Tight Algorithms for the Submodular Multiple Knapsack Problem

Xiaoming Sun * Jialin Zhang * Zhijie Zhang *

July 10, 2020

Abstract

Submodular function maximization has been a central topic in the theoretical computer science community over the last decade. Plenty of well-performing approximation algorithms have been designed for the maximization of monotone/non-monotone submodular functions over a variety of constraints. In this paper, we consider the submodular multiple knapsack problem (SMKP), which is the submodular version of the well-studied multiple knapsack problem (MKP). Roughly speaking, the problem asks to maximize a monotone submodular function over multiple bins (knapsacks). Recently, Fairstein et al. [8] presented a tight $(1 - 1/e - \epsilon)$-approximation randomized algorithm for SMKP. Their algorithm is based on the continuous greedy technique which inherently involves randomness. However, the deterministic algorithm of this problem has not been understood very well previously. In this paper, we present deterministic algorithms with improved approximation ratios for SMKP.

We first consider the case when the number of bins is a constant. Previously a randomized approximation algorithm obtained approximation ratio $(1 - 1/e - \epsilon)$ based on the involved continuous greedy technique. Here we provide a simple combinatorial deterministic algorithm with ratio $(1 - 1/e)$ by directly applying the greedy technique. We then generalize the result to arbitrary number of bins. When the capacity of bins are identical, we design a combinatorial and deterministic algorithm which can achieve the tight approximation ratio $(1 - 1/e - \epsilon)$. In the general case, we provide a $(1/2 - \epsilon)$-approximation algorithm which is also combinatorial and deterministic. We finally show a $1 - 1/e - \epsilon$ randomized algorithm for the general case, thus achieving the result as in Fairstein et al. [8].

*CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences. University of Chinese Academy of Sciences. {sunxiaoming, zhangjialin, zhangzhijie}@ict.ac.cn.

1 Introduction

The multiple knapsack problem (MKP) is defined as follows. We are given a set N of n items and a set of m bins (knapsacks) such that the j-th bin has a capacity B_j and an item $u \in N$ has a size $c(u)$ and a profit $p(u)$. The task is to select a subset of items such that the sum of their profits is maximized and they can be packed into those m bins without exceeding the capacities. It is well-known that the problem admits a PTAS but has no FPTAS assuming P $\neq$ NP [15, 5, 14].

In this paper, we consider the submodular generalization of the above problem, called the submodular multiple knapsack problem (SMKP). Instead of specifying a profit for each item, the profit of each subset of items is described by a monotone submodular function f defined over all subsets of N. The task is again to select a subset of items which maximizes the value of f and can be packed into those m bins without exceeding the capacities. The objective function $f : 2^N \to \mathbb{R}$ is called *submodular* if $f(S+u) - f(S) \geq f(T+u) - f(T)$ for any $S \subseteq T$ and $u \notin T$, which means the marginal value diminishes when the set becomes larger.

By introducing a monotone submodular objective function, SMKP falls into the field of submodular function maximization, which captures many fundamental problems in combinatorial optimization, such as maximum coverage problem, maximum cut problem, submodular welfare, influence maximization, etc. The research of submodular maximization lasts for more than forty years. As early as in 1978, a simple greedy algorithm was proposed which yields a $(1-1/e)$-approximation for the problem of maximizing a monotone submodular function subject to a cardinality constraint $|S| \leq k$ [20], and a $1/2$ approximation for the problem of maximizing a monotone submodular function subject to a matroid constraint [12]. On the other hand, even for the cardinality constraint, submodular maximization problems can not be approximated within a ratio better than $1-1/e$ [19]. Since then, it remains a central open question in this field whether or not the problem of maximizing a monotone submodular function admits a $(1-1/e)$-approximation when subject to a matroid constraint. Until 2008, a groundbreaking work [22] answers this question affirmatively by proposing the so-called *continuous greedy* technique. Since then, a plenty of tight or well-performed approximation algorithms have been devised to maximize (monotone or non-monotone) submodular functions over a variety of constraints [2, 3, 4, 6, 10, 11, 13, 17, 18, 23]. In submodular multiple knapsack problem, this technique also shows great power. Recently, Fairstein et al. [8] presented a tight $(1-1/e-\epsilon)$-approximation randomized algorithm for SMKP in general case. Their algorithm combines the continuous greedy technique and approximation schemes for packing problems.

However, although the continuous greedy technique achieves great success in submodular maximization problem under various constraints, this technique inherently involves randomness, thus it is quite difficult to imply any deterministic algorithms. This makes a huge gap between the performance guarantees of the best known deterministic algorithms and randomized algorithms for several problems. For example, for the monotone submodular maximization under matroid constrain, we have tight $1-1/e$ randomized algorithm while by far the best deterministic ratio is 0.5008; for the non-monotone submodular maximization, the gap between randomized algorithm and deterministic algorithm is 0.385 versus $1/e$ under cardinality constraint and 0.385 versus $1/4-\epsilon$ under matroid constraint.

This gap also exists in the submodular multiple knapsack problem. While the almost optimal randomized algorithm has been proposed for this problem, we know very little about the performance of deterministic algorithms. By generalizing the algorithms for the multiple knapsack problem in [5], it is easy to obtain a 0.468 deterministic algorithm for the identical-size case where the capacity of bins are identical, and a $\frac{1-1/e}{2-1/e} \approx 0.387$ deterministic algorithm for the general case. In this paper, we show how to completely close the gap for the identical-size case and we also significantly reduce the gap for the general case.

1.1 Our Results

In this work, we present several deterministic approximation algorithms with improved ratio for SMKP. We provide 1) a tight $(1-1/e)$-approximation algorithm in the *constant case* where the number of bins m is a constant; 2) an almost-tight $(1-1/e-\epsilon)$-approximation algorithm in the *identical-size case* where all the bins have the same capacity $B_j \equiv B$; 3) a $(1/2-\epsilon)$-approximation algorithm in the general case. For the general case, one of the key ingredient is that we propose a $(1/2-\epsilon)$-approximation algorithm for the *bounded-size case* where the ratio between the maximum capacity and the minimum capacity of the bins is bounded by a constant. We can also boost the approximation ratio to $(1-1/e-\epsilon)$ with the help of standard continuous greedy technique in the general case, which gives a different almost-tight randomized approximation algorithm compared to the work of Fairstein et al. [8]. The following table summarizes our results in this paper and lists the best known results previously as comparison.

Case	Previous result	Our result
Constant bins	$1-1/e-\epsilon$ [16] (randomized, continuous greedy)	$1-1/e$ (deterministic)
Identical-size	0.468 (folklore) (deterministic) $1-1/e-\epsilon$ [8] (randomized, continuous greedy)	$1-1/e-\epsilon$ (deterministic)
General	0.387 (folklore) (deterministic) $1-1/e-\epsilon$ [8] (randomized, continuous greedy)	$1/2-\epsilon$ (deterministic) $1-1/e-\epsilon$ (randomized, continuous greedy)

The main contribution in this paper is the tight $(1-1/e-\epsilon)$-approximation algorithm in the identical-size case and the $1/2$ approximation algorithm in the general case. Both algorithms are deterministic and based on the greedy technique with several new insights in the analysis. We hope our new deterministic algorithms can inspire the design of deterministic algorithm for other constraints in the area, especially multiple linear constraint and the matroid constraint.

1.2 Technique Overview

All the algorithms in this paper are built on the greedy algorithm presented in Section 2.1. Roughly speaking, in each step, the greedy algorithm will select the element with the largest ratio of the marginal value to the size, and pack it into the bin if the bin is not full. Note that, the last selected element might violate the capacity constraint. We call this element a *reserved element*. Though the greedy algorithm might return an infeasible set, called G, the performance of this set is quite well. It is easy to check in the standard knapsack problem where the objective function f is a linear function, we have $f(G) \geq f(OPT)$. Even in the submodular knapsack problem, the solution G returned by the greedy algorithm also enjoys a good approximation. Formally, for any set $T \subseteq N$,

$$f(G) \geq \left(1 - e^{-c(G)/c(T)}\right) f(T).$$

Here $c(G) = \sum_{u \in G} c(u)$ is the size of G. By plugging $T = OPT$ into this inequality and combining the fact that $c(G) \geq$ total capacity $\geq c(OPT)$, we obtain an infeasible solution G which admits $(1-1/e)$-approximation. The algorithms in this paper are all about how one manages to round

the greedy solution into a feasible one by dealing with the reserved elements carefully, such that the rounded solution will not lose too much in the approximation ratio.

One widely used technique to handle this problem is the *enumeration technique*. By enumerating all possible partial solutions, we are able to pack large-valued elements in OPT. Then we can fill the bins by greedily picking small-valued elements, rendering the reserved elements to be small-valued. Since the reserved element is small-valued, we can directly discard this element. This idea provides us a simple polynomial time approximation algorithm when the number of bins is constant.

Theorem 1. *There is a deterministic combinatorial algorithm for SMKP which admits a $(1-1/e)$-approximation and runs in $O((mn)^{em+4})$ time, specially when the number of bins m is a constant, the algorithm runs in polynomial time.*

Remark. When the number of bins m is a constant, the randomized algorithm for the m-knapsack constraint [16] already achieves a $(1-1/e-\epsilon)$-approximation for SMKP. However, such algorithm adopts the continuous greedy technique and requires an involved rounding procedure.

1.2.1 Identical-size Case

When applying such idea to arbitrary m, the obstacle is that the enumeration step requires exponential time. This is because the "constant-bins" algorithm takes all the bins into consideration simultaneously. A natural idea to resolve it is to pack bins one at a time. Let S_j denote the elements packed in the j-th bin and T_j denote the elements packed in the first j bins by an algorithm. To run a greedy algorithm for SMKP is similarly to run a greedy algorithm on an exponential size submodular maximization problem (see the reductions in Section **??**). As a result, to achieve a $(1-1/e)$-approximation, we find it suffices to find the feasible sets $(S_1, S_2, \cdots, S_m$ and $f(S_j \mid T_{j-1}) \geq \frac{B_j}{c(OPT)} f(OPT \mid T_{j-1})$ where OPT is the optimal solution of the SMKP instance, B_j is the capacity of the j-th bin, and T_j denote the elements packed in the first j bins. We find it is possible if we allowed S_j to be the infeasible set returned by the greedy algorithm. Take the first bin as the example, if S_1 is infeasible set returned by the greedy algorithm, we have

$$f(S_1) \geq \left(1 - e^{-B_1/c(OPT)}\right) f(OPT).$$

The ratio $1-e^{-B_1/c(OPT)}$ is approximately $B_1/c(OPT)$ when $B_1/c(OPT)$ is small enough. For the identical-size case, this is exactly the case when m is large. Note, in the general case, this might not be true since it is possible that some bins have large capacity while there are numerous bins with very small capacity.

Our task is again to handle the reserved elements carefully. One thus may expect the enumeration technique still works here. However, this is not the case. The reason is simple: once the bin has packed some elements before the greedy process, the remaining capacity of the bin would become smaller, and therefore no longer a desired fraction of $c(OPT)$.

In order to tackle such difficulty, the first key idea we use is to enumerate large-sized elements rather than large-valued ones, and to greedily pick small-sized elements rather than small-valued ones. There are several advantages to classify elements according to their sizes instead of values. Firstly, the value function is submodular while the size function is linear. Thus we do not need to handle the difficulties coming from the change of marginal values. Such division is more stable and will not change in the whole process. Secondly, dividing elements according to their sizes allows us to manipulate them in a more flexible way, e.g. small-sized elements can be transferred among bins. Thirdly, in the analysis of greedy subroutine, we have $f(S) \geq \left(1 - e^{-c(S)/c(T)}\right) f(T)$. Since

the ratio mainly depends on the ratio between capacities, the based-on-size division is also useful in the analysis.

There are two major difficulties in the size-based-division. Firstly, since the reserved elements are small-sized, not small-valued, they may have large value and we cannot directly discard them. We introduce *reserved bins* to solve this problem. Suppose small-sized elements have size smaller than ϵB where B is the capacity of one bin, then all reserved elements of m bins can be packed into ϵm bins. We thus use ϵm bins to pack those reserved elements, and it is easy to show the loss of approximation ratio is at most ϵ. The second difficulty lies in the combination of enumeration and greedy algorithm. If the remaining capacity of the bin is not large enough, the greedy algorithm cannot guarantee good approximation. In the following, we will explain how to overcome such difficulty in the identical-size case.

For the identical-size case, the basic framework of the algorithm is as follows. For each bin, we adapt the enumeration step by only enumerating all feasible solutions of large-sized elements. Then we fill the remaining part of the bin for each enumerated set by packing only small-sized elements greedily. Finally, the set of the maximum value is returned, and the reserved element will be actually packed into reserved bins.

Take the first bin as an example. Let OPT_l and OPT_s be the set of large-sized and small-sized elements in OPT, respectively. Let $S_{1,l}$ and $S_{1,s}$ be the set returned by the enumeration step and the greedy step, respectively. To prove $f(S_1) \geq \frac{B_1}{c(OPT)} f(OPT) = \frac{f(OPT)}{m}$, the idea is to show that $f(S_{1,l}) \geq \frac{f(OPT_l)}{m}$ and $f(S_{1,s}|S_{1,l}) \geq \frac{f(OPT_s|S_{1,l})}{m}$, respectively. The first inequality is easy to achieve by the enumeration process, while the second one is satisfied if we have $c(S_{1,s}) \geq \frac{c(OPT_s)}{m}$. Equivalently, we hope the algorithm can find a suitable set $S_{1,l}$ such that

$$c(S_{1,l}) \leq \frac{c(OPT_l)}{m} \text{ and } f(S_{1,l}) \geq \frac{f(OPT_l)}{m}.$$

However, this is obviously a far-fetched condition for $S_{1,l}$. If we can only find $S_{1,l}$ with large size and large value, we cannot guarantee anything. But, one key observation is that if we find a set of large-size elements $S_{1,l}$ with large size and large value/size ratio, we can still achieve our goal. More formally, under mild precondition, we can prove $f(S_1) \geq \frac{f(OPT)}{m}$ once the following condition holds for $S_{1,l}$:

$$c(S_{1,l}) \geq \frac{c(OPT_l)}{m} \text{ and } \frac{f(S_{1,l})}{c(S_{1,l})} \geq \frac{f(OPT_l)}{c(OPT_l)}.$$

To summarize, we find two distinct sufficient conditions for finding a good S_1. Unfortunately, there exists a counter-example where there does not exist any set satisfying either conditions, even if we only require that they are satisfied approximately. Let $OPT_l = (OPT_{l,1}, \cdots, OPT_{l,m})$ be its partition into the m identical bins. One can imagine that the elements in OPT_l might be packed into the bins by OPT in an *unbalanced* manner. That is, some of $OPT_{l,i}$ have very large values as well as very large sizes, while others have small values as well as small sizes. In order to circumvent this difficulty, we introduce the *average argument* which shows that there must exists constant number of the sets $OPT_{l,i}$ whose combinations approximately satisfy one of the two conditions. More formally, we show there exists a set $S = OPT_{l,i_1} \cup OPT_{l,i_2} \cup \cdots \cup OPT_{l,i_s}$ where s is a constant, satisfying one of the following two conditions approximately:

$$\frac{c(S)}{s} \leq \frac{c(OPT_l)}{m} \text{ and } \frac{f(S)}{s} \geq \frac{f(OPT_l)}{m}.$$

$$\frac{c(S)}{s} \geq \frac{c(OPT_l)}{m} \text{ and } \frac{f(S)}{c(S)} \geq \frac{f(OPT_l)}{c(OPT_l)}.$$

This is the most intricate part in the proof. But the intuition is simple. Although the optimal solution might pack the large-size elements unbalanced among m bins, there exists a suitable way to group them which will eliminate imbalance quickly. We also change the algorithm according to this idea. In each step, we will consider constant number of bins instead of a single bin, enumerate all feasible solutions of large-sized elements and greedily fill the remaining part of bins with small-sized elements. Combining all these techniques, we have the following result for the identical-size case of SMKP.

Theorem 2. *For the identical-size case of SMKP, there is a polynomial time deterministic algorithm which yields a* $(1 - 1/e - O(\epsilon))$*-approximation.*

1.2.2 General Case

Next, we manage to generalize the above result to the general case. There are two major difficulties in the generalization. Firstly, in the identical-size case, the key idea is to show that for the single bin or constant number of bins, the algorithm can proportionally reach the value of optimal solution, that is, $f(S_j \mid T_{j-1}) \geq \frac{B_j}{c(OPT)} f(OPT \mid T_{j-1})$. However, when the capacities of bins are not identical, the capacity itself will introduce imbalance, and it is impossible to guarantee such global criteria. One extreme example is that there exists some element who can be packed into a large bin while exceeds the capacity of another small bin. If this element has large value, the optimal solution is intrinsic unbalanced and it is quite difficult to guarantee both the small bin and large bin reach the value of optimal solution proportionally. Secondly, when the capacities are not the same, it is difficult to define what is small-sized element, since if we set the criteria of small-sized element according to the capacity of the smallest bin, we may need exponential time to enumerate large-sized elements in the largest bin.

To resolve the first difficult, we will abandon the global criteria, and consider a more local criteria $f(S_j \mid T_{j-1}) \geq f(OPT_j \mid T_{j-1})$. This idea actually comes from the analysis for the classical greedy algorithm under a matroid constraint [12], and based on it, we can show the algorithm admits 1/2-approximation ratio. To resolve the second obstacle, we will first restrict ourselves to the bounded-size case in which the capacity $B_j \in [B, \gamma B]$, where B is the capacity of the smallest bin and $\gamma \geq 1$ is a constant. In the bounded-size case, we can safely define the small-sized elements according to B. We then generalize our result to the general case.

Bounded-size Case:

The algorithm framework for bounded-size case is quite similar as the identical-size case, but the analysis uses different ideas. The algorithm packs bins one at a time. For each bin, the algorithm enumerates large-sized elements, greedily picks small-sized elements, returns the set with largest value and packs the reserved element into the reserved bins. The fact that γ is a constant ensures that the algorithm is polynomial.

For the analysis, same as the identical-sized case, we need a small part of bins as reserved bins. We also draw a small part of bins as *patched bins*, and let m' be the number of remaining bins. We then compare our algorithm with the optimal solution OPT with m' bins and this will cause a small loss to the approximation ratio. We show how to cleverly partition OPT such that it is easier to compare OPT_j with S_j returned by our algorithm. The intuition is that the small-sized elements in OPT can be "arbitrarily" re-partitioned according to our requirements. After re-partition, we cannot guarantee the optimal solution can be packed into m' bins. But since the elements involving in the re-partition are all small-sized elements, we can use small number of bins to pack them. This is the function of patched bins. We will pack all elements in the optimal solution which cannot be packed into m' bins into the patched bins. In the algorithm, we also run greedy algorithm over small-sized elements in the corresponding patched bins. To conclude, we have the following result.

Theorem 3. *For the bounded-size case of SMKP, there is a polynomial time deterministic algorithm which yields a* $(\frac{1}{2} - 2\epsilon)$*-approximation.*

General Case:

Finally, we try to solve the general case based on our result for bounded-size case. The key point is the criteria used in the analysis of bounded-size case is local, that is, $f(S_j \mid T_{j-1}) \geq f(OPT_j \mid T_{j-1})$, thus it is possible to stack up the bins in a hierarchical way. The natural idea is to divide the bins into several groups according to their capacities: $[B, \gamma B], [\gamma B, \gamma^2 B], [\gamma^2 B, \gamma^3 B], \cdots$ where B is the capacity of smallest bin. Within one group, one can call the subroutine of bounded-size case, and can still obtain the local guarantee. However, the main obstacle is that the local guarantee is satisfied only if the number of bins in the group is large enough. If some group contains only small number of bins, although we can use the algorithm of constant bins to obtain $(1 - 1/e)$-approximation ratio within this group, the local guarantee will be no longer satisfied, and we can only expect to have the result $f(S_j \mid T_{j-1}) \geq (1 - 1/e) f(OPT_j \mid T_{j-1})$. Here j should mean some group of bins, but in the extreme case, when the group contains only one bin, it is coincidence with one bin. This makes the analysis fail to obtain the final 1/2-ratio.

There are two key ideas which help us circumvent this problem. Firstly, suppose the capacity of largest bin lies in $[\gamma^{k-1}B, \gamma^k B]$, then consider a group with constant number of bins whose capacity lies in $[\gamma^{i-1}B, \gamma^i B]$. If $i \ll k$, we can directly discard such group since the total capacity of this group is too small compared to the largest bin. Note, we cannot discard such group if it contains large number of bins, since the total capacity cannot be ignored. Thus, what we need to care is the groups with small number of bins but each bin has large capacity. We put all such groups together, and there are in total constant number of bins. Unfortunately, although the ratio of maximum capacity and minimum capacity of those bins are bounded (γ^c for some constant c), the number of bins might not be large enough and we cannot run our algorithm of bounded-size case. We do not know how to guarantee local criteria $f(S_j \mid T_{j-1}) \geq f(OPT_j \mid T_{j-1})$ for those bins. Instead, we use another way to bound the performance of those bins. We show if we simultaneous enumerate large-sized elements for all those bins, and then fill in the remaining space with greedy algorithm over small-sized elements, we can guarantee $f(S_j \mid T_j) \geq f(OPT_j \mid T_j)$. This is enough for the final 1/2-ratio. The disadvantage is that simultaneous enumeration is necessary since if we only handle those bins one by one and return the set of elements with maximum marginal value for each bin, we cannot guarantee this requirement. The lucky thing is that we only need to take care of constant number of bins in this part, thus the simultaneous enumeration is feasible.

Combining all these ideas, we finally generalize the results of bounded-size case to general case.

Theorem 4. *For SMKP, there is a polynomial time deterministic algorithm which yields a* $(\frac{1}{2} - 2\epsilon)$*-approximation.*

Finally, by applying the standard continuous greedy technique, we can boost our approximation ratio to $(1 - 1/e - O(\epsilon))$-approximation. The similar technique of boosting is also used in xxx. The resulting randomized algorithm enjoys almost tight approximation ratio. This gives us a different randomized algorithm compared to the work of Fairstein et al. [8].

Theorem 5. *For SMKP, there is a polynomial time randomized algorithm which yields a* $(1 - 1/e - O(\epsilon))$*-approximation.*

1.3 Related Work

The multiple knapsack problem has been fully studied previously. Kellerer [15] proposed the first PTAS for the identical-size case of the problem. Soon after, Chekuri and Khanna [5] proposed

a PTAS for the general case. The result was later improved to an EPTAS by Jansen [14]. On the other hand, it is easy to see that the problem does not admit an FPTAS even for the case of $m = 2$ bins unless P = NP, by reducing the PARTITION problem to it. For the problem of maximizing a monotone submodular function subject to a knapsack constraint, a tight $(1 - 1/e)$ algorithm was known which runs in $O(n^5)$ time [15, 21]. Later, a fast algorithm was proposed in [1] which achieves a $(1 - 1/e - \epsilon)$-approximation and runs in $n^2(\log n/\epsilon)^{O(1/\epsilon^8)}$ time. This was recently improved in [7] by a new algorithm which runs in $(1/\epsilon)^{O(1/\epsilon^4)} n \log^2 n$ time. For the problem of maximizing a monotone submodular function subject to the m-knapsack constraint, there is a tight $1 - 1/e - \epsilon$ algorithm [16] when the number of knapsacks m is a constant. The problem is NP-hard to approximate within an $n^{1-\epsilon}$ factor when m is a part of input, since it contains the INDEPENDENT-SET problem as a special case.

For the submodular multiple knapsack problem considered in this paper, in his Ph. D thesis [9], Feldman proposed a polynomial time $(1/9 - o(1))$-approximation algorithm and a pseudo polynomial time $1/4$ approximation algorithm for the problem. For the identical-size case where the capacity of bins are identical, he improved the results to a polynomial time $((e-1)/(3e-1) - o(1))$-approximation algorithm and a pseudo polynomial time $(1 - 1/e - o(1))$-approximation algorithm. Recently, Fairstein et al. [8] presented a tight $(1 - 1/e - \epsilon)$-approximation randomized algorithm for SMKP in general case. Their algorithm combines the continuous greedy technique and approximation schemes for packing problems.

Among them, the most relevant constraint to this work is the m-knapsack constraint, also called multiple linear constraints in the literatures.

The m-knapsack constraint assumes that there are m knapsacks and the j-th knapsack has budget B_j. In contrast to our problem, the constraint assumes that each picked element $u \in N$ incurs a cost $c_j(u)$ in the j-th knapsack for all $j \in [m] := \{1, \cdots, m\}$. a set of elements is feasible if it is feasible in each knapsack. In some sense, under the m-knapsack constraint, each picked element is packed into all knapsacks simultaneously; while in our model, under the capacity constraint, each picked element is packed into only one knapsack (we call it bin instead of knapsack in the paper). The m-knapsack constraint is closely related to the capacity constraint in our problem. Firstly, the submodular maximization problem subject to the 1-knapsack constraint (often known as a knapsack constraint) generalizes the classical knapsack problem, and is exactly SMKP with a single bin as input. Secondly, SMKP can actually be reduced to the submodular maximization problem subject to the m-knapsack constraint. The reduction is as follows. For an instance of SMKP, an item $u \in N$ can be divided into m items $u_1, \cdots, u_m$ and for $j \in [m]$, and u_j incurs a cost $c(u_j)$ in the j-th bin and a zero cost in other bins. The function value of a set of new items is equal to the value of the set of the original items from which they are generated. And now we have obtained a submodular maximization instance subject to the m-knapsack constraint. Actually, SMKP is strictly simpler than the submodular maximization problem subject to the m-knapsack constraint. The latter problem is NP-hard to approximate within an $n^{1-\epsilon}$ factor when m is a part of input, while it is not difficult to find a constant approximation algorithm for SMKP for arbitrary m (see the following subsection for details).

1.4 Organization

We first present a formal description of the submodular multiple knapsack problem (SMKP) and some notations in Section 2. The greedy algorithm and a tight $(1 - 1/e)$ algorithm for SMKP assuming the number of bins m is a constant are both presented in this section. The $1 - 1/e - \epsilon$ deterministic algorithms for the identical case and the $1/2 - \epsilon$ deterministic algorithm for the bounded case are presented in Section 3 and Section 4, respectively. Algorithms from these two

sections assume that the number of bins m is "large". The tight randomized algorithm for the general case will be presented in the full version of the paper. Finally, we conclude the paper in Section 5.

2 Preliminaries

Let $f : 2^N \to \mathbb{R}$ be a set function. f is *monotone* if $f(S) \leq f(T)$ for any $S \subseteq T$. f is *submodular* if $f(S+u) - f(S) \geq f(T+u) - f(T)$ for any $S \subseteq T$ and $u \notin T$. Here $S+u$ is a shorthand for $S \cup \{u\}$. Throughout this paper, we also use $f(u \mid S)$ and $f(T \mid S)$ to denote the marginal values $f(S+u) - f(S)$ and $f(S \cup T) - f(S)$, respectively. We assume that f is accessed by a value oracle that returns $f(S)$ given a set $S \subseteq N$.

An instance of the *submodular multiple knapsack problem* (SMKP) is defined as follows. We are given a ground set N of n elements and m bins (knapsacks). For $j \in [m] := \{1, \cdots, m\}$, the j-th bin has a positive capacity B_j. Each element $u \in N$ has a positive size $c(u)$, and the size of a set $S \subseteq N$ is $c(S) = \sum_{u \in S} c(u)$. In addition, there is a non-negative objective function $f : 2^N \to \mathbb{R}_{\geq 0}$ defined over all subsets of N. In this paper, the objective function f is assumed to be *monotone* and *submodular*. A set $S \subseteq N$ is feasible if there exists an ordered partition $(S_1, \cdots, S_m)$ of S such that $c(S_j) \leq B_j$ for each $j \in [m]$. In other words, S is feasible if there is a way to pack it into those m bins. The way how S is packed does not change the value of $f(S)$. The goal is to find a feasible set S (as well as the way it is packed) which maximizes the value of objective function f.

Notations. Throughout the paper, when we refer to a set $S \subseteq N$, we assume that some of its partition $(S_1, \cdots, S_m)$ is implicitly given. The partition may be generated from an algorithm, or be specified explicitly if necessary. Depending on the context, sometimes the partition might not be a feasible solution for the problem, that is, $c(S_j) \leq B_j$ might not hold for some $j \in [m]$. We use $|S|$ to denote the number of elements in S and $b(S) = |\{j \in [m] \mid S_j \neq \emptyset\}|$ to denote the number of bins used in its partition $(S_1, \cdots, S_m)$. The term $b(S)$ actually depends on the partition $(S_1, \cdots, S_m)$ of S, but we slightly abuse the notation here for ease of presentation.

2.1 The Greedy Algorithm

The greedy algorithm is shown as Algorithm 1. It returns a (possibly infeasible) set with a $(1-1/e)$ approximation ratio. All the algorithms in this paper are built based on it. It selects elements to be packed in a greedy manner, according to the ratios of elements' current marginal values to their sizes. It packs the selected elements into bins as long as there is a bin whose capacity has not been fully used. As a result, Algorithm 1 might not return a feasible solution. However, it is easy to see that each bin packs at most one additional element. That is, if $c(S_j) > B_j$ for some $j \in [m]$ and u is the last element added into S_j, then $c(S_j \setminus \{u\}) < B_j$ and hence $S_j \setminus \{u\}$ is feasible in the j-th bin. For convenience, we say Algorithm 1 returns an *almost feasible* solution. For each bin, the last element that violates the capacity constraint is called a *reserved element*. The following lemma lower bounds the quality of the almost feasible set returned by Algorithm 1.

Lemma 1. *Let S be the set returned by Algorithm 1. For any set $T \subseteq N$, we have*

$$f(S) \geq \left(1 - e^{-\frac{c(S)}{c(T)}}\right) \cdot f(T).$$

Proof. We assume that $c(S) \geq \sum_{j=1}^{m} B_j$, since otherwise all elements in N have been packed into S and therefore the lemma holds trivially. Assume that $S = \{u_1, u_2, \cdots, u_l\}$, and for $i \in [l]$,

Algorithm 1: GREEDY

Input: ground set N, objective function f, size function c, number of bins m, capacities $(B_1, \cdots, B_m)$.

Output: An almost feasible set (and the way it is packed).

Let $S = \emptyset$ and $S = (S_1, \cdots, S_m)$ be its partition into m bins.
while $N \setminus S \neq \emptyset$ *and there exists* $j \in [m]$ *such that* $c(S_j) < B_j$ **do**
  $u^* = \arg\max_{u \in N \setminus S} f(u \mid S)/c(u)$.
  $S = S + u^*$ and $S_j = S_j + u^*$.
end
return $S = (S_1, \cdots, S_m)$.

$S^i = \{u_1, u_2, \cdots u_i\}$ denotes the first i elements picked by the algorithm. Then, by the greedy rule,

$$\frac{f(u_i \mid S^{i-1})}{c(u_i)} \geq \frac{f(t \mid S^{i-1})}{c(t)}, \forall\, t \in T \backslash S^{i-1}.$$

This gives us

$$\frac{f(S^i) - f(S^{i-1})}{c(u_i)} \geq \frac{f(T \backslash S^{i-1} \mid S^{i-1})}{c(T \backslash S^{i-1})} \geq \frac{f(T) - f(S^{i-1})}{c(T)}.$$

The first inequality holds since f is submodular. The second inequality holds since f is monotone and $c(T \backslash S^{i-1}) \leq c(T)$.

We also assume that $f(T) > f(S^l)$, since otherwise the lemma already holds. Under this assumption, it must holds that $c(u_i) < c(T)$, since otherwise the inequality

$$\frac{f(S^i) - f(S^{i-1})}{c(u_i)} \geq \frac{f(T) - f(S^{i-1})}{c(T)}$$

implies that $f(T) \leq f(S^i) \leq f(S^l)$. A contradiction!

By rearranging the last inequality, we have

$$f(T) - f(S^i) \leq \left(1 - \frac{c(u_i)}{c(T)}\right)(f(T) - f(S^{i-1})).$$

The recurrence gives us

$$f(T) - f(S^i) \leq \prod_{j=1}^{i} \left(1 - \frac{c(u_j)}{c(T)}\right) \cdot f(T) \leq \prod_{j=1}^{i} e^{-\frac{c(u_j)}{c(T)}} \cdot f(T) = e^{-\frac{c(S^i)}{c(T)}} \cdot f(T).$$

The second inequality holds due to $e^x \geq 1 + x$. Hence we have

$$f(S^i) \geq \left(1 - e^{-\frac{c(S^i)}{c(T)}}\right) \cdot f(T).$$

The lemma follows immediately from it. □

Let OPT be an optimal solution. The above lemma immediately leads to the following corollary.

Corollary 1. *The set S returned by Algorithm 1 satisfies $f(S) \geq (1 - 1/e) f(OPT)$.*

Proof. If set S returned by Algorithm 1 satisfies $c(S) \geq \sum_{j=1}^{m} B_j$, then we have $c(S) \geq c(OPT)$, and Lemma 1 immediately leads to the corollary. If set S returned by Algorithm 1 satisfies $c(S) < \sum_{j=1}^{m} B_j$, this means $S = N$. The corollary also holds. □

Algorithm 2: Constant Number of Bins

Input: ground set N, objective function f, size function c, number of bins m (constant), capacities $(B_1, \cdots, B_m)$, threshold δ.

Output: A feasible solution (and the way it is packed).

$\mathcal{E} =$ Constant-bins-by-value(N, f, c, m, $(B_1, \cdots, B_m)$, δ).

return $S = \arg\max_{S_E \in \mathcal{E}} f(S_E)$ (and the way it is packed).

Procedure Constant-bins-by-value(N, f, c, m, $(B_1, \cdots, B_m)$, δ)**:**

  Enumerate all feasible solutions $E = (E_1, \cdots, E_m)$ such that $|E| \leq \lceil 1/\delta \rceil$.

  foreach *feasible solution* $E = (E_1, \cdots, E_m)$ **do**

    Let $D = \{u \in N \setminus E \mid f(u \mid E) > \delta f(E)\}$.

    $G'_E =$ Greedy$(N \setminus (E \cup D), f(\cdot \mid E), c(\cdot), m, (B_1 - c(E_1), \cdots, B_m - c(E_m)))$.

    Let $R_E \subseteq G'_E$ consist of the reserved elements in G'_E and $G_E = G'_E \setminus R_E$.

    Let $S_E = E \cup G_E$ (which is feasible).

  end

  return $\{S_E \mid$ feasible E that is enumerated$\}$

2.2 Constant Number of Bins

As a warm-up, we present a $(1 - 1/e)$ deterministic algorithm for SMKP, assuming the number of bins m is a constant. This ensures that in the remaining part of the paper, we only need to consider SMKP instances with "large" number of bins.

We already know the greedy algorithm in Section 2.1 returns a set with a $(1-1/e)$ approximation ratio; but this set might be infeasible, with a reserved element in each bin. If the values of these reserved elements are small, we are able to discard them directly without losing too much. However, the greedy algorithm itself cannot guarantee this property. In light of this, Algorithm 2 manages to first pack large-value elements in some optimal solution by the enumeration technique, and then pack elements of small value by the greedy algorithm. In doing so, it ensures that the values of the reserved elements are small and therefore can be safely discarded.

Theorem 6. *If we set $\delta = 1/em$, Algorithm 2 achieves a $(1 - 1/e)$-approximation and runs in $O((mn)^{em+4})$ time. When m is a constant, it runs in polynomial time.*

Proof. Denote by OPT the optimal solution. Assume w.l.o.g. that $|OPT| > \lceil 1/\delta \rceil$, since otherwise OPT will be enumerated in the enumeration step. We order elements in OPT greedily according to their marginal values, i.e. $o_1 = \arg\max_{o \in OPT} f(o)$, $o_2 = \arg\max_{o \in OPT \setminus \{o_1\}} f(o \mid o_1)$, etc. In the enumeration step, the solution $E = (E_1, \cdots, E_m)$ must be visited such that E contains exactly the first $\lceil 1/\delta \rceil$ elements in OPT and these elements are packed in the same way as in OPT. In the following analysis, we focus on this solution and show that $S_E = E \cup G_E$ achieves the desired ratio. Since the algorithm returns the solution with the maximum value, this completes the proof.

We claim that $f(o \mid E) \leq \delta f(E)$ for any $o \in OPT \setminus E$. Let OPT_i be the first i elements in OPT. Then for $j \leq i$ and any $o \notin OPT_i$, $f(o_j \mid OPT_{j-1}) \geq f(o \mid OPT_{j-1}) \geq f(o \mid OPT_i)$. Summing up from $j = 1$ to i, we have $f(OPT_i) \geq i \cdot f(o \mid OPT_i)$. By plugging $i = \lceil 1/\delta \rceil$, $f(E) \geq \lceil 1/\delta \rceil f(o \mid E) \geq 1/\delta f(o \mid E)$.

The above claim implies that $D \cap (OPT \setminus E) = \emptyset$. As a result, elements in $OPT \setminus E$ will not be excluded from the execution of the greedy algorithm. Besides, since elements in E are packed in the same way as in OPT, $OPT \setminus E$ is a feasible (indeed optimal) solution while invoking the greedy algorithm with capacities $(B_1 - c(E_1), \cdots, B_m - c(E_m))$.

By Corollary 1, the set G'_E returned by GREEDY satisfies

$$f(G'_E \mid E) \geq (1 - 1/e) f(OPT \setminus E \mid E).$$

Since G_E is obtained from G'_E by discarding at most m reserved elements in R_E, by the submodularity of f,

$$\begin{aligned} f(G_E \mid E) &\geq f(G'_E \mid E) - f(R_E \mid E) \\ &\geq (1 - 1/e) f(OPT \setminus E \mid E) - \sum_{u \in R_E} f(u \mid E) \\ &\geq (1 - 1/e) f(OPT) - (1 - 1/e) f(E) - \delta m f(E). \end{aligned}$$

Hence we have

$$f(E \cup G_E) \geq (1 - 1/e) f(OPT) + (1/e - \delta m) f(E) \geq (1 - 1/e) f(OPT).$$

The last inequality holds since $1/\delta \geq em$.

Finally, since there are $O((mn)^{1/\delta+2})$ feasible solutions $E = (E_1, \cdots, E_m)$ with $|E| \leq \lceil 1/\delta \rceil$, and GREEDY costs $O(n^2)$ queries, Algorithm 2 uses $O((mn)^{em+4})$ queries. Since m is a constant, Algorithm 2 runs in polynomial time. □

3 The Identical Case

In this section, we present a tight deterministic algorithm for the identical case of SMKP where all bins have the same capacity $B_j \equiv B$. The main algorithm is depicted as Algorithm 3. Given a constant $\epsilon > 0$ as input, it requires that the number of bins $m \geq 4/\epsilon^8$ and achieves a $1 - 1/e - O(\epsilon)$ approximation. When $m < 4/\epsilon^8$, one can run Algorithm 2 to achieve a tight $(1 - 1/e)$-approximation. Combining these two results, we resolve the identical case completely.

In Algorithm 3, the first $(1 - \epsilon)m$ bins are called *working bins*. In contrast, the last ϵm bins are called *reserved bins*. It mainly uses working bins to pack elements. A bin is called *empty* if no elements have been packed in this bin. Whenever there remains at least $4/\epsilon^7$ empty working bins, Algorithm 3 attempts to pack $4/\epsilon^7$ empty working bins simultaneously with the remaining elements by invoking Algorithm 4.

Algorithm 4 divides elements into two classes according to their sizes. Given input ϵ, an element $u \in N$ is *large* if $c(u) > \epsilon B$ and *small* otherwise. Let $N_l = \{u \in N \mid c(u) > \epsilon B\}$ be the set of large elements in N and $N_s = N \setminus N_l$ be the set of small elements. From a high-level viewpoint, Algorithm 4 first enumerates all feasible ways of packing only the large elements. Then, for each enumerated solution, small elements are added into it by invoking the greedy algorithm. Finally, the one with the maximum "average value" will be returned ($b(S_E)$ in line 8 of Algorithm 4 denotes the number of bins used to pack S_E, see Section 2).

Note that although Algorithm 3 invokes Algorithm 4 with $4/\epsilon^7$ bins, the set S returned by Algorithm 4 might use fewer than $4/\epsilon^7$ bins. This is because if an enumerated solution $E = (E_1, \cdots, E_m)$ only uses m' bins, Algorithm 4 will only add small elements to those m' non-empty bins. The only exception is that when $E = \emptyset$ (or equivalently $m' = 0$), Algorithm 4 will use a single bin to pack small elements.

The set S returned by Algorithm 4 might be infeasible. Due to the greedy algorithm, S might contain a reserved element in each of the bins it uses. Nonetheless, we still let Algorithm 4 returns it and tackle the reserved elements in Algorithm 3. At this point, the reserved bins come to rescue.

Algorithm 3: The Identical Case

Input: constant $\epsilon > 0$, ground set N, objective function f, size function c, number of bins $m\,(\geq 4/\epsilon^8)$, capacity B.

Output: A feasible set (and the way it is packed).

1 Divide bins into two classes: the first $(1-\epsilon)m$ bins are called *working bins*, and the last ϵm bins are called *reserved bins*.
2 Let T denote the packed elements so far and initialize it as $T = \emptyset$.
3 **while** *there remains at least* $4/\epsilon^7$ *empty working bins* **do**
4 $\quad S = \textsc{Constant-bins-by-size}(\epsilon, N \setminus T, f(\cdot \mid T), c(\cdot), 4/\epsilon^7, B)$.
5 $\quad$ Pack reserved elements in S into reserved bins.
6 $\quad T = T \cup S$.
7 **end**
8 **return** T (and the way it is packed).

Algorithm 4: Constant-bins-by-size

Input: constant $\epsilon > 0$, ground set N, objective function f, size function c, number of bins m (constant), capacity B.

Output: An almost feasible set (and the way it is packed).

1 Let $N_l = \{u \in N \mid c(u) > \epsilon B\}$ be the set of large elements and $N_s = N \setminus N_l$ be the set of small elements.
2 Enumerate all feasible solutions $E = (E_1, \cdots, E_m)$ such that $E \subseteq N_l$.
3 **foreach** *feasible solution* $E = (E_1, \cdots, E_m)$ **do**
4 $\quad$ Reorder E_j's to ensure that there is an integer $m' \leq m$ such that $E_1 \neq \emptyset, \cdots, E_{m'} \neq \emptyset, E_{m'+1} = \emptyset, \cdots, E_m = \emptyset$.
5 $\quad$ Let $m' = \max\{m', 1\}$.
6 $\quad$ Use the first m' bins to pack small elements and let $G_E = \textsc{Greedy}(N_s, f(\cdot \mid E), c(\cdot), m', (B - c(E_1), \cdots, B - c(E_{m'})))$. Then $S_E = E \cup G_E$ is an almost feasible solution.
7 **end**
8 **return** $S = \arg\max_E f(S_E)/b(S_E)$ (and the way it is packed).

Observe that all reserved elements must be small elements. Thus the ϵm empty reserved bins can pack at least m reserved elements. On the other hand, there are only $(1-\epsilon)m$ working bins; each of them contains at most one reserved element. Consequently, all reserved elements can be packed into the reserved bins without exceeding the capacities, thus, line 5 of Algorithm 3 can always be executed.

We remark that though Algorithm 2 and Algorithm 4 are very similar, there are some differences between them. First, in the enumeration step, Algorithm 2 enumerates feasible solutions containing constant number of elements, while Algorithm 4 enumerates feasible solutions of large elements. Second, in the greedy step, Algorithm 2 directly discards the reserved elements, while Algorithm 4 retains the reserved elements and may return an infeasible solution.

We now introduce some notations for sake of analysis. Assume that the **while** loop in Algorithm 3 has been executed $r \leq (1-\epsilon)m$ times. For $j \in [r]$, denote by S_j the set returned by Algorithm 4 in the j-th iteration of the loop and by T_j the packed elements after the j-th iteration of the loop. Then, $T_j = T_{j-1} \cup S_j$. Recall that $b(S_j)$ is the number of bins used to pack S_j by Algorithm 4. But here $b(S_j)$ does not count the number of reserved bins used to pack the reserved elements in

S_j. Let OPT be the optimal solution. The following lemma bounds the marginal value of S_j with respect to T_{j-1} in terms of the marginal value of OPT.

Lemma 2. *For $1 \leq j \leq r$, we have*

$$f(S_j \mid T_{j-1}) \geq \frac{(1-2\epsilon)b(S_j)}{m} f(OPT \setminus T_{j-1} \mid T_{j-1}).$$

The proof of Lemma 2 is delayed to Section 3.1. Roughly speaking, it means that the marginal value of S_j with respect to T_{j-1} is approximately a $b(S)/m$ fraction of the margin value of OPT. It directly implies the desired approximation ratio:

Theorem 7. *Algorithm 3 achieves a $(1-1/e-O(\epsilon))$ approximation ratio and runs in $O(mn^2 \left(\frac{4n}{\epsilon^7}\right)^{1/\epsilon})$ time.*

Proof. By Lemma 2 and the monotonicity of f, we have

$$f(T_j) - f(T_{j-1}) \geq \frac{(1-2\epsilon)b(S_j)}{m}(f(OPT) - f(T_{j-1})).$$

By rearranging the above inequality, we obtain

$$\left(1 - \frac{(1-2\epsilon)b(S_j)}{m}\right)(f(OPT) - f(T_{j-1})) \geq f(OPT) - f(T_j).$$

This gives us

$$\begin{aligned} f(OPT) - f(T_j) &\leq \prod_{i=1}^{j} \left(1 - \frac{(1-2\epsilon)b(S_j)}{m}\right) f(OPT) \\ &\leq \prod_{i=1}^{j} e^{-\frac{(1-2\epsilon)b(S_j)}{m}} f(OPT) \\ &\leq e^{-\frac{1-2\epsilon}{m}\sum_{i=1}^{j} b(S_j)} f(OPT). \end{aligned}$$

On the other hand, by the ending condition of the **while** loop,

$$\sum_{i=1}^{r} b(S_i) \geq (1-\epsilon)m - 4/\epsilon^7 \geq (1-2\epsilon)m.$$

The last inequality holds since $m \geq 4/\epsilon^8$. Combining the above inequalities, we have

$$f(OPT) - f(T_r) \leq e^{-(1-2\epsilon)^2} f(OPT),$$

which implies that

$$f(T_r) \geq (1 - e^{-(1-2\epsilon)^2}) f(OPT) = (1 - e^{-1} - O(\epsilon)) f(OPT).$$

The running time follows by observing that the algorithm runs in at most m rounds, the enumeration step costs $O(\left(\frac{4n}{\epsilon^7}\right)^{1/\epsilon})$ time and the greedy algorithm needs $O(n^2)$ time. □

3.1 Proof of Lemma 2

This section is dedicated to prove Lemma 2. For simplicity, we show that the lemma holds for the first iteration of the **while** loop in Algorithm 3. A similar argument holds for all subsequent iterations of the loop, in which it suffices to replace a set S by $S\backslash T_{j-1}$ and the function value $f(\cdot)$ by $f(\cdot \mid T_{j-1})$ in the j-th iteration of the loop. Let $OPT_l = \{u \in OPT \mid c(u) > \epsilon B\}$ be the set of large elements in OPT and $OPT_s = OPT\backslash OPT_l$ be the set of small elements in OPT. We prove Lemma 2 by a case analysis, based on the densities of the large and small elements in OPT (Lemma 3 and Lemma 5).

Lemma 3. *If* $\frac{f(OPT_s)}{c(OPT_s)} \geq \frac{f(OPT)}{c(OPT)}$ *holds, then* $f(S_1) \geq \frac{(1-2\epsilon)b(S_1)}{m} f(OPT)$.

Proof. We show by a case analysis that the lemma already holds even if Algorithm 4 takes one bin as input. That is, there is a set S such that $c(S) \leq B$ and $f(S) \geq \frac{1-2\epsilon}{m} f(OPT)$. Since Algorithm 4 returns a set S_E which maximizes $f(S_E)/b(S_E)$, the lemma follows immediately.

Case 1 ($c(OPT_s)$ is "large": $c(OPT_s) \geq \epsilon m B$). Consider the case where $E = \emptyset$ in the enumeration step of Algorithm 4. Recall that in this case the algorithm uses a single bin to pack small elements. It is easy to see that $c(G_\emptyset) \geq B$, since $c(OPT_s) \geq \epsilon m B \geq B$. By Lemma 1,

$$\begin{aligned}
f(G_\emptyset) &\geq \left(1 - e^{-B/c(OPT_s)}\right) \cdot f(OPT_s) \\
&\geq \left(\frac{B}{c(OPT_s)} - \frac{B^2}{2 \cdot c(OPT_s)^2}\right) \cdot f(OPT_s) \\
&\geq \left(\frac{c(OPT)}{m \cdot c(OPT_s)} - \frac{1}{2\epsilon^2 m^2}\right) \cdot f(OPT_s) \\
&\geq \left(\frac{1}{m} - \frac{1}{2\epsilon^2 m^2}\right) \cdot f(OPT) \\
&\geq \frac{1-\epsilon}{m} f(OPT).
\end{aligned}$$

The second inequality holds since $1 - e^{-x} \geq x - x^2/2$ for $x \geq 0$. The third inequality holds since $c(OPT) \leq mB$ and $c(OPT_s) \geq \epsilon m B$. The forth inequality holds due to the condition of the lemma and the monotonicity of f. The last inequality holds as long as $m \geq 1/(2\epsilon^3)$. Thus, in this case the lemma holds.

Case 2 ($f(OPT_s)$ is "large": $f(OPT_s) \geq (1-e^{-B/c(OPT_s)})^{-1} \cdot \frac{f(OPT)}{m}$). If $c(G_\emptyset) < B$, this means all small elements are added into $c(G_\emptyset) < B$. Hence, $f(G_\emptyset) \geq f(OPT_s) \geq (1 - e^{-B/c(OPT_s)})^{-1} \cdot \frac{f(OPT)}{m} \geq \frac{f(OPT)}{m}$. If $c(G_\emptyset) \geq B$, by Lemma 1,

$$f(G_\emptyset) \geq \left(1 - e^{-B/c(OPT_s)}\right) \cdot f(OPT_s) \geq f(OPT)/m.$$

Thus, in this case the lemma holds.

Case 3 (Both $c(OPT_s)$ and $f(OPT_s)$ are "small"). In this case, we assume that $c(OPT_s) < \epsilon m B$ and $f(OPT_s) < \left(1 - e^{-B/c(OPT_s)}\right)^{-1} \cdot \frac{f(OPT)}{m}$. We show that OPT_s only contributes a negligible value in OPT:

$$\begin{aligned}
f(OPT_s) &< (1 - e^{-1/\epsilon m})^{-1} \cdot \frac{f(OPT)}{m} \\
&\leq \left(\frac{1}{\epsilon m} - \frac{1}{2\epsilon^2 m^2}\right)^{-1} \frac{f(OPT)}{m}
\end{aligned}$$

$$\leq \left(\frac{1}{2\epsilon m}\right)^{-1} \frac{f(OPT)}{m}$$
$$= 2\epsilon f(OPT).$$

The first inequality holds since $(1-e^{-B/x})^{-1}$ is monotone increasing. The second inequality holds since $1-e^{-x} \geq x - x^2/2$ for $x \geq 0$. The third inequality holds as long as $m \geq 1/\epsilon$. Hence by the submodularity of f,

$$f(OPT_l) \geq f(OPT) - f(OPT_s) \geq (1-2\epsilon) \cdot f(OPT).$$

Let $E^* = \arg\max\{f(E) \mid E \subseteq N_l \text{ and } c(E) \leq B\}$. In the enumeration step of Algorithm 4, the feasible solution where a single bin is used to pack E^* will be enumerated, and it holds that

$$f(E^* \cup G_{E^*}) \geq f(E^*) \geq \frac{1}{m} f(OPT_l) \geq \frac{1-2\epsilon}{m} \cdot f(OPT).$$

Thus, in this case the lemma holds. □

If the condition of Lemma 3 does not hold, by the submodularity of f, it must hold that $\frac{f(OPT_l)}{c(OPT_l)} \geq \frac{f(OPT)}{c(OPT)}$. If we can show that $f(S_1) \geq \frac{(1-2\epsilon)b(S_1)}{m} f(OPT)$ under this condition, then Lemma 2 is proved. It turns out that this case is much more complicated than the previous case. To resolve it, the first key step of our analysis is to show that Lemma 2 holds under a more special case, as the following lemma suggests.

Lemma 4. *If $\frac{f(OPT_l)}{c(OPT_l)} \geq \frac{f(OPT)}{c(OPT)}$ holds, and additionally in the enumeration step in Algorithm 4 there is a feasible solution $E \subseteq OPT_l$ which satisfies*

$$\frac{c(E)}{b(E)} \geq \frac{c(OPT_l)}{m} \text{ and } \frac{f(E)}{c(E)} \geq \frac{f(OPT_l)}{c(OPT_l)},$$

then $f(S_1) \geq \frac{(1-\epsilon)b(S_1)}{m} f(OPT)$.

Proof. The first condition $\frac{c(E)}{b(E)} \geq \frac{c(OPT_l)}{m}$ for the enumerated set E corresponds to the case where on average the algorithm uses more space to pack large elements than OPT does. Although this ensures that the algorithm collects an enough value from large elements, it may make the algorithm collect an insufficient value from small elements, since there is no enough space for small elements. The key observation is that under the second condition $\frac{f(E)}{c(E)} \geq \frac{f(OPT_l)}{c(OPT_l)}$, one can use the redundant value collected from large elements to cover the deficit incurred when packing small elements. We now give a formal proof of this lemma.

Assume that $\frac{c(E)}{b(E)} \geq \frac{c(OPT)}{m}$. By the conditions of the lemma, $\frac{f(E)}{c(E)} \geq \frac{f(OPT)}{c(OPT)}$. Thus, $f(E) \geq \frac{b(E)}{c(E)} f(OPT)$ and the lemma follows. In the remaining part of the proof, we assume $\frac{c(E)}{b(E)} < \frac{c(OPT)}{m}$.

Next, we construct a set E' which is a surrogate of OPT_l and satisfies $c(E')/m = c(E)/b(E)$. Initially, $E' = OPT_l$. Let $OPT_s = \{u_1, u_2, \cdots\}$ be sorted in *reverse greedy order*, i.e. $u_1 = \arg\min_{u \in OPT_s} f(u \mid OPT_l)/c(u)$, $u_2 = \arg\min_{u \in OPT_s \setminus \{u_1\}} f(u \mid OPT_l \cup \{u_1\})/c(u)$, etc. Then, add elements in OPT_s into E' in this order until there is a u_k such that $c(E')/m \leq c(E)/b(E)$ and $c(E'+u_k)/m > c(E)/b(E)$. Such a u_k must exist since $\frac{c(OPT_l)}{m} \leq \frac{c(E)}{b(E)} < \frac{c(OPT)}{m}$ by our assumption. At this point, we split u_k into two elements u'_k and u''_k such that 1) $c(u'_k) = mc(E)/b(E) - c(E')$ and $c(u''_k) = c(u) - c(u'_k)$; 2) for any $S \subseteq N \setminus \{u_k\}$, $f(u'_k \mid S) = pf(u_k \mid S)$ and $f(u''_k \mid S) = f(u_k \mid S) - f(u'_k \mid S)$, where $p = c(u'_k)/c(u_k)$. It is easy to verify that f is still monotone submodular. Finally, add u'_k into E'. The construction of E' is finished.

We now show that $\frac{f(E)}{b(E)} \geq \frac{f(E')}{m}$. Since elements in OPT_s are added into E' in reverse greedy order, we have

$$\frac{f(E' \setminus OPT_l \mid OPT_l)}{c(E' \setminus OPT_l)} \leq \frac{f(OPT_s \mid OPT_l)}{c(OPT_s)}.$$

On the other hand, by the condition of the lemma,

$$\frac{f(OPT_s \mid OPT_l)}{c(OPT_s)} \leq \frac{f(OPT)}{c(OPT)} \leq \frac{f(OPT_l)}{c(OPT_l)}.$$

The above two inequalities together imply that

$$\frac{f(E') - f(OPT_l)}{c(E' \setminus OPT_l)} \leq \frac{f(OPT_l)}{c(OPT_l)}.$$

Since $c(E' \setminus OPT_l) = c(E') - c(OPT_l)$ due to $OPT_l \subseteq E'$, the last inequality is equivalent to $\frac{f(E')}{c(E')} \leq \frac{f(OPT_l)}{c(OPT_l)}$. Since $\frac{f(E)}{c(E)} \geq \frac{f(OPT_l)}{c(OPT_l)}$ by the condition of the lemma, we have $\frac{f(E)}{c(E)} \geq \frac{f(E')}{c(E')}$. Together with the fact that $\frac{c(E)}{b(E)} = \frac{c(E')}{m}$, we have $\frac{f(E)}{b(E)} \geq \frac{f(E')}{m}$.

On the other hand, let $E'' = (OPT_s \backslash E') - u_k + u''_k$. Clearly, $f(E' \cup E'') = f(OPT_l \cup OPT_s) = f(OPT)$ and $c(E' \cup E'') = c(OPT_l \cup OPT_s) = c(OPT)$. We assume that the set G_E returned by the greedy algorithm satisfies that $c(G_E) \geq b(E) \cdot B - c(E)$, since otherwise all the remaining small elements will be packed into G_E and it follows that $f(G_E \mid E) \geq f(E'' \mid E)$. Under this assumption, we have

$$\frac{c(G_E)}{b(E)} \geq B - \frac{c(E)}{b(E)} = B - \frac{c(E')}{m} \geq \frac{c(OPT) - c(E')}{m} = \frac{c(E'')}{m}.$$

By Lemma 1,

$$\begin{aligned} f(G_E \mid E) &\geq (1 - e^{-c(G_E)/c(E'')}) \cdot f(E'' \mid E) \\ &\geq (1 - e^{-b(E)/m}) \cdot f(E'' \mid E) \\ &\geq \left(\frac{b(E)}{m} - \frac{b(E)^2}{2m^2}\right) \cdot f(E'' \mid E) \\ &\geq \frac{(1-\epsilon)b(E)}{m} \cdot f(E'' \mid E). \end{aligned}$$

The third inequality holds since $1 - e^{-x} \geq x - x^2/2$ for $x \geq 0$. The last inequality holds as long as $b(E) \leq 2\epsilon m$, which is ensured by the facts that $b(E) \leq 4/\epsilon^7$ and $m \geq 4/\epsilon^8$. Finally, by the above argument,

$$\begin{aligned} f(E \cup G_E) &= f(E) + f(G_E \mid E) \\ &\geq \frac{b(E)}{m} f(E') + \frac{(1-\epsilon)b(E)}{m} f(E'' \mid E) \\ &\geq \frac{b(E)}{m} f(E') + \frac{(1-\epsilon)b(E)}{m} f(E'' \mid E') \\ &\geq \frac{(1-\epsilon)b(E)}{m} f(E' \cup E'') \\ &= \frac{(1-\epsilon)b(E)}{m} f(OPT). \end{aligned}$$

This completes the proof. □

In the enumeration step of Algorithm 4, if there is a feasible solution $E \subseteq OPT_l$ satisfying

$$\frac{c(E)}{b(E)} \leq \frac{c(OPT_l)}{m} \text{ and } \frac{f(E)}{b(E)} \geq \frac{f(OPT_l)}{m},$$

then Lemma 2 holds. Intuitively, this is because the solution E collects a sufficiently large value in OPT_l and there remains enough space to pack small elements using the greedy algorithm. Together with Lemma 4, we find two distinct sufficient conditions for $E \subseteq OPT_l$ in the enumeration step which lead to the proof of Lemma 2. The difficulty lies in how one can prove the existence of such a feasible solution E which satisfies one of the two sufficient conditions. We remark that if Algorithm 4 has only one bin as input, there are instances where no feasible subsets of OPT_l satisfy either conditions.

The second key step of our analysis is to prove a technical lemma by an averaging argument, which shows that the desired feasible solution must exist when Algorithm 4 takes multiple bins as input at a time. Indeed, constant bins suffice for the lemma to work. The technical lemma is presented in Section 3.2. Based on it, the final result is proved below in Lemma 5.

Lemma 5. *If* $\frac{f(OPT_l)}{c(OPT_l)} \geq \frac{f(OPT)}{c(OPT)}$ *holds, then* $f(S_1) \geq \frac{(1-2\epsilon)b(S_1)}{m} f(OPT)$.

Proof. For technical reasons, we first show that we can assume w.l.o.g. that $c(OPT_s) \geq \epsilon mB$ and $c(OPT_l) \geq \epsilon mB$, which means that both large and small elements occupy a non-negligible fraction in OPT.

Consider the case $c(OPT_s) \leq \epsilon mB$. We can assume w.l.o.g. that $c(OPT) \geq \frac{1}{2}mB$, since we can assume that a half size of each bin is used to pack elements. Under this assumption, $c(OPT_s) \leq 2\epsilon \cdot c(OPT)$. Due to the condition of this lemma and the submodularity of f,

$$\frac{f(OPT_s \mid OPT_l)}{c(OPT_s)} \leq \frac{f(OPT)}{c(OPT)} \leq \frac{f(OPT_l)}{c(OPT_l)}.$$

The above inequalities imply that $f(OPT_s \mid OPT_l) \leq 2\epsilon f(OPT)$. Thus,

$$f(OPT_l) = f(OPT) - f(OPT_s \mid OPT_l) \geq (1 - 2\epsilon) f(OPT).$$

Let $E^* = \arg\max\{f(E) \mid E \subseteq N_l \text{ and } c(E) \leq B\}$. In the enumeration step of Algorithm 4, the feasible solution where a single bin is used to pack E^* will be enumerated, and it holds that

$$f(E^* \cup G_{E^*}) \geq f(E^*) \geq \frac{1}{m} f(OPT_l) \geq \frac{1-2\epsilon}{m} \cdot f(OPT).$$

Thus the lemma holds in this case.

Consider the case $c(OPT_l) \leq \epsilon mB$. Consider the set $E = \{u_l\}$, where $u_l \in OPT_l$ satisfies $f(u_l)/c(u_l) \geq f(OPT_l)/c(OPT_l)$. It holds that $c(OPT_l)/m \leq \epsilon B \leq c(E) \leq B$ and E can be packed into a single bin. Consequently, E satisfies the conditions of Lemma 4, which implies the lemma.

In the remaining part of the proof, we assume that $c(OPT_s) \geq \epsilon mB$ and $c(OPT_l) \geq \epsilon mB$. Assume that in OPT, OPT_l is partitioned into $(OPT_{l,1}, OPT_{l,2}, \cdots, OPT_{l,m})$ with $c(OPT_{l,j}) \leq B$. We apply the procedure defined in Section 3.2 with $OPT_l = (OPT_{l,1}, OPT_{l,2}, \cdots, OPT_{l,m})$ as the initial partition for $t = \left\lceil \frac{2 \log 1/\epsilon^2}{\log 3/2} \right\rceil$ rounds. By Corollary 2, there is a set $E \subseteq OPT_l$ which is the union of at most 2^{t+1} subsets in the partition $(OPT_{l,1}, OPT_{l,2}, \cdots, OPT_{l,m})$ and satisfies one of the following two conditions.

1. $\frac{c(E)}{b(E)} \leq \frac{c(OPT_l)}{m} + \epsilon^2 B$ and $\frac{f(E)}{b(E)} \geq \frac{f(OPT_l)}{m}$.

2. $\frac{c(E)}{b(E)} \geq \frac{c(OPT_l)}{m} - \epsilon^2 B$ and $\frac{f(E)}{c(E)} \geq \frac{f(OPT_l)}{c(OPT_l)}$.

Since E is the union of at most 2^{t+1} subsets in the form of $OPT_{l,j}$, if E is packed as in OPT_l, the number of bins $b(E)$ used by it satisfies $b(E) \leq 2^{t+1} \leq 4/\epsilon^7$, by the value of t. As a result, in the enumeration step, the set E as well as the way it is packed in OPT_l will be enumerated. Below we complete the proof by showing that the lemma holds whenever the set E satisfies either of the above two conditions

Case 1: $\frac{c(E)}{b(E)} \leq \frac{c(OPT_l)}{m} + \epsilon^2 B$ and $\frac{f(E)}{b(E)} \geq \frac{f(OPT_l)}{m}$. Recall that $c(E) + c(G_E) \leq b(E) \cdot B$ and $c(OPT_l) + c(OPT_s) \leq mB$. We have

$$\begin{aligned}
\frac{c(G_E)}{b(E)} &\geq B - \frac{c(E)}{b(E)} \\
&\geq B - \frac{c(OPT_l)}{m} - \epsilon^2 B \\
&\geq \frac{c(OPT_s)}{m} - \frac{\epsilon c(OPT_s)}{m} \\
&= \frac{(1-\epsilon)c(OPT_s)}{m}.
\end{aligned}$$

The last inequality holds since we have assumed that $c(OPT_s) \geq \epsilon mB$. By Lemma 1,

$$\begin{aligned}
f(G_E \mid E) &\geq (1 - e^{-c(G_E)/c(OPT_s)}) f(OPT_s \mid E) \\
&\geq (1 - e^{-(1-\epsilon)b(E)/m}) f(OPT_s \mid E) \\
&\geq (1 - e^{-(1-\epsilon)b(E)/m}) f(OPT_s \mid OPT_l) \\
&\geq \left(\frac{(1-\epsilon)b(E)}{m} - \frac{(1-\epsilon)^2 b(E)^2}{2m^2} \right) (f(OPT) - f(OPT_l)) \\
&\geq \frac{(1-2\epsilon)b(E)}{m} (f(OPT) - f(OPT_l)).
\end{aligned}$$

The second to last inequality holds since $1 - e^{-x} \geq x - x^2/2$. The last inequality holds as long as $b(E) \leq \frac{2\epsilon m}{(1-\epsilon)^2}$, which follows from the facts that $b(E) \leq 4/\epsilon^7$ and $m \geq 4/\epsilon^8$. Finally,

$$\begin{aligned}
f(E \cup G_E) &= f(E) + f(G_E \mid E) \\
&\geq \frac{b(E)}{m} f(OPT_l) + \frac{(1-2\epsilon)b(E)}{m} (f(OPT) - f(OPT_l)) \\
&\geq \frac{(1-2\epsilon)b(E)}{m} f(OPT).
\end{aligned}$$

The lemma still holds in this case.

Case 2: $\frac{c(E)}{b(E)} \geq \frac{c(OPT_l)}{m} - \epsilon^2 B$ and $\frac{f(E)}{c(E)} \geq \frac{f(OPT_l)}{c(OPT_l)}$. If $\frac{c(E)}{b(E)} \geq \frac{c(OPT_l)}{m}$, then the conditions of Lemma 4 holds and therefore the lemma follows. Hence we assume that $\frac{c(E)}{b(E)} \leq \frac{c(OPT_l)}{m}$ and it follows that

$$\frac{c(G_E)}{b(E)} \geq \frac{c(OPT_s)}{m}.$$

By Lemma 1,

$$\begin{aligned}
f(G_E \mid E) &\geq (1 - e^{-c(G_E)/c(OPT_s)}) \cdot f(OPT_s \mid E) \\
&\geq (1 - e^{-b(E)/m}) f(OPT_s \mid OPT_l)
\end{aligned}$$

$$\geq \frac{(1-\epsilon)b(E)}{m}(f(OPT) - f(OPT_l)).$$

The last inequality holds since $1 - e^{-x} \geq x - x^2/2$, $b(E) \leq 4/\epsilon^7$ and $m \geq 4/\epsilon^8$.

On the other hand, by the condition of this case,

$$\begin{aligned} f(E) &\geq \frac{c(E)}{c(OPT_l)} f(OPT_l) \\ &\geq \left(\frac{b(E)}{m} - \frac{\epsilon^2 B b(E)}{c(OPT_l)}\right) f(OPT_l) \\ &\geq \left(\frac{b(E)}{m} - \frac{\epsilon b(E)}{m}\right) f(OPT) \\ &= \frac{(1-\epsilon)b(E)}{m} f(OPT_l). \end{aligned}$$

The last inequality holds since we have assumed that $c(OPT_l) \geq \epsilon m B$. Finally,

$$\begin{aligned} f(E \cup G_E) &= f(E) + f(G_E \mid E) \\ &\geq \frac{(1-\epsilon)b(E)}{m} f(OPT_l) + \frac{(1-\epsilon)b(E)}{m}(f(OPT) - f(OPT_l)) \\ &\geq \frac{(1-\epsilon)b(E)}{m} f(OPT). \end{aligned}$$

The lemma still holds in this case. □

3.2 A Technical Lemma

Below we define an auxiliary procedure to assist the analysis of Lemma 2. Note that the procedure itself will never be executed in the algorithm.

The procedure starts with a feasible set X^0 which enjoys a partition $X^0 = (X_1^0, X_2^0, \cdots, X_{m_0}^0)$ with m_0 subsets and satisfies $c(X_j^0) \leq B$ for all $j \in [m_0]$. Then it runs t rounds in order to obtain a new partition of X^0 with desired properties. Assume that at the beginning of the k-th round, the partition of X^0 has been updated to $X^{k-1} = (X_1^{k-1}, X_2^{k-1}, \cdots, X_{m_{k-1}}^{k-1})$, where each X_j^{k-1} is a union of b_j^{k-1} subsets in X^0. The procedure proceeds by applying the so-called Couple operation to construct a new partition $X^k = (X_1^k, X_2^k, \cdots, X_{m_k}^k)$ as follows.

1. Reorder the subsets in X^{k-1} according to their *average size* such that
$$\frac{c(X_1^{k-1})}{b_1^{k-1}} \leq \frac{c(X_2^{k-1})}{b_2^{k-1}} \leq \cdots \leq \frac{c(X_{m_{k-1}}^{k-1})}{b_{m_{k-1}}^{k-1}}.$$
2. If m_{k-1} is odd, let $c_{k-1} = \left|\left\{X_j^{k-1} : \frac{c(X_j^{k-1})}{b_j^{k-1}} \geq \frac{c(X^0)}{m_0}\right\}\right|$ and $d_{k-1} = \left|\left\{X_j^{k-1} : \frac{c(X_j^{k-1})}{b_j^{k-1}} < \frac{c(X^0)}{m_0}\right\}\right|$. If $c_{k-1} < d_{k-1}$, replace X_2^{k-1} by $X_1^{k-1} \cup X_2^{k-1}$, which contains $b_1^{k-1} + b_2^{k-1}$ subsets in X^0. Let $X_j = X_{j+1}$ for $j \in [m_{k-1} - 1]$ and finally let $m_{k-1} = m_{k-1} - 1$. If $c_{k-1} > d_{k-1}$, replace the last two sets $X_{m_{k-1}-1}^{k-1}$ and $X_{m_{k-1}}^{k-1}$ in X^{k-1} by $X_{m_{k-1}-1}^{k-1} \cup X_{m_{k-1}}^{k-1}$, which contains $b_{m_{k-1}-1}^{k-1} + b_{m_{k-1}}^{k-1}$ subsets in X^0, and let $m_{k-1} = m_{k-1} - 1$.
3. Let $X_1^k = X_1^{k-1} \cup X_{m_{k-1}}^{k-1}, X_2^k = X_2^{k-1} \cup X_{m_{k-1}-1}^{k-1}, \cdots, X_{m_k}^k = X_{\frac{m_{k-1}}{2}}^{k-1} \cup X_{\frac{m_{k-1}}{2}+1}^{k-1}$.

Clearly, $m_k = \lfloor \frac{m_{k-1}}{2} \rfloor$. Let $u_k = \max_j \frac{c(X_j^k)}{b_j^k} - \frac{c(X^0)}{m_0}$ and $l_k = \frac{c(X^0)}{m_0} - \min_j \frac{c(X_j^k)}{b_j^k}$. Since $\sum_{j=1}^{m_k} b_j^k = m_0$ and $\sum_{j=1}^{m_k} c(X_j^k) = c(X^0)$ for any $0 \le k \le t$, by a simple averaging argument, it holds that $u_k, l_k \ge 0$. The following lemma proves several properties about the above procedure.

Lemma 6. *The above procedure has the following properties.*

1. *For any $0 \le k \le t$ and $j \in [m_k]$, $2^k \le b_j^k < 2^{k+1}$.*
2. *Both u_k and l_k are monotone non-increasing in k.*
3. *If in X^{k-1}, $c_{k-1} \le d_{k-1}$, then $u_k \le \frac{2}{3} u_{k-1}$.*
4. *If in X^{k-1}, $c_{k-1} \ge d_{k-1}$, then $l_k \le \frac{2}{3} l_{k-1}$.*

Proof. 1) Since $b_j^0 = 1$ for any $j \in [m_0]$, property 1 holds trivially in the base case where $k = 0$. Assume that property 1 holds for $k-1$. For any $j \in [m_k]$, by the construction of X_j^k, there are either two indexes j_1 and j_2 such that $X_j^k = X_{j_1}^{k-1} \cup X_{j_2}^{k-1}$, or three indexes j_1, j_2, j_3 such that $X_j^k = X_{j_1}^{k-1} \cup X_{j_2}^{k-1} \cup X_{j_3}^{k-1}$. In either case, $b_j^k \ge 2^{k-1} + 2^{k-1} = 2^k$.

Next, we prove that $b_j^k < 2^{k+1}$. We first prove by induction that $m_0 - 2^k m_k < 2^k$ for $0 \le k \le t$. The claim holds trivially for $k = 0$. Assume it holds for $k-1$. If m_{k-1} is even, $m_0 - 2^k m_k = m_0 - 2^{k-1} m_{k-1} < 2^{k-1} < 2^k$. If m_{k-1} is odd, $m_0 - 2^k m_k = m_0 - 2^k \frac{m_{k-1}-1}{2} = m_0 - 2^{k-1} m_{k-1} + 2^{k-1} < 2^k$. Thus the claim holds. Let $b_j^k = 2^k + a_j^k$ for any $j \in [m_k]$. Then $m_0 = \sum_{j \in [m_k]} b_j^k = 2^k m_k + \sum_{j \in [m_k]} a_j$. Thus $\sum_{j \in [m_k]} a_j = m_0 - 2^k m_k < 2^k$, which implies $b_j^k < 2^{k+1}$ immediately.

2) For any $j \in [m_k]$, by the construction of X_j^k, there are either two indexes j_1 and j_2 such that $X_j^k = X_{j_1}^{k-1} \cup X_{j_2}^{k-1}$, or three indexes j_1, j_2, j_3 such that $X_j^k = X_{j_1}^{k-1} \cup X_{j_2}^{k-1} \cup X_{j_3}^{k-1}$. In the former case, we have

$$\frac{c(X_j^k)}{b_j^k} = \frac{c(X_{j_1}^{k-1}) + c(X_{j_2}^{k-1})}{b_{j_1}^{k-1} + b_{j_2}^{k-1}} \le \max_j \frac{c(X_j^{k-1})}{b_j^{k-1}}.$$

The same argument holds for the latter case. Thus we have $\max_j \frac{c(X_j^k)}{b_j^k} \le \max_j \frac{c(X_j^{k-1})}{b_j^{k-1}}$ and therefore $u_k \le u_{k-1}$. The same holds for l_k by a similar argument.

3) When $c_{k-1} \le d_{k-1}$, a set $X_{j_1}^{k-1}$ with $\frac{c(X_{j_1}^{k-1})}{b_{j_1}^{k-1}} \ge \frac{c(X^0)}{m_0}$ will not merge with another set in step 2 and will be coupled with another (possibly updated) set $X_{j_2}^{k-1}$ with $\frac{c(X_{j_2}^{k-1})}{b_{j_2}^{k-1}} < \frac{c(X^0)}{m_0}$ in step 3 to obtain a new set X_j^k. And we have

$$\begin{aligned}
&\frac{c(X_j^k)}{b_j^k} - \frac{c(X^0)}{m_0} \\
&= \frac{c(X_{j_1}^{k-1}) + c(X_{j_2}^{k-1})}{b_{j_1}^{k-1} + b_{j_2}^{k-1}} - \frac{c(X^0)}{m_0} \\
&= \frac{b_{j_1}^{k-1}}{b_{j_1}^{k-1} + b_{j_2}^{k-1}} \left(\frac{c(X_{j_1}^{k-1})}{b_{j_1}^{k-1}} - \frac{c(X^0)}{m_0} \right) + \frac{b_{j_2}^{k-1}}{b_{j_1}^{k-1} + b_{j_2}^{k-1}} \left(\frac{c(X_{j_2}^{k-1})}{b_{j_2}^{k-1}} - \frac{c(X^0)}{m_0} \right)
\end{aligned}$$

$$\leq \frac{b_{j_1}^{k-1}}{b_{j_1}^{k-1}+b_{j_2}^{k-1}}\left(\frac{c(X_{j_1}^{k-1})}{b_{j_1}^{k-1}}-\frac{c(X^0)}{m_0}\right)$$
$$\leq \frac{2}{3}u_{k-1}.$$

The last inequality holds since $b_{j_1}^{k-1} < 2b_{j_2}^{k-1}$, due to $b_{j_1}^{k-1} < 2^k$ and $b_{j_2} \geq 2^{k-1}$.

4) This is true by a similar argument as in case 3).

□

Given a monotone submodular function f defined on X^0, we have the following corollary.

Corollary 2. *Assume that* $t = \left\lceil \frac{2\log 1/\epsilon}{\log 3/2} \right\rceil$, *then in* X^t *there is a set* X_j^t *which contains at most* 2^{t+1} *subsets in* X^0 *and satisfies one of the following two conditions.*

1. $\frac{c(X_j^t)}{b_j^t} \leq \frac{c(X^0)}{m_0} + \epsilon B$ *and* $\frac{f(X_j^t)}{b_j^t} \geq \frac{f(X^0)}{m_0}$.
2. $\frac{c(X_j^t)}{b_j^t} \geq \frac{c(X^0)}{m_0} - \epsilon B$ *and* $\frac{f(X_j^t)}{c(X_j^t)} \geq \frac{f(X^0)}{c(X^0)}$.

Proof. By Lemma 6, any X_j^t contains $b_j^t < 2^{t+1}$ subsets in X^0. If the procedure runs t rounds, then either property 3 or property 4 in Lemma 6 holds for at least $t/2$ rounds. If property 3 holds for $t/2$ rounds, since u_k is non-increasing, $u_t \leq (\frac{2}{3})^{t/2}u_0 \leq (\frac{2}{3})^{t/2}B \leq \epsilon B$. Hence all X_j^t satisfy $\frac{c(X_j^t)}{b_j^t} \leq \frac{c(X^0)}{m_0} + \epsilon B$, and the one with the largest $f(X_j^t)/b_j^t$ satisfies $\frac{f(X_j^t)}{b_j^t} \geq \frac{f(X^0)}{m_0}$ simultaneously. Thus condition 1 in this corollary holds. If property 4 in Lemma 6 holds for $t/2$ rounds, by a similar argument, all X_j^t satisfy $\frac{c(X_j^t)}{b_j^t} \geq \frac{c(X^0)}{m_0} - \epsilon B$, and the one with the largest $f(X_j^t)/c(X_j^t)$ satisfies $\frac{f(X_j^t)}{c(X_j^t)} \geq \frac{f(X^0)}{c(X^0)}$ simultaneously. Thus condition 2 in this corollary holds. □

4 The General Case: 1/2 Ratio

In this section, we present a 1/2 deterministic algorithm for the general case of SMKP. A set of bins $\{B_1, B_2, \cdots, B_k\}$ is γ-*bounded* (or simply *bounded*) if $\max B_j / \min B_j \leq \gamma$, where γ is a constant. An instance of SMKP belongs to the *bounded case* if its bins $\{B_1, B_2, \cdots, B_m\}$ is γ-bounded for some constant γ. We first present a 1/2 deterministic algorithm for the bounded case in Section 4.1. Then, we generalize this result to the general case in Section 4.2.

4.1 A 1/2 Deterministic Algorithm for the Bounded Case

In this section, we consider the bounded case of SMKP where the ratio between the maximum capacity and the minimum capacity of the bins is bounded by a constant $\gamma \geq 1$. Equivalently, we assume all capacities $B_j \in [B, \gamma B]$, where B is the size of the smallest bin. We present a deterministic algorithm with a $(1-5\epsilon)/2$-approximation guarantee. The algorithm requires that the number of bins $m \geq \gamma \ln(1/\epsilon)/(2\epsilon^2)$. When this is not the case, we can use Algorithm 2 to obtain a tight $(1-1/e)$-approximation.

Algorithm 5 first divides bins into three classes as follows. It sorts bins in an decreasing order by their capacities. Let $m' = m - \gamma\ln(1/\epsilon)/(2\epsilon) - 3\epsilon m$. The first m' bins are *working bins*, the following $\gamma\ln(1/\epsilon)/(2\epsilon)$ are *patched bins*, and the last $3\epsilon m$ bins are *reserved bins*. Algorithm 5

Algorithm 5: The Bounded Case, 1/2-Ratio (BOUNDED)

Input: constant $\epsilon \in (0,1)$, ground set N, objective function f, size function c, the number of bins m, capacities $(B_1, \cdots, B_m)$ with $B_1 = B$ and $B_j \in [B, \gamma B]$.

Output: A feasible set (and the way it is packed).

Reorder bins such that $B_1 \geq B_2 \geq \cdots \geq B_m$. Let $m' = m - \gamma \ln(1/\epsilon)/(2\epsilon) - 3\epsilon m$. Divide bins into three classes: the first m' bins are *working bins*, the following $\gamma \ln(1/\epsilon)/(2\epsilon)$ are *patched bins*, and the last $3\epsilon m$ bins are *reserved bins*.

$T_0 = \emptyset$.

for $j = 1$ *to* m' **do**

  $S_j = \text{CONSTANT-BINS-BY-SIZE}(\epsilon, N \backslash T_{j-1}, f(\cdot \mid T_{j-1}), c(\cdot), 1, B_j + \epsilon B)$.

  Pack the elements in S_j that violate the capacity B_j into reserved bins.

  $T_j = T_{j-1} \cup S_j$.

end

$S_{m'+1} = \text{GREEDY}(N_s \backslash T_{m'}, f(\cdot \mid T_{m'}), c(\cdot), 2\gamma\epsilon \ln(1/\epsilon), (B_{m'+1}, \cdots, B_{m'+2\gamma\epsilon \ln(1/\epsilon)}))$.

Pack the reserved elements in $S_{m'+1}$ into reserved bins.

$T_{m'+1} = T_{m'} \cup S_{m'+1}$.

return $T_{m'+1}$ (and the way it is packed).

mainly uses working bins and patched bins to pack elements. It packs working bins one at a time by invoking Algorithm 4 (CONSTANT-BINS-BY-SIZE) with a single bin as input. But note that instead of taking the capacity B_j in the j-th iteration of the loop, Algorithm 4 takes $B_j + \epsilon B$ as input for technical reasons. After all working bins are packed, it uses patched bins to pack *small elements* that have not been packed so far by invoking the greedy algorithm.

Algorithm 5 requires the notions of large elements and small elements as in the identical case. We directly adopt these notions in the bounded case. That is, given input ϵ, an element $u \in N$ is *large* if $c(u) > \epsilon B$ and *small* otherwise. Again, $N_l = \{u \in N \mid c(u) > \epsilon B\}$ denotes the set of large elements in N and $N_s = N \backslash N_l$ denotes the set of small elements. Finally, for $j \in [m']$, there are other elements than the reserved element in S_j that violate the capacity B_j, since we replace B_j by $B_j + \epsilon B$ during the execution of Algorithm 4. However, it is easy to verify that these elements (including the reserved element) have a total size of at most $3\epsilon B$. Since there are $3\epsilon m$ reserved bins, by a similar argument as in the identical case, all the elements that violate the capacities can be packed into the reserved bins during the execution of Algorithm 5.

Analysis. For $j \in [m']$, denote by S_j the set returned by Algorithm 4 in the j-th iteration of the **for** loop and by T_j the packed elements after the j-th iteration of the loop. Then, $T_j = T_{j-1} \cup S_j$. Let $S_{m'+1}$ be the set returned by the greedy algorithm and $T_{m'+1} = T'_m \cup S_{m'+1}$. Assume that X is a feasible set over working bins, which means there is a partition $(X'_1, X'_2, \cdots, X'_{m'})$ such that $c(X'_j) \leq B_j$ for $j \in [m']$. If we can show that the marginal value of S_j with respect to T_{j-1} is at least the marginal value of X'_j, then $T_{m'}$ is a 1/2 approximation of X. However, this is not always possible. To circumvent this difficulty, we show that we can re-partition X into $m'+1$ parts $(X_1, \cdots, X_{m'}, X_{m'+1})$ to obtain the desired property. This is achieved by transferring small elements in X among the bins. The extra $X_{m'+1}$ contains small elements that are not packed into the working bins, which will be bounded by $S_{m'+1}$. This is why we need the patched bins. Finally, note that $(X_1, \cdots, X_{m'}, X_{m'+1})$ may not be a feasible partition over working bins and patched bins. A formal statement of this idea is formulated as Lemma 7

Lemma 7. *For a feasible set X over working bins, there is a partition $(X_1, \cdots, X_{m'}, X_{m'+1})$ of X such that for $j \in [m'+1]$,*

$$f(S_j \mid T_{j-1}) \geq (1-\epsilon) f(X_j \mid T_{j-1}).$$

Proof. Since X is feasible over working bins, there is a partition $(X'_1, \cdots, X'_{m'})$ such that $c(X'_j) \leq B_j$ for $j \in [m']$. We now construct the partition $(X_1, \cdots, X_{m'}, X_{m'+1})$ from it. Let X_l and X_s be the set of large elements and small elements in X, respectively. For $j \in [m'+1]$, let $X_{l,j} = X_j \cap X_l$ and $X_{s,j} = X_j \cap X_s$, respectively. For each element in X_l, we pack it exactly the same as $(X'_1, \cdots, X'_{m'})$ does. Thus, for $j \in [m']$, $X_{l,j}$ contains the same set of large elements as in $(X'_1, \cdots, X'_{m'})$, and $X_{l,m'+1} = \emptyset$. To construct $X_{s,j}$'s, intuitively, we will shuffle elements in X_s and pack them in the order of increasing marginal value. Assume that we have constructed $X_{s,1}, \cdots, X_{s,j-1}$, and hence $X_1, \cdots, X_{j-1}$ have been constructed. Define $\tilde{X}_j = X_s \setminus (X_1 \cup \cdots \cup X_{j-1})$ that contains the remaining small elements we need to pack. We then pack elements in $\tilde{X}_j$ into the j-th bin in the order of increasing marginal values with respect to $X_{l,j} \cup T_{j-1}$ until the bin is full. Specifically, Let $\tilde{X}_j = \{x_1, x_2, \cdots\}$ be sorted in *reverse greedy order* such that $x_1 = \arg\min_{x \in \tilde{X}_j} f(x \mid X_{l,j} \cup T_{j-1})/c(x)$, $x_2 = \arg\min_{x \in \tilde{X}_j \setminus \{x_1\}} f(x \mid \{x_1\} \cup X_{l,j} \cup T_{j-1})/c(x)$, etc. Let k be the first index such that $c(\{x_1, \cdots, x_k\}) \geq B_j - c(X_{l,j})$. Then, define $X_{s,j} = \{x_1, \cdots, x_k\}$. Since each working bin is fully used in the construction of $X_{s,j}$, we could have packed all elements in X_s into working bins by repeating the above procedure. But we will stop when we find $c(\tilde{X}_j) \leq \gamma B/(2\epsilon)$ for some $j \in [m']$. Let J be the index of such bin. We put all the elements in $\tilde{X}_J$ into $X_{m'+1}$ and finish our construction. To summarize, by the above construction, for any $1 \leq j < J$, we have $X_j = X_{l,j} \cup X_{s,j}$, $B_j \leq c(X_j) \leq B_j + \epsilon B$ since the last small element in the j-th bin violates the capacity, and $c(\tilde{X}_j) > \gamma B/(2\epsilon)$. For any $J \leq j \leq m'$, we have $X_j = X_{l,j}$ containing only large elements. For $j = m'+1$, $X_{m'+1} = \tilde{X}_J = X_s \setminus (X_1 \cup \cdots \cup X_{J-1})$ where $c(X_{m'+1}) = c(\tilde{X}_J) \leq \gamma B/(2\epsilon)$.

We now show that $f(S_j \mid T_{j-1}) \geq (1-\epsilon) f(X_j \mid T_{j-1})$ for $j < J$. Recall that $X_{l,j}$ contains the large elements packed in the j-th working bins by $(X'_1, \cdots, X'_{m'})$. Thus, when Algorithm 4 is invoked, $X_{l,j} \setminus T_{j-1}$ will be enumerated in the enumeration step. Let G_j be the set returned by the greedy algorithm which begins with $X_{l,j} \setminus T_{j-1}$. Since Algorithm 4 is invoked with parameter $B_j + \epsilon B$ instead of B_j, and G_j contains one reserved element that violates the capacity, we have $c(G_j) \geq B_j - c(X_{l,j} \setminus T_{j-1}) + \epsilon B$. On the other hand, $c(X_{s,j}) \leq B_j - c(X_{l,j}) + \epsilon B$ by the construction of $X_{s,j}$. Thus, $c(G_j) \geq c(X_{s,j})$.

By Lemma 1, we have

$$\begin{aligned}
&f(G_j \mid (X_{l,j} \setminus T_{j-1}) \cup T_{j-1}) \\
&= f(G_j \mid X_{l,j} \cup T_{j-1}) \\
&\geq (1 - e^{-c(G_j)/c(\tilde{X}_j \setminus T_{j-1})}) f(\tilde{X}_j \setminus T_{j-1} \mid X_{l,j} \cup T_{j-1}) \\
&\geq (1 - e^{-c(G_j)/c(\tilde{X}_j)}) f(\tilde{X}_j \mid X_{l,j} \cup T_{j-1}) \\
&\geq \left(\frac{c(G_j)}{c(\tilde{X}_j)} - \frac{c(G_j)^2}{2c(\tilde{X}_j)^2} \right) f(\tilde{X}_j \mid X_{l,j} \cup T_{j-1}) \\
&\geq \frac{(1-\epsilon) c(G_j)}{c(\tilde{X}_j)} f(\tilde{X}_j \mid X_{l,j} \cup T_{j-1}).
\end{aligned}$$

The second inequality holds since $1 - e^{-x}$ is non-decreasing. The third inequality holds since $1 - e^{-x} \geq x - x^2/2$ for $x \geq 0$. The last inequality holds as long as $c(G_j) \leq 2\epsilon \cdot c(\tilde{X}_j)$, which is guaranteed by the fact that $c(\tilde{X}_j) > \gamma B/(2\epsilon)$ and $c(G_j) \leq \gamma B$.

On the other hand, Since elements in $\tilde{X}_j$ are added into $X_{s,j}$ in reverse greedy order,

$$\frac{f(\tilde{X}_j \mid X_{l,j} \cup T_{j-1})}{c(\tilde{X}_j)} \geq \frac{f(X_{s,j} \mid X_{l,j} \cup T_{j-1})}{c(X_{s,j})}.$$

Combining the above inequalities, we have

$$f(G_j \mid X_{l,j} \cup T_{j-1}) \geq \frac{(1-\epsilon)c(G_j)}{c(X_{s,j})} f(X_{s,j} \mid X_{l,j} \cup T_{j-1}) \geq (1-\epsilon) f(X_{s,j} \mid X_{l,j} \cup T_{j-1}).$$

By adding $f(X_{l,j} \mid T_{j-1})$ on both sides of the last inequality, we obtain

$$f(S_j \mid T_{j-1}) \geq f(X_{l,j} \cup G_j \mid T_{j-1}) \geq (1-\epsilon) f(X_j \mid T_{j-1}).$$

Next, we show that $f(S_j \mid T_{j-1}) \geq f(X_j \mid T_{j-1})$ for $J \leq j \leq m'$. Since $X_j = X_{l,j}$ and $X_{l,j} \setminus T_{j-1}$ will be enumerated in the enumeration step, we have

$$f(S_j \mid T_{j-1}) \geq f(X_{l,j} \setminus T_{j-1} \mid T_{j-1}) = f(X_{l,j} \mid T_{j-1}) = f(X_j \mid T_{j-1}).$$

Finally, we show that $f(S_{m'+1} \mid T_{m'}) \geq (1-\epsilon) f(X_{m'+1} \mid T_{m'})$. Recall that $S_{m'+1}$ is obtained by running the greedy algorithm with $N_s \setminus T_{m'}$ and patched bins as input. Since there are $\gamma \ln(1/\epsilon)/(2\epsilon)$ patched bins and the capacity of each bin is at least B, the total capacity of patched bins is at least $\gamma B \ln(1/\epsilon)/(2\epsilon)$. Assume that $c(S_{m'+1}) \geq \gamma B \ln(1/\epsilon)/(2\epsilon)$, since otherwise $S_{m'+1} = N_s \setminus T_{m'} \supseteq X_s \setminus T_{m'}$ and the lemma holds trivially. On the other hand, $c(X_{m'+1}) \leq \gamma B/(2\epsilon)$ by its construction. As a result, $c(S_{m'+1})/c(X_{m'+1}) \geq \ln(1/\epsilon)$. By Lemma 1, we have

$$\begin{aligned}
&f(S_{m'+1} \mid T_{m'}) \\
&\geq (1 - e^{-c(S_{m'+1})/c(X_{m'+1} \setminus T_{m'})}) f(X_{m'+1} \setminus T_{m'} \mid T_{m'}) \\
&\geq (1 - e^{-c(S_{m'+1})/c(X_{m'+1})}) f(X_{m'+1} \mid T_{m'}) \\
&\geq (1 - e^{-\ln(1/\epsilon)}) f(X_{m'+1} \mid T_{m'}) \\
&= (1-\epsilon) f(X_{m'+1} \mid T_{m'}).
\end{aligned}$$

The second and third inequalities hold since $1 - e^{-x}$ is non-decreasing. □

Lemma 7 will be used in the analysis of the 1/2 algorithm for the general case of SMKP. For now, we use it to show that Algorithm 5 achieves a 1/2 approximation ratio in the bounded case.

Theorem 8. *Algorithm 5 achieves a $(1-5\epsilon)/2$ approximation ratio and runs in $O(mn^{\gamma/\epsilon+3})$ time. Since γ is a constant, it runs in polynomial time.*

Proof. Let OPT be an optimal solution of a bounded case SMKP instance and X^* be an optimal solution when one can only pack elements into working bins. Recall that there are $3\epsilon m$ reserved bins. Since $m \geq \gamma \ln(1/\epsilon)/(2\epsilon^2)$, the number of patched bins is $\gamma \ln(1/\epsilon)/(2\epsilon) \leq \epsilon m$. Hence, by the construction, the working bins consist of at least $(1-4\epsilon)m$ bins with the largest capacities. Thus, $f(X^*) \geq (1-4\epsilon) f(OPT)$.

By Lemma 7, there is a partition $(X_1^*, \cdots, X_{m'}^*, X_{m'+1}^*)$ of X^* such that for $j \in [m'+1]$,

$$f(S_j \mid T_{j-1}) \geq (1-\epsilon) f(X_j^* \mid T_{j-1}) \geq (1-\epsilon) f(X_j^* \mid T_{m'+1}).$$

The last inequality is due to the submodularity of f. Summing up these $m'+1$ inequalities and again by submodularity, we have $f(T_{m'+1}) \geq (1-\epsilon) f(X^* \mid T_{m'+1})$ and therefore $f(T_{m'+1}) \geq \frac{1-\epsilon}{2} f(X^*)$.

Thus, it holds that

$$f(T_{m'+1}) \geq \frac{1-\epsilon}{2} f(X^*) \geq \frac{1-5\epsilon}{2} f(OPT).$$

For complexity, Algorithm 5 needs to pack at most m working bins. For each working bin, there are $O(n^{\gamma/\epsilon+1})$ possible solutions in the enumeration step. Each enumerated solution is augmented by the $O(n^2)$ greedy algorithm. Thus the total running time is bounded by $O(mn^{\gamma/\epsilon+3})$. □

4.2 A 1/2 Deterministic Algorithm for the General Case

In this section, we present a 1/2 deterministic algorithm for the general case. From a high-level viewpoint, we manage to "reduce" a general-case instance to a bounded-case instance. For this purpose, Algorithm 6 will divide bins into groups and apply Algorithm 5 to each group of bins in sequence. For Algorithm 5 to work, we require that 1) each group of bins are bounded, and 2) each group contains enough bins. As lemma 8 shows, we can actually find a partition which almost satisfies our requirement by discarding some bins. But there may be a group in this partition which is not bounded. Fortunately, this group contains only constant number of bins, and we will pack this group by Constant-bins-by-value.

Lemma 8. *For any $\epsilon > 0$, let $\gamma = \ln(1/\epsilon)/(2\epsilon^2)$. There is a subset $C \subseteq \{B_1, B_2, \cdots, B_m\}$ such that C can be divided into groups $P_1, \cdots, P_k, P_{k+1}$ for some k with the following properties:*

1. *For $j \in [k]$, P_j are γ-bounded, and the number of bins in P_j satisfies $|P_j| \geq \gamma^2$.*
2. *P_{k+1} may not be bounded, but the number of bins in P_{k+1} satisfies $|P_{k+1}| < 5\gamma^2$.*
3. *Let OPT be an optimal solution over bins $\{B_1, \cdots, B_m\}$ and OPT^* be an optimal solution over bins C. Then, $f(OPT^*) \geq (1-\epsilon) f(OPT)$.*

Proof. Assume that $B_1 \leq B_2 \leq \cdots \leq B_m$. We first divide bins into groups $P'_1, \cdots, P'_{k'}, P'_{k'+1}$, where group P_i contains bins with their capacities lying in the interval $[\gamma^{i-1}B_1, \gamma^i B_1)$, and $B_m \in P_{k'+1}$. Let $k'' = k' - 4$. A group P'_i is *large* if the number of bins in it is at least γ^2; otherwise it is *small*. For small groups P'_i with index $i \leq k''$, we directly discard them. For small groups P'_i with index $k'' < i \leq k' + 1 = k'' + 5$, we merge them into one group. The large groups remain unchanged. In this way, we construct groups $P_1, \cdots, P_k, P_{k+1}$, where $P_1, \cdots, P_k$ are exactly the large groups in $P'_1, \cdots, P'_{k'}, P'_{k'+1}$ and P_{k+1} is the group merged from small groups P'_i with index $k'' < i \leq k' + 1 = k'' + 5$.

By construction, $|P_j| \geq \gamma^2$ for $j \in [k]$ and $|P_{k+1}| < 5\gamma^2$. Finally, note that B_m will never be discarded and the total capacity of the discarded bins is at most

$$\gamma^2 \sum_{i=1}^{k''} \gamma^i B_1 = \frac{\gamma^3(\gamma^{k''} - 1)}{\gamma - 1} B_1 \leq \gamma^{k''+3} B_1 \leq \epsilon \gamma^{k''+4} B_1 \leq \epsilon B_m.$$

Thus, $f(OPT^*) \geq (1-\epsilon) f(OPT)$. □

To analyze Algorithm 6, we first give an alternative analysis for Constant-bins-by-value in Lemma 9. This will give us a similar inequality as in Lemma 7. Recall that we use E to denote an enumerated feasible solution and G'_E the set returned by the greedy algorithm which begins with E. Note that G'_E contains the reserved elements.

Lemma 9. *Assume we run the procedure* Constant-bins-by-value *in Algorithm 2 with parameters $f(\cdot)$, m bins and $\delta = 1/(2m)$. For any $X \subseteq N$, there is a set E in the enumeration step such that $f(G'_E \mid E) \geq f(X \mid E \cup G'_E)$.*

Algorithm 6: The General Case, 1/2-Ratio

Input: constant $\epsilon \in (0,1)$, ground set N, objective function f, size function c, the number of bins m, capacities $(B_1, \cdots, B_m)$.

Output: A feasible set (and the way it is packed).

Divide bins into groups $P_1 \cdots, P_k, P_{k+1}$ as in Lemma 8.
Initialize $T = \emptyset$.
for $i = 1$ *to* k **do**
  $S = \textsc{Bounded}(\epsilon, N \setminus T, f(\cdot \mid T), c(\cdot), |P_i|, P_i)$.
  $T = T \cup S$.
end
$\mathcal{E} = \textsc{Constant-bins-by-value}(N \setminus T, f(\cdot \mid T), c(\cdot), |P_{k+1}|, P_{k+1}, 1/(2|P_{k+1}|))$.
return $\arg\max_{S \in \mathcal{E}} f(T \cup S)$ (and the way it is packed).

Proof. Assume w.l.o.g. that $|X| \geq 2m$, since otherwise X will be enumerated in the enumeration step and $E = X$ satisfies the inequality. We order elements in X greedily according to their marginal values, i.e. $x_1 = \arg\max_{x \in X} f(x)$, $x_2 = \arg\max_{x \in X \setminus \{x_1\}} f(x \mid x_1)$, etc. In the enumeration step, the solution $E = (E_1, \cdots, E_m)$ must be visited such that E contains exactly the first m elements in X and these elements are packed in the same way as in X. In the following analysis, we show that E and G'_E satisfies the desired inequality.

By a similar argument as in Theorem 6, $X \setminus E$ is a feasible (indeed optimal) solution while the greedy algorithm begins with the (partial) solution $E = (E_1, \cdots, E_m)$. If $c(G'_E) < c(X \setminus E)$, this means the greedy algorithm packs all elements in $N \setminus E$ and therefore $X \setminus E \subseteq G'_E$. Then, $f(G'_E \mid E) \geq f(X \setminus E \mid E) = f(X \mid E) \geq f(X \mid E \cup G'_E)$. Thus, in the following we assume that $c(G'_E) \geq c(X \setminus E)$. Assume that $G'_E = \{u_1, u_2, \cdots, u_l\}$, and for $i \in [l]$, $G_i = \{u_1, u_2, \cdots u_i\}$ denotes the first i elements picked by the greedy algorithm. By the greedy rule, for any $x \in X \setminus E$,

$$\frac{f(u_i \mid E \cup G_{i-1})}{c(u_i)} \geq \frac{f(x \mid E \cup G_{i-1})}{c(x)}.$$

By submodularity,

$$\frac{f(u_i \mid E \cup G_{i-1})}{c(u_i)} \geq \frac{f(X \setminus E \mid E \cup G_{i-1})}{c(X \setminus E)} \geq \frac{f(X \setminus E \mid E \cup G'_E)}{c(X \setminus E)}.$$

This gives us

$$f(G'_E \mid E) \geq c(G'_E) \cdot \frac{f(X \mid E \cup G'_E)}{c(X \setminus E)} \geq f(X \mid E \cup G'_E).$$

□

We now show that Algorithm 6 is a 1/2 approximation algorithm.

Theorem 9. *Algorithm 6 achieves a* $(1 - 6\epsilon)/2$ *approximation ratio and runs in xxx time.*

Proof. Define C and $P_1, \cdots, P_k, P_{k+1}$ as in Lemma 8. By construction, for $i \in [k]$, P_i is γ-bounded and there are at least $\gamma^2 = \gamma \ln(1/\epsilon)/(2\epsilon^2)$ bins in P_i. Thus, it is legal to use Algorithm 5 (Bounded) to pack bins in P_i. And Algorithm 5 defines working bins and patched bins in each P_i.

Let OPT be an optimal solution of the original instance. Let OPT^* be an optimal solution when one can only use bins in $P_1, \cdots, P_k, P_{k+1}$ to pack elements. Let X^* be an optimal solution when one can only use working bins in $P_1, \cdots, P_k$ and bins in P_{k+1} to pack elements. By Lemma

8, $f(OPT^*) \geq (1-\epsilon)f(OPT)$. Besides, since each P_i contains at least $(1-4\epsilon)|P_i|$ working bins with the largest capacities, we have $f(X^*) \geq (1-4\epsilon)f(OPT^*)$.

Assume there are l working bins and patched bins in total in $P_1, P_2, \cdots, P_k$ and they are sorted in the order they are processed. For $j \in [l]$, let S_j be the set of elements packed into the j-th (working or patched) bin by the algorithm and T_j be the set of elements packed into the first j bins. Then, $T_j = T_{j-1} \cup S_j$.

By the definition of X^*, there is a partition $(X^*_{P_1}, \cdots, X^*_{P_k}, X^*_{P_{k+1}})$ such that $X^*_{P_i}$ is a feasible solution over working bins in P_i for $i \in [k]$, and $X^*_{P_{k+1}}$ is a feasible solution over bins in P_{k+1}. Therefore, for each $X^*_{P_i}$, we can re-partition it into working bins and patched bins in P_i as in Lemma 7 and finally obtain a partition $(X^*_1, \cdots, X^*_l, X^*_{l+1})$ of X^*, where X^*_{l+1} is exactly the same as $X^*_{P_{k+1}}$. Besides, by Lemma 7, for $j \in [l]$,

$$f(S_j \mid T_{j-1}) \geq (1-\epsilon)f(X^*_j \mid T_{j-1}).$$

Now we consider the bins in P_{k+1}. For each E in the enumeration step of Constant-bins-by-value, let G'_E be the set returned by the greedy algorithm which begins with $T_l \cup E$. Let R_E be the reserved elements in G'_E. Define $T'_E = T_l \cup E \cup G'_E$ and $T_E = T'_E \setminus R_E$. By Lemma 9, there is a set E such that $f(G'_E \mid T_l \cup E) \geq f(X \mid T_l \cup E \cup G'_E)$. Adding $f(E \mid T_l)$ into both sides, we obtain $f(E \cup G'_E \mid T_l) \geq f(X \mid T_l \cup E \cup G'_E) + f(E \mid T_l)$. By submodularity, $f(S_j \mid T_{j-1}) \geq (1-\epsilon)f(X^*_j \mid T'_E)$ for $j \in [l]$. Summing up these $l+1$ inequalities, again by submodularity,

$$f(T'_E) \geq (1-\epsilon)f(X^* \mid T'_E) + f(E \mid T_l).$$

Therefore,

$$f(T'_E) \geq \frac{1-\epsilon}{2} f(X^*) + \frac{f(E \mid T_l)}{2}.$$

Since we enumerate all feasible E with $|E| \leq 2|P_{k+1}|$, by a similar argument as in Theorem 6, $f(u \mid T_l \cup E) \leq f(E \mid T_l)/(2|P_{k+1}|)$ for any $u \in R_E$. Hence, by submodularity,

$$\begin{aligned}
f(T_E) &= f(T'_E) - f(R_E \mid T_E) \\
&\geq \frac{1-\epsilon}{2} f(X^*) + \frac{f(E \mid T_l)}{2} - f(R_E \mid T_l \cup E) \\
&\geq \frac{1-\epsilon}{2} f(X^*) + \frac{f(E \mid T_l)}{2} - \sum_{u \in R_E} f(u \mid T_l \cup E) \\
&\geq \frac{1-\epsilon}{2} f(X^*) + \frac{f(E \mid T_l)}{2} - |P_{k+1}| \frac{f(E \mid T_l)}{2|P_{k+1}|} \\
&= \frac{1-\epsilon}{2} f(X^*).
\end{aligned}$$

Let $S_{l+1} = \arg\max_{S \in \mathcal{E}} f(T_l \cup S)$ and $T_{l+1} = T_l \cup S_{l+1}$. We have

$$f(T_{l+1}) \geq f(T_E) \geq \frac{1-\epsilon}{2} f(X^*) \geq \frac{(1-\epsilon)^2(1-4\epsilon)}{2} f(OPT) \geq \frac{1-6\epsilon}{2} f(OPT).$$

□

5 Conclusion

In this paper, we present a $1 - 1/e - \epsilon$ deterministic algorithm for the identical case of SMKP and a $1/2 - \epsilon$ deterministic algorithm for the general case. We further show there is a $1 - 1/e - \epsilon$ randomized algorithm for the general case.

There remains some interesting open problems. For the identical case, our algorithm packs a constant number of bins in each iteration, leading to the high complexity of the algorithm. It is interesting to investigate whether the algorithm has the same performance when it packs bins one at a time. For the general case, though there is a tight $1 - 1/e - \epsilon$ randomized algorithm, the deterministic algorithm can only achieve a $1/2$ ratio. It is appealing to find deterministic algorithms which can break the $1/2$ barrier and finally achieve the optimal $1 - 1/e$ approximation ratio. Finally, SMKP is closely related to submodular maximization under m-knapsack constraint. The result of [8] is based on reducing SMKP to a special case of the latter problem. Besides, most of our algorithms for SMKP are deterministic. Thus, we believe our results may inspire the design of deterministic algorithms for submodular maximization under m-knapsack constraint.

References

[1] Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1497–1514, 2014.

[2] Niv Buchbinder and Moran Feldman. Deterministic algorithms for submodular maximization problems. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 392–403, 2016.

[3] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 649–658, 2012.

[4] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1433–1452, 2014.

[5] Chandra Chekuri and Sanjeev Khanna. A PTAS for the multiple knapsack problem. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA.*, pages 213–222, 2000.

[6] Alina Ene and Huy L. Nguyen. Constrained submodular maximization: Beyond 1/e. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 248–257, 2016.

[7] Alina Ene and Huy L. Nguyen. A nearly-linear time algorithm for submodular maximization with a knapsack constraint. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece.*, pages 53:1–53:12, 2019.

[8] Yaron Fairstein, Ariel Kulik, Joseph Naor, Danny Raz, and Hadas Shachnai. A (1-e^{-1}-ϵ)-approximation for the monotone submodular multiple knapsack problem. *CoRR*, abs/2004.12224, 2020.

[9] Moran Feldman. *Maximization problems with submodular objective functions.* Technion-Israel Institute of Technology, Faculty of Computer Science, 2013.

[10] Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 570–579, 2011.

[11] Yuval Filmus and Justin Ward. A tight combinatorial algorithm for submodular maximization subject to a matroid constraint. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 659–668, 2012.

[12] Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. An analysis of approximations for maximizing submodular set functions - II. In *Polyhedral combinatorics*, pages 73–87. Springer, 1978.

[13] Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1098–1116, 2011.

[14] Klaus Jansen. Parameterized approximation scheme for the multiple knapsack problem. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 665–674, 2009.

[15] Hans Kellerer. A polynomial time approximation scheme for the multiple knapsack problem. In *Randomization, Approximation, and Combinatorial Algorithms and Techniques, RANDOM-APPROX'99, Berkeley, CA, USA, August 8-11, 1999, Proceedings*, pages 51–62, 1999.

[16] Ariel Kulik, Hadas Shachnai, and Tami Tamir. Maximizing submodular set functions subject to multiple linear constraints. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 545–554, 2009.

[17] Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 323–332, 2009.

[18] Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Math. Oper. Res.*, 35(4):795–806, 2010.

[19] George L. Nemhauser and Laurence A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Res.*, 3(3):177–188, 1978.

[20] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978.

[21] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004.

[22] Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 67–74, 2008.

[23] Jan Vondrák, Chandra Chekuri, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. In *Proceedings of the 43rd ACM*

Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011, pages 783–792, 2011.