Online Submodular Maximization Problem with Vector Packing Constraint * †

T-H. Hubert Chan[1], Shaofeng H.-C. Jiang[1], Zhihao Gavin Tang[1], and Xiaowei Wu[1]

1 **Department of Computer Sicence, The University of Hong Kong, Hong Kong**
hubert,sfjiang,zhtang,xwwu@cs.hku.hk

Abstract

We consider the online vector packing problem in which we have a d dimensional knapsack and items u with weight vectors $\mathbf{w}_u \in \mathbb{R}_+^d$ arrive online in an arbitrary order. Upon the arrival of an item, the algorithm must decide immediately whether to discard or accept the item into the knapsack. When item u is accepted, $\mathbf{w}_u(i)$ units of capacity on dimension i will be taken up, for each $i \in [d]$. To satisfy the knapsack constraint, an accepted item can be later disposed of with no cost, but discarded or disposed of items cannot be recovered. The objective is to maximize the utility of the accepted items S at the end of the algorithm, which is given by $f(S)$ for some non-negative monotone submodular function f.

For any small constant $\epsilon > 0$, we consider the special case that the weight of an item on every dimension is at most a $(1-\epsilon)$ fraction of the total capacity, and give a polynomial-time deterministic $O(\frac{k}{\epsilon^2})$-competitive algorithm for the problem, where k is the (column) sparsity of the weight vectors. We also show several (almost) tight hardness results even when the algorithm is computationally unbounded. We first show that under the ϵ-slack assumption, no deterministic algorithm can obtain any $o(k)$ competitive ratio, and no randomized algorithm can obtain any $o(\frac{k}{\log k})$ competitive ratio. We then show that for the general case (when $\epsilon = 0$), no randomized algorithm can obtain any $o(k)$ competitive ratio.

In contrast to the $(1+\delta)$ competitive ratio achieved in Kesselheim et al. (STOC 2014) for the problem with random arrival order of items and under large capacity assumption, we show that in the arbitrary arrival order case, even when $\|\mathbf{w}_u\|_\infty$ is arbitrarily small for all items u, it is impossible to achieve any $o(\frac{\log k}{\log \log k})$ competitive ratio.

1998 ACM Subject Classification F.1.2 Modes of Computation, G.1.6 Optimization

Keywords and phrases Submodular Maximization, Free-disposal, Vector Packing

Digital Object Identifier 10.4230/LIPIcs.ESA.2017.1

1 Introduction

Online Vector Packing Problem. We consider the following online submodular maximization problem with vector packing constraint. Suppose we have a d dimensional knapsack, and items arrive online in an arbitrary order. Each item $u \in \Omega$ has a *weight vector* $\mathbf{w}_u \in \mathbb{R}_+^d$, i.e., when item $u \in \Omega$ is accepted, for each $i \in [d]$, item u will take up $\mathbf{w}_u(i)$ units of capacity on every dimension i of the knapsack. By rescaling the weight vectors, we can assume that each of the d dimensions has capacity 1. Hence we can assume w.l.o.g. that $\mathbf{w}_u \in [0,1]^d$ for all $u \in \Omega$. The *(column) sparsity* [6, 26] is defined as the minimum number k such that every

* This work was partially supported by the Hong Kong RGC under the grant 17202715.
† The full version of this paper can be found at https://arxiv.org/abs/1706.06922.

weight vector $\mathbf{w}_u$ has at most k non-zero coordinates. The objective is to pack a subset of items with the maximum utility into the knapsack, where the utility of a set S of items is given by a non-negative monotone submodular function $f : 2^\Omega \to \mathbb{R}_+$.

The vector packing constraint requires that the accepted items can take up a total amount of at most 1 capacity on each of the d dimensions of the knapsack. However, as items come in an arbitrary order, it can be easily shown that the competitive ratio is arbitrarily bad, if the decision of acceptance of each item is decided online and cannot be revoked later. In the literature, when the arrival order is arbitrary, the *free disposal* feature [19] is considered, namely, an accepted item can be disposed of when later items arrive. On the other hand, we cannot recover items that are discarded or disposed of earlier.

We can also interpret the problem as solving the following program online, where variables pertaining to u arrive at step $u \in \Omega$. We assume that the algorithm does not know the number of items in the sequence. The variable $x_u \in \{0, 1\}$ indicates whether item u is accepted. During the step u, the algorithm decides to set x_u to 0 or 1, and may decrease $x_{u'}$ from 1 to 0 for some $u' < u$ in order to satisfy the vector packing constraints.

$$\begin{aligned} \max \quad & f(\{u \in \Omega : x_u = 1\}) \\ \text{s.t.} \quad & \textstyle\sum_{u\in\Omega} \mathbf{w}_u(i) \cdot x_u \le 1, \qquad \forall i \in [d] \\ & x_u \in \{0, 1\}, \qquad \forall u \in \Omega. \end{aligned}$$

In some existing works [8, 18, 27, 26], the items are decided by the adversary, who sets the value (the utility of a set of items is the summation of their values) and the weight vector of each item, but the items arrive in a uniformly random order. This problem is sometimes referred to as Online Packing LPs with random arrival order, and each choice is irrevocable. To emphasize our setting, we refer to our problem as Online Vector Packing Problem (with submodular objective and free disposal).

Competitive Ratio. After all items have arrived, suppose $S \subset \Omega$ is the set of items currently accepted (excluding those that are disposed of) by the algorithm. The objective is $\mathsf{ALG} := f(S)$. Note that to guarantee feasibility, we have $\sum_{u\in S} \mathbf{w}_u \le \mathbf{1}$, where $\mathbf{1}$ denotes the d dimensional all-one vector. The competitive ratio is defined as the ratio between the optimal objective OPT that is achievable by an offline algorithm and the (expected) objective of the algorithm: $\mathsf{r} := \frac{\mathsf{OPT}}{\mathsf{E[ALG]}} \ge 1$.

1.1 Our Results and Techniques

We first consider the Online Vector Packing Problem *with slack*, i.e., there is a constant $\epsilon > 0$ such that for all $u \in \Omega$, we have $\mathbf{w}_u \in [0, 1 - \epsilon]^d$, and propose a deterministic $O(\frac{k}{\epsilon^2})$-competitive algorithm, where k is the sparsity of weight vectors.

▶ **Theorem 1.** *For the Online Vector Packing Problem with ϵ slack, there is a (polynomial-time) deterministic $O(\frac{k}{\epsilon^2})$-competitive algorithm for the Online Vector Packing Problem.*

Observe that by scaling weight vectors, Theorem 1 implies a bi-criteria $(1 + \epsilon, \frac{k}{\epsilon^2})$-competitive algorithm for general weight vectors, i.e., by relaxing the capacity constraint by an ϵ fraction, we can obtain a solution that is $O(\frac{k}{\epsilon^2})$-competitive compared to the optimal solution (with the augmented capacity).

We show that our competitive ratio is optimal (up to a constant factor) for deterministic algorithms, and almost optimal (up to a logarithmic factor) for any (randomized) algorithms. Moreover, all our following hardness results (Theorem 2, 3 and 4) hold for algorithms with **unbounded computational power**.

▶ **Theorem 2** (Hardness with Slack). *For the Online Vector Packing Problem with slack $\epsilon \in (0, \frac{1}{2})$, any deterministic algorithm has a competitive ratio $\Omega(k)$, even when the utility function is linear and all items have the same value, i.e., $f(S) := |S|$; for randomized algorithms, the lower bound is $\Omega(\frac{k}{\log k})$.*

We then consider the hardness of the Online Vector Packing Problem (without slack) and show that no (randomized) algorithm can achieve any $o(k)$-competitive ratio.

▶ **Theorem 3** (Hardness without Slack). *Any (randomized) algorithm for the Online Vector Packing Problem has a competitive ratio $\Omega(k)$, even when $f(S) := |S|$.*

As shown by [26], for the Online Vector Packing Problem with random arrival order, if we have $\|\mathbf{w}_u\|_\infty = O(\frac{\epsilon^2}{\log k})$ for all items $u \in \Omega$, then a $(1+\epsilon)$ competitive ratio can be obtained. Hence, a natural question is whether better ratio can be achieved under this "small weight" assumption. For example, if $\max_{u\in\Omega}\{\|\mathbf{w}_u\|_\infty\}$ is arbitrarily small, is it possible to achieve a $(1+\epsilon)$ competitive ratio like existing works [18, 16, 27, 26]? Unfortunately, we show that even when all weights are arbitrarily small, it is still not possible to achieve any constant competitive ratio.

▶ **Theorem 4** (Hardness under Small Weight Assumption). *There does not exist any (randomized) algorithm with an $o(\frac{\log k}{\log\log k})$ competitive ratio for the Online Vector Packing Problem, even when $\max_{u\in\Omega}\{\|\mathbf{w}_u\|_\infty\}$ is arbitrarily small and $f(S) := |S|$.*

Our hardness result implies that even with free disposal, the problem with arbitrary arrival order is strictly harder than its counter part when the arrival order is random. For space reason, we defer the proof the Theorem 3 and 4 to the full version of the paper [10].

Our Techniques. To handle submodular functions, we use the standard technique by considering marginal cost of an item, thereby essentially reducing to linear objective functions. However, observe that the hardness results in Theorems 2, 3 and 4 hold even for linear objective function where every item has the same value. The main difficulty of the problem comes from the weight vectors of items, i.e., when items conflict with one another due to multiple dimensions, it is difficult to decide which items to accept. Indeed, even the offline version of the problem has an $\Omega(\frac{k}{\log k})$ NP-hardness of approximation result [24, 28].

For the case when $d = 1$, i.e., $\mathbf{w}_u \in [0, 1]$, it is very natural to compare items based on their *densities* [23], i.e., the value per unit of weight, and accept the items with maximum densities. A naive solution is to use the maximum weight $\|w_u\|_\infty$ to reduce the problem to the 1-dimensional case, but this can lead to $\Omega(d)$-competitive ratio, even though each weight vector has sparsity $k \ll d$. To overcome this difficulty, we define for each item a density on **each** of the d dimensions, and make items comparable on any particular dimension.

Even though our algorithm is deterministic, we borrow techniques from randomized algorithms for a variant of the problem with matroid constraints [7, 9]. Our algorithm maintains a fractional solution, which is rounded at every step to achieve an integer solution. When a new item arrives, we try to accept the item by **continuously** increasing its accepted fraction (up to 1), while for each of its k non-zero dimensions, we decrease the fraction of the currently least dense accepted item, as long as the rate of increase in value due to the new item is at least some factor times the rate of loss due to disposing of items fractionally.

The rounding is simple after every step. If the new item is accepted with a fraction larger than some threshold α, then the new item will be accepted completely in the integer solution; at the same time, if the fraction of some item drops below some threshold β, then the corresponding item will be disposed of completely in the integer solution. The ϵ slack

assumption is used to bound the loss of utility due to rounding. The high level intuition of why the competitive ratio depends on the sparsity k (as opposed to the total number d of dimensions) is that when a new item is fractionally increased, at most k dimensions can cause other items to be fractionally disposed of.

Then, we apply a standard argument to compare the value of items that are eventually accepted (the utility of our algorithm) with the value of items that are **ever** accepted (but maybe disposed of later). The value of the latter is in turn compared with that of an optimal solution to give the competitive ratio.

1.2 Related Work

The Online Vector Packing Problem (with free disposal) is general enough to subsume many well-known online problems. For instance, the special case $d = 1$ becomes the Online Knapsack Problem [23]. The offline version of the problem captures the k-Hypergraph b-Matching Problem (with sparsity k and $\mathbf{w}_u \in \{0, \frac{1}{b}\}^d$, where d is the number of vertices), for which an $\Omega(\frac{k}{b \log k})$ NP-hardness of approximation is known [24, 28], for any $b \leq \frac{k}{\log k}$. In contrast, our hardness results are due to the online nature of the problem and hold even if the algorithms have unbounded computational power.

Free Disposal. The free disposal setting was first proposed by Feldman et al. [19] for the online edge-weighted bipartite matching problem with arbitrary arrival order, in which the decision whether an online node is matched to an offline node must be made when the online node arrives. However, an offline node can dispose of its currently matched node, if the new online node is more beneficial. They showed that the competitive ratio approaches $1 - \frac{1}{e}$ when the number of online nodes each offline node can accept approaches infinity. It can be shown that in many online (edge-weighted) problems with arbitrary arrival order, no algorithm can achieve any bounded competitive ratio without the free disposal assumption. Hence, this setting has been adopted by many other works [13, 5, 17, 14, 22, 20, 11, 23, 7].

Online Generalized Assignment Problem (OGAP). Feldman et al. [19] also considered a more general online biparte matching problem, where each edge e has both a value v_e and a weight w_e, and each offline node has a capacity constraint on the sum of weights of matched edges (assume without loss of generality that all capacities are 1). It can be easily shown that the problem is a special case of the Online Vector Packing Problem with d equal to the total number of nodes, and sparsity $k = 2$: every item represents an edge e, and has value v_e, weight 1 on the dimension corresponding to the online endpoint, and weight w_e on the dimension corresponding to the offline endpoint.

For the problem when each edge has arbitrary weight and each offline node has capacity 1, it is well-known that the greedy algorithm that assigns each online node to the offline node with maximum marginal increase in the objective is 2-competitive, while no algorithm is known to have a competitive ratio strictly smaller than 2. However, several special cases of the problem were analyzed and better competitive ratios have been achieved [2, 15, 11, 1].

Apart from vector packing constraints, the online submodular maximization problem with free disposal has been studied under matroid constraints [7, 9]. In particular, the uniform and the partition matroids can be thought of special cases of vector packing constraints, where each item's weight vector has sparsity one and the same value for non-zero coordinate. However, using special properties of partition matroids, the exact optimal competitive ratio can be derived in [9], from which we also borrow relevant techniques to design our online algorithm.

Other Online Models. Kesselheim et al. [26] considered a variant of the problem when items (which they called requests) arrive in random order and have small weights compared to the total capacity; this is also known as the *secretary setting*, and free disposal is not allowed. They considered a more general setting in which an item can be accepted with more than one option, i.e., each item has different utilities and different weight vectors for different options. For every $\delta \in (0, \frac{1}{2})$, for the case when every weight vector is in $[0, \delta]^d$, they proposed an $O(k^{\frac{\delta}{1-\delta}})$-competitive algorithm, and a $(1+\epsilon)$-competitive algorithm when $\delta = O(\frac{\epsilon^2}{\log k})$, for $\epsilon \in (0, 1)$. In the random arrival order framework, many works assumed that the weights of items are much smaller than the total capacity [18, 16, 27, 26]. In comparison, our algorithm just needs the weaker ϵ slack assumption that no weight is more than $1 - \epsilon$ fraction of the total capacity.

The Online Vector Bin Packing problem [4, 3] is similar to the problem we consider in this paper. In the problem, items (with weight $\mathbf{w}_u \in [0, 1]^d$) arrive online in an arbitrary order and the objective is to pack all items into a minimum number of knapsacks, each with capacity $\mathbf{1}$. The current best competitive ratio for the problem is $O(d)$ [21] while the best hardness result is $\Omega(d^{1-\epsilon})$ [4], for any constant $\epsilon > 0$.

Future Work. We believe that it is an interesting open problem to see whether an $O(k)$-competitive ratio can be achieved for general instances, i.e., $\mathbf{w}_u \in [0, 1]^d$. However, at least we know that it is impossible to do so using deterministic algorithms (see Lemma 5).

Actually, it is interesting to observe that similar slack assumptions on the weight vectors of items have been made by several other literatures [12, 4, 26]. For example, for the Online Packing LPs problem (with random arrival order) [26], the competitive ratio $O(k^{\frac{\delta}{1-\delta}})$ holds only when $\mathbf{w}_u \in [0, \delta]^d$ for all $u \in \Omega$, for some $\delta \leq \frac{1}{2}$. For the Online Vector Bin Packing problem [4], while a hardness result $\Omega(d^{1-\epsilon})$ on the competitive ratio is proof for general instances with $\mathbf{w}_u \in [0, 1]^d$; when $\mathbf{w}_u \in [0, \frac{1}{B}]^d$ for some $B \geq 2$, they proposed an $O(d^{\frac{1}{B-1}} (\log d)^{\frac{B}{B-1}})$-competitive algorithm.

Another interesting open problem is whether the $O(k)$-competitive ratio can be improved for the problem under the "small weight assumption". Note that we have shown in Theorem 4 that achieving a constant competitive ratio is impossible.

2 Preliminaries

We use Ω to denote the set of items, which are not known by the algorithm initially and arrive one by one. Assume that each of the d dimensions of the knapsack has capacity 1. For $u \in \Omega$, the weight vector $\mathbf{w}_u \in [0, 1]^d$ is known to the algorithm only when item u arrives. A set $S \subset \Omega$ of items is *feasible* if $\sum_{u \in S} \mathbf{w}_u \leq \mathbf{1}$. The utility of S is $f(S)$, where f is a non-negative monotone submodular function. For a positive integer t, we use $[t]$ to denote $\{1, 2, \ldots, t\}$. We say that an item u is *discarded* if it is not accepted when it arrives; it is *disposed of* if it is accepted when it arrives, but later dropped to maintain feasibility.

Note that in general (without constant slack), no deterministic algorithm for the problem is competitive, even with linear utility function and when $d = k$. A similar result when $k = 1$ has been shown by Iwama and Zhang [25].

▶ **Lemma 5** (Generalization of [25]). *Any deterministic algorithm has a competitive ratio $\Omega(\sqrt{\frac{k}{\epsilon}})$ for the Online Vector Packing Problem with weight vectors in $[0, 1 - \epsilon]^d$, even when the utility function is linear and $d = k$.*

Proof. Since the algorithm is deterministic, we can assume that the instance is adaptive.

Consider the following instance with $k = d$. Let the first item have value 1 and weight $1 - \epsilon$ on all d dimensions; the following (small) items have value $\sqrt{\frac{\epsilon}{k}}$ and weight 2ϵ on one of the d dimension (and 0 otherwise). Stop the sequence immediately if the first item is not accepted. Otherwise let there be $\frac{1}{2\epsilon}$ items on each of the d dimensions. Note that to accept any of the "small" items, the first item must be disposed of. We stop the sequence immediately once the first item is disposed of.

It can be easily observe that we have either $\mathsf{ALG} = 1$ and $\mathsf{OPT} = \sqrt{\frac{k}{4\epsilon}}$, or $\mathsf{ALG} = \sqrt{\frac{\epsilon}{k}}$ and $\mathsf{OPT} \geq 1$, in both cases the competitive ratio is $\Omega(\sqrt{\frac{k}{\epsilon}})$. ◀

Note that the above hardness result (when $k = 1$) also holds for the Online Generalized Assignment Problem (with one offline node). We use OPT to denote both the optimal utility, and the feasible set that achieves this value. The meaning will be clear from the context.

3 Online Algorithm for Weight Vectors with Slack

In this section, we give an online algorithm for weight vectors with constant slack $\epsilon > 0$. Specifically, the algorithm is given some constant parameter $\epsilon > 0$ initially such that for all items $u \in \Omega$, its weight vector satisfies $\|\mathbf{w}_u\|_\infty \leq 1 - \epsilon$. On the other hand, the algorithm does not need to know upfront the upper bound k on the sparsity of the weight vectors.

3.1 Deterministic Online Algorithm

Notation. During the execution of an algorithm, for each item $u \in \Omega$, we use S^u and A^u to denote the feasible set of maintained items and the set of items that have ever been accepted, respectively, at the moment **just before** the arrival of item u.

We define the *value* of u as $\mathsf{v}(u) := f(u|A^u) = f(A^u \cup \{u\}) - f(A^u)$. Note that the value of an item depends on the algorithm and the arrival order of items. For $u \in \Omega$, for each $i \in [d]$, define the *density* of u at dimension i as $\rho_u(i) := \frac{\mathsf{v}(u)}{\mathbf{w}_u(i)}$ if $\mathbf{w}_u(i) \neq 0$ and $\rho_u(i) := \infty$ otherwise. By considering a lexicographical order on Ω, we may assume that all ties in values and densities can be resolved consistently.

For a vector $\mathbf{x} \in [0,1]^\Omega$, we use $\mathbf{x}(u)$ to denote the component corresponding to coordinate $u \in \Omega$. We overload the notation $\lceil \mathbf{x} \rceil$ to mean either the support $\lceil \mathbf{x} \rceil := \{u \in \Omega : \mathbf{x}(u) > 0\}$ or its indicator vector in $\{0,1\}^\Omega$ such that $\lceil \mathbf{x} \rceil(u) = \lceil \mathbf{x}(u) \rceil$.

Online Algorithm. The details are given in Algorithm 1, which defines the parameters $\beta := 1 - \epsilon$, $\alpha := \sqrt{\beta} = 1 - \Theta(\epsilon)$ and $\gamma := \frac{1}{2}(1 - \frac{\beta}{\alpha}) = \Theta(\epsilon)$. The algorithm keeps a (fractional) vector $\mathbf{s} \in [0,1]^\Omega$, which is related to the actual feasible set S maintained by the algorithm via the loop invariant (of the **for** loop in lines 2-24): $S = \lceil \mathbf{s} \rceil$. Specifically, when an item u arrives, the vector $\mathbf{s}$ might be modified such that the coordinate $\mathbf{s}(u)$ might be increased and/or other coordinates might be decreased; after one iteration of the loop, the feasible set S is changed according to the loop invariant. The algorithm also maintains an auxiliary vector $\mathbf{a} \in [0,1]^\Omega$ that keeps track of the maximum fraction of item u that has ever been accepted.

Algorithm Intuition. The algorithm solves a fractional variant behind the scenes using a linear objective function defined by v. For each dimension $i \in [d]$, it assumes that the capacity is $\beta < 1$. Upon the arrival of a new element $u \in \Omega$, the algorithm tries to increase the fraction of item u accepted via the parameter $\theta \in [0,1]$ in the **do**...**while** loop starting

at line 16. For each dimension $i \in [d]$ whose capacity is saturated (at β) and $\mathbf{w}_u(i) > 0$, to further increase the fraction of item u accepted, some item u_i^θ with the least density ρ_i will have its fraction decreased in order to make room for item u. Hence, with respect to θ, the value decreases at a rate at most $\sum_i \mathbf{w}_u(i) \cdot \rho_i(u_i^\theta)$ due to disposing of fractional items. We keep on increasing θ as long as this rate of loss is less than γ times $\mathsf{v}(u)$ (which is the rate of increase in value due to item u).

After trying to increase the fraction of item u (and disposing of other items fractionally), the algorithm commits to this change only if at least α fraction of item u is accepted, in which case any item whose accepted fraction is less than β will be totally disposed of.

Parameters: $\alpha := \sqrt{1-\epsilon}, \beta := 1-\epsilon, \gamma := \frac{1}{2}(1-\sqrt{1-\epsilon})$

1 initialize $\mathbf{s}, \mathbf{a} \in [0,1]^\Omega$ as all zero vectors; ▷ $\lceil \mathbf{s} \rceil$ is the current feasible solution

2 **for** *each round when* u *arrives* **do**

3 Define $\mathsf{v}(u) := f(u|\lceil \mathbf{a} \rceil)$;

4 Initialize $\theta \leftarrow 0$, $\mathbf{x}^0 \leftarrow \mathbf{s}$;

5 **do**

6 Increase θ continuously (variables $\mathbf{x}^\theta$ and u_i^θ all depend on θ):

7 **for** *every* $i \in [d]$ **do**

8 **if** $\sum_{v \in \Omega} \mathbf{x}^\theta(v)\mathbf{w}_v(i) = \beta$ *and* $\mathbf{w}_u(i) > 0$ **then**

9 Set $u_i^\theta \leftarrow \arg\min\{\rho_i(v) : v \in \Omega \setminus \{u\}, \mathbf{x}^\theta(v)\mathbf{w}_v(i) > 0\}$;

10 **end**

11 **if** $\sum_{v \in \Omega} \mathbf{x}^\theta(v)\mathbf{w}_v(i) < \beta$ *or* $\mathbf{w}_u(i) = 0$ **then**

12 Set $u_i^\theta \leftarrow \perp$ and $\rho_i(u_i^\theta) \leftarrow 0$;

13 **end**

14 **end**

15 Change $\mathbf{x}^\theta(v)$ (for all $v \in \Omega$) at rate:

$$\frac{d\mathbf{x}^\theta(v)}{d\theta} = \begin{cases} 1, & v = u; \\ -\max_{i \in [d]: u_i^\theta = v} \left\{ \frac{\mathbf{w}_u(i)}{\mathbf{w}_{u_i^\theta}(i)} \right\}, & v \in \{u_i^\theta\}_{i \in [d]}; \\ 0, & \text{otherwise.} \end{cases}$$

16 **while** $\theta < 1$ *and* $\gamma \cdot \mathsf{v}(u) > \sum_{i \in [d]} \mathbf{w}_u(i) \cdot \rho_i(u_i^\theta)$;

17 **if** $\theta \geq \alpha$ **then**

18 $\mathbf{s} \leftarrow \mathbf{x}^\theta$, $\mathbf{a}(u) \leftarrow \mathbf{x}^\theta(u)$; ▷ update phase

19 **for** $v \in \Omega$ *with* $\mathbf{s}(v) < \beta$ **do**

20 $\mathbf{s}(v) \leftarrow 0$; ▷ dispose of small fractions

21 **end**

22 **end**

23 ▷ if $\theta < \alpha$, then $\mathbf{s}$ and $\mathbf{a}$ will not be changed

24 **end**

25 **return** $\lceil \mathbf{s} \rceil$.

Algorithm 1: Online Algorithm

3.2 Competitive Analysis

For notational convenience, we use the superscripted versions (e.g., $\mathbf{s}^u$, $\mathbf{a}^u$, $S^u = \lceil \mathbf{s}^u \rceil$, $A^u = \lceil \mathbf{a}^u \rceil$) to indicate the state of the variables at the beginning of the iteration in the **for** loop (starting at line 2) when item u arrives. When we say the **for** loop, we mean the one

that runs from lines 2 to 24. When the superscripts of the variables are removed (e.g., S and A), we mean the variables at some moment just before or after an iteration of the **for** loop.

We first show that the following properties are loop invariants of the **for** loop.

▶ **Lemma 6** (Feasibility Loop Invariant)**.** *The following properties are loop invariants of the **for** loop:*

(a) For every $i \in [d]$, $\sum_{v\in\Omega} \mathbf{s}(v) \cdot \mathbf{w}_v(i) \leq \beta$, i.e., for every dimension, the total capacity consumed by the fractional solution $\mathbf{s}$ is at most β.

(b) The set $S = \lceil \mathbf{s} \rceil \subset \Omega$ is feasible for the original problem.

Proof. Statement (a) holds initially because $\mathbf{s}$ is initialized to $\vec{0}$. Next, assume that for some item $u \in \Omega$, statement (a) holds for $\mathbf{s}^u$. It suffices to analyze the non-trivial case when the changes to $\mathbf{s}$ are committed at the end of the iteration. Hence, we show that statement (a) holds throughout the execution of the **do**...**while** loop starting at line 16. It is enough show that for each $i \in [d]$, $g_i(\theta) := \sum_{v\in\Omega} \mathbf{x}^\theta(v) \cdot \mathbf{w}_v(i) \leq \beta$ holds while θ is being increased.

To this end, it suffices to prove that if $g_i(\theta) = \beta$, then $\frac{dg_i(\theta)}{d\theta} \leq 0$. We only need to consider the case $\mathbf{w}_u(i) > 0$, because otherwise $g_i(\theta)$ cannot increase. By the rules updating $\mathbf{x}$, we have in this case $\frac{dg_i(\theta)}{d\theta} \leq \frac{d\mathbf{x}^\theta(u)}{d\theta}\mathbf{w}_u(i) + \frac{d\mathbf{x}^\theta(u_i^\theta)}{d\theta}\mathbf{w}_{u_i^\theta}(i) \leq 0$, as required.

We next show that statement (b) follows from statement (a). Line 20 ensures that between iterations of the **for** loop, for all $v \in S = \lceil \mathbf{s} \rceil$, $\mathbf{s}(v) \geq \beta$.

Hence, for all $i \in [d]$, we have $\sum_{v\in S} \mathbf{w}_v(i) \leq \frac{1}{\beta}\sum_{v\in S} \mathbf{s}(v)\cdot\mathbf{w}_v(i) = \frac{1}{\beta}\sum_{v\in\Omega} \mathbf{s}(v)\cdot\mathbf{w}_v(i) \leq 1$, where the last inequality follows from statement (a). ◀

For a vector $\mathbf{x} \in [0,1]^\Omega$, we define $\mathsf{v}(\mathbf{x}) := \sum_{u\in\Omega} \mathsf{v}(u) \cdot \mathbf{x}(u)$; for a set $X \subset \Omega$, we define $\mathsf{v}(X) := \sum_{u\in X} \mathsf{v}(u)$. Note that the definitions of $\mathsf{v}(\lceil \mathbf{x} \rceil)$ are consistent under the set and the vector interpretations.

The following simple fact (which is similar to Lemma 2.1 of [9]) establishes the connection between the values of items (defined by our algorithm) and the utility of the solution (defined by the submodular function f).

▶ Fact 3.1 (Lemma 2.1 in [9]). The **for** loop maintains the invariants $f(A) = f(\emptyset) + \mathsf{v}(A)$ and $f(S) \geq f(\emptyset) + \mathsf{v}(S)$, where $A = \lceil \mathbf{a} \rceil$ and $S = \lceil \mathbf{s} \rceil$.

Our analysis consists of two parts. We first show that $\mathsf{v}(\mathbf{a})$ is comparable to the value of our real solution S in Lemma 7. Then, we compare in Lemma 8 the value of an (offline) optimal solution with $\mathsf{v}(\mathbf{a})$. Combining the two lemmas we are able to prove Theorem 9.

▶ **Lemma 7.** *The **for** loop maintains the invariant: $(1 - \frac{\beta}{\alpha}) \cdot \mathsf{v}(S) \geq (1 - \frac{\beta}{\alpha} - \gamma) \cdot \mathsf{v}(\mathbf{a})$, where $S = \lceil \mathbf{s} \rceil$. In particular, our choice of the parameters implies that $\mathsf{v}(\mathbf{a}) \leq 2 \cdot \mathsf{v}(S)$.*

Proof. We prove the stronger loop invariant that:

$$\mathsf{v}(\mathbf{s}) \geq (1 - \gamma - \frac{\beta}{\alpha}) \sum_{r\in A\setminus S} \mathsf{v}(r) \cdot \mathbf{a}(r) + (1 - \gamma) \sum_{r\in S} \mathsf{v}(r) \cdot \mathbf{a}(r),$$

where $S = \lceil \mathbf{s} \rceil$ is the current feasible set and $A \setminus S$ is the set of items that have been accepted at some moment but are already discarded.

The invariant holds trivially initially when $S = A = \emptyset$ and $\mathbf{s} = \vec{0}$. Suppose the invariant holds at the beginning of the iteration when item $u \in \Omega$ arrives. We analyze the non-trivial case when the item u is accepted into S, i.e., $\mathbf{s}$ and $\mathbf{a}$ are updated at the end of the iteration. Recall that $\mathbf{s}^u$ and $\mathbf{a}^u$ refer to the variables at the beginning of the iteration, and for the rest of the proof, we use the $\widehat{\mathbf{s}}$ and $\widehat{\mathbf{a}}$ to denote their states at the end of the iteration.

Suppose in the **do**...**while** loop, the parameter θ is increased from 0 to $\mathbf{a}(u) \geq \alpha$. Since for all $r \neq u$, $\mathbf{a}^u(r) = \widehat{\mathbf{a}}(r)$, we can denote this common value by $\mathbf{a}(r)$ without risk of ambiguity. We use $\mathbf{x}^u$ to denote the vector $\mathbf{x}^\theta$ when $\theta = \mathbf{a}(u)$. Then, we have

$$\begin{aligned}\mathsf{v}(\mathbf{x}^u) - \mathsf{v}(\mathbf{s}^u) &\geq \mathsf{v}(u) \cdot \mathbf{a}(u) - \int_0^{\mathbf{a}(u)} \sum_{i \in [d]: u_i^\theta \neq \perp} \left(\frac{\mathbf{w}_u(i)}{\mathbf{w}_{u_i^\theta}(i)} \cdot \mathsf{v}(u_i^\theta)\right) d\theta \\ &> \mathsf{v}(u) \cdot \mathbf{a}(u) - \int_0^{\mathbf{a}(u)} \gamma \cdot \mathsf{v}(u) d\theta = (1-\gamma) \cdot \mathsf{v}(u) \cdot \mathbf{a}(u),\end{aligned}$$

where the second inequality holds by the criteria of the **do**...**while** loop.

Next, we consider the change in value $\mathsf{v}(\widehat{\mathbf{s}}) - \mathsf{v}(\mathbf{x}^u)$, because some (fractional) items are disposed of in line 20. Let $D \subseteq S^u$ be such discarded items. Since an item is discarded only if its fraction is less than β, the value lost is at most $\beta \sum_{r \in D} \mathsf{v}(r) \leq \frac{\beta}{\alpha} \sum_{r \in D} \mathsf{v}(r) \cdot \mathbf{a}(r)$, where the last inequality follows because $\mathbf{a}(r) \geq \alpha$ for all items r that are ever accepted. Therefore, we have

$$\mathsf{v}(\widehat{\mathbf{s}}) - \mathsf{v}(\mathbf{x}^u) \geq -\frac{\beta}{\alpha} \sum_{r \in D} \mathsf{v}(r) \cdot \mathbf{a}(r).$$

Combining the above two inequalities, we have

$$\mathsf{v}(\widehat{\mathbf{s}}) - \mathsf{v}(\mathbf{s}^u) \geq (1-\gamma) \cdot \mathsf{v}(u) \cdot \mathbf{a}(u) - \frac{\beta}{\alpha} \sum_{r \in D} \mathsf{v}(r) \cdot \mathbf{a}(r).$$

Hence, using the induction hypothesis that the loop invariant holds at the beginning of the iteration, it follows that

$$\begin{aligned}\mathsf{v}(\widehat{\mathbf{s}}) \geq& (1-\gamma-\frac{\beta}{\alpha}) \sum_{r \in A^u \setminus S^u} \mathsf{v}(r) \cdot \mathbf{a}(r) + (1-\gamma) \sum_{r \in S^u} \mathsf{v}(r) \cdot \mathbf{a}(r) + (1-\gamma) \cdot \mathsf{v}(u) \cdot \mathbf{a}(u) \\ &- \frac{\beta}{\alpha} \sum_{r \in D} \mathsf{v}(r) \cdot \mathbf{a}(r) \\ \geq& (1-\gamma-\frac{\beta}{\alpha}) \sum_{r \in \widehat{A} \setminus \widehat{S}} \mathsf{v}(r) \cdot \mathbf{a}(r) + (1-\gamma) \sum_{r \in \widehat{S}} \mathsf{v}(r) \cdot \mathbf{a}(r),\end{aligned}$$

where $\widehat{A} = \lceil \widehat{\mathbf{a}} \rceil$ and $\widehat{S} = \lceil \widehat{\mathbf{s}} \rceil$, as required.

We next show that the stronger invariant implies the result of the lemma. Rewriting the invariant gives
$\mathsf{v}(\mathbf{s}) \geq (1-\gamma-\frac{\beta}{\alpha}) \sum_{r \in A} \mathsf{v}(r) \cdot \mathbf{a}(r) + \frac{\beta}{\alpha} \sum_{r \in S} \mathsf{v}(r) \cdot \mathbf{a}(r) \geq (1-\gamma-\frac{\beta}{\alpha}) \sum_{r \in A} \mathsf{v}(r) \cdot \mathbf{a}(r) + \frac{\beta}{\alpha} \cdot \mathsf{v}(\mathbf{s})$, where the last inequality follows because $\mathbf{a}(r) \geq \mathbf{s}(r)$ for all $r \in S$. Finally, the lemma follows because $\mathsf{v}(S) = \mathsf{v}(\lceil \mathbf{s} \rceil) \geq \mathsf{v}(\mathbf{s})$. ◀

The following lemma gives an upper bound on the value of the items in a feasible set that are discarded right away by the algorithm.

▶ **Lemma 8.** *The* ***for*** *loop maintains the invariant that if* OPT *is a feasible subset of items that have arrived so far, then* $\gamma \cdot \mathsf{v}(\mathsf{OPT} \setminus A) \leq \frac{k}{\beta(1-\alpha)} \cdot \mathsf{v}(\mathbf{a})$*, where* $A = \lceil \mathbf{a} \rceil$*. In particular, our choice of the parameters implies that* $\mathsf{v}(\mathsf{OPT} \setminus A) \leq O(\frac{k}{\epsilon^2}) \cdot \mathsf{v}(S)$*.*

Proof. Consider some $u \in \mathsf{OPT} \setminus A$. Since $u \notin A$, in iteration u of the **for** loop, we know that at the end of the **do**...**while** loop, we must have $\theta < \alpha$, which implies $\gamma \cdot \mathsf{v}(u) \leq \sum_{i \in [d]} \mathbf{w}_u(i) \cdot \rho_i(u_i^\theta)$ at this moment.

Recall that by definition, $\rho_i(u_i^\theta)$ is either (i) 0 in the case $\sum_{v\in\Omega} \mathbf{x}^\theta(v)\cdot\mathbf{w}_v(i) < \beta$ and $\mathbf{w}_u(i) > 0$, or (ii) the minimum density $\rho_i(v)$ in dimension i among items $v \neq u$ such that $\mathbf{x}^\theta(v)\cdot\mathbf{w}_v(i) > 0$.

Hence, in the second case, we have

$$\rho_i(u_i^\theta) \le \frac{\sum_{v\neq u:\mathbf{x}^\theta(v)\mathbf{w}_v(i)>0} \mathbf{x}^\theta(v)\cdot\mathsf{v}(v)}{\sum_{v\neq u:\mathbf{x}^\theta(v)\mathbf{w}_v(i)>0} \mathbf{x}^\theta(v)\cdot\mathbf{w}_v(i)} = \frac{\sum_{v\neq u:\mathbf{x}^\theta(v)\mathbf{w}_v(i)>0} \mathbf{x}^\theta(v)\cdot\mathsf{v}(v)}{\beta-\theta\cdot\mathbf{w}_u(i)}$$
$$\le \frac{\sum_{v:\mathbf{w}_v(i)>0}\mathbf{a}(v)\cdot\mathsf{v}(v)}{\beta(1-\alpha)} = \frac{V_i}{\beta(1-\alpha)},$$

where $V_i := \sum_{v:\mathbf{w}_v(i)>0}\mathbf{a}(v)\cdot\mathsf{v}(v)$ depends only on the current $\mathbf{a}$ and $i\in[d]$. In the last inequality, we use $\theta\cdot\mathbf{w}_u(i)\le\alpha\beta$ and a very loose upper bound on the numerator. Observe that for the case (i) $\rho_i(u_i^\theta)=0$, the inequality $\rho_i(u_i^\theta)\le\frac{V_i}{\beta(1-\alpha)}$ holds trivially.

Hence, using this uniform upper bound on $\rho_i(u_i^\theta)$, we have $\gamma\cdot\mathsf{v}(u)\le\sum_{i\in[d]}\mathbf{w}_u(i)\cdot\frac{V_i}{\beta(1-\alpha)}$.

Therefore, we have

$$\gamma\cdot\mathsf{v}(\mathsf{OPT}\setminus A) \le \sum_{u\in\mathsf{OPT}\setminus A}\sum_{i\in[d]}\mathbf{w}_u(i)\cdot\frac{V_i}{\beta(1-\alpha)} = \sum_{i\in[d]}\left(\sum_{u\in\mathsf{OPT}\setminus A}\mathbf{w}_u(i)\right)\cdot\frac{V_i}{\beta(1-\alpha)}$$
$$\le \sum_{i\in[d]}\frac{V_i}{\beta(1-\alpha)} \le \frac{k\cdot\mathsf{v}(\mathbf{a})}{\beta(1-\alpha)},$$

where the second to last inequality follows because $\mathsf{OPT}\setminus A$ is feasible, and $\sum_{i\in[d]}V_i\le k\cdot\mathsf{v}(\mathbf{a})$, because for each $v\in\Omega$, $|\{i\in[d]:\mathbf{w}_v(i)>0\}|\le k$. ◀

▶ **Theorem 9.** *Algorithm 1 is $O(\frac{k}{\epsilon^2})$-competitive.*

Proof. Suppose OPT is a feasible subset. Recall that S is the feasible subset currently maintained by the algorithm. Then, by the monotonicity and the submodularity of f, we have $f(\mathsf{OPT})\le f(\mathsf{OPT}\cup A)\le f(A)+\sum_{u\in\mathsf{OPT}\setminus A}f(u|A)\le f(\emptyset)+\mathsf{v}(A)+\mathsf{v}(\mathsf{OPT}\setminus A)$, where we use Fact 3.1 and submodularity $f(u|A)\le f(u|A^u)=\mathsf{v}(u)$ in the last inequality.

Next, observe that for all $u\in A$, $\mathbf{a}(u)\ge\alpha$. Hence, we have $\mathsf{v}(A)\le\frac{\mathsf{v}(\mathbf{a})}{\alpha}=O(1)\cdot\mathsf{v}(\mathbf{a})$. Combining with Lemma 8, we have $f(\mathsf{OPT})\le f(\emptyset)+O(\frac{k}{\epsilon^2})\cdot\mathsf{v}(\mathbf{a})$.

Finally, using Lemma 7 and Fact 3.1 gives $f(\mathsf{OPT})\le O(\frac{k}{\epsilon^2})\cdot f(S)$, as required. ◀

3.3 Hardness Results: Proof of Theorem 2

We show that for the Online Vector Packing Problem with slack $\epsilon\in(0,\frac{1}{2})$, no deterministic algorithm can achieve $o(k)$-competitive ratio, and no randomized algorithm can achieve $o(\frac{k}{\log k})$-competitive ratio. To prove the hardness result for randomized algorithms, we apply Yao's principle [29] and construct a distribution of hard instances, such that any deterministic algorithm cannot perform well in expectation. Specifically, we shall show that each instance in the support of the distribution has offline optimal value $\Theta(\frac{k}{\log k})$, but any deterministic algorithm has expected objective value $O(1)$, thereby proving Theorem 2.

In our hard instances, the utility function is linear, and all items have the same value, i.e., the utility function is $f(S):=|S|$. Moreover, we assume all weight vectors are in $\{0,1-\epsilon\}^d$, for any arbitrary $\epsilon\in(0,\frac{1}{2})$. Hence, we only need to describe the arrival order of items, and the non-zero dimensions of weight vectors. In particular, we can associate each item u with a k-subset of $[d]$. We use $\binom{[d]}{k}$ to denote the collection of k-subsets of $[d]$.

Notations. We say that two items are *conflicting*, if they both have non-zero weights on some dimension i (in which case, we say that they *conflict* with each other on dimension i). We call two items *non-conflicting* if they do not conflict with each other on any dimension.

Our hard instances show that in some case when items conflict with one another on different dimensions, the algorithm might be forced to make difficult decisions on choosing which item to accept. By utilizing the nature of unknown future, we show that it is very unlikely for any algorithm to make the right decisions on the hard instances. Although accepted items can be later disposed of to make room for (better) items, by carefully setting the weights and arrival order, we show that disposing of accepted items cannot help to get a better objective (hence in a sense, disabling free-disposal).

Hard instance for deterministic algorithms. Let $d := 2k^2$. Recall that each item is specified by an element of $\binom{[d]}{k}$, indicating which k dimensions are non-zero. Consider any deterministic algorithm. An arriving sequence of length at most $2k$ is chosen adaptively. The first item is picked arbitrarily, and the algorithm must select this item, or else the sequence stops immediately. Subsequently, in each round, the non-zero dimensions for the next arriving item u are picked according to the following rules.

1. Exactly $k-1$ dimensions from $[d]$ are chosen such that no previous item has picked them.
2. Suppose $\widehat{u} \in \binom{[d]}{k}$ is the item currently kept by the algorithm. Then, the remaining dimension i is picked from $\widehat{u}$ such that no other arrived item conflicts with $\widehat{u}$ on dimension i. If no such dimension i can be picked, then the sequence stops.

▶ **Lemma 10.** *Any deterministic algorithm can keep at most 1 item, while there exist at least k items that are mutually non-conflicting, implying that an offline optimal solution contains at least k items.*

Proof. By adversarial choice, every arriving item conflicts with the item currently kept by the algorithm. Hence, the algorithm can keep at most 1 item at any time.

We next show that when the sequence stops, there exist at least k items in the sequence that are mutually non-conflicting. For the case when there are $2k$ items in the sequence, consider the items in reversed order of arrival. Observe that each item conflicts with only one item that arrives before it. Hence, we can scan the items one by one backwards, and while processing a remaining item, we remove any earlier item that conflicts with it. After we finish with the scan, there are at least k items remaining that are mutually non-conflicting.

Suppose the sequence stops with less than $2k$ items. It must be the case that while we are trying to add a new item u, we cannot find a dimension i contained in the item $\widehat{u}$ currently kept by the algorithm such that no already arrived item conflicts with $\widehat{u}$ on dimension i. This implies that for every non-zero dimension i of $\widehat{u}$, there is already an item u_i conflicting with $\widehat{u}$ on that dimension. Since by choice, each dimension can cause a conflict between at most 2 items, these k items u_i's must be mutually non-conflicting. ◀

Distribution of Hard Instances. To use Yao's principle [29], we give a procedure to sample a random sequence of items. For some large enough integer ℓ that is a power of 2, define $k := 100\ell \log_2 \ell + 1$, which is the sparsity of the weight vectors. Observe that $\ell = \Theta(\frac{k}{\log k})$, and define $d := \ell + 400\ell^2 \log_2 \ell = O(\frac{k^2}{\log k})$ to be the number of dimensions. We express the set of dimensions $[d] = I \cup J$ as the disjoint union of $I := [\ell]$ and $J := [d] \setminus I$. The items arrive in ℓ phases, and for each $i \in [\ell]$, $4\ell - i + 1$ items arrive. Recall that each item is characterized by its k non-zero dimensions (where the non-zero coordinates all equal

$1-\epsilon > \frac{1}{2}$). We initialize $J_1 := J$. For i from 1 to ℓ, we describe how the items in phase i are sampled as follows.

1. Each of the $4\ell - i + 1$ items will have $i \in I = [\ell]$ as the only non-zero dimension in I.
2. Observe that (inductively) we have $|J_i| = (4\ell - i + 1) \cdot 100\ell \log_2 \ell$. We partition J_i randomly into $4\ell - i + 1$ disjoint subsets, each of size exactly $k - 1 = 100\ell \log_2 \ell$. Each such subset corresponds to the remaining $(k-1)$ non-zero dimensions of an item in phase i. These items in phase i can be revealed to the algorithm one by one.
3. Pick S_i from those $4\ell - i + 1$ subsets uniformly at random; define $J_{i+1} := J_i \setminus S_i$. Observe that the algorithm does not know S_i until the next phase $i+1$ begins.

▶ Claim 3.1. In the above procedure, the items corresponding to S_i's for $i \in [\ell]$ are mutually non-conflicting. This implies that there is an offline optimal solution containing $\ell = \Theta(\frac{k}{\log k})$ items. We say that those ℓ items are *good*, while other items are *bad*.

We next show that bad items are very likely to be conflicting.

▶ **Lemma 11.** *Let $\mathcal{E}$ be the event that there exist two bad items that are non-conflicting. Then,* $\Pr[\mathcal{E}] \leq \frac{1}{\ell^2}$.

Proof. An alternative view of the sampling process is that the subsets $S_1, S_2, \ldots, S_\ell$ are first sampled for the good items. Then, the remaining bad items can be sampled independently across different phases (but note that items within the same phase are sampled in a dependent way).

Suppose we condition on the subsets $S_1, S_2, \ldots, S_\ell$ already sampled. Consider phases i and j, where $i < j$. Next, we further condition on all the random subsets generated in phase j for defining the corresponding items. We fix some bad item v in phase j.

We next use the remaining randomness (for picking the items) in phase i. Recall that each bad item in phase i corresponds to a random subset of size $k-1 = 100\ell \log_2 \ell$ in $J_i \setminus S_i$, where $|J_i \setminus S_i| \leq 4(k-1)\ell$. If we focus on such a particular (random) subset from phase i, the probability that it is disjoint from the subset corresponding to item v (that we fixed from phase j) is at most $(1 - \frac{k-1}{4(k-1)\ell})^{k-1} \leq \exp(-25 \log_2 \ell) \leq \frac{1}{\ell^7}$.

Observe that there are in total at most $4\ell^2$ items. Hence, taking a union over all possible pairs of bad items, the probability of the event $\mathcal{E}$ is at most $(4\ell^2)^2 \cdot \frac{1}{\ell^7} \leq \frac{1}{\ell^2}$. ◀

▶ **Lemma 12.** *For any deterministic algorithm* ALG *applied to the above random procedure, the expected number of items kept in the end is $O(1)$.*

Proof. Let X denote the number of good items and Y denote the number of bad items kept by the algorithm at the end.

Observe that the sampling procedure allows the good item (corresponding to S_i) in phase i to be decided after the deterministic algorithm finishes making all its decisions in phase i. Hence, the probability that the algorithm keeps the good item corresponding to S_i is at most $\frac{1}{4\ell - i + 1} \leq \frac{1}{3\ell}$. Since this holds for every phase, it follows that $E[X] \leq \frac{1}{3\ell} \cdot \ell = \frac{1}{3}$.

Observe that conditioning on the complementing event $\overline{\mathcal{E}}$ (refer to Lemma 11), at most 1 bad item can be kept by the algorithm, because any two bad items are conflicting. Finally, because the total number of items is at most $4\ell^2$, we have $E[Y] = \Pr[\mathcal{E}]E[Y|\mathcal{E}] + \Pr[\overline{\mathcal{E}}]E[Y|\overline{\mathcal{E}}] \leq \frac{1}{\ell^2} \cdot 4\ell^2 + 1 \cdot 1 \leq 5$.

Hence, $E[X + Y] \leq 6$, as required. ◀

▶ **Corollary 13.** *By Claim 3.1 and Lemma 12, Yao's principle implies that for any randomized algorithm, there exists a sequence of items such that the value of an offline optimum is at least $\Theta(\frac{k}{\log k})$, but the expected value achieved by the algorithm is $O(1)$.*

References

1. Melika Abolhassani, T.-H. Hubert Chan, Fei Chen, Hossein Esfandiari, MohammadTaghi Hajiaghayi, Hamid Mahini, and Xiaowei Wu. Beating ratio 0.5 for weighted oblivious matching problems. In *ESA*, volume 57 of *LIPIcs*, pages 3:1–3:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
2. Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *SODA*, pages 1253–1264, 2011.
3. Yossi Azar, Ilan Reuven Cohen, Amos Fiat, and Alan Roytman. Packing small vectors. In *SODA*, pages 1511–1525. SIAM, 2016.
4. Yossi Azar, Ilan Reuven Cohen, Seny Kamara, and F. Bruce Shepherd. Tight bounds for online vector bin packing. In *STOC*, pages 961–970. ACM, 2013.
5. Moshe Babaioff, Jason D. Hartline, and Robert D. Kleinberg. Selling ad campaigns: online algorithms with cancellations. In *EC*, pages 61–70. ACM, 2009.
6. Nikhil Bansal, Nitish Korula, Viswanath Nagarajan, and Aravind Srinivasan. On k-column sparse packing programs. In *IPCO*, volume 6080 of *Lecture Notes in Computer Science*, pages 369–382. Springer, 2010.
7. Niv Buchbinder, Moran Feldman, and Roy Schwartz. Online submodular maximization with preemption. In *SODA*, pages 1202–1216. SIAM, 2015.
8. Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34(2):270–286, 2009.
9. T.-H. Hubert Chan, Zhiyi Huang, Shaofeng H.-C. Jiang, Ning Kang, and Zhihao Gavin Tang. Online submodular maximization with free disposal: Randomization beats ¼ for partition matroids. In *SODA*, pages 1204–1223. SIAM, 2017.
10. T.-H. Hubert Chan, Shaofeng H.-C. Jiang, Zhihao Gavin Tang, and Xiaowei Wu. Online submodular maximization problem with vector packing constraint. *CoRR*, abs/1706.06922, 2017.
11. Moses Charikar, Monika Henzinger, and Huy L. Nguyen. Online bipartite matching with decomposable weights. In *ESA*, volume 8737 of *Lecture Notes in Computer Science*, pages 260–271. Springer, 2014.
12. Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM J. Comput.*, 43(6):1831–1879, 2014.
13. Florin Constantin, Jon Feldman, S. Muthukrishnan, and Martin Pál. An online mechanism for ad slot reservations with cancellations. In *SODA*, pages 1265–1274. SIAM, 2009.
14. Nikhil R. Devanur, Zhiyi Huang, Nitish Korula, Vahab S. Mirrokni, and Qiqi Yan. Whole-page optimization and submodular welfare maximization with online bidders. In *EC*, pages 305–322. ACM, 2013.
15. Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of ranking for online bipartite matching. In *SODA*, pages 101–107, 2013.
16. Nikhil R. Devanur, Kamal Jain, Balasubramanian Sivan, and Christopher A. Wilkens. Near optimal online algorithms and fast approximation algorithms for resource allocation problems. In *EC*, pages 29–38. ACM, 2011.
17. Leah Epstein, Asaf Levin, Danny Segev, and Oren Weimann. Improved bounds for online preemptive matching. In *STACS*, volume 20 of *LIPIcs*, pages 389–399. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.
18. Jon Feldman, Monika Henzinger, Nitish Korula, Vahab S. Mirrokni, and Clifford Stein. Online stochastic packing applied to display ad allocation. In *ESA (1)*, volume 6346 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 2010.

19 Jon Feldman, Nitish Korula, Vahab S. Mirrokni, S. Muthukrishnan, and Martin Pál. Online ad assignment with free disposal. In *WINE*, volume 5929 of *Lecture Notes in Computer Science*, pages 374–385. Springer, 2009.

20 Stanley P. Y. Fung. Online scheduling with preemption or non-completion penalties. *J. Scheduling*, 17(2):173–183, 2014.

21 M. R. Garey, Ronald L. Graham, David S. Johnson, and Andrew Chi-Chih Yao. Resource constrained scheduling as generalized bin packing. *J. Comb. Theory, Ser. A*, 21(3):257–298, 1976.

22 Xin Han, Yasushi Kawase, and Kazuhisa Makino. Online unweighted knapsack problem with removal cost. *Algorithmica*, 70(1):76–91, 2014.

23 Xin Han, Yasushi Kawase, and Kazuhisa Makino. Randomized algorithms for online knapsack problems. *Theor. Comput. Sci.*, 562:395–405, 2015.

24 Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-set packing. *Computational Complexity*, 15(1):20–39, 2006.

25 Kazuo Iwama and Guochuan Zhang. Optimal resource augmentations for online knapsack. In *APPROX-RANDOM*, volume 4627 of *Lecture Notes in Computer Science*, pages 180–188. Springer, 2007.

26 Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. Primal beats dual on online packing lps in the random-order model. In *STOC*, pages 303–312. ACM, 2014.

27 Marco Molinaro and R. Ravi. Geometry of online packing linear programs. In *ICALP (1)*, volume 7391 of *Lecture Notes in Computer Science*, pages 701–713. Springer, 2012.

28 Mourad El Ouali, Antje Fretwurst, and Anand Srivastav. Inapproximability of b-matching in k-uniform hypergraphs. In *WALCOM*, volume 6552 of *Lecture Notes in Computer Science*, pages 57–69. Springer, 2011.

29 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *FOCS*, pages 222–227. IEEE Computer Society, 1977.