Online Budgeted Maximum Coverage*

Dror Rawitz†
Bar-Ilan University
Ramat Gan 52900, Israel
dror.rawitz@biu.ac.il

Adi Rosén‡
CNRS and Université de Paris
France
adiro@irif.fr

Abstract

We study the Online Budgeted Maximum Coverage (OBMC) problem. Subsets of a weighted ground set U arrive one by one, where each set has a cost. The online algorithm has to select a collection of sets, under the constraint that their cost is at most a given *budget*. Upon arrival of a set the algorithm must decide whether to accept or to irrevocably reject the arriving set, and it may also irrevocably drop previously accepted sets. The goal is to maximize the total weight of the elements covered by the sets in the chosen collection.

We give a deterministic $\frac{4}{1-r}$-competitive algorithm where r is the maximum ratio between the cost of a set and the total budget, and show that the competitive ratio of any deterministic online algorithm is $\Omega(\frac{1}{1-r})$. We further give a randomized $O(1)$-competitive algorithm. We also give a deterministic $O(\Delta)$-competitive algorithm, where Δ is the maximum weight of a set and a modified version of it with competitive ratio of $O(\min\{\Delta, \sqrt{w(U)}\})$ for the case that the total weight of the elements, $w(U)$, is known in advance. A matching lower bound of $\Omega(\min\{\Delta, \sqrt{w(U)}\})$ is given. Finally, our results, including the lower bounds, apply also to Removable Online Knapsack.

Keywords: budgeted coverage, maximum coverage, online algorithms, competitive analysis, removable online knapsack

1 Introduction

The Budgeted Maximum Coverage problem (abbreviated BMC) is the dual of the classical Set Cover problem. In Set Cover the input consists of a collection of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$ over the ground set $U = \{u_1, \ldots, u_n\}$ with cost for each set, and the goal is to find a sub-collection of sets of minimal cost, whose union covers all the elements in the ground set. In BMC we are, in addition, given weights for the elements and a budget cap B. The goal is to find a subcollection of the sets that maximizes the total weight of the union of those sets, under the constraint that the total cost of the subcollection is at most B. In other words, the goal is to find a subcollection that maximizes coverage given a knapsack constraint. In fact the Knapsack problem can be viewed as the special case of BMC in which the sets are pairwise disjoint.

In the Online Budgeted Maximum Coverage problem (OBMC) sets arrive online and the goal of an online algorithm for this problem is to find (in an online manner) a sub-collection of the sets, that maximizes the weight of the covered items, while adhering to the budget constraint. Preemption

*A preliminary version of this paper appeared in the proceedings of ESA 2016 [36].

†Supported in part by a grant from the Israeli Ministry of Science, Technology, and Space, Israel (French-Israeli project Maimonide No. 3-10996) and by the Israel Science Foundation (grant No. 497/14).

‡Research supported in part by ANR project NeTOC, and by a French-Israeli grant PHC Maimonide 31768XL.

of previously used sets is allowed, but a preempted (or rejected at arrival) set cannot be used again later. We note that preemption is necessary in order to achieve a bounded competitive ratio for this problem by a deterministic or randomized online algorithm. BMC has many applications both in its offline and online versions, such as facility location [33, 9, 8], where the budget is the number of facilities, and web streaming [38], where the budget is the number (or total size) of web objects that can be stored in memory.

The Budgeted Maximum Coverage problem, as well as its dual problem, the Set Cover problem, both in their weighted (general costs) and unweighted (unit costs) versions, are fundamental, widely studied problems (cf. [24, 41]). Even in their unweighted versions they are NP-hard problems [21]; approximation algorithms exist for their weighted versions (with approximation ratios of $O(\log n)$ for Set Cover [14] and $\frac{e}{e-1}$ for BMC [29]). These approximation ratios are best possible unless P = NP [19, 3]. The online version of the Set Cover problem has been studied in many variants (see, e.g., [16, 2, 12, 30]). As to the OBMC problem, only the unweighted case has been studied in the online setting, where a 4-competitive deterministic algorithm is given [38].

In the present paper we give the first results for the *budgeted* (i.e., when sets have varying costs) online maximum coverage problems. We give both upper and lower bounds on the competitive ratio of deterministic and randomized algorithms for this problem, in terms of a number of parameters of the instance.

1.1 Our Contributions

We present a deterministic $\frac{4}{1-r}$-competitive algorithm, where $r \triangleq \max_i \frac{c(S_i)}{B}$ is the maximum fraction of the budget needed for any single set. Our algorithm is inspired by the online algorithms for the case of unit costs [38, 4]. However, several new ideas are needed to cope with general costs, such as working with a fractional solution in the background and rounding it in an online manner to an integral solution while incurring only a small penalty. Furthermore, the natural algorithm, that results from reducing the weighted (set costs) case to the unit cost case by duplicating the sets, does not necessarily yield fractional solutions that can be readily converted to integral ones (i.e., many sets are sometimes used fractionally in those fractional solutions). Instead, we give an online algorithm for the weighted fractional setting that computes a solution which has at most one set used fractionally. Such solution can be converted to an integral one in an online manner while incurring only a small penalty (a $1/(1-r)$ factor).

On the negative side, we show that the competitive ratio of any deterministic online algorithm for OBMC must depend on r, by showing a lower bound of $\Omega(\frac{1}{1-r})$. Building on our deterministic algorithms we then also give an $O(1)$-competitive randomized algorithm for OBMC.

We further give a deterministic (Δ+1)-competitive algorithm for OBMC, where $\Delta \triangleq \max_{S \in \mathcal{S}} w(S)$ is the maximum weight of a set (defined under the assumption that all element weights are at least 1). Note that for unit weights we have that Δ is the maximum set size. If $w(U)$, i.e., the total weight of all elements in the ground set, is known in advance, we show that a slight modification of that algorithm is $O(\min\{\Delta, \sqrt{w(U)}\})$-competitive. We give a matching lower bound, namely that the competitive ratio of any deterministic online algorithm for OBMC is $\Omega(\min\{\Delta, \sqrt{w(U)}\})$, even for the special case of unit weights. Note that $w(U) = n$ in the unit weights case.

We note that by applying our deterministic upper bounds to the special case of OBMC in which the sets are pairwise disjoint, we obtain results for the Removable Online Knapsack problem. This problem is the version of Online Knapsack in which preemption is allowed. Furthermore, our deterministic lower bounds apply to this problem since they can be obtained using constructions containing pairwise disjoint sets.

1.2 Related Work

In the MAXIMUM COVERAGE problem the goal is, given an integer parameter k, to cover as many elements as possible, using at most k sets. In this case the natural greedy algorithm computes solutions whose weight is within a factor of $1-(1-\frac{1}{k})^k > 1-\frac{1}{e}$ from the optimum (see [34, 25, 24]). This ratio holds even in the more general case of nonnegative, nondecreasing, submodular set function maximization [35, 20].[1] Khuller, Moss and Naor [29] showed that MAXIMUM COVERAGE cannot be approximated to within a factor better than $\frac{e}{e-1}$, unless NP $\subseteq$ DTIME($n^{O(\log\log n)}$). Feige [19] did the same under the weaker assumption of P$\neq$NP. Ageev and Sviridenko [1] presented an approximation algorithm for MAXIMUM COVERAGE that computes solutions whose weight is within a factor of $1-(1-\frac{1}{\Delta})^{\Delta}$ from the optimum, where Δ is the maximum size of a set. Khuller et al. [29] showed that BMC can be approximated to within $\frac{e}{e-1}$. Sviridenko [40] extended this result to maximization of a monotone submodular set function subject to a budget constraint.

Saha and Getoor [38] presented a deterministic 4-competitive algorithm for the ONLINE MAXIMUM COVERAGE problem. Ausiello et al. [4] analyzed a variant of the above algorithm and showed that its competitive ratio is strictly less than 4, but that it tends to 4 as k increases. They also considered the special case of ONLINE MAXIMUM COVERAGE in which vertices are used to cover edges (i.e., an element appears in exactly two sets) and provided a simple deterministic 2-competitive algorithm for the latter that simply chooses the k largest sets seen so far. Ausiello et al. [4] also gave lower bounds 2 and $\frac{3}{2}$ for ONLINE MAXIMUM COVERAGE and for that special case, respectively, on the competitive ratio of any deterministic online algorithm. Badanidiyuru [6] and Chakrabarti and Kale [13] presented a streaming 4-approximation algorithm for maximizing a monotone submodular function subject to cardinality constraint. This algorithm can be implemented as an online algorithm. Buchbinder et al. [10] studied submodular maximization with cardinality constraints which contain MAXIMUM COVERAGE as a special case, but they mainly focused on the non-monotone case. Buchbinder, Feldman, and Schwartz [11] provided constant competitive ratio algorithms for online submodular maximization with preemption and a cardinality constraint. They also gave a deterministic 4-competitive algorithm for the monotone case.

We note that Awerbuch et al. [5] studied a problem they called ONLINE SET COVER. However they actually consider a variant of ONLINE MAXIMUM COVERAGE in which the elements arrive in an online manner, and the sets are revealed during this process. The goal is to cover as many elements as possible using k sets without preemption, where an element is considered covered only by a set that contains it which is added to the solution after the arrival of the element. Awerbuch et al. [5] gave a randomized $O(\log n \log \frac{m}{k})$-competitive algorithm for this problem.

The dual of MAXIMUM COVERAGE is the classical SET COVER problem. For the unweighted SET COVER problem, Johnson [28] and Lovász [31] showed that the greedy algorithm is an H_{Δ}-approximation algorithm, where H_n the nth harmonic number. This result was generalize by Chvátal [14] to the weighted case. Feige [19] proved a lower bound of $(1-o(1))\ln n$ on the approximability of that problem (unless NP $\subseteq$ DTIME($n^{O(\log\log n)}$)). In [37, 3] it was shown that SET COVER cannot be approximated within a factor of $c\log n$, for some $c>0$, unless P=NP. SET COVER can also be approximated to within a factor of $\Delta_U \triangleq \max_{u\in U} |\{S : u \in S\}|$ [23, 7]. However, it is NP-hard to approximate it within $\Delta_U - 1 - \varepsilon$, for any $\varepsilon > 0$, assuming $\Delta_U > 2$ [17], or within 1.36 for $\Delta_U = 2$ [18].

A certain online version of SET COVER was studied by Alon et al. [2]. In this problem the sets are known in advance, subsets of the elements arrives in an online manner, and the goal is to cover

[1] A function f is called submodular if $f(T)+f(T') \geq f(T\cup T')+f(T\cap T')$ for every two sets T and T' in the domain of f.

all seen elements with a sub collection of sets of minimal cardinality. A deterministic $O(\log m \log n)$-competitive algorithm and a nearly matching lower bound are given in [2] (n is the number of elements and m the number of sets).

Knapsack is a special case of BMC in which the sets are pairwise disjoint. Knapsack is known to be NP-hard, but admits an FPTAS [39, 26]. Removable Online Knapsack (ROK) is a special case of OBMC in which the sets are pairwise disjoint. In other words, in ROK items arrive one by one, each with its load and value. An online algorithm is required to accept or to reject an incoming item upon arrival, and it is allowed to drop previously accepted items to make room for a new item. The goal is to maximize the value accrued by the accepted items, under the constraint that their total load is within a given maximum load. Iwama and Taketomi [27] considered the special case of ROK in which the value of and item is equal to its load. They provided a deterministic competitive algorithm whose ratio is $\frac{\sqrt{5}+1}{2} \approx 1.62$, and a matching lower bound. Han, Kawase, and Makino [22] and Cygan, Jeż, and Sgall [15] gave a randomized 2-competitive algorithm and showed that the competitive ratio of any randomized online algorithm is at least $\frac{e+1}{e}$. Both lower bounds apply to OBMC. Non-removable Online Knapsack is the variant in which accepted items cannot be dropped. In this case the deterministic [32] and randomized [42] competitive ratios are known to be unbounded.

1.3 The Model

An instance of the BMC problem is composed of a weighted ground set $U = \{u_1, \ldots, u_n\}$, with each element having a known weight $w(u_i) \geq 1$ (we define all weights to be at least 1 to avoid arbitrary scaling of the weights). The instance is further composed of a collection $\mathcal{S} = \{S_1, \ldots, S_m\}$ of sets, where $S_i \subseteq U$, for every i. The cost of a set S_i is denoted $c(S_i)$. Without loss of generality we assume that the budget cap is 1, and therefore $0 < c(S_i) \leq 1$, for every i.

In OBMC the sets of $\mathcal{S}$ arrive online, where each set S_i is given by the elements that it contains, as well as its cost, $c(S_i)$. When a set arrives the online algorithm has to decide whether to *accept* it or to *reject* it, under the constraint that the currently accepted sets at any given time should have a cumulative cost not larger than 1. The algorithm can *drop* a previously accepted set, i.e., extract it from the currently accepted sets. However, a rejected or dropped set cannot later be re-accepted. The goal of the online algorithm is to maximize the total weight of the elements covered by the accepted sets.

Given a set $S \subseteq U$, we define its weight to be $w(S) = \sum_{u \in S} w(u)$. Given a sub-collection $\mathcal{C} \subseteq \mathcal{S}$, we define its cost to be $c(\mathcal{C}) = \sum_{S \in \mathcal{C}} c(S)$. In addition, we sometimes write $w(\mathcal{C})$ to denote $w(\cup_{S \in \mathcal{C}} S)$. Given an instance of the OBMC problem we define $\Delta \triangleq \max_{S \in \mathcal{S}} w(S)$, i.e., the maximum weight of a set. Note that for unit weights we have $\Delta = \max_{S \in \mathcal{S}} |S|$. Further we define $r \triangleq \max_i c(S_i)$ to be the maximum cost of a set.

2 Deterministic Lower Bound

In this section we give lower bounds on the competitive ratio of deterministic online algorithms for OBMC in terms of three parameters: the number of elements n, the weight of the heaviest set, Δ, and the maximum cost of a set, r.

We start with our deterministic lower bound in terms of n and Δ.

Theorem 2.1. *The competitive ratio of any deterministic online algorithm for OBMC is* $\Omega(\min\{\sqrt{n}, \Delta\})$.

Proof. Let ALG be any deterministic online algorithm for OBMC. Consider an input sequence containing a collection of subsets of a ground set U that contains k^2 unit weight elements, for an arbitrary

integer k. We define the input sequence given by an adversary. The input sequence starts with a set S_0, where $S_0 = \{u_1, \ldots, u_k\}$ and $c(S_0) = 1$. If ALG does not accept S_0, then the sequence terminates. Otherwise, the sequence continues with $S_1, S_2, \ldots$, such that $S_i = \{u_i\}$ and $c(S_i) = 1/k^2$, for $i \geq 1$, until either ALG drops S_0 or $i = k^2$.

To analyze the competitive ratio of ALG, note that, as defined above, there are three options as to the actual input sequence given by the adversary. We analyze the costs of ALG and OPT for each one of these sequences:

- If ALG rejects S_0, then OPT covers k elements using S_0 while ALG covers nothing.
- If ALG accepts S_0 and when S_i, for some $i \geq 1$, arrives, ALG drops S_0 and (possibly) accepts S_i, then OPT covers k elements using S_0, while ALG covers at most one element.
- If ALG accepts S_0 and never drops it, then ALG covers k elements, while OPT covers k^2 elements using $S_1, \ldots, S_{k^2}$.

Hence the competitive ratio of ALG is at least k, and the theorem follows, since $k = \sqrt{n}$ and $k = \Delta$. □

Next we move to a lower bound in terms of r.

Theorem 2.2. *The competitive ratio of any deterministic online algorithm for OBMC is $\Omega(\frac{1}{1-r})$. In particular, if $r = 1$ the competitive ratio of any deterministic algorithm is unbounded.*

Proof. Let ALG be any deterministic online algorithm for OBMC. Let $k \in \mathbb{N}$ be a sufficiently large integer with respect to r (as comes out from the calculations below). Consider the following sequence that contains a collection of subsets of a ground set U of size $k^3 + 1$. First, let $S_0 = \{u_0\}$, $w(u_0) = 1$, and $c(S_0) = r$. Also, let $S_i = \{u_i\}$, $w(u_i) = i \cdot k^{-3}$, $c(S_i) = k^{-1}$, for every $i \in \{1, \ldots, k^3\}$. The adversary gives the sequence until S_0 is dropped (or rejected initially), or continues to give the sequence until all $k^3 + 1$ sets are given, if S_0 is never dropped. To analyze the competitive ratio of ALG, note that, as defined above, there are various options as to the actual input sequence given by the adversary. We analyze the costs of ALG and OPT considering several cases as to if and when ALG drops S_0:

- ALG rejects S_0 upon arrival.

  In this case, OPT covers u_0 using S_0, while ALG covers nothing.

- ALG accepts S_0 and drops it after the arrival of S_i, for some $i \in \{1, \ldots, k^2\}$.

  In this case,
  $$w(\text{ALG}) < (1 - r + k^{-1})k \cdot i \cdot k^{-3} \leq (1 - r + k^{-1}) \ ,$$
  while $w(\text{OPT}) = 1$.

- ALG accepts S_0 and drops it after the arrival of S_i, for some $i \in \{k^2 + 1, \ldots, k^3\}$.

  In this case,
  $$w(\text{ALG}) < (1 - r + k^{-1})k \cdot i \cdot k^{-3} = i \cdot (1 - r + k^{-1}) \cdot k^{-2} \ ,$$
  while
  $$w(\text{OPT}) \geq k \cdot (i - k) \cdot k^{-3} = (i - k) \cdot k^{-2} \ .$$

- ALG accepts S_0 and never drops it.

  In this case,
  $$w(\text{ALG}) < 1 + (1-r)k \cdot k^3 \cdot k^{-3} = 1 + (1-r) \cdot k \ ,$$
  while
  $$w(\text{OPT}) \geq k \cdot (k^3 - k) \cdot k^{-3} = k - k^{-1} \ .$$

In all cases we get a ratio which is $\Omega(\frac{1}{1-r})$. □

We can conclude with the following theorem.

Theorem 2.3. *Let* ALG *be a c-competitive deterministic online algorithm for OBMC. Then* $c = \Omega(\min\{\sqrt{n}, \Delta, \frac{1}{1-r}\})$.

We note that the construction in Theorem 2.1 can be slightly modified to consist of pairwise disjoint sets. Hence, the lower bound apply to Removable Online Knapsack.

Theorem 2.4. *Let* ALG *be a c-competitive deterministic online algorithm for ROK. Then* $c = \Omega(\min\{\sqrt{n}, \Delta, \frac{1}{1-r}\})$.

3 $O(\frac{1}{1-r})$-competitive Algorithm

In this section we present a deterministic $\frac{4}{1-r}$-competitive algorithm for OBMC. In what follows we assume that $r < 1$. Otherwise, the competitive ratio of any deterministic algorithm is unbounded (see Theorem 2.2).

Roughly speaking, our algorithm is inspired by the online algorithm for the case of unit costs [38]. We use a similar greedy rule: a set joins the solution if its marginal benefit, with respect to the current solution, is high enough. However, in contrast to the unit cost algorithm, we sort the sets in the current solution by cost effectiveness (to be defined later), and consider only sets that are contained in the prefix of the sets of the current solution which sums to a cost of at most 1. This prefix may be fractional, namely there may be a set that is only partly considered, which then complicates both the algorithm and the analysis. In what follows we define some notations and then present formally the algorithm.

Definitions and notations. Our algorithm is defined based on an imaginary fractional solution to the problem that we maintain throughout receiving the input. In this fractional solution, the algorithm can use only a fraction $x(S) \in [0,1]$ of a given set S, paying only $x(S) \cdot c(S)$, and covering by set S at most an $x(S)$ fraction of every element $v \in S$. This fractional solution can be defined using the following variables. In what follows we refer by *time* i to the time *after* the algorithm has processed the ith input set. A variable with a subscript i refers to the value of the variable at time i.

- $x_i(S) \in [0,1]$, for $S \in \mathcal{S}$, is the fraction of set S used by the algorithm at time i.
- $z_i(v,S) \in [0,1]$, for $S \in \mathcal{S}$ and $v \in U$, is the fraction of v that is covered by the algorithm using set S at time i. We set $z_i(v,S) = 0$ if $v \notin S$.

We further define, given any $\vec{z}_i$, for $v \in U$, $\hat{z}_i(v) \triangleq \sum_{S \in \mathcal{S}} z_i(v,S)$.

Algorithm 1: insert$(S, \vec{x}, \vec{z})$

$\vec{x'} \leftarrow \vec{x}$; $\vec{z'} \leftarrow \vec{z}$

$x'(S) \leftarrow 1$

foreach $v \in U$ **do** $z'(v,S) \leftarrow \begin{cases} 1 - \hat{z'}(v) & v \in S, \\ 0 & v \notin S \end{cases}$

$\hat{\mathcal{S}} \leftarrow \{S : x'(S) > 0\}$; $\ell \leftarrow |\hat{\mathcal{S}}|$

Order the sets in $\hat{\mathcal{S}}$ by non-increasing value of $\rho(\vec{z'}, \vec{x'}, S)$; let $S_{j_1}, S_{j_2}, \ldots, S_{j_\ell}$ be the ordering.

$k \leftarrow \max\left\{k' \leq \ell : \sum_{i=1}^{k'-1} x'(S_{j_i}) c(S_{j_i}) < 1\right\}$

$\chi \leftarrow \min\left\{\frac{1-\sum_{i=1}^{k-1} x'(S_{j_i}) c(S_{j_i})}{c(S_{i_k})}, x'(S_{i_k})\right\}$

foreach $v \in S_{i_k}$ **do** $z'(v, S_{i_k}) \leftarrow \frac{\chi}{x'(S_{i_k})} \cdot z'(v, S_{i_k})$

$x'(S_{i_k}) \leftarrow \chi$

for $i = k+1$ **to** ℓ **do**

  $x'(S_{j_i}) \leftarrow 0$

  foreach $v \in U$ **do** $z'(v, S_{j_i}) \leftarrow 0$

return $(\vec{x'}, \vec{z'})$

The (fractional) optimization problem can now be defined by the following linear program:

$$\begin{array}{lll} \max & \sum_{v \in U} \hat{z}(v) \cdot w(v) & \\ \text{s.t.} & \sum_{S \in \mathcal{S}} x(S) \cdot c(S) \leq 1 & \\ & \hat{z}(v) \leq 1 & \forall v \in U \\ & z(v,S) \leq x(S) & \forall v \in U, S \in \mathcal{S}, \\ & x(S) \in [0,1] & \forall S \in \mathcal{S} \\ & z(v,S) \geq 0 & \forall v \in U, S \ni v \\ & z(v,S) = 0 & \forall v \in U, S \not\ni v \end{array}$$

Before presenting the algorithm we need the following further notations.

- $w(\vec{z}) \triangleq \sum_v \hat{z}(v) w(v)$, i.e., the total weight covered in a solution defined by the matrix $\vec{z}$.
- $\rho(\vec{z}, \vec{x}, S) \triangleq \frac{\sum_v z(v,S) w(v)}{x(S) c(S)}$, if $x(S) > 0$, and $\rho(\vec{z}, \vec{x}, S) \triangleq 0$, otherwise. That is, $\rho(\vec{z}, \vec{x}, S)$ stands for the total weight covered by set S in a solution defined by the matrix $\vec{z}$, divided by the "actual cost" paid for S. We call this quantity the *efficiency* of set S with respect to the solution $(\vec{z}, \vec{x})$.

3.1 The Algorithm

Our algorithm maintains two variables $\vec{z}$ and $\vec{x}$, corresponding to the variables by the same names described above, and which represent a current fractional (imaginary) solution held by the online algorithm. As the algorithm is an online algorithm, we allow it to increase $z(\cdot, S)$ and $x(S)$ only when set S arrives.

We first define a procedure **insert** that we use in the algorithm. This procedure takes a set S and inserts it into the current solution represented by the variables z and x. This changes the values of z and x to represent the new solution.

Algorithm 2: α**-greedy**; operations when set S_i arrives.

1 $\vec{x'} \leftarrow \vec{x}$, $\vec{z'} \leftarrow \vec{z}$

2 $x'(S_i) \leftarrow 1$

3 $z'(v, S_i) \leftarrow \begin{cases} 1 - \hat{z}(v) & \text{if } v \in S_i \\ 0 & \text{otherwise} \end{cases}$

4 **if** $\rho(\vec{z'}, \vec{x'}, S_i) > \alpha \cdot w(\vec{z})$ **then** $(\vec{x}, \vec{z}) \leftarrow \text{insert}(S_i, \vec{x}, \vec{z})$

We can now define the online algorithm, that we call α-greedy, for any $\alpha > 1$. The optimal value for α will be defined later in the analysis.

α-greedy. We initialize the two (vector) variables $\vec{z} \leftarrow \vec{0}$, $\vec{x} \leftarrow \vec{0}$. Then, for every set S_i that arrives, we use the operations defined in the pseudocode in Algorithm 2.

At any given time, the solution held by the online (regular, integral) algorithm consists of all the sets S for which $x(S) = 1$. As we later prove, the algorithm has, at any given time, at most one set S "used fractionally", i.e., with $x(S) \in (0, 1)$.

We claim that the algorithm is a well defined online algorithm for our problem. That is, that

1. the algorithm accepts a set only when this set arrives, i.e., a set that is not accepted when it arrives, or accepted but subsequently dropped, cannot later be part of the solution; and
2. the solution held by the algorithm at any given time is feasible, i.e., the total budget used by the algorithm is at most 1 at any given time.

To see that these two points hold observe that all the changes in the variables held by the algorithm are done in procedure **insert**. Procedure **insert** assigns a value of 1 to variable $x'(S)$ only for S which is the inserted set: an explicit assignment of 1 is only done in Line 1, and in Line 9 the value of $x'(S_{i_k})$ cannot grow compared to the previous round. This proves point (1). Point (2) requires a bit more of formalism, which is given in the proof of the following lemma.

Lemma 3.1. *For any time i, $\sum_{j \in O_i} c(S_j) \leq 1$, where $O_i = \{j : x_i(S_j) = 1\}$.*

Proof. We prove this lemma by induction on i. For the base of the induction, i.e., when $i = 0$, we have that $\vec{x}_0 = \vec{0}$. It follows that $O_0 = \emptyset$ and we are done. For the inductive step, we assume that the claim is true for $i - 1$, for $i \geq 1$ and prove it for i. If **insert** is not called during iteration i, then $x_i = x_{i-1}$ and $O_i = O_{i-1}$, and the claim follows from the inductive hypothesis. If **insert** is activated, then x_i is constructed such that $x_i(S_{j_i}) = 0$ if $j_i > k$ and $\sum_{i=1}^{k} x_i(S_{j_i}) c(S_{j_i}) \leq 1$. Hence $\sum_{j \in O_i} c(S_j) \leq 1$. □

We now give two technical claims which we use in the analysis. The first claim is immediate from the code of **insert**.

Claim 3.2. *The efficiency of a given set S remains the same throughout the period when $x(S) > 0$.*

The next claim says that at any time i, there may be at most one set S such that $x(S) \in (0, 1)$, and that if such set exists then the whole budget is used. Furthermore, if such set exists then that set is the one with minimum efficiency among the sets with non-zero x-coefficient. We note that these properties of our algorithm are the properties that allow one to obtain in an online manner an integral solution, and that a simple, natural reduction of the weighted fractional case to the unweighted case does not yield an algorithm with such properties.

Claim 3.3. *Let $\mathcal{S}_i^+ = \{S : x_i(S) > 0\}$. There is at most one set $S \in \mathcal{S}^+$ such that $x(S) < 1$. If such a set S exists then*

1. $\sum_{S \in \mathcal{S}_i^+} x_i(S)c(S) = 1$*; and*
2. $\rho(\vec{z}_i, \vec{x}_i, S) \leq \rho(\vec{z}_i, \vec{x}_i, S')$*, for any* $S' \in \mathcal{S}^+ \setminus \{S\}$.

Proof. We prove the claim by induction on i. For the basis of the induction, i.e., for $i = 0$, the claim is trivial in the empty sense. We now assume that the claim holds for $i-1$, for $i \geq 1$, and we prove it for i.

If **insert** is not invoked during iteration i, then the claim clearly holds by the induction hypothesis. If procedure **insert** is invoked we have that (according to Line 3 of α-greedy)

$$\rho(\vec{z'}, \vec{x'}, S_i) > \alpha \cdot w(z_{i-1}) = \alpha \cdot \sum_{S \in \mathcal{S}_{i-1}^+} \rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S) \cdot x_{i-1}(S) \cdot c(S) \ .$$

We now consider two cases. The first case is when a set S with $x_{i-1}(S) \in (0,1)$ exists. In that case, by the induction hypothesis we have that $\sum_{S \in \mathcal{S}_{i-1}^+} x_{i-1}(S)c(S) = 1$. It follows that

$$\rho(\vec{z'}, \vec{x'}, S_i) > \min_{S \in \mathcal{S}_{i-1}^+} \{\rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S)\}$$

(recall that $\alpha > 1$), and therefore the new set S_i is not the last set in the non-increasing order of efficiency defined in Line 5 of procedure **insert**. Points (1) and (2) for time i therefore follow from the code of **insert** and the induction hypothesis of point (2). The second case is when there is no set S with $x_{i-1}(S) \in (0,1)$. In that case the code of **insert** directly guarantees both points (1) and (2) for time i. □

3.2 Competitive Analysis

We analyze the competitive ratio of the algorithm using a charging scheme argument. We first describe the scheme in general terms – more details are given in the following paragraphs. Let OPT be an optimal solution, i.e., an optimal collection of sets. As the sets of OPT arrive, each new element u that is covered by OPT is allotted $w(u)$ monetary units (or simply money). We show that the allotted money can be moved between elements of U such that all the money is allotted to the items covered by ALG, and such that each element v that is covered by ALG has at most $\frac{4}{1-r} \cdot w(v)$ amount of money. This gives an upper bound of $\frac{4}{1-r}$ on the competitive ratio.

We now formally define the charging scheme. Let $\text{OPT}_k = \text{OPT} \cap \{S_1, \ldots, S_k\}$. If $S_i \in \text{OPT}$, let $F_i \triangleq S_i \setminus \cup_{S \in \text{OPT}_{i-1}} S$, i.e., F_i contains all the elements covered by OPT that, when S_i arrives, are covered for the first time by OPT (according to the order of arrival of the sets). When a set $S_i \in \text{OPT}$ arrives, each element $u \in F_i$ is allotted $w(u)$ money. The money that is allotted this way is always held by items covered by ALG, and may sometimes be moved between items covered by ALG; furthermore this money is partitioned into *blue money* and *red money*. We denote by $\text{blue}_i(v)$ and by $\text{red}_i(v)$ the amount of blue and red money, respectively, held by $v \in U$ at time i.

Intuitively, after the arrival of S_i, $w(v)$ money is created for any element $v \in F_i$. This amount is partitioned into blue money and red money: the amount of blue money is proportional to the coverage of v by the current solution, and the rest is red money. Blue money stays with v, while red money is distributed among the currently covered items proportionally to their contribution to the total weight of the solution. Moreover, when the coverage of an element drops, it gives both

money types to currently covered items, again proportionally to the contribution to the solution. The greedy nature of the algorithm makes sure that both money types can be bounded.

Now, the rules that govern the allotment and movement of the money are the following rules, applied when a set S_i arrives.

- **Transfer of money**. This rule is applied if the "then" part of Line 3 of **α-greedy** is reached, i.e., if procedure **insert** is invoked. In this case elements that lost coverage relinquish part of their money, and this money is transferred to elements that gain coverage. Let $\hat{z}'_i(v) \triangleq \sum_{j=1}^{i-1} z_i(v, S_j)$ and $Z_i \triangleq \{v : \hat{z}'_i(v) < \hat{z}_{i-1}(v)\}$. That is, we look at the coverage of an item v by all but the last arriving set, and observe if this coverage reduced during the course of iteration i.

  1. **Out-transfer of money**. For each $v \in Z_i$ let $\delta_v = (1 - \frac{\hat{z}'_i(v)}{\hat{z}_{i-1}(v)})$. We remove from v an amount of R_v red money which is a δ_v-fraction of its current red money, and an amount of B_v blue money which is a δ_v-fraction of its current blue money.
  2. **In-transfer of money**. These total amounts of removed red and blue money are distributed to the various red and blue money variables of $u \in S_i$ proportionally to $z_i(u, S_i) \cdot w(u)$. That is, each element $u \in S_i$ gets additional $\frac{z_i(u,S_i)\cdot w(u)}{\sum_{v\in S_i} z_i(v,S_i)\cdot w(v)} \cdot \sum_{v\in Z_i} R_v$ red money, and gets additional $\frac{z_i(u,S_i)\cdot w(u)}{\sum_{v\in S_i} z_i(v,S_i)\cdot w(v)} \cdot \sum_{v\in Z_i} B_v$ blue money.

- **Creation of new money**. If $S_i \in \text{OPT}$, $w(v)$ money is distributed for each $v \in F_i$ as follows:

  1. v gets $\hat{z}_i(v)w(v)$ newly created blue money; and
  2. a total amount of $R_i(v) = (1-\hat{z}_i(v))w(v)$ red money is created, and distributed among the items u currently covered by the algorithm, i.e., each element $u \in U$ gets additional red money in the amount of $R_i(v)\frac{\hat{z}_i(u)w(u)}{w(\vec{z}_i)}$ (i.e., the red money is distributed proportionally to the contribution of each element to the total weight of the current solution).

We now prove upper bounds on the amount of red and blue money held by any element at any given time. We start with a technical claim. In the next lemma we show that if S_i is accepted, then the amount of coverage provided by set S_i is more than α times the coverage lost due to the sets (or part of sets) pushed out.

Lemma 3.4. *Assume that $x_i(S_i) > 0$. Then,*

$$\rho(\vec{z}_i, \vec{x}_i, S_i) > \alpha \cdot \frac{\sum_{j<i} \sum_{u\in U} w(u)[z_{i-1}(u, S_j) - z_i(u, S_j)]}{x_i(S_i)c(S_i)} \ .$$

Proof. Since $x_i(S) > 0$ we know that $\rho(\vec{z}_i, \vec{x}_i, S_i) = \frac{\sum_{u\in U}(1-\hat{z}_{i-1}(u))w(u)}{c(S_i)} > \alpha \cdot w(z_{i-1})$ and that **insert** was invoked. Consider what happens to the solution $(\vec{z}_{i-1}, \vec{x}_{i-1})$ during the invocation of **insert** at iteration i. Some sets do not lose coverage (i.e., sets S for which $x_i(S) = x_{i-1}(S) = 1$). Other sets may leave the cover and their cover is lost (i.e., $x_{i-1}(S) > x_i(S) = 0$). In addition, by Claim 3.3, there may be at most one set that partly leaves the cover, and so it both retains and loses coverage (i.e., $x_{i-1}(S) > x_i(S) > 0$). Define $\mathcal{Y}_i \triangleq \{S_j : x_{i-1}(S_j) > x_i(S_j)\}$, namely $\mathcal{Y}_i$ contains the sets that lose coverage due to the invocation of **insert** during the ith iteration.

If no coverage is lost during iteration i, namely if $\mathcal{Y}_i = \emptyset$ (or $Z_i = \emptyset$), then

$$\sum_{j<i} \sum_{u\in U} w(u)[z_{i-1}(u, S_j) - z_i(u, S_j)] = 0 \ ,$$

and we are done since $x_i(S) > 0$ implies that $\rho(\vec{z}_i, \vec{x}_i, S_i) > 0$.

If $\mathcal{Y}_i \neq \emptyset$, then by Claim 3.3 and the code of **insert** it follows that $\rho(z_{i-1}, x_{i-1}, S) \leq \rho(z_i, x_i, S')$, for every $S \in \mathcal{Y}_i$ and S' such that $x_i(S') > 0$. In words, the efficiency of the sets that retain coverage is at least as high as the efficiency of the sets that lost coverage. As mentioned above, observe that there may be at most one set S_j, such that $S_j \in \mathcal{Y}_i$ and $x_i(S_j) > 0$. Define

$$\rho^i_{\min} \triangleq \min\left\{\rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S_j) : x_i(S_j) > 0, j < i\right\}$$
$$\rho(\mathcal{Y}_i) \triangleq \max\left\{\rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S_j) : S_j \in \mathcal{Y}_i\right\},$$

and we have that $\rho(\mathcal{Y}_i) \leq \rho^i_{\min}$.

Define W^L_i to be the total weight of lost coverage during the ith iteration and define W^R_i to be the total weight of retained coverage at the ith iterations. That is, define:

$$W^L_i \triangleq \sum_{j<i} \sum_{u \in U} w(u)[z_{i-1}(u, S_j) - z_i(u, S_j)]$$
$$W^R_i \triangleq \sum_{j<i} \sum_{u \in U} w(u) z_i(u, S_j)$$

Observe that $w(\vec{z}_{i-1}) = W^L_i + W^R_i$. Also, recall that $\rho(\vec{z}_i, \vec{x}_i, S_j) = \rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S_j)$, for $j < i$ and $x_i(S_j) > 0$ due to Claim 3.2. We have that

$$W^R_i = \sum_{j<i} \rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S_j) x_i(S_j) c(S_j) \geq \rho^i_{\min} \sum_{j<i} x_i(S_j) c(S_j) \ ,$$

and

$$\begin{aligned}
W^L_i &= \sum_{j<i} \rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S_j)[x_{i-1}(S_j)c(S_j) - x_i(S_j)c(S_j)] \\
&= \sum_{S_j \in \mathcal{Y}_i} \rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S_j)[x_{i-1}(S_j)c(S_j) - x_i(S_j)c(S_j)] \\
&\leq \rho(\mathcal{Y}_i) \sum_{S_j \in \mathcal{Y}_i} [x_{i-1}(S_j)c(S_j) - x_i(S_j)c(S_j)] \\
&= \rho(\mathcal{Y}_i) \sum_{j<i} [x_{i-1}(S_j)c(S_j) - x_i(S_j)c(S_j)] \\
&\leq \rho^i_{\min} \left[1 - \sum_{j<i} x_i(S_j)c(S_j)\right] \\
&\leq W^R_i \cdot \frac{1 - \sum_{j<i} x_i(S_j)c(S_j)}{\sum_{j<i} x_i(S_j)c(S_j)} \ .
\end{aligned}$$

It follows that

$$w(\vec{z}_{i-1}) = W^L_i + W^R_i \geq W^L_i + W^L_i \cdot \frac{\sum_{j<i} x_i(S_j)c(S_j)}{1 - \sum_{j<i} x_i(S_j)c(S_j)} = \frac{W^L_i}{1 - \sum_{j<i} x_i(S_j)c(S_j)} \ .$$

Now since $\mathcal{Y}_i \neq \emptyset$, we have that **insert** decreased coverage of at least one set and by the code of **insert** this implies that $\sum_S x_i(S)c(S) = 1$. Hence we have that

$$w(\vec{z}_{i-1}) \geq \frac{W^L_i}{x_i(S_i)c(S_i)} \ ,$$

and the lemma follows. □

We are now ready to give in the next lemma an upper bound on the amount of blue money that is allotted to any element $u \in U$ at any given time.

Lemma 3.5. *At any given time i and for every $u \in U$,* $\text{blue}_i(u) \leq w(u)\hat{z}_i(u) \cdot \frac{\alpha}{\alpha-1}$.

Proof. In this proof we consider separately new blue money, that was not transferred yet, and old blue money, that was transferred at least once. Observe that for each element $u \in U$ there is at most one index j such that $u \in F_j$. We denote this index by $f(u)$. We prove by induction on i that at any given time i and for every $u \in U$, we have that

1. $\text{blue-old}_i(u) \leq w(u)\hat{z}_i(u) \cdot \frac{1}{\alpha-1}$, and
2. $\text{blue-new}_i(u) \leq \begin{cases} 0 & i < f(u), \\ w(u)\hat{z}_i(u) & i \geq f(u). \end{cases}$

Observe that the lemma follows from the above two inequalities.

For $i = 0$, i.e., before the first input set arrives, the claims hold as there is no (blue) money. We now prove the claims for time $i \geq 1$, assuming that the claims hold for time $i-1$. We analyze the changes in $\text{blue-old}_i(\cdot)$ and $\text{blue-new}_i(\cdot)$ taking into account one by one, in their order, the operations of the charging scheme defined above. Recall that $\hat{z}'_i(v) \triangleq \sum_{j=1}^{i-1} z_i(v, S_j)$ and that $Z_i \triangleq \{v : \hat{z}'_i(v) < \hat{z}_{i-1}(v)\}$, and observe that the out-transfer and in-transfer phases are performed only if procedure **insert** is invoked.

- **Out-transfer of money**. Blue money (old or new) is removed from elements $u \in Z_i$. For such an element u we have at the end of the out-transfer phase that $\text{blue}_i(u) = \text{blue}_{i-1}(u) \cdot \frac{\hat{z}'_i(u)}{\hat{z}_{i-1}(u)}$. By the induction hypothesis this is at most

$$\text{blue-old}_i(u) \leq \text{blue-old}_i(u) \cdot \frac{\hat{z}'_i(u)}{\hat{z}_{i-1}(u)} \leq w(u)\hat{z}_{i-1}(u) \cdot \frac{1}{\alpha-1} \cdot \frac{\hat{z}'_i(u)}{\hat{z}_{i-1}(u)} = w(u)\hat{z}'_i(u) \cdot \frac{1}{\alpha-1}$$

  and

$$\text{blue-new}_i(u) \leq \text{blue-new}_i(u) \cdot \frac{\hat{z}'_i(u)}{\hat{z}_{i-1}(u)} \leq w(u)\hat{z}_{i-1}(u) \cdot \frac{\hat{z}'_i(u)}{\hat{z}_{i-1}(u)} = w(u)\hat{z}'_i(u) \ ,$$

  for $i > f(u)$, and $\text{blue-new}_i(u) = 0$, otherwise.

  For $u \in U \setminus Z_i$, $\text{blue-old}_i(u) = \text{blue-old}_{i-1}(u)$ and $\text{blue-old}_i(u) = \text{blue-old}_{i-1}(u)$. Using the induction hypothesis, at the end of the out-transfer phase, the same bound holds for $u \in U \setminus Z_i$ as well.

- **In-transfer of money**. First notice that there is no in-transfer of new blue money.

  Each element $u \in S_i$ gets old blue money in the amount of

$$\frac{z_i(u, S_i) \cdot w(u)}{\sum_{v \in S_i} z_i(v, S_i) \cdot w(v)} \cdot \sum_{v \in Z_i} \left(1 - \frac{\hat{z}'_i(v)}{\hat{z}_{i-1}(v)}\right) \text{blue}_{i-1}(v) \ .$$

  By the inductive hypothesis this is at most

$$\frac{z_i(u, S_i) \cdot w(u)}{\sum_{v \in S_i} z_i(v, S_i) \cdot w(v)} \cdot \sum_{v \in Z_i} (\hat{z}_{i-1}(v) - \hat{z}'_i(v)) \cdot w(v) \cdot \frac{\alpha}{\alpha-1} \ ,$$

and by Lemma 3.4 it follows that this is at most

$$\frac{z_i(u,S_i)\cdot w(u)}{\sum_{v\in S_i} z_i(v,S_i)\cdot w(v)}\cdot\frac{1}{\alpha-1}\cdot\rho(\vec{z}_i,\vec{x}_i,S_i)x_i(S_i)c(S_i) \;=\; z_i(u,S_i)\cdot w(u)\cdot\frac{1}{\alpha-1}\ .$$

Hence, using the upper bound on blue-old$_i(u)$ at the end of the out-transfer phase, we have that at the end of in-transfer phase:

$$\text{blue-old}_i(u)\le w(u)\hat{z}'_i(u)\cdot\frac{1}{\alpha-1}+w(u)z_i(u,S_i)\cdot\frac{1}{\alpha-1}=w(u)\hat{z}_i(u)\cdot\frac{1}{\alpha-1}\ .$$

Since no transfer of new blue money occurs by the scheme, it holds at the end of the out transfer phase, like at the end of the in-transfer phase that:

$$\text{blue-new}_i(u)\le\begin{cases}0 & i-1<f(u),\\ w(u)\hat{z}'_i(u) & i-1\ge f(u).\end{cases}$$

- **Creation of new money**. If $S_i\in\text{OPT}$, then any $u\in F_i$ receives $\hat{z}_i(u)w(u)$ new blue money. Observe that for such u, $f(u)=i$. Hence, using our claims as to the end of the in-transfer phase, the inductive claim holds at the end of the creation-of-new-money phase.

□

We now give an upper bound on the amount of red money an element may have.

Lemma 3.6. *At any given time i and for every $u\in U$,* $\text{red}_i(u)\le w(u)\hat{z}_i(u)\cdot\alpha\cdot c(\text{OPT}_i)$.

Proof. We prove the claim by induction on i. For $i=0$, i.e., before the first input set arrives, the claim holds as there is no (red) money. We now prove the claim for time $i\ge 1$, assuming that the claim holds for time $i-1$. We analyze the changes in $\text{red}_i(\cdot)$ taking into account one by one, in their order, the operations of the charging scheme defined above. Recall that $\hat{z}'_i(v)\triangleq\sum_{j=1}^{i-1}z_i(v,S_j)$ and $Z_i\triangleq\{v:\hat{z}'_i(v)<\hat{z}_{i-1}(v)\}$, and observe that the out-transfer and in-transfer phases are performed only if procedure **insert** is invoked.

- **Out-transfer of money**. Red money is removed from items $u\in Z_i$. For such u we have at the end of the out-transfer phase that

$$\text{red}_i(u)=\text{red}_{i-1}(u)\cdot\frac{\hat{z}'_i(u)}{\hat{z}_{i-1}(u)}\le w(u)\hat{z}_{i-1}(u)\cdot\alpha\cdot c(\text{OPT}_{i-1})\cdot\frac{\hat{z}'_i(u)}{\hat{z}_{i-1}(u)}=w(u)\hat{z}'_i(u)\cdot\alpha\cdot c(\text{OPT}_{i-1})$$

  where the inequality is due to the induction hypothesis. For $u\in U\setminus Z_i$, $\text{red}_i(u)=\text{red}_{i-1}(u)$ and using the induction hypothesis, at the end of the out-transfer phase, the same bound holds for $u\in U\setminus Z_i$ as well.

- **In-transfer of money.** The proof for this phase formalizes the following rather simple argument, roughly stated in what follows. The cost of the sets that are pushed out during iteration i is at most the cost used for the new set S_i inserted into the current solution. On the other hand the efficiency of S_i is at least as high as the efficiency of the sets that are pushed out. Therefore, and using the induction hypothesis, we have that the items (or parts of items) covered by set S_i can take upon themselves all the red money discarded from the pushed-out sets. We now give the formal proof.

We first give an upper bound on the total amount of red money that is transferred to the elements of S_i during the in-transfer phase. We denote this quantity by R.

$$\begin{aligned}
R = \sum_{v \in Z_i} R_v &= \sum_{v \in Z_i} (1 - \frac{\hat{z}'_i(v)}{\hat{z}_{i-1}(v)}) \mathrm{red}_{i-1}(v) \\
&= \sum_{v \in Z_i} \frac{\mathrm{red}_{i-1}(v)}{\hat{z}_{i-1}(v)} \sum_{j=1}^{i-1} (z_{i-1}(v, S_j) - z_i(v, S_j)) \\
&\leq \alpha \cdot c(\mathrm{OPT}_{i-1}) \sum_{v \in U} w(v) \sum_{j=1}^{i-1} (z_{i-1}(v, S_j) - z_i(v, S_j)) \\
&= \alpha \cdot c(\mathrm{OPT}_{i-1}) \sum_{j=1}^{i-1} \sum_{v \in U} w(v) z_{i-1}(v, S_j) \frac{x_{i-1}(S_j) - x_i(S_j)}{x_{i-1}(S_j)} \\
&= \alpha \cdot c(\mathrm{OPT}_{i-1}) \sum_{j=1}^{i-1} \rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S_j) c(S_j) (x_{i-1}(S_j) - x_i(S_j)) \ ,
\end{aligned}$$

where the inequality follows by the induction hypothesis.

Recall that $\mathcal{Y}_i \triangleq \{S_j : x_{i-1}(S_j) > x_i(S_j)\}$. Observe that by the code of **insert** it follows that for $S_j \in \mathcal{Y}_i$, $\rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S_j) \leq \rho(\vec{z}_i, \vec{x}_i, S_i)$ and that $\sum_{S_j \in \mathcal{Y}_i} c(S_j) \cdot (x_{i-1}(S_j) - x_i(S_j)) \leq c(S_i) x_i(S_i)$. Therefore the total amount of red money transferred is

$$R \leq \alpha \cdot c(\mathrm{OPT}_{i-1}) \cdot \rho(\vec{z}_i, \vec{x}_i, S_i) c(S_i) x_i(S_i) = \alpha \cdot c(\mathrm{OPT}_{i-1}) \cdot \sum_{v \in S_i} w(v) z_i(v, S_i) \ .$$

An element $u \in S_i$ therefore gets additional red money in the amount of

$$\frac{z_i(u, S_i) \cdot w(u)}{\sum_{v \in S_i} z_i(v, S_i) \cdot w(v)} \cdot R \leq \alpha \cdot c(\mathrm{OPT}_{i-1}) \cdot z_i(u, S_i) \cdot w(u)$$

(and an element $u \notin S_i$ does not get any additional red money during the in-transfer phase). Using the fact that $z_{i-1}(u, S_i) = 0$, and our claim on an upper bound on $\mathrm{red}_i(\cdot)$ at the end of the in-transfer phase, we conclude that at the end of the in-transfer phase, for any $u \in U$, $\mathrm{red}_i(u) \leq w(u) \hat{z}_i(u) \cdot \alpha \cdot c(\mathrm{OPT}_{i-1})$.

- **Creation of new money.** If $S_i \in \mathrm{OPT}$ then each element $u \in U$ gets additional red money in the amount of $R' \cdot \frac{\hat{z}_i(u) w(u)}{w(\vec{z}_i)}$, where $R' = \sum_{v \in F_i} (1 - \hat{z}_i(v)) w(v)$ (recall that $F_i \triangleq S_i \setminus \cup_{S \in \mathrm{OPT}_{i-1}} S$). We consider two cases depending whether or not procedure **insert** was activated.

  If **insert** is not invoked for S_i, then we have that

$$\begin{aligned}
R' = \sum_{v \in F_i} (1 - \hat{z}_i(v)) w(v) = \sum_{v \in F_i} (1 - \hat{z}_{i-1}(v)) w(v) &\leq \sum_{v \in S_i} (1 - \hat{z}_{i-1}(v)) w(v) \\
&\leq \alpha \cdot w(z_{i-1}) c(S_i) \qquad (1) \\
&= \alpha \cdot w(z_i) c(S_i) \ , \qquad (2)
\end{aligned}$$

  where (1) follows from the fact that procedure **insert** is not invoked only if the condition on Line 3 of α-greedy fails, and (2) follows since no change in $\vec{z}$ occurs in iteration i. We

therefore have that in this case during the creation of new money phase each element $u \in U$ gets additional red money in the amount of at most $\hat{z}_i(u)w(u)\alpha c(S_i)$.

We now consider the case where **insert** is invoked. Observe that new red money is created both if $x_i(S_i) \in (0,1)$ and if there is at least one set $S \in \mathcal{Y}_i$. (Recall that $\mathcal{Y}_i = \{S_j : x_{i-1}(S_j) > x_i(S_j)\}$.) If at least one of these two conditions occurs then by the code of **insert** $\sum_S c(S)x_i(S) = 1$.

By the code of **insert** we also have that for $v \in S_i$,

$$1 - \hat{z}_i(v) = 1 - \hat{z}_{i-1}(v) - z_i(v, S_i) + \sum_{S_j \in \mathcal{Y}_i} [z_{i-1}(v, S_j) - z_i(v, S_j)] \ .$$

We thus have that

$$\begin{aligned}
R' &= \sum_{v \in F_i} (1 - \hat{z}_i(v))w(v) \\
&\leq \sum_{v \in S_i} (1 - \hat{z}_i(v))w(v) \\
&= \sum_{v \in S_i} w(v) \cdot \left(1 - \hat{z}_{i-1}(v) - z_i(v, S_i) + \sum_{S_j \in \mathcal{Y}_i} [z_{i-1}(v, S_j) - z_i(v, S_j)] \right) \\
&\leq \sum_{v \in U} w(v) \cdot [1 - \hat{z}_{i-1}(v) - z_i(v, S_i)] + \sum_{v \in U} w(v) \cdot \sum_{S_j \in \mathcal{Y}_i} [z_{i-1}(v, S_j) - z_i(v, S_j)] \\
&= \sum_{v \in U} w(v) \cdot \left[\frac{z_i(v, S_i)}{x_i(S_i)} - z_i(v, S_i) \right] + \sum_{v \in U} w(v) \sum_{S_j \in \mathcal{Y}_i} \left[z_{i-1}(v, S_j) - z_{i-1}(v, S_j) \frac{x_i(S_j)}{x_{i-1}(S_j)} \right] \\
&= \sum_{v \in U} w(v) \cdot \frac{z_i(v, S_i)(1 - x_i(S_i))}{x_i(S_i)} + \sum_{v \in U} w(v) \sum_{S_j \in \mathcal{Y}_i} z_{i-1}(v, S_j) \cdot \frac{x_{i-1}(S_j) - x_i(S_j)}{x_{i-1}(S_j)} \\
&= \frac{1 - x_i(S_i)}{x_i(S_i)} \sum_{v \in U} w(v) \cdot z_i(v, S_i) + \sum_{S_j \in \mathcal{Y}_i} \frac{x_{i-1}(S_j) - x_i(S_j)}{x_{i-1}(S_j)} \sum_{v \in U} w(v) \cdot z_{i-1}(v, S_j) \\
&= \rho(\vec{z}_i, \vec{x}_i, S_i) c(S_i)(1 - x_i(S_i)) + \sum_{S_j \in \mathcal{Y}_i} \rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S_j) c(S_j)(x_{i-1}(S_j) - x_i(S_j)) \ ,
\end{aligned}$$

where the third equality follows directly from the code of **insert**, and the last equality follows from the definition of $\rho(\vec{z}, \vec{x}, S)$.

Now observe that by the code of **insert** and by Claim 3.3, and since $\sum_S c(S)x_i(S) = 1$ (which follows since new red money is created iteration i), we have that

- $\sum_{S_j \in \mathcal{Y}_i} c(S_j) \cdot (x_{i-1}(S_j) - x_i(S_j)) \leq x_i(S_i)c(S_i)$. (Informally, the total amount of cost pushed out when S_i is inserted is at most the cost used for S_i.)
- $\rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S') \leq \rho(\vec{z}_i, \vec{x}_i, S)$, for every $S' \in \mathcal{Y}_i$, and S such that $x_i(S) > 0$, and therefore $\rho(\vec{z}_{i-1}, \vec{x}_{i-1}, S') \leq w(\vec{z}_i)$, for every $S' \in \mathcal{Y}_i$.
- If $x_i(S_i) \in (0,1)$, then $\rho(\vec{z}_i, \vec{x}_i, S_i) \leq \rho(\vec{z}_i, \vec{x}_i, S)$, for every set S such that $x_i(S) = 1$, and therefore $\rho(\vec{z}_i, \vec{x}_i, S_i) \leq w(\vec{z}_i)$.

Notice that $\rho(\vec{z}_i, \vec{x}_i, S_i)(1 - x_i(S_i)) \leq w(\vec{z}_i)(1 - x_i(S_i))$, since either $\rho(\vec{z}_i, \vec{x}_i, S_i) \leq w(\vec{z}_i)$ or $1 - x_i(S_i) = 0$. Hence

$$R' \leq w(\vec{z}_i) \cdot \left((1 - x_i(S_i))c(S_i) + \sum_{S_j \in \mathcal{Y}_i} c(S_j) \cdot (x_{i-1}(S_j) - x_i(S_j)) \right) \leq w(\vec{z}_i)c(S_i) \ .$$

We therefore have that during the creation of new money phase, for the case when **insert** is invoked, each element $u \in U$ gets additional red money in the amount of at most $\hat{z}_i(u)w(u)c(S_i)$.

We thus get that in any of the two cases (whether **insert** is invoked or not), at the end of iteration i, for any $u \in U$, $\text{red}_i(u) \leq w(u)\hat{z}_i(u) \cdot \alpha \cdot c(\text{OPT}_i)$ (recall that $\alpha > 1$).

□

Using Lemmas 3.5 and 3.6 we now give a lower bound on the weight of the fractional solution $(\vec{z}, \vec{x})$.

Lemma 3.7. *Let $\alpha = 2$. Then, $w(\vec{z}) \geq \frac{1}{4} w(\cup_{S \in \text{OPT}} S)$ at termination.*

Proof. First observe that the total amount of blue and red money created during the course of the run is equal to the weight of the elements covered by OPT, and that all created money remains in the system, held by the various elements $u \in U$, until the end of the run. We therefore compare the total amount of money held at the end by the elements $u \in U$, to the weight of the elements covered by the online algorithm.

Let n be the number of sets in the input sequence. Since $c(\text{OPT}) = c(\text{OPT}_n) \leq 1$, we have, for $\alpha = 2$ and using Lemma 3.5 and Lemma 3.6, that for each $u \in U$, $\text{blue}_n(u) + \text{red}_n(u) \leq w(u)\hat{z}_n(u) \cdot (\alpha + \frac{\alpha}{\alpha - 1}) = w(u)\hat{z}_n(u) \cdot 4$. □

We now give a lower bound on the weight of the (integral) solution returned by the online algorithm, in terms of the value of the fractional solution $(\vec{z}, \vec{x})$.

Lemma 3.8. *Let ALG_i be the integral solution returned by the online algorithm after processing set S_i. Then, $w(\cup_{S \in \text{ALG}_i}) \geq (1 - r) \cdot w(\vec{z}_i)$.*

Proof. By definition we have that $\text{ALG}_i = \{S : x_i(S) = 1\}$. Each such set (fully) covers all its elements, hence $w(\cup_{S \in \text{ALG}_i}) \geq \sum_u \sum_{S \in \text{ALG}_i} w(u) z_i(u, S)$. By Claim 3.3 there may be at most one set S' such that $x_i(S') \in (0, 1)$. If such a set does not exists, then $w(\hat{z}_i) = \sum_u \sum_{S \in \text{ALG}_i} w(u) z_i(u, S)$, and we are done. If there exists a set S' such that $x_i(S') \in (0, 1)$, then by Claim 3.3 we have that $\sum_{S \in \text{ALG}_i} c(S) = 1 - x_i(S')c(S') > 1 - c(S')$. Also, due to Claim 3.3, $\rho(\vec{z}_i, \vec{x}_i, S') \leq \rho(\vec{z}_i, \vec{x}_i, S)$, for any $S \in \text{ALG}_i$. Therefore

$$w(\cup_{S \in \text{ALG}_i}) \geq \sum_{S \in \text{ALG}_i} \sum_u w(u) z_i(u, S) \geq \frac{1 - c(S')}{c(S')} \cdot \sum_u w(u) z_i(u, S') \geq \frac{1 - r}{r} \cdot \sum_u w(u) z_i(u, S') \ .$$

(Recall that r is the highest set-cost appearing in the input sequence). It follows that $w(\cup_{S \in \text{ALG}_i}) \geq (1 - r) \cdot \sum_{S \in \text{ALG}_i \cup \{S'\}} \sum_u w(u) z_i(u, S) = (1 - r) \cdot w(\vec{z}_i)$. □

It remains to give an upper bound on the competitive ratio of the algorithm.

Theorem 3.9. *Algorithm 2-greedy is $\frac{4}{1-r}$-competitive.*

Proof. By Lemma 3.7 we have that $w(\vec{z}) \geq \frac{1}{4} w(\cup_{S \in \text{OPT}} S)$ at termination. By Lemma 3.8 the total weight of the elements covered by the online algorithm is at least $(1 - r) \cdot w(\vec{z})$. □

4 $O(1)$-competitive Randomized Algorithm

In this section we give a randomized online algorithm with an $O(1)$ competitive ratio. The algorithm is based on the deterministic $\frac{4}{1-r}$-competitive algorithm from Section 3.

The algorithm is a barely random algorithm that chooses to run one of the following two algorithms, the first with probability $\frac{1}{4}$ and the second with probability $\frac{3}{4}$:

1. Always keep a single set S_j, which is the set with the highest weight seen so far.
2. Run algorithm α-**greedy** of Section 3, with $\alpha = 2$, only on sets with cost at most $\frac{1}{3}$.

It remains to analyze the competitive ratio.

Theorem 4.1. *There is an 8-competitive randomized online algorithm for OBMC.*

Proof. Let OPT be an optimal solution, and define $\text{OPT}_{>} = \{S \in \text{OPT} : c(S) > \frac{1}{3}\}$ and $\text{OPT}_{\leq} = \{S \in \text{OPT} : c(S) \leq \frac{1}{3}\}$. Observe that $|\text{OPT}_{>}| \leq 2$.

If at a certain time S_j is the set with the highest weight seen so far, then $w(S_j) \geq w(S)$, for every $S \in \text{OPT}_{>}$. Hence $w(S_j) \geq \frac{1}{2} \cdot w(\cup_{S \in \text{OPT}_{>}} S)$. Moreover, the ratio of α-**greedy** is $\frac{4}{1-1/3} = 6$ with respect to sets whose cost is at most $\frac{1}{3}$. Since $\text{OPT}_{\leq}$ is a solution with respect to sets whose cost is at most $\frac{1}{3}$, we have that the weight of a solution computed by α-**greedy** is at least $\frac{1}{3} \cdot w(\cup_{S \in \text{OPT}_{\leq}} S)$.

It follows that

$$\mathbb{E}[w(\text{ALG})] \geq \frac{1}{4} \cdot \frac{w(\cup_{S \in \text{OPT}_{>}} S)}{2} + \frac{3}{4} \cdot \frac{w(\cup_{S \in \text{OPT}_{\leq}} S)}{6} = \frac{w(\text{OPT})}{8} \ .$$

□

We note that Han, Kawase, and Makino [22] and Cygan, Jeż, and Sgall [15] showed that the competitive ratio of any randomized online algorithm for ROK is at least $\frac{e+1}{e}$, and this lower bound applies to OBMC.

5 $O(\Delta)$-competitive Algorithm

In this section we present an $O(\Delta)$-competitive algorithm for OBMC. Given an OBMC instance we define a bipartite graph $G = (\mathcal{S}, U, E)$, where $(S, u) \in E$ if and only if $u \in S$. Given a collection of sets $\mathcal{S}'$, let $G[\mathcal{S}']$ be the subgraph of G that is induced by $\mathcal{S}'$ and U. The algorithm is based on computing maximum cardinality matchings between sets ($\mathcal{S}'$, the left side of G) and elements (U, the right side of G). Let **MaxMatch** be an algorithm that solves the Maximum Cardinality Matching problem in bipartite graphs.

The OBMC algorithm works as follows. Upon arrival of a set S_i, $i \geq 1$, the algorithm constructs a solution $\mathcal{S}_i$ using the previous solution $\mathcal{S}_{i-1}$ (we initialize $\mathcal{S}_0 = \emptyset$). The algorithm looks for an element to be matched to S_i, where S_i takes precedence over sets that cost more than $c(S_i)$. This is done as follows. First a maximum matching is computed for the collection of sets that includes those sets in the already-computed solution that have cost at most $c(S_i)$, and S_i. If it is impossible to match all these sets, or their total cost exceeds 1, S_i is *rejected*, and the current solution remains unchanged. Otherwise S_i is *accepted*, and all the sets in the already-computed solution that have cost at most $c(S_i)$ are not dropped. In the latter case the algorithm then tries to extend the matching by assigning an element to those sets in $\mathcal{S}_{i-1}$ that cost more than $c(S_i)$. Such a set S_j is *dropped* if a matching cannot be obtained or if the total cost exceeds 1. Algorithm 3 shows a formal pseudo-code of this algorithm.

Algorithm 3: Match; operations when set S_i arrives.

```
1  S_i ← {S ∈ S_{i−1} : c(S) ≤ c(S_i)}
2  M ← MaxMatch(G[S_i ∪ {S_i}])
3  if |M| = |S_i ∪ {S_i}| then
4      if c(S_i ∪ {S_i}) ≤ 1 then
5          S_i ← S_i ∪ {S_i}; M_i ← M
6          S'_i ← S_{i−1} \ S_i
7          while S'_i ≠ ∅ do
8              j ← argmin_j {c(S_j) : S_j ∈ S'_i}
9              S'_i ← S'_i \ {S_j}
10             M ← MaxMatch(G[S_i ∪ {S_j}])
11             if |M| = |S_i ∪ {S_j}| then
12                 if c(S_i ∪ {S_j}) ≤ 1 then
13                     S_i ← S_i ∪ {S_j}; M_i ← M
14     else
15         S_i ← S_{i−1}; M_i ← M_{i−1}
16 else
17     S_i ← S_{i−1}; M_i ← M_{i−1}
```

In what follows ALG denotes the collection of sets that is output by the algorithm. We first prove that the solution is feasible.

Lemma 5.1. $c(\text{ALG}) \leq 1$.

Proof. We prove by induction on i that after each iteration we have $c(\mathcal{S}_i) \leq 1$. The base case is trivial, since $\mathcal{S}_0 = \emptyset$. For the inductive step, assume that $c(\mathcal{S}_{i-1}) \leq 1$. $\mathcal{S}_i$ is initially a subset of $\mathcal{S}_{i-1}$ (Line 1) and therefore it is feasible at this stage due to the inductive hypothesis. Furthermore, we add sets to $\mathcal{S}_i$ only if the budget constraint is not violated (Lines 4 and 12). □

We now show that the number of sets in the solution never decreases, and may increase by at most one set in any step.

Lemma 5.2. $|\mathcal{S}_{i-1}| \leq |\mathcal{S}_i| \leq |\mathcal{S}_{i-1}| + 1$, *for every* $i > 0$.

Proof. The second inequality is obvious as the sets that the algorithm is trying to include in $\mathcal{S}_i$ are $\mathcal{S}_{i-1} \cup \{S_i\}$. As to the first inequality, observe that if S_i is rejected, then $\mathcal{S}_i = \mathcal{S}_{i-1}$. Furthermore, if S_i is taken into $\mathcal{S}_i$, then at most one set from $\mathcal{S}_{i-1}$ is dropped and not taken into $\mathcal{S}_i$.

To see the latter claim, observe that $c(S_i)$ is less than the cost of any set considered in $\mathcal{S}'_i$ when it is initially defined (Line 9). Hence, since $\mathcal{S}_i$ was feasible, only the last, most expensive, set in $\mathcal{S}'_i$ could fail the test of Line 12 (i.e., if a set fails the test of Line 12, no further set will fail either the test of Line 11, or the test of Line 12.) Furthermore, if one set from $\mathcal{S}'_i$ fails the test of Line 11, then no further set will fail the test of Line 12.

In addition, we show that if a set S_j fails the test of Line 11 during the ith iteration, no further set $S_{j'} \in \mathcal{S}'_i$ will fail the test of Line 11 during the ith iteration. Let S_j be the first set that fails the test of Line 11 during the ith iteration (if such a set exists). We consider two matchings: the

value of M_i when set S_j failed the test of Line 11 (i.e., the matching which matches the sets of the current $\mathcal{S}_i$), and the matching induced by M_{i-1} on the sets in $\mathcal{S}_i \cup \{S_j\}$. Construct a path as follows. Start with $\ell = 0$ and $T_0 = S_j$. If $T_\ell = S_i$ stop, and otherwise increase ℓ by 1, move to the element $u_{\ell+1}$ that is assigned to T_ℓ by M_{i-1}. From u_ℓ move to the set that is assigned to it by M_i. Such a set exists, since otherwise there exists an augmenting path for S_j with respect to M_i. Since a cycle cannot be formed, the path must terminate at S_i. It follows that M_i may be extended to a matching of $\mathcal{S}_{i-1} \setminus \{S_j\} \cup \{S_i\}$ by matching the sets that are on the above path as in M_i and the rest of the sets as in M_{i-1}. Therefore, no further set $S_{j'} \in \mathcal{S}'_i$ will fail the test of Line 11. □

Observation 5.3. $|\cup_{S \in \text{ALG}} S| \geq |\text{ALG}|$.

Let OPT denote an optimal collection of sets. We partition OPT $\setminus$ ALG into two collections. Let τ be the cost of the cheapest set that was either rejected upon arrival due to the budget constraint, or dropped later due to the budget constraint. More formally, define

$$\begin{aligned}
\tau_1 &= \min\{\{c(S_i) : S_i \text{ was rejected by Line 4 at the } i\text{th iteration}\} \cup \{\infty\}\} \ , \\
\tau_2 &= \min\{\{c(S_i) : S_i \text{ was dropped by Line 12 at the } j\text{th iteration}\} \cup \{\infty\}\} \ , \\
\tau &= \min\{\tau_1, \tau_2\} \ ,
\end{aligned}$$

and define

$$\begin{aligned}
\text{OPT}' &= \{S \in \text{OPT} \setminus \text{ALG} : c(S) < \tau\} \ , \\
\text{OPT}'' &= \{S \in \text{OPT} \setminus \text{ALG} : c(S) \geq \tau\} \ .
\end{aligned}$$

The next two lemmas essentially give the upper bound on the competitive ratio of the algorithm. The first lemma shows that the elements that are covered by the sets in OPT$'$ are covered by the sets in ALG.

Lemma 5.4. $\cup_{S \in \text{OPT}'} S \subseteq \cup_{S \in \text{ALG}} S$.

Proof. We prove by induction on i that if $j \leq i$ and $c(S_j) < \tau$, then $S_j \subseteq \cup_{S \in \mathcal{S}_i} S$. The base case of $i = 1$ is trivial since $\mathcal{S}_1 = \{S_1\}$. For the inductive step, assume that the claim holds for $i - 1$.

Consider first the case that S_i is rejected. Then $\mathcal{S}_i = \mathcal{S}_{i-1}$. If $c(S_i) \geq \tau$ the claim holds by the induction hypothesis. If $c(S_i) < \tau$, by the definition of τ S_i must have been rejected by the test of Line 3. It follows that all the elements of S_i are covered by M_{i-1}, i.e., by the sets of $\mathcal{S}_{i-1}$, and the claim holds by the induction hypothesis.

Now consider the second case when S_i is accepted. If no set $S_j \in \mathcal{S}_{i-1}$ is dropped we are done. Assume then that a set S_j, for $j < i$, is dropped. If $c(S_j) \geq \tau$, then we are done. Otherwise, by the definition of τ, it must be that S_j is dropped due to Line 11. In this case, all elements of S_j are matched by sets in M_i, and the claim holds by the induction hypothesis. □

Next we show that the size of OPT$''$ is bounded from above by the size of ALG.

Lemma 5.5. $|\text{OPT}''| \leq |\text{ALG}|$.

Proof. If OPT$'' = \emptyset$, then we are done. Otherwise, let $S_j \in$ OPT$''$ be the least costly set in OPT$''$. By the definition OPT$''$, there exists an index $i \geq j$ such that during the ith iteration set S_j is either rejected or dropped by a budget constraint (i.e., it does not pass the test of either Line 12 or Line 4).

If S_j fails the budget test of Line 12 during iteration i, for $i > j$, it follows that $\sum_{S \in \mathcal{S}_i} c(S) + c(S_j) > 1$ and $c(S) \leq c(S_j)$ for every, $S \in \mathcal{S}_i$ (see proof of Lemma 5.2). Hence, $c(S_j) > 1/(|\mathcal{S}_i| + 1)$,

and due to Lemma 5.2, it follows that $c(S_j) > 1/(|\text{ALG}| + 1)$. On the other hand, if S_j is rejected upon arrival by Line 4, then it follows that $\sum_{S \in \mathcal{S}_j} c(S) + c(S_j) > 1$ and $c(S) \leq c(S_j)$ for every, $S \in \mathcal{S}_j$, where $\mathcal{S}_j$ is as it was initialized in Line 1. Hence, $c(S_j) > 1/(|\mathcal{S}_j| + 1)$, and since S_j may only increase in size during the iteration, we have also that $c(S_j) > 1/(|\mathcal{S}_j| + 1)$ at the end of the iteration. As in the previous case, we have that $c(S_j) > 1/(|\text{ALG}| + 1)$ by Lemma 5.2.

Since $c(S_j) > 1/(|\text{ALG}|+1)$, it follows that $|\text{OPT}''| \cdot \frac{1}{|\text{ALG}|+1} < c(\text{OPT}'') \leq 1$. Thus $|\text{OPT}''| < |\text{ALG}|+1$, and the lemma follows. □

It remains to give an upper bound on the competitive ratio of Algorithm **Match**.

Theorem 5.6. *Algorithm **Match** is $(\Delta + 1)$-competitive.*

Proof. We have that

$$\begin{aligned} w(\cup_{S \in \text{OPT}} S) &\leq w(\cup_{S \in (\text{OPT} \cap \text{ALG}) \cup \text{OPT}'} S) + w(\cup_{S \in \text{OPT}''} S) && (3) \\ &\leq w(\cup_{S \in \text{ALG}} S) + w(\cup_{S \in \text{OPT}''} S) && (4) \\ &\leq w(\cup_{S \in \text{ALG}} S) + |\text{OPT}''| \cdot \Delta && (5) \\ &\leq w(\cup_{S \in \text{ALG}} S) + |\text{ALG}| \cdot \Delta && (6) \\ &\leq (\Delta + 1) \cdot w(\cup_{S \in \text{ALG}} S) \ , && (7) \end{aligned}$$

where (3) is since $\text{OPT} \setminus \text{ALG} = \text{OPT}' \cup \text{OPT}''$, (4) follows from Lemma 5.4, (5) follows from the definition of Δ, (6) is due to Lemma 5.5, and (7) is due to Observation 5.3 and the assumption that the weight of an element is at least 1. □

5.1 $w(U)$ is known in advance

We obtain a deterministic algorithm by running algorithm **Match** with the additional rule that if a set S_i with $w(S_i) \geq \sqrt{w(U)}$ arrives, then we drop all currently taken sets, take set S_i, and never change further the solution (i.e., the final output solution is $\{S_i\}$).

Observe that if there exists a set $S_i \in \mathcal{S}$ with $w(S_i) \geq \sqrt{w(U)}$, then the total weight of the elements in the sets of ALG is at least $\sqrt{w(U)}$, while the total weight of the elements in the sets of OPT is at most $w(U)$. If no such set exists then ALG is equivalent to **Match** which is $(\Delta + 1) < (\sqrt{w(U)} + 1)$-competitive. This leads to the following result.

Theorem 5.7. *There exists a deterministic online algorithm whose competitive ratio is $O(\min\{\Delta, \sqrt{w(U)}\}$, provided that $w(U)$ is known in advance.*

Note that for the unit-weight case $w(U)$ equals n, and we thus get an $O(\min\{\Delta, \sqrt{n}\})$-competitive deterministic algorithm for this case.

6 Conclusion

We have studied the Online Budgeted Maximum Coverage problem. We presented a deterministic online algorithms in terms of three parameters of the given instance, and we gave deterministic matching lower bounds bases on these parameters. Both our upper and lower bounds on the deterministic competitive ratio apply to Removable Online Knapsack which is the preemptive version of the Online Knapsack problem. We also provided a randomized $O(1)$-competitive algorithm. which is based on the one of our deterministic online algorithms.

We briefly mention some possible future research directions. It would be interesting to design an $O(\sqrt{w(U)})$-competitive deterministic algorithm that does not require advance knowledge of $w(U)$,

and to devise a single deterministic algorithm that obtains as a competitive ratio the minimum of all deterministic competitive ratios shown in this paper. In addition, it would be interesting to extend the results to the online monotone submodular maximization subject to a budget constraint.

Acknowledgments

We would like to thank an anonymous referee for improving the lower bound to $\Omega(1/(1-r))$. (The lower bound that appeared in the conference version [36] was $\Omega(1/\sqrt{1-r})$.) We also thank the referee for a suggestion that lead to an improvement in the constant randomized competitive ratio.

References

[1] A. A. Ageev and M. Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.

[2] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. The online set cover problem. *SIAM Journal on Computing*, 39(2):361–370, 2009.

[3] N. Alon, D. Moshkovitz, and S. Safra. Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms*, 2(2):153–177, 2006.

[4] G. Ausiello, N. Boria, A. Giannakos, G. Lucarelli, and V. T. Paschos. Online maximum k-coverage. *Discrete Applied Mathematics*, 160(13-14):1901–1913, 2012.

[5] B. Awerbuch, Y. Azar, A. Fiat, and F. T. Leighton. Making commitments in the face of uncertainty: How to pick a winner almost every time. In *28th Annual ACM Symposium on the Theory of Computing*, pages 519–530, 1996.

[6] A. Badanidiyuru. Buyback problem - approximate matroid intersection with cancellation costs. In *38th Annual Intl. Colloquium on Automata, Languages and Programming*, volume 6755 of *LNCS*, pages 379–390, 2011.

[7] R. Bar-Yehuda and S. Even. A linear time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981.

[8] O. Berman, D. Bertsimas, and R. C. Larson. Locating discretionary service facilities, ii: Maximizing market size, minimizing inconvenience. *Operations Research*, 43(4):623–632, 1995.

[9] O. Berman, R. C. Larson, and N. Fouska. Optimal location of discretionary service facilities. *Transportation Science*, 26(4):201–611, 1992.

[10] N. Buchbinder, M. Feldman, J. Naor, and R. Schwartz. Submodular maximization with cardinality constraints. In *25th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 1433–1452, 2014.

[11] N. Buchbinder, M. Feldman, and R. Schwartz. Online submodular maximization with preemption. *ACM Transactions on Algorithms*, 15(3):30:1–30:31, 2019.

[12] N. Buchbinder and J. Naor. Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34(2):270–286, 2009.

[13] A. Chakrabarti and S. Kale. Submodular maximization meets streaming: Matchings, matroids, and more. In *17th Intl. Conference on Integer Programming and Combinatorial Optimization*, volume 8494 of *LNCS*, pages 210–221, 2014.

[14] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[15] M. Cygan, L. Jeż, and J. Sgall. Online knapsack revisited. *Theory of Computing Systems*, 2016. To appear.

[16] M. Demange and V. T. Paschos. On-line vertex-covering. *Theoretical Computer Science*, 332(1–3):83–108, 2005.

[17] I. Dinur, V. Guruswami, S. Khot, and O. Regev. A new multilayered PCP and the hardness of hypergraph vertex cover. *SIAM Journal on Computing*, 34(5):1129–1146, 2005.

[18] I. Dinur and S. Safra. The importance of being biased. In *34th Annual ACM Symposium on the Theory of Computing*, pages 33–42, 2002.

[19] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[20] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. Analysis of approximation algorithms for maximizing submodular set function II. *Mathematical Programming Study*, 8:73–87, 1978.

[21] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.

[22] X. Han, Y. Kawase, and K. Makino. Randomized algorithms for online knapsack problems. *Theoretical Computer Science*, 562:395–405, 2015.

[23] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982.

[24] D. S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problem*. PWS Publishing Company, 1997.

[25] D. S. Hochbaum and A. Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Naval Research Logistics*, 45(6):615–627, 1998.

[26] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, 1975.

[27] K. Iwama and S. Taketomi. Removable online knapsack problems. In *29th Annual Intl. Colloquium on Automata, Languages and Programming*, volume 2380 of *LNCS*, pages 293–305, 2002.

[28] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[29] S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

[30] D. Komm, R. Královic, and T. Mömke. On the advice complexity of the set cover problem. In *7th International Computer Science Symposium in Russia*, pages 241–252, 2012.

[31] L. Lovász. On the ratio of optimal integeral and fractional solutions. *Discrete Mathematics*, 13:383–390, 1975.

[32] A. Marchetti-Spaccamela and C. Vercellis. Stochastic on-line knapsack problems. *Math. Program.*, 68:73–104, 1995.

[33] N. Megiddo, E. Zemel, and S. L. Hakimi. The maximum coverage location problem. *SIAM Journal on Algebraic and Discrete Methods*, 4(2):253–261, 1983.

[34] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., 1988.

[35] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions I. *Mathematical Programming*, 14(1):265–294, 1978.

[36] D. Rawitz and A. Rosén. Online budgeted maximum coverage. In *24th Annual European Symposium on Algorithms*, volume 57 of *LIPIcs*, pages 73:1–73:17, 2016.

[37] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *29th Annual ACM Symposium on the Theory of Computing*, pages 475–484, 1997.

[38] B. Saha and L. Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *SIAM International Conference on Data Mining*, pages 697–708, 2009.

[39] S. Sahni. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM*, 22(1):115–124, 1975.

[40] M. Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters*, 32(1):41–43, 2004.

[41] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin Heidelberg New York, 2001.

[42] Y. Zhou, D. Chakrabarty, and R. M. Lukose. Budget constrained bidding in keyword auctions and online knapsack problems. In *4th international Workshop on Internet and Network Economics*, volume 5385 of *LNCS*, pages 566–576, 2008.