t} > 0$ 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 P_W be the set of ads A_i with $Sup(A_i) = W$. In particular, each ad $A_i \in P_W$ is compatible with any type $t \in W$. For each type $t \in W$, we define the total flow received by t from P_W as

$$z_t = \sum_{A_i \in P_W} F_{A_i,t}.$$

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

$$\sum_{A_i \in P_W} w_i s_i \geq \sum_{A_i \in P_W} \sum_{t \in W} F_{A_i,t} = \sum_{t \in W} z_t.$$

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

The process of rounding the fractional assignment is summarized in Algorithm 1, which receives as input a fractional assignment of ads to types W and returns an integer assignment.

Algorithm 1 Algorithm for rounding ad assignment.

procedure ROUNDING(F)
  for each $A_i \in P$ and $t \in \mathcal{T}$ **do**
    $F'_{A_i,t} \leftarrow 0$
  for each $W \subseteq \mathcal{T}$ **do**
    $P_W \leftarrow$ all ads A_i with $Sup(A_i) = W$
    for each $t \in W$ **do**
      $z_t \leftarrow \sum_{A_i \in P_W} F_{A_i,t}$
      $Z_t \leftarrow \emptyset$
      for each $A_i \in P_W$ **do**
        if $|t| f(Z_t) + w_i s_i \leq z_t$ **then**
          $Z_t \leftarrow Z_t \cup \{A_i\}$
          $F'_{A_i,t} \leftarrow w_i s_i$
          $P_W \leftarrow P_W \setminus \{A_i\}$
    discard remaining ads in P_W
  return F'

We observe that Algorithm 1 is polynomial in the number of ads.

Lemma 3.1 *Algorithm 1 runs in polynomial time.*

Lemma 3.2 bounds the total size of ads discarded by Algorithm 1 in each iteration.

Lemma 3.2 *Let $W \subseteq \mathcal{T}$ and let P_W be the set of ads with support W at the beginning of the algorithm. Then the total assignment of P_W after the execution is*

$$\sum_{A_i \in P_W} \sum_{t \in W} F'_{A_i,t} \geq \sum_{A_i \in P_W} \sum_{t \in W} F_{A_i,t} - |W|\varepsilon/(2^{2^K} 2^K).$$

Proof. Let P_W and z_t be as in the algorithm, and recall that

$$\sum_{A_i \in P_W} w_i s_i \geq \sum_{A_i \in P_W} \sum_{t \in W} F_{A_i,t} = \sum_{t \in W} z_t.$$

If no ad in P_W is discarded, then the lemma holds trivially. Thus, assume that there exists an ad A_j which was discarded in the iteration. Since A_j was discarded in every iteration of the Line 6, we know that for any type t,

$$|t| f(Z_t) + w_j s_j > z_t.$$

Therefore,

$$\begin{aligned}\sum_{A_i \in P_W} \sum_{t \in W} F'_{A_i,t} &= \sum_{t \in W} \sum_{A_i \in Z_t} w_i s_i = \sum_{t \in W} |t| f(Z_t) \\ &> \sum_{t \in W} (z_t - w_j s_j) = \sum_{A_i \in P_W} \sum_{t \in W} F_{A_i,t} - |W| w_j s_j.\end{aligned}$$

Since A_j is small, $s_j < \varepsilon/(2^{2^K} 2^K K)$, and since $w_j \leq K$, the lemma follows. □

Corollary 3.3 is obtained from Lemma 3.2.

Corollary 3.3 *The difference between the maximum fractional flow and modified flow is not larger than ε. That is,*

$$\sum_{A_i \in P} \sum_{t \in \mathcal{T}} F'_{A_i,t} \geq \sum_{A_i \in P} \sum_{t \in \mathcal{T}} F_{A_i,t} - \varepsilon.$$

Proof. Consider the value of variables W and P_W of Algorithm 1. Using Lemma 3.2, we have that

$$\begin{aligned}\sum_{A_i \in P} \sum_{t \in \mathcal{T}} F'_{A_i,t} &= \sum_{W \subseteq \mathcal{T}} \sum_{A_i \in P_W} \sum_{t \in W} F'_{A_i,t} \\ &\geq \sum_{W \subseteq \mathcal{T}} \left(\sum_{A_i \in P_W} \sum_{t \in W} F_{A_i,t} - |W| \varepsilon/(2^{2^K} 2^K) \right) \\ &\geq \sum_{A_i \in P} \sum_{t \in \mathcal{T}} F_{A_i,t} - \sum_{W \subseteq \mathcal{T}} 2^K (\varepsilon/(2^{2^K} 2^K)) \\ &= \sum_{A_i \in P} \sum_{t \in \mathcal{T}} F_{A_i,t} - \varepsilon,\end{aligned}$$

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 small ads is presented in Algorithm 2. Given parameter $\varepsilon > 0$, this algorithm receives as input a set of small ads P and a vector c which contains the capacity of each type. The algorithm returns a feasible schedule for $P' \subseteq P$ in K slots. Based on vector c, the algorithm creates a flow graph and executes a maximum flow algorithm to assign ads to type. The Algorithm Rounding transforms a fractional assignment into an integral assignment. Note that this assignment can be easily converted into a solution for SMALL-MAXSPACE-RD.

Algorithm 2 Algorithm for small ads.

```
procedure ALGP_ε(P, c)
    H ← create flow graph using values of c
    F ← MAXFLOW(H)
    F′ ← ROUDING(F)
    Create a solution S′ according to integral assignment F′
    return S′
```

In Lemma 3.4 and Lemma 3.5 we prove that Algorithm 2 is polynomial in the instance size, and that it discards at most ε of the space of an optimal schedule.

Lemma 3.4 *Algorithm 2 executes in polynomial time.*

Lemma 3.5 *Let S_P be the solution Algorithm 2 returns and let Opt_P be an optimal solution for SMALL-MAXSPACE-RD for a set P of ads. Then, $f(S_P) \geq f(Opt_P) - \varepsilon$.*

Proof. Let F be a maximum flow in H and F' be the output of ROUNDING. Define

$$f(F) = \sum_{A_i \in P} \sum_{t \in \mathcal{T}} F_{A_i,t} \quad \text{and} \quad f(F') = \sum_{A_i \in P} \sum_{t \in \mathcal{T}} F'_{A_i,t}.$$

Observe that Opt_P induces a feasible flow for H with value $f(Opt_P)$. This implies that $f(F) \geq f(Opt_P)$, as F is a maximum flow. Also, note that $f(S_P) = f(F')$, then using Corollary 3.3 we have

$$f(S_P) = f(F') \geq f(F) - \varepsilon \geq f(Opt_P) - \varepsilon.$$

□

4 A PTAS for the general case

In the following, we derive a PTAS for the general case, which tackles both small and large ads. Consider an optimal solution *Opt* which schedules a subset of ads $\mathcal{A}^*$ into slots $B_1^*, B_2^*, \ldots, B_K^*$. Note that *Opt* induces a feasible configuration of large ads S_G into slots $B_1, B_2, \ldots, B_K$ such that $B_j = B_j^* \cap G$. Since all candidate configurations can be listed in polynomial time by Lemma 2.1, we may assume that we guessed the configuration S_G of large ads induced by *Opt*. We are left with the residual problem of placing small ads.

For each slot j, $1 \leq j \leq k$, the space which is unused by large ads is

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

While these values do not completely specify the capacity c_t for each type t, which is part of the input of SMALL-MAXSPACE-RD, the number of possible capacity possibilities is a constant defined by equation (1), and thus, again, we can guess the vector c. We say that a vector c is *compatible* with S_G if, for each $1 \leq j \leq K$,

$$\sum_{\substack{t \in \mathcal{T}: \\ j \in t}} c_t \varepsilon \leq u_j.$$

Let o_t^* be the occupation of small ads with type t in the solution Opt. Observe that the optimal solution induces a vector c which is compatible with S_G and such that $c_t\varepsilon \geq o_t^* - \varepsilon$ for each type t. Since we try each capacity vector, we may assume that we guessed c and solved the instance of SMALL-MAXSPACE-RD with c.

Given parameter $\varepsilon > 0$, Algorithm 3 receives as input a set of ads $\mathcal{A}$ and returns a feasible solution. First of all, the algorithm tries to schedule all ads with the first-fit heuristic and, if it is possible, the solution is returned. Otherwise, for each configuration of large ads S_G and each capacity vector c compatible with S_G, the algorithm calls the algorithm $\textsc{AlgP}_\varepsilon$ and obtains a schedule for small ads S_P. By combining the solution for large ads in S_G and the solution for small ads in S_P, it obtains a feasible solution S to the problem. The algorithm returns the best solution S_{max} found among all pair of configuration and capacity vector.

Algorithm 3 A PTAS for whole set of ads A.

procedure $\textsc{Alg}(A, \varepsilon)$
    Create a solution S_{max} using first-fit
    if S_{max} schedules the whole set $\mathcal{A}$ **then**
        return S_{max}
    $G = \{A_i \in \mathcal{A} \mid s_i \geq \varepsilon/(2^{2^K} 2^K K)\}$
    $P \leftarrow \mathcal{A} \setminus G$
    $R \leftarrow$ enumerate the set of configurations for large ads as in Lemma 2.1
    for each $S_G \in R$ **do**
        $C \leftarrow$ enumerate the set of capacity vectors c compatible with S_G
        for each $c \in C$ **do**
            $S_P \leftarrow \textsc{AlgP}_\varepsilon(P, c, \varepsilon)$
            $S \leftarrow S_G \cup S_P$
            if $f(S) > f(S_{max})$ **then**
                $S_{max} \leftarrow S'$
    return S_{max}

We show that Algorithm 3 is a PTAS. First, Lemma 4.1 shows Algorithm 3 runs in time polynomial in the size of the instance. Then, Lemma 4.2 highlights that the returned solution is feasible.

Lemma 4.1 *Algorithm 3 executes in polynomial time when K is a constant.*

Lemma 4.2 *Algorithm 3 returns a feasible solution.*

Proof. Since each configuration for large ads is feasible, S_G respects release date and deadline restrictions. Solution S_P returned by Algorithm $\textsc{AlgP}_\varepsilon$ also respects the release date and deadline restrictions to small ads. Thus, it only remains to show that for the union of S_G and S_P, the capacity of each slot capacity is not exceeded. Indeed, consider $1 \le j \le K$ and let B_j be slot of large ads scheduled by S_G. Also, let $\mathcal{A}'_t$ be the set of small ads scheduled by S_P with type t. The occupation of slot j in the combined solution is then

$$\sum_{A_i \in B_j} s_i + \sum_{\substack{t \in \mathcal{T}: \\ j \in t}} \sum_{A_i \in \mathcal{A}'_t} s_i \le \sum_{A_i \in B_j} s_i + \sum_{\substack{t \in \mathcal{T}: \\ j \in t}} c_t \varepsilon \le \sum_{A_i \in B_j} s_i + u_j \le 1,$$

where the first inequality holds because S_P is a feasible solution for SMALL-MAXSPACE-RD with vector c, and the second inequality holds because c is compatible with S_G. □

Theorem 4.3 *Algorithm 3 is a PTAS to MAXSPACE-RD with constant K.*

Proof. By Lemma 4.1 and Lemma 4.2, the algorithm executes in polynomial time and returns a feasible solution.

Let Opt be an optimal schedule and let S_{max} be the schedule returned by Algorithm 3. If the first-fit heuristic schedules the whole set $\mathcal{A}$, $f(S_{max}) = f(Opt)$ and the proof ends. Thus, assume the algorithm continues and, by Lemma 2.2, that $f(Opt) \ge 1/2$.

Denote by $\mathcal{A}^*$ the set of ads scheduled by Opt and let S'_G and S'_P be the configurations of large and small ads induced by $\mathcal{A}^* \cap G$ and $\mathcal{A}^* \cap P$, respectively, such that $f(Opt) = f(S'_G) + f(S'_P)$. Also, let c' be the capacity vector such that for each type t, c'_t is the largest integer with $c'_t\varepsilon \le o^*_t$, where o^*_t is the occupation of ads in $\mathcal{A}^* \cap P$ with type t. Clearly, c'_t is compatible with S'_G. Therefore, the algorithm constructs a solution S' for the pair of configuration S'_G and capacity vector c'.

Let Opt_P be an optimal solution for SMALL-MAXSPACE-RD with ads P and vector c'. We claim that $f(Opt_P) \ge f(S'_P) - 2 \cdot 2^K K\varepsilon$. To prove this, for each type t, let $\mathcal{A}'_t$ be a maximal subset of ads in $\mathcal{A}^* \cap P$ with type t and such that

$$\sum_{A_i \in \mathcal{A}'_t} s_i \le c'_t \varepsilon.$$

Thus, the sets $\mathcal{A}'_t$ form a feasible solution S''_P for SMALL-MAXSPACE-RD with vector c'. Since $\mathcal{A}'_t$ is maximal, either $\mathcal{A}'_t$ contains all ads in $\mathcal{A}^* \cap P$ with type t and $\sum_{A_i \in \mathcal{A}'_t} s_i = o^*_t \ge c'_t\varepsilon$, or there is a small ad $A_j \in \mathcal{A}^* \cap P$ with type t which could not be added to $\mathcal{A}'_t$ and $\sum_{A_i \in \mathcal{A}'_t} s_i > c'_t\varepsilon - s_j$. In either case,

$$\sum_{A_i \in \mathcal{A}'_t} s_i > c'_t\varepsilon - \varepsilon/(2^{2^K} 2^K K).$$

This implies that the fullness of Opt_P, which is at least the fullness of S''_P, is

$$\begin{aligned} f(Opt_P) \geq f(S''_P) &= \sum_{t\in\mathcal{T}} \sum_{A_i\in\mathcal{A}'_t} |t|s_i \\ &> \sum_{t\in\mathcal{T}} (|t|c'_t\varepsilon - |t|\varepsilon/(2^{2^K}2^K K)) \\ &> \sum_{t\in\mathcal{T}} (|t|(o^*_t - \varepsilon) - |t|\varepsilon/(2^{2^K}2^K K)) \\ &\geq f(S'_P) - 2^K K(\varepsilon + \varepsilon/(2^{2^K}2^K K)). \end{aligned}$$

Therefore, indeed $f(Opt_P) \geq f(S'_P) - 2 \cdot 2^K K\varepsilon$.

If S_P is the solution obtained by Algorithm ALGP_ε for vector c', then, by Lemma 3.5, $f(S_P) \geq f(Opt_P) - \varepsilon$. Let S' be the solution considered by combining S'_G and S_P. Since the algorithm returns the best found solution,

$$\begin{aligned} f(S_{max}) \geq f(S') &= f(S'_G) + f(S_P) \\ &\geq f(S'_G) + f(Opt_P) - \varepsilon \\ &\geq f(S'_G) + f(S'_P) - 2 \cdot 2^K K\varepsilon - \varepsilon \\ &\geq f(Opt) - 3 \cdot 2^K K\varepsilon \\ &\geq f(Opt) - 6 \cdot 2^K K\varepsilon \cdot f(Opt) \\ &= (1 - 6 \cdot 2^K K\varepsilon) f(Opt), \end{aligned}$$

where the fourth inequality holds because $2 \cdot 2^K K > 1$, and the last inequality holds because $f(Opt) > 1/2$.

For any $\varepsilon' > 0$, by letting $\varepsilon = \varepsilon'/(6 \cdot 2^K K)$, the obtained solution has fullness at least $(1 - \varepsilon')f(Opt)$. Therefore, Algorithm 3 is a PTAS for MAXSPACE-RD. □

5 Final remarks

This paper presented a PTAS for MAXSPACE-RD, which is a generalization of MAXSPACE that deals with release dates and deadlines. To our knowledge, this is the first approximation scheme to this MAXSPACE variant. When the number of bins is given in the input, we can show that MAXSPACE-RD is strongly NP-hard, and thus does not admit an FPTAS. We left open the question of whether the problem with a constant number of slots admits an FPTAS. In future works, we will consider the variant in which the value of an ad is given in the input, and may be unrelated to its size. This variant is a generalization of the Multiple Knapsack Problem [3], which is strongly NP-hard even for $K = 2$.

References

[1] M. Adler, P. B. Gibbons, and Y. Matias. Scheduling space-sharing for internet advertising. *Journal of Scheduling*, 5(2):103–119, 2002.

[2] R. K. Ahuja. *Network flows: theory, algorithms, and applications.* Pearson Education, 2017.

[3] C. Chekuri and S. Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.

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

[5] B. C. Dean and M. 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.

[6] A. Freund and J. S. Naor. Approximating the advertisement placement problem. In *Proceedings of International Conference on Integer Programming and Combinatorial Optimization*, pages 415–424, 2002.

[7] S. Kumar. *Optimization Issues in Web and Mobile Advertising: Past and Future Trends*. Springer, 2015.

[8] D. Silverman. Iab internet advertising revenue report. *In Proceedings of Interactive Advertising Bureau. New York*, page 26, 2010.

[9] V. V. Vazirani. *Approximation algorithms.* Springer Science & Business Media, 2013.

A Omitted proofs

Proof. [of Lemma 3.1] The loops of Lines 4 and 6 execute a constant number of iterations, since $|\mathcal{T}| = 2^K$ and the number of subsets of $\mathcal{T}$ is 2^{2^K}. The inner loop (Line 9) executes a polynomial number of iterations since $|P_W|$ is polynomial. Then, the algorithm executes in polynomial time. $\square$

Proof. [of Lemma 3.4] The maximum flow is solved in polynomial time in the size of graph H [2] and H is polynomial in the size of the instance since it has exactly one vertex per small ad and a constant number of vertices for types. The ROUDING algorithm is also polynomial, by Lemma 3.1. Then, Algorithm 2 is polynomial in the instance size. $\square$

Proof. [of Lemma 4.1] The number of configurations for large ads is polynomial, by Lemma 2.1. Thus, the loop of Line 8 executes a polynomial number of iterations. Also, the number of capacity vectors which are compatible with each such configuration is at most a constant, by equation (1). Thus, the loop of Line 10 executes a polynomial number of iterations. The call to $\mathrm{ALGP}_\varepsilon$ also runs in polynomial time, by Lemma 3.4. Therefore, the algorithm runs in polynomial time. $\square$