Average Sensitivity of the Knapsack Problem

Soh Kumabe
The University of Tokyo, Japan

Yuichi Yoshida
National Institute of Informatics, Tokyo, Japan

Abstract

In resource allocation, we often require that the output allocation of an algorithm is stable against input perturbation because frequent reallocation is costly and untrustworthy. Varma and Yoshida (SODA'21) formalized this requirement for algorithms as the notion of average sensitivity. Here, the average sensitivity of an algorithm on an input instance is, roughly speaking, the average size of the symmetric difference of the output for the instance and that for the instance with one item deleted, where the average is taken over the deleted item.

In this work, we consider the average sensitivity of the knapsack problem, a representative example of a resource allocation problem. We first show a $(1-\epsilon)$-approximation algorithm for the knapsack problem with average sensitivity $O(\epsilon^{-1}\log\epsilon^{-1})$. Then, we complement this result by showing that any $(1-\epsilon)$-approximation algorithm has average sensitivity $\Omega(\epsilon^{-1})$. As an application of our algorithm, we consider the incremental knapsack problem in the random-order setting, where the goal is to maintain a good solution while items arrive one by one in a random order. Specifically, we show that for any $\epsilon > 0$, there exists a $(1-\epsilon)$-approximation algorithm with amortized recourse $O(\epsilon^{-1}\log\epsilon^{-1})$ and amortized update time $O(\log n + f_\epsilon)$, where n is the total number of items and $f_\epsilon > 0$ is a value depending on ϵ.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Average Sensitivity, Knapsack Problem, FPRAS

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.75

Funding *Soh Kumabe*: Supported by JST, PRESTO Grant Number JPMJPR192B.
Yuichi Yoshida: Supported by JST, PRESTO Grant Number JPMJPR192B.

1 Introduction

1.1 Background and Motivation

The *knapsack problem* is one of the most fundamental models in *resource allocation*, which handles the selection of good candidates under a budget constraint. For example, it can model the hiring process of employees in a company and the selection process of government projects. The knapsack problem is formally defined as follows. The input is a pair (V, W), where V is a set of n items, $W \in \mathbb{R}_{>0}$ is a weight limit, and each item $i \in V$ has a positive weight $w(i) \leq W$ and value $v(i)$. The goal of the problem is to find a set of items $S \subseteq V$ that maximizes the total value $\sum_{i \in S} v(i)$, subject to the weight constraint $\sum_{i \in S} w(i) \leq W$.

Sometimes, the information used to allocate resources is uncertain or outdated. For example, suppose that a satellite isolated from the Earth is taking actions. The satellite has a list of potential actions V, and each action has a fixed value $v(i)$ and fuel consumption $w(i)$. Some of the actions may be infeasible at the moment due to the satellite's conditions, such as the atmosphere or surrounding space debris. The satellite's objective is to find a combination of feasible actions with the highest possible total value without running out of W amount of fuel.

The control room on the Earth wants to know the satellite's action for future planning. Since direct communication to the satellite is not always possible, the control room would simulate the satellite's decision process. However, the list V that the control room has as potential actions for the satellite may be different from the list that the satellite uses. Even in such a situation, the control room would like to predict the actual actions taken by the satellite with some accuracy. Such a purpose can be achieved by designing algorithms with small *average sensitivity* [24, 30].

The following definitions are necessary to formally define the average sensitivity. Let V be a finite set. Then, the *Hamming distance*[1] between two sets $S, S' \subseteq V$ is $|S \triangle S'|$, where $S \triangle S' = (S \setminus S') \cup (S' \setminus S)$ is the *symmetric difference.* For two probability distributions $\mathcal{S}$ and $\mathcal{S}'$ over sets in V, the *earth mover's distance* $\mathrm{EM}(\mathcal{S}, \mathcal{S}')$ between them is given by

$$\min_{\mathcal{D}} \mathop{\mathbb{E}}_{(S,S')\sim\mathcal{D}} |S \triangle S'|, \tag{1}$$

where the minimum is taken over distributions of a pair of sets such that its marginal distributions on the first and the second coordinates are equal to $\mathcal{S}$ and $\mathcal{S}'$, respectively. Let $\mathcal{A}$ be a (randomized) algorithm for the knapsack problem, and let (V, W) be an instance of the knapsack problem. We abuse the notation $\mathcal{A}$ (resp., $\mathcal{A}^i$) to represent the output (distribution) of the algorithm $\mathcal{A}$ on the instance (V, W) (resp., $(V \setminus \{i\}, W)$). Then, the *average sensitivity* of the algorithm (on the instance (V, W)) is

$$\frac{1}{n} \sum_{i \in V} \mathrm{EM}(\mathcal{A}, \mathcal{A}^i). \tag{2}$$

An algorithm is informally called *stable-on-average* if its average sensitivity is small.

1.2 Our Contributions

Our main contribution is a fully polynomial-time randomized approximation scheme (FPRAS) for the knapsack problem with a small average sensitivity:

▶ **Theorem 1.** *For any $\epsilon > 0$, there is a $(1-\epsilon)$-approximation algorithm for the knapsack problem with time complexity polynomial in $\epsilon^{-1} n$ and average sensitivity $O(\epsilon^{-1} \log \epsilon^{-1})$, where n is the number of items.*

It is noteworthy that Kumabe and Yoshida [20] have presented a $(1-\epsilon)$-approximation algorithm with $O(\epsilon^{-1} \log^3(nW))$ average sensitivity for the knapsack problem with integer weights. In contrast, our algorithm can be applied to an instance with non-integer weight and has a smaller average sensitivity bound independent of the weight limit W.

The following proposition states that the algorithm that outputs an optimal solution has an unbounded average sensitivity. Thus to bound the average sensitivity, we should allow for approximation as in Theorem 1.

▶ **Proposition 2.** *The average sensitivity of the algorithm that outputs an optimal solution can be as large as $\Omega(n)$.*

Let $\mathrm{opt}(V)$ be the optimal value of the original instance. The basic idea behind the algorithm of Theorem 1 is classifying items into two categories according to whether their values are more than a threshold $\approx \epsilon \cdot \mathrm{opt}(V)$. It is not difficult to design stable-on-average

[1] The Hamming distance is usually defined for strings, but our definition matches the formal definition by considering a set $S \subseteq V$ as a binary string of length $|V|$.

$(1-\epsilon)$-approximation algorithms when all the items belong to one of the two categories. Indeed, if all the items are small, i.e., with values at most $\epsilon \cdot \mathrm{opt}(V)$, then we can prove that a variant of the greedy algorithm has an approximation ratio $(1-O(\epsilon))$ and an $O(\epsilon^{-1})$ average sensitivity. If all items are large, i.e., with values at least $\epsilon \cdot \mathrm{opt}(V)$, then any algorithm has a small average sensitivity because any feasible solution contains at most ϵ^{-1} items. To combine these two algorithms, we add a subset of the large items selected by the *exponential mechanism* [23] to the output. However, if we apply the exponential mechanism to all possible sets of large items, the average sensitivity becomes too high because the number of such sets is exponentially large. Therefore, we need to reduce the number of candidate sets appropriately without sacrificing the approximation guarantee. A more detailed overview is provided in Section 4.1.

Next, we show that our upper bound on the average sensitivity is tight up to a logarithmic factor in ϵ^{-1}.

▶ **Theorem 3.** *Let $\epsilon > 0$. Then any $(1-\epsilon)$-approximation algorithm for the knapsack problem has average sensitivity $\Omega(\epsilon^{-1})$.*

The *simple knapsack problem* is a special case of the general knapsack problem in which the value of each item is proportional to its weight. For this problem, we obtain a deterministic algorithm with a better average sensitivity:

▶ **Theorem 4.** *For any $\epsilon > 0$, there is a deterministic $(1-\epsilon)$-approximation algorithm for the simple knapsack problem with time complexity polynomial in $\epsilon^{-1}n$ and average sensitivity $O(\epsilon^{-1})$, where n is the number of items.*

Finally, we discuss the connection to dynamic algorithms. In the *incremental dynamic knapsack problem*, items arrive one by one, and the goal is to maintain an approximate solution for the current set of items. The amortized recourse of a dynamic algorithm is the average Hamming distance between the outputs before and after an item arrives. More formally, the *amortized recourse* of a deterministic algorithm over a stream $v_1, \ldots, v_n$ of items is defined as $\frac{1}{n}\sum_{i=1}^{n} |X_{i-1} \triangle X_i|$, where X_i is the solution of the algorithm right after v_i is added. The amortized recourse of a randomized algorithm is the expectation of amortized recourse over the randomness of the algorithm. When the arrival order is random, we can construct an algorithm for the incremental dynamic knapsack problem using our stable-on-average algorithm:

▶ **Theorem 5.** *For any $\epsilon > 0$, there exists $f_\epsilon > 0$ such that there is a $(1-\epsilon)$-approximation algorithm with amortized recourse $O(\epsilon^{-1}\log \epsilon^{-1})$ and update time $O(f_\epsilon + \log n)$ for the incremental knapsack problem in the random-order setting.*

The fact that the output of our stable-on-average algorithm does not depend on the arrival order of the items but depends only on the current set of items implies that this result can also be applied to the *decremental knapsack problem*, in which we are to maintain a good solution while the items are removed one by one from the initial set of items.

1.3 Related Work

1.3.1 Knapsack Problem

The *knapsack problem* is one of the 21 problems that was first proved to be NP-hard by Karp [16], but admits a pseudo-polynomial time algorithm via dynamic programming [19]. Ibarra and Kim [13] proposed the first fully polynomial-time approximation scheme (FPTAS) for the knapsack problem. The main idea is to round the values of items into multiples of a small value, and run a dynamic programming algorithm on the resulting instance. Since then, several faster algorithms have been developed [5, 15, 17, 18, 21, 27].

The knapsack problem has been studied in the context of online settings. In the most basic *online knapsack problem*, items arrive one by one, and when an item arrives, it is irrevocably added it to the knapsack or discarded. The difference from our dynamic model is that, in the online knapsack problem, once an item is accepted or rejected, the decision cannot be reverted. However, this is not the case in our model. While Marchetti-Spaccamela and Vercellis [22] showed that no algorithm has a bounded competitive ratio for this problem in general, Buchbinder and Naor [4] proposed an $O(\log(U/L))$-competitive algorithm for the case where all weights are much smaller than the weight limit, where U and L are the upper and lower bounds on the *efficiency* $w(i)/v(i)$. Zhou, Chakrabarty, and Lukose showed that this bound is tight [32]. Several variations of the online knapsack problem have been investigated, such as the *removable online knapsack problem* [14], which allows removing items added from the knapsack, the *knapsack secretary problem* [2], which concerns items arriving in random order, and the *online knapsack problem with a resource buffer* [12], in which we have a buffer that can contain higher weights of items than the weight limit, and at the end, we can select a subset of the stored items as the output.

The problem most closely related to ours is the *dynamic setting*. In this problem, the goal is to maintain a good solution while items are added or deleted dynamically. In this model, there is no restriction on adding items that were previously deleted from the knapsack (and vice versa). While designing fast algorithms that maintain an exact solution is a major issue of focus in the context of data structures [1], the problem of maintaining approximate solutions has also been investigated in a variety of problems including the shortest path problem [28, 29], maximum matching problem [11, 25], and set cover problem [9]. Recently, dynamic approximation algorithms with bounded recourse have been studied for several problems including the bin-packing problem [8], set cover problem [9], and submodular cover problem [10]. As for the knapsack problem, a recent work of Böhm et al. [3] presented an algorithm that maintains a $(1-\epsilon)$-approximate solution with polylogarithmic update time in the fully dynamic setting, where both additions and deletions of items are allowed. Because this model considers both additions and deletions, the solution must be drastically changed even after a single update operation. Therefore, they proposed an algorithm that maintains a data structure from which a good solution can be recovered (with a time proportional to the size of the output), rather than explicitly maintaining the solution as a set of items.

1.3.2 Average Sensitivity

Murai and Yoshida [24] introduced the notion of average sensitivity for centralities, i.e., importance of nodes and edges, on networks to compare various notions of centralities. The notion of average sensitivity for graph problems was recently introduced by Varma and Yoshida [30], in which they studied various graph problems, including the minimum spanning tree problem, minimum cut problem, maximum matching problem, and minimum vertex cover problem. Zhou and Yoshida [31] presented a $(1-\epsilon)$-approximation algorithm for the maximum matching problem with sensitivity solely depending on ϵ, where the *(worst-case) sensitivity* is defined as (2) with the average replaced by the maximum over i. Kumabe and Yoshida [20] presented a stable-on-average algorithm for the maximum weight chain problem on directed acyclic graphs. The approximation ratio of their algorithm is $(1-\epsilon)$ and average sensitivity is polylogarithmic in the number of vertices in the graph, which roughly corresponds to the number of states of the dynamic programming. They designed stable-on-average algorithms for a wide range of dynamic programming problems, including the knapsack problem with integer weights, by reducing them to the maximum weight chain problem. Peng and Yoshida [26] analyzed the average sensitivity of spectral clustering and showed that it is proportional to λ_2/λ_3^2, where λ_i is the i-th smallest eigenvalue of the (normalized) Laplacian of the input graph. Intuitively, this bound indicates that spectral clustering is stable on average when the input graph can be well partitioned into two communities but not into three. This implies that spectral clustering is stable on average at relevant instances. The relation between *Differential privacy*, proposed by Dwork *et al.* [7], is given in the full version.

1.4 Organization

The rest of this paper is organized as follows. In Section 2, we review the basics of the knapsack problem, especially focusing on the fractional knapsack problem. In Section 3, we present a stable-on-average $(1 - O(\epsilon))$-approximation algorithm when all items have value at most ϵ, where the analysis is given in the full version. In Section 4, we present an (inefficient) stable-on-average algorithm for the general setting, while some proofs are given in the full version. In the full version, we discuss on improving the time complexity to obtain an FPRAS, prove the $O(\epsilon^{-1})$ lower bound on the average sensitivity of $(1 - \epsilon)$-approximation algorithms, present a deterministic $(1-\epsilon)$-approximation algorithm with $O(\epsilon^{-1})$ sensitivity for the simple knapsack problem, and discuss the application to the dynamic knapsack problem in the random-order setting.

2 Basic Facts about the Knapsack Problem

For an instance (V, W) of the knapsack problem, let $\mathrm{opt}(V, W)$ be the optimal value for the instance. As W is often set to 1, we define $\mathrm{opt}(V) := \mathrm{opt}(V, 1)$ for convenience. In this work, we assume that each item has a unique (comparable) identifier that remains unchanged with the deletion of other items. This naturally defines the ordering of items. In our algorithms, we implicitly use this ordering for the tiebreak. For example, when we sort items, we use a sorting algorithm that is stable with respect to this ordering. For a set $S \subseteq V$, let $v(S)$ and $w(S)$ denote $\sum_{i \in S} v(i)$ and $\sum_{i \in S} w(i)$, respectively.

We consider the following fractional relaxation of the knapsack problem, which we call the *fractional knapsack problem*:

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in V} v(i)x(i), \\ \text{subject to} \quad & \sum_{i \in V} w(i)x(i) \leq W, \\ & 0 \leq x(i) \leq 1 \quad (i \in V). \end{aligned}$$

Then, we denote the optimal value of this problem by $\mathrm{fopt}(V, W)$. We also define $\mathrm{fopt}(V) := \mathrm{fopt}(V, 1)$, and the *efficiency* of an item i is $v(i)/w(i)$. The following is well known:

▶ **Lemma 6** ([6]). *Suppose that items in V are sorted in non-increasing order of efficiency, i.e., $v(1)/w(1) \geq \cdots \geq v(n)/w(n)$. Let k be the largest index with $w(1) + \cdots + w(k) \leq 1$. Then, $\mathrm{fopt}(V, 1)$ is achieved by the solution*

$$x(i) = \begin{cases} 1 & (i = 1, \ldots, k), \\ \frac{1 - \sum_{j=1}^{k} w(j)}{w(i+1)} & (i = k + 1), \\ 0 & (i \geq k + 2). \end{cases}$$

Algorithm 1 A $(1 - O(\epsilon))$-approximation algorithm with small average sensitivity.

```
Procedure ModifiedGreedy($\epsilon, V$)
    Reorder the items of $V$ so that $v(1)/w(1) \geq \cdots \geq v(n)/w(n)$;
    Let $W$ be sampled from $[1-\epsilon, 1]$ uniformly at random;
    Let $t$ be the maximum index with $\sum_{i=1}^{t} w(i) \leq W$;
    return $\{1, \ldots, t\}$.
```

Using Lemma 6, $\mathrm{fopt}(V, W)$ can be computed in polynomial time. The following is a direct consequence of Lemma 6:

▶ **Lemma 7.** *For $W \leq 1$, we have $\mathrm{fopt}(V, W) \geq W \cdot \mathrm{fopt}(V)$.*

We give the proofs in the following two lemmas in full version.

▶ **Lemma 8.** $\mathrm{opt}(V) \leq \mathrm{fopt}(V) \leq 2\mathrm{opt}(V)$ *holds.*

▶ **Lemma 9.** *For an item set U and weight limit W, we have*

$$\sum_{i \in U} (\mathrm{fopt}(U, W) - \mathrm{fopt}(U \setminus \{i\}, W)) \leq \mathrm{fopt}(U, W).$$

3 Items with Small Values

In this section, we provide a stable-on-average algorithm for instances with each item having a small value. Specifically, we show the following:

▶ **Theorem 10.** *For any $\epsilon, \delta > 0$, there exists an algorithm for the knapsack problem with average sensitivity $O(\epsilon^{-1})$ that, given an instance (V, W) with each item having value at most δ, outputs a solution with value at least $(1-\epsilon)\mathrm{fopt}(V, W) - \delta$.*

This algorithm will be used in our algorithm for the general case as a subroutine in Section 4. Note that it suffices to design an algorithm for the case in which the weight limit is 1 because we can first divide the weight of each item by W and feed the resulting instance to the algorithm. Hence in the rest of this paper, we assume that the weight limit is 1 (and use the symbol W for a different purpose). A pseudocode of our algorithm is given in Algorithm 1. The proof of Theorem 10 is given in the full version.

4 General Case

In this section, we consider the general case of the knapsack problem, and prove the following:

▶ **Theorem 11.** *For any $\epsilon > 0$, there exists a $(1-\epsilon)$-approximation algorithm for the knapsack problem with average sensitivity $O(\epsilon^{-1} \log \epsilon^{-1})$.*

Note that this algorithm takes exponential time. We will discuss on improving time complexity in the full version.

4.1 Technical Overview and Algorithm Description

First, we explain the intuition behind our algorithm (Algorithm 2). As in Section 3, we assume that the weight limit is 1 without loss of generality. We fix a parameter $0 < \epsilon < 0.05$. We say that an item is *large* if its value is at least $\epsilon \cdot \mathrm{fopt}(V)$ and *small* otherwise. If all items are small, ModifiedGreedy (Algorithm 1) with the parameter ϵ has an approximation ratio $1 - O(\epsilon)$ and average sensitivity $O(\epsilon^{-1})$, and we are done. On the contrary, if all items are large, the procedure of outputting the optimal solution has average sensitivity $O(\epsilon^{-1})$ because any feasible solution has cardinality at most ϵ^{-1}. We combine these two observations to obtain an algorithm for the general case as follows. Note that here we don't concern about the running time; we accelerate it in the full version.

Let $L \subseteq V$ be the set of large items. First, we take a subset $R \subseteq L$ with $w(R) \leq 1$ that maximizes $v(R) + \mathrm{opt}(V \setminus L, 1 - w(R))$. (Note that in this section, we do not take efficiency into consideration and therefore we assume that $\mathrm{opt}(V \setminus L, 1 - w(R))$ can be computed.) Then, we output $R \cup \textsc{ModifiedGreedy}(\epsilon, V \setminus L, 1 - w(R))$, where $\textsc{ModifiedGreedy}(\epsilon, U, W)$ runs $\textsc{ModifiedGreedy}(\epsilon, U)$ after replacing $w(i)$ with $w(i)/W$ for each $i \in U$.

However, this algorithm has the following two issues.

1. If the value of $\mathrm{fopt}(V)$ changes upon deleting an item, the set L may drastically change. For example, consider the case when there are many items with values slightly less than $\epsilon \cdot \mathrm{fopt}(V)$. Then, the set of small items added to the output may drastically change.
2. Even when the value $\mathrm{fopt}(V)$ does not change, if the choice of R changes upon deleting an item, the value of $1 - w(R)$ also changes. Thus, the set of small items added to the output may drastically change.

To resolve the first issue, we sample a value threshold $c = O(\epsilon \cdot \mathrm{fopt}(V))$ that classifies items as large or small from an appropriate distribution, instead of fixing it to $\epsilon \cdot \mathrm{fopt}(V)$.

Now, we consider the second issue. Suppose we have sampled the same value threshold c both before and after deleting an item $i \in V$. There are two cases in which R changes after deleting the item i.

1. If the item i is large and $i \in R$, then the algorithm should change the choice of R because the item i would no longer exist.
2. If the item i is small, then the algorithm may change the choice of R because the value $\mathrm{fopt}(V \setminus L, 1 - w(R))$ (for the original R) may decrease.

The first case is easy to resolve; R contains $O(\epsilon^{-1})$ many items and therefore, by taking average over i, this case contributes to the average sensitivity by $O(\epsilon^{-1})$.

To address the second case (deleting small items), we ensure that the distribution of R does not change significantly with small decreases in $\mathrm{fopt}(V \setminus L, 1 - w(R))$. To this end, instead of considering the R with the maximum value of $v(R) + \mathrm{fopt}(V \setminus L, 1 - w(R))$ among $R \subseteq L$, we apply the exponential mechanism [23]. Specifically, we sample R with probability proportional to the exponential of this value with appropriate scaling and rounding (see Line 13 in Algorithm 2). To ensure that the mechanism outputs a $(1 - \epsilon)$-approximate solution, we reduce the number of candidates that can possibly be R to a constant. This is implemented by taking only one candidate A_t from the family of sets R with $w(R) \in [tc, (t + 1)c)$ for each integer t (see Line 7 in Algorithm 2). For technical reasons, we apply an exponential mechanism for the value $tc + \mathrm{fopt}(V \setminus L, 1 - w(A_t))$, rather than the exact value $v(A_t) + \mathrm{opt}(V \setminus L, 1 - w(A_t))$ (see Line 9).

In Sections 4.2 and 4.3, we analyze the approximation ratio and average sensitivity of Algorithm 2, respectively.

Algorithm 2 A $(1-O(\epsilon))$-approximation algorithm with small average sensitivity.

```
1  Procedure StableOnAverageKnapsack($\epsilon, V$)
2      Sample $c$ from the uniform distribution over $[\epsilon \cdot \mathrm{fopt}(V), 2\epsilon \cdot \mathrm{fopt}(V)]$;
3      Let $L \subseteq V$ be the set of items with value at least $c$;
4      Let $l = \lfloor \mathrm{fopt}(V)/c \rfloor$;
5      for $t = 0, \ldots, l$ do
6          if there is a subset of $L$ with value in $[tc, (t+1)c)$ then
7              Let $A_t \subseteq L$ be the set of items with smallest weight that satisfies
                 $tc \leq v(A_t) < (t+1)c$;
8              (if there are multiple choices, choose the lexicographically smallest one);
9              Let $x_t = tc + \mathrm{fopt}(V \setminus L, 1 - w(A_t))$;
10         else
11             Let $A_t = \emptyset$ and $x_t = -\infty$;
12     Let $d = \frac{c}{10\log(\epsilon^{-1})} = O\left(\frac{\epsilon}{\log(\epsilon^{-1})}\mathrm{fopt}(V)\right)$;
13     Sample $t^\circ \in \{0, \ldots, l\}$ with probability proportional to $\exp(x_{t^\circ}/d)$ and let
         $R = A_{t^\circ}$;
14     return $R \cup$ ModifiedGreedy($V \setminus L, 1 - w(A_t)$);
```

4.2 Approximation Ratio

First, we analyze the approximation ratio of Algorithm 2. Let S^* be a set of items that attains $\mathrm{opt}(V)$. Let $t^* \geq 0$ be an integer such that $v(S^* \cap L) \in [t^*c, (t^*+1)c)$.

The following lemma bounds the loss in the approximation ratio caused by considering only $A_0, \ldots, A_l$ instead of all subsets of L as candidate sets that can possibly be R.

▶ **Lemma 12.** *$x_{t^*} \geq (1-4\epsilon)\,\mathrm{opt}(V)$ holds.*

Proof. We have

$$\begin{aligned} x_{t^*} &= t^*c + \mathrm{fopt}(V \setminus L, 1 - w(A_{t^*})) \geq t^*c + \mathrm{fopt}(V \setminus L, 1 - w(S^* \cap L)) \\ &\geq t^*c + v(S^* \setminus L) \geq v(S^* \cap L) - c + v(S^* \setminus L) = \mathrm{opt}(V) - c \geq (1-4\epsilon)\mathrm{opt}(V), \end{aligned}$$

where the first equality is from the definition of x_{t^*}, the first inequality is from $w(A_{t^*}) \leq w(S^* \cap L)$, which is ensured by Line 7 in the algorithm, the second inequality is from $\mathrm{fopt}(V \setminus L, 1 - w(S^* \cap L)) \geq \mathrm{opt}(V \setminus L, 1 - w(S^* \cap L)) = v(S^* \setminus L)$, the third inequality is from the definition of t^*, the second equality is from the optimality of S^*, and the last inequality is from $c \leq 2\epsilon \cdot \mathrm{fopt}(V) \leq 4\epsilon \cdot \mathrm{opt}(V)$. ◀

Next we analyze the loss in the approximation ratio caused by the exponential method applied at Line 13. We prove the following.

▶ **Lemma 13.** *$\mathbb{E}[x_{t^\circ}] \geq (1-3\epsilon)x_{t^*}$ holds.*

Proof. We have

$$\begin{aligned}
&\Pr[x_{t^\circ} \le (1-\epsilon)x_{t^*}] = \frac{\sum_{t\in\{0,\dots,l\}: x_t \le (1-\epsilon)x_{t^*}} \exp\left(\frac{x_t}{d}\right)}{\sum_{t\in\{0,\dots,l\}} \exp\left(\frac{x_t}{d}\right)} \le \frac{(l+1)\exp\left(\frac{(1-\epsilon)x_{t^*}}{d}\right)}{\exp\left(\frac{x_{t^*}}{d}\right)} \\
&= (l+1)\exp\left(-\frac{\epsilon x_{t^*}}{d}\right) \\
&\le (l+1)\exp\left(-\frac{\epsilon(1-4\epsilon)\mathrm{opt}(V)}{d}\right) \le (l+1)\exp\left(-\frac{5}{2}\log\epsilon^{-1}(1-4\epsilon)\right) \\
&= (l+1)\epsilon^{\frac{5}{2}(1-4\epsilon)} \le 2\epsilon^{-1}\cdot\epsilon^2 = 2\epsilon. \qquad (3)
\end{aligned}$$

Here, the first equality is from Line 13 of the algorithm, the first inequality is from the fact that there are at most $l+1$ indices t with $x_t \le (1-\epsilon)x_{t^*}$, the second inequality is from Lemma 12, the third inequality is from

$$d = \frac{c}{10\log\epsilon^{-1}} \le \frac{\epsilon}{5\log\epsilon^{-1}}\cdot\mathrm{fopt}(V) \le \frac{5}{2}\cdot\frac{\epsilon}{\log\epsilon^{-1}}\cdot\mathrm{opt}(V),$$

and the last inequality is from $l+1 \le \epsilon^{-1}+1 \le 2\epsilon^{-1}$ and $\frac{5}{2}(1-4\epsilon) \ge 2$, which is from $\epsilon \le 0.05$. Therefore, we have

$$\mathbb{E}[x_{t^\circ}] \ge (1-\epsilon)x_{t^*}\Pr[x_{t^\circ} > (1-\epsilon)x_{t^*}] \ge (1-\epsilon)(1-2\epsilon)x_{t^*} \ge (1-3\epsilon)x_{t^*}.$$

Here, the second inequality is from (3). ◀

Combining the lemmas above yields the following.

▶ **Lemma 14.** $\mathbb{E}[v(A_{t^\circ}) + \mathrm{fopt}(V\setminus L, 1-w(A_{t^\circ}))] \ge (1-7\epsilon)\mathrm{opt}(V)$ *holds.*

Proof. We have

$$\begin{aligned}
&\mathbb{E}[v(A_{t^\circ}) + \mathrm{fopt}(V\setminus L, 1-w(A_{t^\circ}))] \\
&\ge \mathbb{E}[x_{t^\circ}] \ge (1-3\epsilon)x_{t^*} \ge (1-3\epsilon)(1-4\epsilon)\mathrm{opt}(V) \ge (1-7\epsilon)\mathrm{opt}(V),
\end{aligned}$$

where the first inequality is from Line 9 of the algorithm, the second inequality is from Lemma 13, and the third inequality is from Lemma 12. ◀

Now we bound the approximation ratio of Algorithm 2. Let $\mathcal{A}_{\mathrm{SMALL}}(V', W')$ be the output of Algorithm 1 on V', where all weights in the input are divided by W'.

▶ **Lemma 15.** *The approximation ratio of Algorithm 2 is at least* $1-12\epsilon$*.*

Proof. Let $\mathcal{A}$ be the output distribution of Algorithm 2 applied on the instance $(V,1)$. We have

$$\begin{aligned}
\mathbb{E}[\mathcal{A}] &= \mathbb{E}[v(A_{t^\circ}) + v(\mathcal{A}_{\mathrm{SMALL}}(V\setminus L, 1-w(A_{t^\circ})))] \\
&\ge \mathbb{E}[v(A_{t^\circ}) + (1-\epsilon)\mathrm{fopt}(V\setminus L, 1-w(A_{t^\circ})) - c] \\
&\ge \mathbb{E}[(1-\epsilon)(v(A_{t^\circ}) + \mathrm{fopt}(V\setminus L, 1-w(A_{t^\circ}))) - c] \\
&\ge \mathbb{E}[(1-\epsilon)(1-7\epsilon)\mathrm{opt}(V) - c] \ge (1-12\epsilon)\mathrm{opt}(V),
\end{aligned}$$

where the first inequality is from the analysis of Algorithm 1 given in the full version, the third inequality is from Lemma 14, and the last inequality is from $c \le 2\mathrm{fopt}(V) \le 4\mathrm{opt}(V)$. ◀

4.3 Average Sensitivity

In this section, we discuss bounding the average sensitivity of Algorithm 2. For the parameters used in the algorithm applied on the instance $(V \setminus \{i\}, 1)$, we use symbols c^i, d^i, and R^i to denote c, d, and R, respectively (and use c, d, and R for the instance $(V, 1)$). Similarly, we use the symbols A_t^i and x_t^i for $t = 0, \ldots, l$ to denote A_t and x_t, respectively, for the instance $(V \setminus \{i\}, 1)$. We first prove that the distributions of R and R^i are sufficiently close on average, where the average is taken over $i \in V$ (Lemma 22). Subsequently, we combine the analysis for large and small items (Lemma 24).

The distributions of R and R^i may differ for the following two reasons: the difference between the distributions of c and c^i, and the absence of the item i in the instance $V \setminus \{i\}$. We first forcus on the second reason. Specifically, we fix the value $\hat{c}$ and assume $c = c^i = \hat{c}$. In this case, we prove that the distribution of R is sufficiently close to that of R^i on average, where the average is taken over $i \in V$. To this end, we analyze the following quantity:

$$\sum_{i \in V} \sum_{A \subseteq V} \max\left(0, \Pr[R = A \mid c = \hat{c}] - \Pr[R^i = A \mid c^i = \hat{c}]\right). \tag{4}$$

Because $\Pr[R = A \mid c = \hat{c}]$ is positive only if $A = A_t$ for some $t \in \{0, \ldots, l\}$, we have

$$(4) = \sum_{i \in V} \sum_{t \in \{0,\ldots,l\}} \max\left(0, \Pr[R = A_t \mid c = \hat{c}] - \Pr[R^i = A_t \mid c^i = \hat{c}]\right). \tag{5}$$

We analyze the sum in (5) by dividing it into two cases: $i \in L$ and $i \in V \setminus L$. We first show that $x_t \geq x_t^i$ holds for all i and t.

▶ **Lemma 16.** *For any $t \in \{0, \ldots, l\}$ and $i \in V$, we have $x_t \geq x_t^i$.*

Proof. If $x_t = -\infty$, then we have $x_t^i = -\infty$ due to Line 6 of Algorithm 2. Hereafter, we assume $x_t \neq -\infty$. We prove the lemma by considering the following three cases:

▷ Claim 17. If $i \in L \setminus A_t$, we have $x_t = x_t^i$.

▷ Claim 18. If $i \in A_t$, we have $x_t \geq x_t^i$.

▷ Claim 19. If $i \in V \setminus L$, we have $x_t \geq x_t^i$.

We give the proofs of these claims in the full version. Then we complete the case analysis and the lemma is proved. ◀

The next lemma handles the case $i \in L$. We give the proof in the full version.

▶ **Lemma 20.** *For any $\hat{c} \in \mathbb{R}$, we have*

$$\sum_{i \in L} \sum_{t \in \{0,\ldots,l\}} \max\left(0, \Pr[R = A_t \mid c = \hat{c}] - \Pr[R^i = A_t \mid c^i = \hat{c}]\right) \leq \epsilon^{-1}.$$

Now we consider the case $i \in V \setminus L$.

▶ **Lemma 21.** *For any $\hat{c} \in \mathbb{R}$, we have*

$$\sum_{i \in V \setminus L} \sum_{t \in \{0,\ldots,l\}} \max\left(0, \Pr[R = A_t \mid c = \hat{c}] - \Pr[R^i = A_t \mid c^i = \hat{c}]\right) \leq 10\epsilon^{-1} \log \epsilon^{-1}.$$

Proof. Let $\hat{d} = \frac{\hat{c}}{10\log(\epsilon^{-1})}$. Then, we have

$$\begin{aligned}
&\Pr[R = A_t \mid c = \hat{c}] - \Pr[R^i = A_t \mid c^i = \hat{c}] \\
&= \frac{\exp(x_t/\hat{d})}{\sum_{t\in\{0,\dots,l\}}\exp(x_t/\hat{d})} - \frac{\exp(x_t^i/\hat{d})}{\sum_{t\in\{0,\dots,l\}}\exp(x_t^i/\hat{d})} \leq \frac{\exp(x_t/\hat{d}) - \exp(x_t^i/\hat{d})}{\sum_{t\in\{0,\dots,l\}}\exp(x_t/\hat{d})} \\
&= \left(1 - \exp\left(-\frac{x_t - x_t^i}{\hat{d}}\right)\right)\frac{\exp(x_t/\hat{d})}{\sum_{t\in\{0,\dots,l\}}\exp(x_t/\hat{d})} \leq \frac{x_t - x_t^i}{\hat{d}}\Pr[R = A_t \mid c = \hat{c}]. \qquad (6)
\end{aligned}$$

Here, the first equality is from Line 13 of Algorithm 2, the first inequality is from Lemma 16, and the last inequality is from $1 - \exp(-x) \leq x$. Now, we have

$$\begin{aligned}
&\sum_{i\in V\setminus L}\sum_{t\in\{0,\dots,l\}}\max\left(0, \Pr[R = A_t \mid c = \hat{c}] - \Pr[R^i = A_t \mid c^i = \hat{c}]\right) \\
&\leq \sum_{i\in V\setminus L}\sum_{t\in\{0,\dots,l\}}\frac{x_t - x_t^i}{\hat{d}}\cdot\Pr[R = A_t \mid c = \hat{c}] \\
&\leq \sum_{t\in\{0,\dots,l\}}\frac{\mathrm{fopt}(V\setminus L, 1 - w(A_t))}{\hat{d}}\cdot\Pr[R = A_t \mid c = \hat{c}] \\
&\leq \frac{\mathrm{fopt}(V)}{\hat{d}}\sum_{t\in\{0,\dots,l\}}\Pr[R = A_t \mid c = \hat{c}] \leq 10\epsilon^{-1}\log\epsilon^{-1}.
\end{aligned}$$

Here, the first inequality is obtained from (6) and Claim 19. The second inequality is obtained from:

$$\begin{aligned}
\sum_{i\in V\setminus L}(x_t - x_t^i) &= \sum_{i\in V\setminus L}(\mathrm{fopt}(V\setminus L, 1 - w(A_t)) - \mathrm{fopt}((V\setminus L)\setminus\{i\}, 1 - w(A_t))) \\
&\leq \mathrm{fopt}(V\setminus L, 1 - w(A_t)),
\end{aligned}$$

which is obtained from Lemma 9. The last inequality is from Line 12 of Algorithm 2. ◀

Combining Lemmas 20 and 21, we obtain the following.

▶ **Lemma 22.** *We have*

$$\sum_{i\in V}\sum_{A\subseteq V}\max\left(0, \Pr[R = A \mid c = \hat{c}] - \Pr[R^i = A \mid c^i = \hat{c}]\right) \leq 11\epsilon^{-1}\log\epsilon^{-1}.$$

Proof. The claim immediately follows from Lemmas 20 and 21 and the fact that V is a disjoint union of $V\setminus L$ and L. ◀

Now, we evaluate the average sensitivity. Let $\mathcal{A}^i$ be the output distribution of Algorithm 2 applied on the instance $V\setminus\{i\}$.

To bound the earth mover's distance, we consider transporting the probability mass in such a way the mass of $\mathcal{A}$ corresponding to a particular choice of c is transported to that of $\mathcal{A}^i$ for the same c^i (as far as we can). For the same choice of c and c^i, we consider transporting the probability mass in such a way that the mass corresponding to a particular choice of R is transported to that for the same R^i (as far as we can). For the same choice of c, R and c^i, R^i, we consider transporting the probability mass in a manner similar to the analysis of Algorithm 1. The remaining mass is transported arbitrarily.

First, we bound the contribution of the difference between the distributions of c and c^i to the earth mover's distance. Let f and f^i be the probability density functions of c and c^i, respectively. Precisely, $f(c) = \frac{1}{\epsilon\cdot\mathrm{fopt}(V)}$ if $\epsilon\cdot\mathrm{fopt}(V) \leq c \leq 2\epsilon\cdot\mathrm{fopt}(V)$ and 0 otherwise. Similarly, $f^i(c^i) = \frac{1}{\epsilon\cdot\mathrm{fopt}(V\setminus\{i\})}$ if $\epsilon\cdot\mathrm{fopt}(V\setminus\{i\}) \leq c^i \leq 2\epsilon\cdot\mathrm{fopt}(V\setminus\{i\})$ and 0 otherwise.

▶ **Lemma 23.** *We have*

$$\sum_{i\in V}\int_{\epsilon\cdot\mathrm{fopt}(V)}^{2\epsilon\cdot\mathrm{fopt}(V)} \max\left(0, f(\hat{c}) - f^i(\hat{c})\right) \mathrm{d}\hat{c} \leq 2.$$

Proof. We have

$$\sum_{i\in V}\int_{\epsilon\cdot\mathrm{fopt}(V)}^{2\epsilon\cdot\mathrm{fopt}(V)} \max\left(0, f(\hat{c}) - f^i(\hat{c})\right) \mathrm{d}\hat{c} = \sum_{i\in V}\int_{\max\{\epsilon\cdot\mathrm{fopt}(V),2\epsilon\cdot\mathrm{fopt}(V\setminus\{i\})\}}^{2\epsilon\cdot\mathrm{fopt}(V)} \frac{1}{\epsilon\cdot\mathrm{fopt}(V)}\mathrm{d}\hat{c}$$
$$\leq \frac{1}{\epsilon\cdot\mathrm{fopt}(V)}\sum_{i\in V}(2\epsilon\cdot(\mathrm{fopt}(V)-\mathrm{fopt}(V\setminus\{i\}))) \leq \frac{1}{\epsilon\cdot\mathrm{fopt}(V)}\cdot 2\epsilon\cdot\mathrm{fopt}(V) = 2,$$

where the first inequality is from the fact that $f(\hat{c}) \leq f^i(\hat{c})$ holds if $\epsilon\cdot\mathrm{fopt}(V) \leq \hat{c} \leq 2\epsilon\cdot\mathrm{fopt}(V\setminus\{i\})$, and the last inequality is from Lemma 9. ◀

Finally, we bound the average sensitivity.

▶ **Lemma 24.** *The average sensitivity of Algorithm 2 is at most* $12\epsilon^{-1}\log\epsilon^{-1}$.

Proof. We have

$$\begin{aligned}
&\frac{1}{n}\sum_{i\in V}\mathrm{EM}(\mathcal{A},\mathcal{A}^i)\\
&\leq \frac{1}{n}\sum_{i\in V}\int_{\epsilon\cdot\mathrm{fopt}(V)}^{2\epsilon\cdot\mathrm{fopt}(V)} \max\left(0, f(\hat{c}) - f^i(\hat{c})\right) \mathrm{d}\hat{c}\cdot n\\
&\quad+\frac{1}{n}\sum_{i\in V}\int_{\epsilon\cdot\mathrm{fopt}(V)}^{2\epsilon\cdot\mathrm{fopt}(V)}\sum_{A\subseteq V}\Big(\max\left(0,\Pr[R=A\mid c=\hat{c}]-\Pr[R^i=A\mid c^i=\hat{c}]\right)\cdot n\\
&\quad+\Pr[R=A\mid c=\hat{c}]\cdot\mathrm{EM}\left(\mathcal{A}_{\mathrm{SMALL}}(V\setminus L,1-w(A)),\mathcal{A}_{\mathrm{SMALL}}((V\setminus L)\setminus\{i\},1-w(A))\right)\Big)f(\hat{c})\mathrm{d}\hat{c}\\
&\leq 2+\frac{1}{n}\int_{\epsilon\cdot\mathrm{fopt}(V)}^{2\epsilon\cdot\mathrm{fopt}(V)}\left(11\epsilon^{-1}\log\epsilon^{-1}n+\sum_{i\in V}\sum_{A\subseteq V}\Pr[R=A\mid c=\hat{c}](\epsilon^{-1}+1)\right)f(\hat{c})\mathrm{d}\hat{c}\\
&= 2+\frac{1}{n}\int_{\epsilon\cdot\mathrm{fopt}(V)}^{2\epsilon\cdot\mathrm{fopt}(V)}\left(11\epsilon^{-1}\log\epsilon^{-1}n+(\epsilon^{-1}+1)\cdot n\right)f(\hat{c})\mathrm{d}\hat{c}\\
&= 2+11\epsilon^{-1}\log\epsilon^{-1}+\epsilon^{-1}+1\leq 12\epsilon^{-1}\log\epsilon^{-1},
\end{aligned}$$

where the first inequality is due to the transport of the probability mass, and the second inequality is from Lemmas 23, 22, and the analysis of Algorithm 1 given in the full version. ◀

Proof of Theorem 11. Applying Lemma 15 and Lemma 24 and replacing ϵ with $\min(0.05, \epsilon/12)$ proves this theorem. ◀

References

1 Mikhail Atallah and Marina Blanton. *Algorithms and theory of computation handbook*. CRC press, 2009.

2 Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *Approximation, randomization, and combinatorial optimization. Algorithms and techniques*, pages 16–28. Springer, 2007.

3 Martin Böhm, Franziska Eberle, Nicole Megow, Bertrand Simon, Lukas Nölke, Jens Schlöter, and Andreas Wiese. Fully dynamic algorithms for knapsack problems with polylogarithmic update time. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, 2021.

4 Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Mathematics of Operations Research*, 34(2):270–286, 2009.

5 Timothy Chan. Approximation schemes for 0-1 knapsack. In *Symposium on Simplicity in Algorithms*, 2018.

6 George Dantzig. Discrete variable extremum problems. *Operations Research*, 5:266–277, 1957.

7 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, pages 265–284. Springer, 2006.

8 Björn Feldkord, Matthias Feldotto, Anupam Gupta, Guru Guruganesh, Amit Kumar, Sören Riechers, and David Wajc. Fully-dynamic bin packing with little repacking. In *International Colloquium on Automata, Languages, and Programming*, 2018.

9 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *ACM SIGACT Symposium on Theory of Computing*, pages 537–550, 2017.

10 Anupam Gupta and Roie Levin. Fully-dynamic submodular cover with bounded recourse. *IEEE Symposium on Foundations of Computer Science*, pages 1147–1157, 2020.

11 Manoj Gupta and Richard Peng. Fully dynamic $(1+\epsilon)$-approximate matchings. In *IEEE Symposium on Foundations of Computer Science*, pages 548–557, 2013.

12 Xin Han, Yasushi Kawase, Kazuhisa Makino, and Haruki Yokomaku. Online knapsack problems with a resource buffer. In *International Symposium on Algorithms and Computation*, 2019.

13 Oscar Ibarra and Chul Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, 1975.

14 Kazuo Iwama and Shiro Taketomi. Removable online knapsack problems. In *International Colloquium on Automata, Languages, and Programming*, pages 293–305, 2002.

15 Ce Jin. An improved fptas for 0-1 knapsack. In *International Colloquium on Automata, Languages, and Programming*, 2019.

16 Richard Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

17 Hans Kellerer and Ulrich Pferschy. A new fully polynomial time approximation scheme for the knapsack problem. *Journal of Combinatorial Optimization*, 3(1):59–71, 1999.

18 Hans Kellerer and Ulrich Pferschy. Improved dynamic programming in connection with an fptas for the knapsack problem. *Journal of Combinatorial Optimization*, 8(1):5–11, 2004.

19 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

20 Soh Kumabe and Yuichi Yoshida. Average sensitivity of dynamic programming. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1925–1961, 2022.

21 Eugene Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979.

22 Alberto Marchetti-Spaccamela and Carlo Vercellis. Stochastic on-line knapsack problems. *Mathematical Programming*, 68(1):73–104, 1995.

23 Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *IEEE Symposium on Foundations of Computer Science*, pages 94–103, 2007.

24 Shogo Murai and Yuichi Yoshida. Sensitivity analysis of centralities on unweighted networks. In *The World Wide Web Conference*, pages 1332–1342, 2019.

25 Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *ACM Symposium on Theory of Computing*, pages 457–464, 2010.

26 Pan Peng and Yuichi Yoshida. Average sensitivity of spectral clustering. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1132–1140. ACM, 2020.

27 Donguk Rhee. *Faster fully polynomial approximation schemes for knapsack problems*. PhD thesis, Massachusetts Institute of Technology, 2015.

28 Yossi Shiloach and Shimon Even. An on-line edge-deletion problem. *Journal of the ACM*, 28(1):1–4, 1981.

29 Jan van den Brand and Danupon Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *IEEE Symposium on Foundations of Computer Science*, pages 436–455, 2019.

30 Nithin Varma and Yuichi Yoshida. Average sensitivity of graph algorithms. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 684–703, 2021.

31 Yuichi Yoshida and Samson Zhou. Sensitivity analysis of the maximum matching problem. In *Innovations in Theoretical Computer Science*, pages 58:1–58:20, 2021.

32 Yunhong Zhou, Deeparnab Chakrabarty, and Rajan Lukose. Budget constrained bidding in keyword auctions and online knapsack problems. In *International Workshop on Internet and Network Economics*, pages 566–576. Springer, 2008.