Journal of Artificial Intelligence Research 82 (2025) 39-75 Submitted 07/2024; published 01/2025

Practical Parallel Algorithms for Non-Monotone Submodular Maximization

Shuang Cui SCUI@SUDA.EDU.CN
School of Computer Science and Technology,
Soochow University,
Suzhou, Jiangsu, China

Kai Han (Corresponding Author) HANKAI@SUDA.EDU.CN
School of Computer Science and Technology,
Soochow University,
Suzhou, Jiangsu, China

Jing Tang JINGTANG@UST.HK
The Hong Kong University of Science and Technology (Guangzhou),
The Hong Kong University of Science and Technology,
China

Xueying Li XIAOMING.LXY@ALIBABA-INC.COM
Aakas Zhiyuli AAKAS.LZY@ALIBABA-INC.COM
Alibaba Group,
Hangzhou, Zhejiang, China

Hanxiao Li 2023060204@MAILS.QUST.EDU.CN
Qingdao University of Science and Technology,
Qingdao, Shandong, China

Abstract

Submodular maximization has found extensive applications in various domains within the field of artificial intelligence, including but not limited to machine learning, computer vision, and natural language processing. With the increasing size of datasets in these domains, there is a pressing need to develop efficient and parallelizable algorithms for submodular maximization. One measure of the parallelizability of a submodular maximization algorithm is its adaptive complexity, which indicates the number of sequential rounds where a polynomial number of queries to the objective function can be executed in parallel. In this paper, we study the problem of non-monotone submodular maximization subject to a knapsack constraint, and propose a low-adaptivity algorithm achieving an $(1/8 - \epsilon)$-approximation with practical $\tilde{\mathcal{O}}(n)$ query complexity. Moreover, we also propose the first algorithm with both provable approximation ratio and sublinear adaptive complexity for the problem of non-monotone submodular maximization subject to a k-system constraint. As a by-product, we show that our two algorithms can also be applied to the special case of submodular maximization subject to a cardinality constraint, and achieve performance bounds comparable with those of state-of-the-art algorithms. Finally, the effectiveness of our algorithms is demonstrated by extensive experiments on real-world applications.

1. Introduction

Submodular maximization algorithms have played a critical role in advancing the field of artificial intelligence, particularly in the areas of machine learning (e.g., non-parametric

learning (Qian et al., 2019; Kulesza & Taskar, 2012; Lawrence et al., 2002), active learning (Golovin & Krause, 2010; Wei et al., 2015; Golovin & Krause, 2011), data summarization (Balkanski et al., 2018; Mirzasoleiman et al., 2018; Mitrovic et al., 2018; Amanatidis et al., 2022)), computer vision (e.g., object detection (Angelova & Zhu, 2013; Zhu et al., 2014), image segmentation (Kim et al., 2011)), and natural language processing (e.g., text classification (Lei et al., 2019), document summarization (Lin & Bilmes, 2012; Kulesza & Taskar, 2012)). As a result, submodular maximization has been widely studied under various constraints such as cardinality, knapsack, matroid, and k-system constraints. Many algorithms in this area adopt the greedy search strategy (e.g., continuous greedy algorithms in (Calinescu et al., 2011)), but may have large *query complexity* to achieve a good approximation ratio, where query complexity refers to the number of evaluations to the objective function. In practice, evaluating the objective function may be time-consuming (Dueck & Frey, 2007; Das & Kempe, 2008; Kazemi et al., 2018), and this situation is further exacerbated by the proliferation of "big data", for which simply reducing query complexity is often insufficient to get efficient algorithms. Thus, parallelization has received increased attention for submodular maximization.

Unfortunately, traditional greedy algorithms for submodular maximization are inherently sequential and adaptive, which makes them unsuitable for being parallelized. Some efforts have been devoted to designing distributed submodular maximization algorithms using parallel models such as MapReduce (Mirzasoleiman et al., 2013; Kumar et al., 2013; Barbosa et al., 2015; Mirzasoleiman et al., 2016; Epasto et al., 2017; Kazemi et al., 2021), but these algorithms can still be highly adaptive, as they usually run sequential greedy algorithms on each of the machines. Recently, (Balkanski & Singer, 2018) proposed submodular maximization algorithms with low *adaptive complexity* (a.k.a. "adaptivity"), where only a sub-linear number of *adaptive rounds* are incurred and polynomially-many queries can be executed in parallel in each adaptive round. Subsequently, a lot of studies have appeared to design low-adaptivity algorithms; many of them concentrate on the submodular maximization with a cardinality constraint (**SMC**) problem (e.g., (Kazemi et al., 2019; Fahrbach et al., 2019b; Balkanski et al., 2019a)).

Besides the SMC, one of the most fundamental problems in submodular optimizations is the problem of submodular maximization subject to a knapsack constraint (**SKP**), which has many applications both for monotone and non-monotone submodular functions (Kulik et al., 2009; Lee et al., 2010; Badanidiyuru & Vondrák, 2014). Surprisingly, although the SKP has been extensively studied since the 1980s (Wolsey, 1982), there exist only few studies on designing low-adaptivity algorithms for it. In particular, (Chekuri & Quanrud, 2019b) provides a $(1 - 1/e - \epsilon)$-approximation in $\mathcal{O}(\log n)$ adaptive rounds for monotone SKP, while (Ene et al., 2019) presents a $(e - \epsilon)$-approximation in $\mathcal{O}(\log^2 n)$ adaptive rounds for non-monotone SKP. However, both (Chekuri & Quanrud, 2019b) and (Ene et al., 2019) assume oracle access to the multilinear extension of a submodular function and its gradient, which incurs high query complexity for estimating multilinear extensions accurately, making their algorithms impractical in real applications (Amanatidis et al., 2021). Very recently, (Amanatidis et al., 2021) study the non-monotone SKP problem and present a parallelizable algorithm dubbed ParKnapsack with $\tilde{\mathcal{O}}(n)$[1] query complexity and an approximation ratio

1. Throughout the paper, we use the $\tilde{O}$ notation to suppress poly-logarithmic factors.

of $\frac{-3\epsilon(\epsilon(8\epsilon-2\sqrt{3}-13)+4\sqrt{3}+2)+4\sqrt{3}-6}{3((2\sqrt{3}-3)\epsilon^3+(11-6\sqrt{3})\epsilon+4(\sqrt{3}-1))} = (3-\sqrt{3})/12 - \Theta(\epsilon) \approx 0.106 - \Theta(\epsilon)$. Unfortunately, as pointed out by our conference version of this paper (Cui et al., 2023a), the performance analysis of (Amanatidis et al., 2021) contains critical errors and hence their performance bounds do not hold. Thus, our work is the first to provide a low-adaptivity algorithm that can achieve a provable approximation ratio with practical query complexity for the non-monotone SKP. After the publication of our conference version (Cui et al., 2023a, 2023b), (Amanatidis et al., 2023) (which is the full version of (Amanatidis et al., 2021)) provided a revised algorithm achieving the same $0.106-\Theta(\epsilon)$ ratio as that in (Amanatidis et al., 2021), but this ratio is still worse than the $0.125-\epsilon$ ratio provided by this paper.

Another fundamental problem in submodular optimizations is the problem of submodular maximization subject to a k-system constraint (**SSP**), as k-system constraints capture a wide variety of constraints, including matroid constraints, intersection of matroids, spanning trees, graph matchings, scheduling, and even planar subgraphs. Since the 1970s, significant efforts have been devoted to solving the SSP. The state-of-the-art approximation ratios are $1/(k+1)$ (Fisher et al., 1978) and $(1+\epsilon)^{-1}(\sqrt{k}+1)^{-2}$ (Cui et al., 2021) for monotone submodular functions and non-monotone submodular functions, respectively. Recently, (Quinzan et al., 2021) proposes the first low-adaptivity algorithm for the non-monotone SSP. Their algorithm can achieve an approximation ratio of $(1+\epsilon)^{-1}(1-\epsilon)^2(k+2\sqrt{2(k+1)}+5)^{-1}$ under the adaptive complexity of $\mathcal{O}(\sqrt{k}\log^2 n \log\frac{n}{k})$ and the query complexity of $\tilde{\mathcal{O}}(\sqrt{k}n)$. Regrettably, as demonstrated in Appendix B, their analysis contains serious errors that invalidated their algorithm's approximation ratio. Therefore, it remains an open question whether there exists a low-adaptivity algorithm that can achieve a provable approximation ratio for the non-monotone SSP.

Contributions. In this paper, we present two practical low-adaptivity algorithms named ParSKP and ParSSP that provide confirmative answers to the open problem mentioned above and significantly improve the existing results. Our contributions include:

- For the non-monotone SKP, we propose ParSKP algorithm that uses $\mathcal{O}(\log^2 n)$ adaptivity and practical $\tilde{\mathcal{O}}(n)$ query complexity (or uses near-optimal $\mathcal{O}(\log n)$ adaptivity and $\tilde{\mathcal{O}}(n^2)$ query complexity) to achieve an approximation ratio of $1/8-\epsilon$ which is best among the existing practical parallel algorithms for the SKP problem.

- For the non-monotone SSP, we propose ParSSP algorithm that uses $\mathcal{O}(\sqrt{k}\log^3 n)$ adaptivity and practical $\tilde{\mathcal{O}}(\sqrt{k}n)$ query complexity (or uses $\mathcal{O}(\sqrt{k}\log^2 n)$ adaptivity and $\tilde{\mathcal{O}}(\sqrt{k}n^2)$ query complexity) to achieve an approximation ratio of $(1-\epsilon)^5(\sqrt{k+1}+1)^{-2}$. To the best of our knowledge, our ParSSP is the first low-adaptivity algorithm that can achieve a provable approximation ratio for the non-monotone SSP.

- As a by-product, our two algorithms can be directly used to address the non-monotone SMC, where the approximation ratio of ParSSP can be tightened to $1/4-\epsilon$ under the same complexities listed above. That is to say, our ParSSP algorithm achieves the best approximation ratio among existing practical low-adaptivity algorithms for the SMC problem.

- We conduct extensive experiments using several applications including revenue maximization, movie recommendation and image summarization. The experimental results

demonstrate that our algorithms can achieve comparable utility using significantly fewer adaptive rounds than existing non-parallel algorithms, while also achieving significantly better utility than existing low-adaptivity algorithms.

Challenges and Techniques. Naively, it seems that slightly adjusting the low-adaptivity algorithms for the monotone submodular maximization problems can address their non-monotone variants. However, we find that the techniques developed under the monotone setting heavily rely on monotonicity, which may incur subtle issues when applying to the non-monotone scenarios. For instance, as pointed out in Appendix B and (Chen & Kuhnle, 2022), the approximation claims in some recent work (Fahrbach et al., 2019a; Quinzan et al., 2021) are flawed by the abuse of existing techniques developed for monotone functions. Moreover, (Amanatidis et al., 2021) proposes the ParKnapsack algorithm sophisticatedly tailored to non-montone functions, which again gives flawed analysis on approximation guarantees as pointed in our conference version. As can be seen, designing approximation parallel algorithms for non-monotone SKP/SSP achieving both low-adaptivity and low-complexity is challenging. To overcome this challenge, our ParSKP algorithm adopts a novel method using a more sophisticated filtering procedure dubbed Probe where multiple solutions are created in different ways given an ideal threshold, which can provide theoretical guarantees correctly. Moreover, our ParSSP algorithm adopts another method where there is no need to guess an ideal threshold, but we use decreasing thresholds and introduce a "random batch selection" operation to bypass the difficulties caused by non-monotonicity.

This manuscript provides an expanded and refined presentation of the research that was initially introduced in a preliminary conference paper at AAAI-2023 (Cui et al., 2023a). The main revisions can be summarized as follows:

- We present the ParSSP algorithm (Section 4.3), which is a modification of an algorithm previously introduced in the conference version. Our ParSSP algorithm is the first to achieve both a provable approximation ratio and sublinear adaptive complexity for the non-monotone SSP. Moreover, we perform a set of comparative experiments to validate the performance of our ParSSP algorithm for this problem (Section 6.4).

- We have added a set of comparative experiments on a small dataset to compare our ParSKP algorithm with a parallel algorithm that has a theoretically optimal approximation ratio but impractical query complexity (Section 6.3). Additionally, we also add experiments for the SMC problem (Section 6.5). These experiments further demonstrate the superiority of our algorithm in terms of efficiency and effectiveness.

- Due to page limitations, the complete theoretical analysis of our algorithms and the discussions of theoretical analysis errors in (Quinzan et al., 2021; Amanatidis et al., 2021) were not included in the conference version. However, we provide all of the complete theoretical analysis (Sections 4-5) and discussion mentioned above in this journal version (Appendix A-B).

- We have restructured all sections to provide a clearer and more comprehensive explanation of our research objectives and findings.

The rest of our paper is organized as follows. The related studies are reviewed in Section 2 and our problem definitions are presented in Section 3. Our proposed approximation

Table 1: Low-adaptivity algorithms for non-monotone submodular maximization.

Constraint	Reference	Ratio	Adaptivity	Queries
Knapsack	(Ene et al., 2019)	$\boldsymbol{1/e-\epsilon}$	$\mathcal{O}(\log^2 n)$	$\tilde{\mathcal{O}}(n^3)$
	(Amanatidis et al., 2023)	$(3-\sqrt{3})/12-\Theta(\epsilon)$	$\boldsymbol{\mathcal{O}(\log n)}$ || $\mathcal{O}(\log^2 n)$	$\tilde{\mathcal{O}}(n^2)$ || $\boldsymbol{\tilde{\mathcal{O}}(n)}$
	ParSKP(Alg. 3)	$1/8-\epsilon$	$\boldsymbol{\mathcal{O}(\log n)}$ || $\mathcal{O}(\log^2 n)$	$\tilde{\mathcal{O}}(n^2)$ || $\boldsymbol{\tilde{\mathcal{O}}(n)}$
k-System	(Quinzan et al., 2021)	$(1+\epsilon)^{-1}(1-\epsilon)^2(k+2\sqrt{2(k+1)}+5)^{-1}$ [#]	$\mathcal{O}(\sqrt{k}\log^2 n\log\frac{n}{k})$	$\boldsymbol{\tilde{\mathcal{O}}(\sqrt{k}n)}$
	ParSSP(Alg. 5)	$\boldsymbol{(1-\epsilon)^5(\sqrt{k+1}+1)^{-2}}$	$\boldsymbol{\mathcal{O}(\sqrt{k}\log^2 n)}$ || $\mathcal{O}(\sqrt{k}\log^3 n)$	$\mathcal{O}(\sqrt{k}n^2)$ || $\boldsymbol{\tilde{\mathcal{O}}(\sqrt{k}n)}$
Cardinality	(Chekuri & Quanrud, 2019a)	$3-2\sqrt{2}-\epsilon$	$\mathcal{O}(\log^2 n)$	$\tilde{\mathcal{O}}(nr^4)$
	(Balkanski et al., 2018)	$1/2e-\epsilon$	$\mathcal{O}(\log^2 n)$	$\tilde{\mathcal{O}}(nr^2)$
	(Ene & Nguyen, 2020)	$\boldsymbol{1/e-\epsilon}$	$\boldsymbol{\mathcal{O}(\log n)}$	$\tilde{\mathcal{O}}(nr^2)$
	(Chen & Kuhnle, 2022)	$1/6-\epsilon$ || $0.193-\epsilon$	$\boldsymbol{\mathcal{O}(\log n)}$ || $\mathcal{O}(\log n\log r)$	$\boldsymbol{\tilde{\mathcal{O}}(n)}$
	ParSKP(Alg. 3)	$1/8-\epsilon$	$\boldsymbol{\mathcal{O}(\log n)}$ || $\mathcal{O}(\log n\log r)$	$\tilde{\mathcal{O}}(nr)$ || $\boldsymbol{\tilde{\mathcal{O}}(n)}$
	ParSSP(Alg. 5)	$1/4-\epsilon$	$\mathcal{O}(\log^2 n)$ || $\mathcal{O}(\log^2 n\log r)$	$\tilde{\mathcal{O}}(nr)$ || $\boldsymbol{\tilde{\mathcal{O}}(n)}$

[1] Bold font indicates the best result(s) in each setting, r is the largest cardinality of any feasible solution.
[#] The approximation ratio is derived from flawed analysis as explained in Appendix B.

algorithms and their theoretical analysis are introduced in Section 4. In Section 5, we analyze the performance bounds of our algorithms when applied to the SMC. We present the experimental results in Section 6 before concluding our work in Section 7. Discussion on the theoretical analysis error in relevant literature has been relegated to Appendix A-B.

2. Related Work

In the following, we review several lines of related studies, and list the performance bounds of some representative ones in Table 1.

2.1 Algorithms for the SKP

The traditional SKP has been extensively studied, both for monotone and non-monotone submodular functions (e.g., (Sviridenko, 2004; Kulik et al., 2013; Gupta et al., 2010; Ene & Nguyen, 2019a; Yaroslavtsev et al., 2020)). For non-monotone SKP, (Buchbinder & Feldman, 2019) achieves the best-known approximation ratio of 0.385, while some other studies (Mirzasoleiman et al., 2016; Amanatidis et al., 2022; Han et al., 2021; Cui et al., 2024) provide faster algorithms with weaker approximation ratios. However, all these algorithms have super-linear adaptive complexity unsuitable for parallelization. In terms of low-adaptivity algorithms for SKP, (Chekuri & Quanrud, 2019b) provides a $(1-1/e-\epsilon)$-approximation in $\mathcal{O}(\log n)$ adaptive rounds for monotone SKP. For the non-monotone SKP, (Ene et al., 2019) provides an $(1/e-\epsilon)$-approximation with $\mathcal{O}(\log^2 n)$ adaptivity, based on a continuous optimization approach using multi-linear extension. However, an excessively large and impractical number of $\Omega(nr^2\log^2 n)$ function valuations are needed in (Ene et al., 2019) for simulating a query to the multilinear extension of a submodular function or its gradient with sufficient accuracy, as described in (Fahrbach et al., 2019a). Observing this, (Amanatidis et al., 2021) provides a combinatorial algorithm that uses $\mathcal{O}(\log^2 n)$ adaptivity and practical $\tilde{\mathcal{O}}(n)$ query complexity to achieve an approximation ratio of $\frac{-3\epsilon(\epsilon(8\epsilon-2\sqrt{3}-13)+4\sqrt{3}+2)+4\sqrt{3}-6}{3((2\sqrt{3}-3)\epsilon^3+(11-6\sqrt{3})\epsilon+4(\sqrt{3}-1))}=(3-\sqrt{3})/12-\Theta(\epsilon)\approx 0.106-\Theta(\epsilon)$. Unfortunately, their approximation ratio is derived from flawed analysis as explained in our conference

version (Cui et al., 2023a). The work of (Amanatidis et al., 2023) (which is the full version of (Amanatidis et al., 2021)) has adopted a more complex sampling procedure to achieve an approximation ratio of $0.106 - \Theta(\epsilon)$, but this ratio is still worse than the $0.125 - \epsilon$ ratio provided by this paper.

2.2 Algorithms for the SSP

Since the 1970s, the SSP has been a topic of widespread interest among the academic community. Research related to the SSP has included (Fisher et al., 1978; Gupta et al., 2010; Calinescu et al., 2011; Mirzasoleiman et al., 2016; Feldman et al., 2017, 2023; Cui et al., 2021). Among the existing proposals, (Cui et al., 2021) achieves the best-known approximation ratio of $(1+\epsilon)^{-1}(k+2\sqrt{k}+1)^{-1}$ under $\mathcal{O}(\frac{n}{\epsilon}\log\frac{r}{\epsilon})$ time complexity. However, all the studies mentioned above only provide algorithms with super-linear adaptive complexity unsuitable for parallelization. In terms of parallelizable algorithms, to the best of our knowledge, only (Quinzan et al., 2021) has proposed a low-adaptivity algorithm for the SSP that achieves an approximation ratio of $(1+\epsilon)^{-1}(1-\epsilon)^2(k+2\sqrt{2(k+1)}+5)^{-1}$ under the adaptive complexity of $\mathcal{O}(\sqrt{k}\log^2 n\log\frac{n}{k})$ and the query complexity of $\tilde{\mathcal{O}}(\sqrt{k}n)$. Unfortunately, their approximation ratio is derived from flawed analysis as pointed in our conference version.

2.3 Algorithms for the SMC

For the monotone SMC, several parallelizable algorithms have been proposed such as (Ene & Nguyen, 2019b; Balkanski et al., 2019b; Chekuri & Quanrud, 2019a; Breuer et al., 2020; Chen et al., 2021). For the non-monotone SMC, (Chekuri & Quanrud, 2019a) and (Balkanski et al., 2018) propose parallelizable algorithms with $3-2\sqrt{2}-\epsilon$ and $1/(2e)-\epsilon$ approximation ratios, respectively, both under $\mathcal{O}(\log^2 n)$ adaptivity, while (Ene & Nguyen, 2020) achieve an improved ratio of $1/e-\epsilon$ under $\mathcal{O}(\log n)$ adaptivity. However, all these studies use multilinear extensions and have high query complexity (larger than $\Omega(nr^2)$). Another study in this line (Fahrbach et al., 2019a) aims to reduce both adaptivity and query complexity, but achieving a relatively large ratio of $0.039-\epsilon$. Subsequently, (Kuhnle, 2021) claims a $(0.193-\epsilon)$-approximation with $\mathcal{O}(\log^2 n)$ adaptivity. However, (Chen & Kuhnle, 2022) identifies non-trivial errors in both (Fahrbach et al., 2019a) and (Kuhnle, 2021), and propose a new adaptive algorithm to fix the $(0.193-\epsilon)$-approximation. Despite these efforts, our ParSSP algorithm still achieves a superior $(1/4-\epsilon)$-approximation for the non-monotone SMC.

3. Preliminaries

We provide the some basic definitions in the following:

Definition 1 (Submodular Function, defined by (Fisher et al., 1978))**.** *Given a finite ground set $\mathcal{N}$ with $|\mathcal{N}| = n$, a set function $f\colon 2^{\mathcal{N}} \mapsto \mathbb{R}$ is submodular if for all sets $X, Y \subseteq \mathcal{N}\colon f(X)+f(Y) \geq f(X\cup Y)+f(X\cap Y)$.*

In this paper, we allow $f(\cdot)$ to be non-monotone, i.e., $\exists X \subseteq Y \subseteq \mathcal{N}\colon f(X) > f(Y)$, and we assume that $f(\cdot)$ is non-negative, i.e., $f(X) \geq 0$ for all $X \subseteq \mathcal{N}$.

Definition 2 (Independence System). *Given a finite ground set $\mathcal{N}$ and a collection of sets $\mathcal{I} \subseteq 2^{\mathcal{N}}$, the pair $(\mathcal{N}, \mathcal{I})$ is called an independence system if it satisfies: (1) $\emptyset \in \mathcal{I}$; (2) if $X \subseteq Y \subseteq \mathcal{N}$ and $Y \in \mathcal{I}$, then $X \in \mathcal{I}$.*

Given an independence system $(\mathcal{N}, \mathcal{I})$ and any two sets $X \subseteq Y \subseteq \mathcal{N}$, X is called a *base* of Y if $X \in \mathcal{I}$ and $X \cup \{u\} \notin \mathcal{I}$ for all $u \in Y \setminus X$. A k-system is a special independence system defined as:

Definition 3 (k-system). *An independence system $(\mathcal{N}, \mathcal{I})$ is called a k-system ($k \geq 1$) if $|X_1| \leq k|X_2|$ holds for any two bases X_1 and X_2 of any set $Y \subseteq \mathcal{N}$.*

We assume that each element $u \in \mathcal{N}$ has a cost $c(u) > 0$ and there is a budget $B > 0$. Without loss of generality, we also assume $\forall u \in \mathcal{N}\colon c(u) \leq B$. Then a knapsack constraint can be modeled as an independence system as follows:

Definition 4 (Knapsack Constraint). *An independence system $(\mathcal{N}, \mathcal{I})$ capturing a knapsack constraint is defined as the collection of sets $X \subseteq \mathcal{N}$ obeying $c(X) \leq B$ for some non-negative modular function $c(X) = \sum_{u \in X} c(u)$.*

Based on the above definitions, the SKP, SMC and SSP can be written as:

- SKP: $\max\{f(S)\colon S \in \mathcal{I}\}$, where $(\mathcal{N}, \mathcal{I})$ is a knapsack constraint
- SMC: $\max\{f(S)\colon S \in \mathcal{I}\}$, where $(\mathcal{N}, \mathcal{I})$ is a knapsack constraint and $\forall u \in \mathcal{N} : c(u) = 1$
- SSP: $\max\{f(S)\colon S \in \mathcal{I}\}$, where $(\mathcal{N}, \mathcal{I})$ is a k-system and $\forall u \in \mathcal{N} : c(u) = 1$

For convenience, we denote by $\mathrm{OPT} = f(O)$ the optimal value of the objective function for the SKP/SSP/SMC, where O is an optimal solution, and use w to denote an element with maximum cost in O. Moreover, we let r denote the maximum cardinality of any feasible solution to the same problem. For any $u \in \mathcal{N}$ and any $X \subseteq \mathcal{N}$, we use $f_X(\cdot)$ to denote the function defined as $f_X(Y) = f(X \cup Y)$ for any $Y \subseteq \mathcal{N}$; and we use $f(u \mid X)$ to denote the "marginal gain" of u with respect to X, i.e., $f(u \mid X) = f(X \cup \{u\}) - f(X)$; we also call $f(u \mid X)/c(u)$ as the "marginal density" of u with respect to X.

Suppose that $f(S)$ can be returned by an oracle query for any given $S \subseteq \mathcal{N}$, the *query complexity* of any algorithm ALG denotes the number of oracle queries to $f(\cdot)$ incurred in ALG, and its *adaptive complexity* denotes the number of *adaptive rounds* of ALG, where $\mathcal{O}(\mathrm{poly}(n))$ oracle queries are allowed in each adaptive round, but all these queries can only depend on the results of previous adaptive rounds. We assume that there exists an algorithm $\mathsf{USM}(X)$ addressing the *unconstrained submodular maximization (USM)* problem of $\max\{f(Y)\colon Y \subseteq X\}$ for any $X \subseteq \mathcal{N}$, and assume that it achieves the following performance bounds:

Theorem 1 (Theorem A.1 in the full version of (Chen et al., 2019)). *For every constant $\epsilon > 0$, there is an algorithm without using multi-linear extension that achieves a $(1/2 - \epsilon)$-approximation for the unconstrained submodular maximization problem using $\mathcal{O}(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ adaptive rounds with $\mathcal{O}(\frac{n}{\epsilon^4} \log^3 \frac{1}{\epsilon})$ query complexity.*

4. Approximation Algorithms

In this section, we propose our ParSKP and ParSSP algorithms. Both of them call a submodule RandBatch based on the "adaptive sequencing" technique, which was originally proposed by (Balkanski et al., 2019b). It is worth noting that the "adaptive sequencing" technique alone is far from sufficient to achieve approximation ratios for any submodular maximization problem, thus numerous studies such as (Amanatidis et al., 2021, 2023; Breuer et al., 2020; Quinzan et al., 2021; Gong et al., 2024) built upon this technique to develop low-adaptivity algorithms for submodular maximization under different constraints as we have done. In contrast to the original technique of (Balkanski et al., 2019b) and its variants such as that in (Amanatidis et al., 2021, 2023), our RandBatch introduces a "random batch selection" method to allow for a more flexible adaptive sequencing. This enhancement expands the applicability of our algorithm to a wider range of constraints and objective functions. More importantly, our methods for generating candidate solutions (i.e., ParSKP and ParSSP) differ from existing work, which allows us to avoid the issues encountered in (Amanatidis et al., 2021; Quinzan et al., 2021) and achieve better performance bounds than their algorithm.

For clarity, we first introduce RandBatch (Sec. 4.1), then introduce ParSKP (Sec. 4.2) and ParSSP (Sec. 4.3), respectively.

4.1 The RandBatch Procedure

The RandBatch procedure (Algorithm 1) takes as input a threshold ρ, candidate element set I, submodular function $f(\cdot)$, cost function $c(\cdot)$, a number M to control the adaptivity, and $p, \epsilon \in (0, 1]$. It runs in iterations to find a solution set A and uses U to store all elements considered for addition to A. Besides, it maintains a set L of "valuable elements" in I that have not been considered throughout the procedure, where any element is called a *valuable element* w.r.t. A if it can be added into A with marginal density no less than ρ under the budget constraint (Line 2). In each iteration, RandBatch first neglects $f(\cdot)$ and calls a simple function GetSEQ (Algorithm 2) to get a random sequence of elements $(v_1, \ldots, v_d)$ from L without violating the constraint (Line 4), and then finds a subsequence $V_{t^*} = (v_1, \ldots, v_{t^*})$ with "good quality" by considering $f(\cdot)$ (Lines 5–11), where $t^* = \min\{t_1, t_2\}$ will be explained shortly. After that, it invokes a "random batch selection" operation by adding V_{t^*} into A with probability of p and abandoning V_{t^*} with probability of $1-p$ (Line 12). All the elements in V_{t^*} are recorded into U no matter they are accepted or abandoned. Then RandBatch enters a new iteration and repeats the above process with an updated L (Line 15). RandBatch uses a variable *count* to control its adaptive complexity (Line 14), and returns (A, U, L) either when $L = \emptyset$ or $count = M$. Note that RandBatch returns $L \neq \emptyset$ only when $count = M$.

As mentioned above, RandBatch uses $t^* = \min\{t_1, t_2\}$ to control the quality of the elements in V_{t^*}, where t_1, t_2 depend on E_i^+ (elements in L with enough density), E_i^- (elements in L with negative marginal gain) and D_i (elements in V_i with negative marginal gain) defined in Lines 7-9. Intuitively, the setting of t_1 (Line 10) ensures that the total cost of valuable elements w.r.t. $A \cup V_{t_1}$ (i.e., elements in $E_{t_1}^+$) is sufficiently small, and the setting of t_2 (Line 10) ensures that the total marginal gain of valuable elements w.r.t. $A \cup V_{t_1}$ (i.e., elements in $E_{t_2}^+$) is sufficiently small. Through the selection of V_{t^*}, RandBatch strikes

Algorithm 1: RandBatch$(\rho, I, M, p, \epsilon, f(\cdot), c(\cdot))$

Input: density threshold ρ, set I, maximum number of iterations $M \in \mathbb{Z}_{>0}$, probability p, precision $\epsilon \in (0,1)$, submodular function $f(\cdot)$, and cost function $c(\cdot)$;

1 $A \leftarrow \emptyset$; $U \leftarrow \emptyset$; $count \leftarrow 0$;
2 $L \leftarrow \{u \in I \colon \frac{f(u|A)}{c(u)} \geq \rho \wedge A \cup \{u\} \in \mathcal{I}\}$;
3 **while** $L \neq \emptyset \wedge count < M$ **do**
4 $\quad \{v_1, v_2, \ldots, v_d\} \leftarrow$ GetSEQ$(A, L, c(\cdot))$;
5 $\quad$ **foreach** $i \in \{0, 1, \ldots, d\}$ **do**
6 $\quad\quad V_i \leftarrow \{v_1, v_2, \ldots, v_i\}$; $G_i \leftarrow A \cup V_i$;
7 $\quad\quad E_i^+ \leftarrow \{u \in L \colon \frac{f(u|G_i)}{c(u)} \geq \rho \wedge G_i \cup \{u\} \in \mathcal{I}\}$;
8 $\quad\quad E_i^- \leftarrow \{u \in L \colon f(u \mid G_i) < 0\}$;
9 $\quad\quad D_i \leftarrow \{v_j \colon j \in [i] \wedge f(v_j \mid A \cup V_{j-1}) < 0\}$;
10 $\quad$ Find $t_1 \leftarrow \min_{i \leq d}\{c(E_i^+) \leq (1-\epsilon)c(L)\}$ and $t_2 \leftarrow \min_{i \leq d}\{\epsilon \sum_{u \in E_i^+} f(u \mid G_i) \leq \sum_{u \in E_i^-} |f(u \mid G_i)| + \sum_{v_j \in D_i} |f(v_j \mid A \cup V_{j-1})|\}$;
11 $\quad t^* \leftarrow \min\{t_1, t_2\}$; $U \leftarrow U \cup \{V_{t^*}\}$;
12 $\quad$ **with** *probability* p **do**
13 $\quad\quad A \leftarrow A \cup V_{t^*}$;
14 $\quad\quad$ **if** $t_2 < t_1$ **then** $count \leftarrow count + 1$;
15 $\quad L \leftarrow \{u \colon u \in L \setminus U \wedge \frac{f(u|A)}{c(u)} \geq \rho \wedge A \cup \{u\} \in \mathcal{I}\}$;
16 **return** (A, U, L);

Algorithm 2: GetSEQ(A, I)

Input: sets A and I;

1 $V \leftarrow \emptyset$;
2 **while** $I \neq \emptyset$ **do**
3 $\quad$ Randomly permute I into $\{v_1, \ldots, v_{|I|}\}$;
4 $\quad s \leftarrow \max\{i \in [|I|] : A \cup V \cup \{v_1, \ldots, v_i\} \in \mathcal{I}\}$;
5 $\quad V \leftarrow V \cup \{v_1, \ldots, v_s\}$;
6 $\quad I \leftarrow \{u \colon u \in I \setminus V \wedge A \cup V \cup \{u\} \in \mathcal{I}\}$;
7 **return** V;

a balance between solution quality and adaptive complexity, as shown by the following lemma:

Lemma 1. *The sets A and L output by* RandBatch$(\rho, I, M, f(\cdot), c(\cdot), p, \epsilon)$ *satisfy (1) $A \in \mathcal{I}$, (2) $\mathbb{E}[f(A)] \geq (1-\epsilon)^2 \rho \cdot \mathbb{E}[c(A)]$ and (3) $\epsilon \cdot M \cdot \sum_{u \in L} f(u \mid A) \leq \mathrm{OPT}$ for any $I \subseteq \mathcal{N}$.*

It can be readily observed that property 1 of Lemma 1 holds as GetSEQ always returns feasible sequences. However, the proof of properties 2 and 3 of Lemma 1 are a bit complex due to the randomness introduced in GetSEQ function and in Line 12 of RandBatch.

To overcome this challenge, we first introduce Lemma 2, which shows that the expected marginal density of each element that RandBatch attempts to add to the candidate solution is sufficiently large. We then use Lemma 2 to prove Lemma 1. The proof is inspired by (Balkanski et al., 2019b; Amanatidis et al., 2021), but is more involved due to the randomness introduced by Line 12 of Algorithm 1.

Lemma 2. *For any V_{t^*} found in Lines 10–11 of RandBatch and any $u \in V_{t^*}$, let $\lambda(u)$ denote the set of elements in V_{t^*} selected before u (note that V_{t^*} is an ordered list according to Line 6), and $\lambda(u)$ does not include u. Let A and U denote the sets returned by* $\mathsf{RandBatch}(\rho, I, M, p, \epsilon, f(\cdot), c(\cdot))$*, where the elements in U are $\{u_1, u_2, \ldots, u_s\}$ (listed according to the order they are added into U). Let u_j be a "dummy element" with zero marginal gain and zero cost for all $s < j \leq |I|$. Then we have:*

$$\forall j \in [|I|]\colon \qquad \mathbb{E}[f(u_j \mid \{u_1 \ldots, u_{j-1}\} \cap A \cup \lambda(u_j)) \mid \mathcal{F}_{j-1}] \geq (1-\epsilon)^2\rho \cdot \mathbb{E}[c(u_j) \mid \mathcal{F}_{j-1}],$$

where $\mathcal{F}_{j-1}$ denotes the filtration capturing all the random choices made until the moment right before selecting u_j.

Proof of Lemma 2. Note that $\mathcal{F}_{j-1}$ determines whether $u_j \in U$, so the lemma trivially follows for all $j > s$. In the sequel, we consider the case of $j \leq s$. Recall that RandBatch runs in iterations and adds a batch of elements V_{t^*} into U in each iteration (Line 11). Consider the specific iteration in which u_j is added into U and the sets G_q, E_q^+, E_q^- defined by Lines 6–8 of RandBatch in that iteration, where $q = |\lambda(u_j)|$; and let $H = \{v\colon v \in L \setminus G_q \wedge G_q \cup \{v\} \in \mathcal{I}\}$, where L is the set considered at the beginning of that iteration. So we have $\{u_1, \ldots, u_{j-1}\} \cap A \cup \lambda(u_j) = G_q$. Note that both G_q and H are deterministic given $\mathcal{F}_{j-1}$, and that u_j is drawn uniformly at random from H. So we have

$$\begin{aligned}
&\mathbb{E}\big[f(u_j \mid \{u_1, \ldots, u_{j-1}\} \cap A \cup \lambda(u_j)) \mid \mathcal{F}_{j-1}\big] - (1-\epsilon)^2\rho \cdot \mathbb{E}\big[c(u_j) \mid \mathcal{F}_{j-1}\big] \\
&= \sum\nolimits_{v \in H} \Pr[u_j = v \mid \mathcal{F}_{j-1}] f(v \mid G_q) - (1-\epsilon)^2\rho \sum\nolimits_{v \in H} \Pr[u_j = v \mid \mathcal{F}_{j-1}] c(v) \\
&\geq |H|^{-1} \cdot \left(\sum\nolimits_{v \in E_q^+} f(v \mid G_q) + \sum\nolimits_{v \in E_q^-} f(v \mid G_q) - (1-\epsilon)^2\rho \cdot \sum\nolimits_{v \in H} c(v) \right) \\
&= |H|^{-1} \cdot \left(\epsilon \sum\nolimits_{v \in E_q^+} f(v \mid G_q) - \sum\nolimits_{v \in E_q^-} |f(v \mid G_q)| \right) && (1) \\
&\quad + |H|^{-1} \cdot (1-\epsilon) \cdot \sum\nolimits_{v \in E_q^+} (f(v \mid G_q) - \rho \cdot c(v)) && (2) \\
&\quad + |H|^{-1} \cdot (1-\epsilon) \cdot \rho \cdot \left(\sum\nolimits_{v \in E_q^+} c(v) - (1-\epsilon) \sum\nolimits_{v \in H} c(v) \right), && (3)
\end{aligned}$$

where the first inequality is due to $E_q^+ \cup E_q^- \subseteq H$. Note that Eqn. (2) is non-negative due to the definition of E_q^+. According to the definition of t^* in Lines 10–11 of Algorithm 1, we have $c(E_q^+) > (1-\epsilon)c(L)$ due to $q \leq t^* - 1$, so Eqn. (3) is non-negative as $H \subseteq L$. Similarly, Eqn. (1) is also non-negative due to the definition of t^*. Combining these completes the proof. □

Proof of Lemma 1. We first prove $\mathbb{E}[f(A)] \geq (1-\epsilon)^2\rho \cdot \mathbb{E}[c(A)]$. Consider the sequence $\{u_1, \ldots, u_s, u_{s+1}, \ldots, u_{|I|}\}$ defined in Lemma 2. For each $j \in [|I|]$, define a random variable

$\delta_1(u_j) = f(u_j \mid \{u_1, \ldots, u_{j-1}\} \cap A \cup \lambda(u_j))$ if $u_j \in A$ and $\delta_1(u_j) = 0$ otherwise, and also define $\delta_2(u_j) = c(u_j)$ if $u_j \in A$ and $\delta_2(u_j) = 0$ otherwise. So we have $f(A) \geq \sum_{j=1}^{|I|} \delta_1(u_j)$ and $c(A) = \sum_{j=1}^{|I|} \delta_2(u_j)$. Due to the linearity of expectation and the law of total expectation, we only need to prove

$$\forall j \in [|I|], \forall \mathcal{F}_{j-1} \colon \mathbb{E}[\delta_1(u_j) \mid \mathcal{F}_{j-1}] \geq (1-\epsilon)^2 \rho \cdot \mathbb{E}[\delta_2(u_j) \mid \mathcal{F}_{j-1}], \tag{4}$$

where $\mathcal{F}_{j-1}$ is the filtration defined in Lemma 2. This trivially holds for $j > s$. For any $j \leq s$, we have

$$\begin{aligned} \mathbb{E}[\delta_1(u_j)|\mathcal{F}_{j-1}] &= \mathbb{E}[\mathbb{E}[\delta_1(u_j)|\mathcal{F}_{j-1}, u_j] \mid \mathcal{F}_{j-1}] \\ &= p\mathbb{E}[f(u_j \mid \{u_1, \ldots, u_{j-1}\} \cap A \cup \lambda(u_j)) \mid \mathcal{F}_{j-1}], \end{aligned}$$

where the second equality is due to the reason that, each $u_j \in U$ is added into A with probability of p, which is independent of the selection of u_j. Similarly, we can prove $\mathbb{E}[\delta_2(u_j) \mid \mathcal{F}_{j-1}] = p \cdot \mathbb{E}[c(u_j) \mid \mathcal{F}_{j-1}]$. Combining these results with Lemma 2 proves Eqn. (4) and hence $\mathbb{E}[f(A)] \geq (1-\epsilon)^2 \rho \cdot \mathbb{E}[c(A)]$.

Next, we prove $\epsilon \cdot M \cdot \sum_{u \in L} f(u \mid A) \leq f(O)$. This trivially holds if $L = \emptyset$, otherwise we must have $count = M$ when Algorithm 1 returns (A, U, L) due to Line 3. Note that *count* is increased by 1 only in Line 14 of the while-loop in Algorithm 1. Consider the i-th iteration among the specific M iterations of the while-loop in which *count* gets increased, and let $E^+_{[i]}, E^-_{[i]}, D_{[i]}$ denote the sets $E^+_{t^*}, E^-_{t^*}, D_{t^*}$ in that iteration, and let $A_{[i]}$ denote the set of elements already added into A at the end of that iteration. Besides, let $A^<(u)$ denote the elements in A that are selected before u for any $u \in A$, and let $A^+ = \{u \in A \colon f(u \mid A^<(u)) \geq 0\}$ and $A^- = \{u \in A \colon f(u \mid A^<(u)) < 0\}$. As the sets in $\{E^-_{[i]}, D_{[i]} \colon i \in [M]\}$ are mutually disjoint according to their definitions and $\bigcup_{i=1}^M E^-_{[i]} \cap A = \emptyset$, we can use submodularity of $f(\cdot)$ to get

$$\begin{aligned} &f(\textstyle\bigcup_{i=1}^M E^-_{[i]} \cup A) \\ &\leq f(A) + \textstyle\sum_{i=1}^M \sum_{u \in E^-_{[i]}} f(u \mid A) \\ &\leq \textstyle\sum_{u \in A^+} f(u \mid A^<(u)) + \sum_{u \in A^-} f(u \mid A^<(u)) + \sum_{i=1}^M \sum_{u \in E^-_{[i]}} f(u \mid A_{[i]}) \\ &\leq \textstyle\sum_{u \in A^+} f(u \mid A^<(u)) + \sum_{i=1}^M \sum_{u \in D_{[i]}} f(u \mid A^<(u)) \\ &+ \textstyle\sum_{i=1}^M \sum_{u \in E^-_{[i]}} f(u \mid A_{[i]}), \end{aligned} \tag{5}$$

where the last inequality follows from the fact that $\cup_{i \in [M]} D_{[i]} \subseteq A^-$. Combining Eqn. (5) with $\sum_{u \in A^+} f(u \mid A^<(u)) \leq f(A^+)$ (due to submodularity) and $f(\bigcup_{i=1}^M E^-_{[i]} \cup A) \geq 0$, we can get

$$\sum_{i=1}^M \Big(\sum_{u \in D_{[i]}} |f(u \mid A^<(u))| + \sum_{u \in E^-_{[i]}} |f(u \mid A_{[i]})| \Big) \leq f(A^+) \leq \mathrm{OPT}. \tag{6}$$

Besides, according to Lines 10,11,14 of Algorithm 1, we must have

$$\forall i \in [M]\colon \epsilon \sum_{u \in E^+_{[i]}} f(u \mid A_{[i]}) \leq \sum_{u \in E^-_{[i]}} |f(u \mid A_{[i]})| + \sum_{u \in D_{[i]}} |f(u \mid A^{<}(u))|. \tag{7}$$

Moreover, we have $\sum_{i=1}^{M} \sum_{u \in E^+_{[i]}} f(u \mid A_{[i]}) \geq M \cdot \sum_{u \in E^+_{[M]}} f(u \mid A)$ due to submodularity of $f(\cdot)$ and $E^+_{[M]} \subseteq E^+_{[M-1]} \subseteq \cdots \subseteq E^+_{[1]}$. Combining this with Eqn. (6) and Eqn. (7) yields $\mathrm{OPT} \geq \epsilon \cdot M \cdot \sum_{u \in E^+_{[M]}} f(u \mid A)$, which completes the proof due to $E^+_{[M]} = L$ when Algorithm 1 returns a non-empty L. □

The complexity of RandBatch (shown in Lemma 3) can be proved by using the fact that, when A enlarges, either $c(L)$ is decreased by a $1-\epsilon$ factor, or *count* is increased by 1 (Line 14).

Lemma 3. *RandBatch has $\mathcal{O}((\frac{1}{\epsilon}\log(|I|\cdot\beta(I))+M)/p)$ adaptivity, and its query complexity is $\mathcal{O}(|I|\cdot r)$ times of its adaptive complexity, where $\beta(I) \triangleq \max_{u,v\in I}\frac{c(u)}{c(v)}$. If we use binary search in Line 10, then RandBatch has $\mathcal{O}((\frac{1}{\epsilon}\log(|I|\cdot\beta(I))+M)\cdot(\log r)/p)$ adaptivity, and its query complexity is $\mathcal{O}(|I|)$ times of its adaptivity.*

Proof. First, we analyze the number of while-loops in RandBatch. Note that RandBatch needs $\mathcal{O}(1/p)$ while-loops (in expectation) to trigger Lines 13–14 once. Each time when Lines 13–14 are executed, either $c(L)$ is decreased by at least a $1-\epsilon$ factor, or *count* is increased by 1. Besides, note that RandBatch terminates either when $L=\emptyset$ or $\mathit{count}=M$. At the beginning of the algorithm, we have $c(L) \leq c_{max}\cdot|I|$ where $c_{max}=\max_{u\in I} c(u)$, while we need $c(L) < \min_{u\in I} c(u)$ to ensure $L=\emptyset$. Based on the above discussions, it can be seen that the total number of while-loops in RandBatch is at most $\mathcal{O}((\frac{1}{\epsilon}\log(|I|\cdot\beta(I))+M)/p)$ in expectation.

Second, we explain why binary search can be used in Line 10 of Algorithm 1. Recall that Line 10 need to find the smallest $i\in[d]$ satisfying Eqn. (8) to determine t_1, and to find the smallest $i\in[d]$ satisfying Eqn. (9) to determine t_2:

$$c(E_i^+) \leq (1-\epsilon)c(L), \tag{8}$$

$$\epsilon\sum\nolimits_{u\in E_i^+} f(u\mid G_i) \leq \sum\nolimits_{u\in E_i^-} |f(u\mid G_i)| + \sum\nolimits_{v_j\in D_i} |f(v_j\mid A\cup V_{j-1})|. \tag{9}$$

By submodularity and the definitions of E_i^+, E_i^-, D_i, it can be verified that the LHS of Eqn. (8) or Eqn. (9) decreases when i increases, and the RHS of Eqn. (9) increases with i. This makes it possible to use binary search.

Third, we analyze the adaptive complexity and query complexity caused by each while-loop in RandBatch. Note that the GetSEQ function causes zero adaptive complexity. Therefore, if binary search is not used in Line 10, then the seeking of t^* in Lines 10–11 of RandBatch can be fully parallelized and hence causes $\mathcal{O}(1)$ adaptive complexity, but under $\mathcal{O}(|I|\cdot r)$ query complexity because $\mathcal{O}(|I|)$ oracle queries are needed to calculate E_i^+, E_i^- for each $i\leq d\leq r$. On the other side, if binary search is used, then we need $\mathcal{O}(\log r)$ adaptive rounds to find t^* due to $t^*\leq d\leq r$, causing $\mathcal{O}(|I|\log r)$ oracle queries. The lemma then follows by synthesizing all the above discussions. □

Algorithm 3: ParSKP$(\alpha, \epsilon, B, f(\cdot), c(\cdot))$

Input: parameter $\alpha \in (0,1)$, precision $\epsilon \in (0,1)$, budget B, submodular function $f(\cdot)$, and cost function $c(\cdot)$;

1 $\mathcal{N}_1 \leftarrow \{u \in \mathcal{N} \colon c(u) > \epsilon \frac{B}{n}\}$; $\mathcal{N}_2 \leftarrow \mathcal{N} \setminus \mathcal{N}_1$;
2 $u^* \leftarrow \arg\max_{u \in \mathcal{N}} f(u)$; $S \leftarrow \mathsf{USM}(\mathcal{N}_2)$;
3 $S \leftarrow \arg\max_{X \in \{S, \{u^*\}\}} f(X)$;
4 $\rho_{min} \leftarrow \frac{\alpha f(u^*)}{B}$; $\rho_{max} \leftarrow \frac{n^2 \cdot \alpha f(u^*)}{\epsilon B}$;
5 $Z \leftarrow \{(1-\epsilon)^{-z} \colon z \in \mathbb{Z} \wedge (1-\epsilon)^{-z} \in [\rho_{min}, \rho_{max}]\}$;
6 **foreach** $\rho \in Z$ *in parallel* **do**
7 **for** $i \leftarrow 1$ **to** $\lceil \log_{1-\epsilon} \epsilon \rceil$ *in parallel* **do**
8 $T \leftarrow \mathsf{Probe}(\rho, \mathcal{N}_1, \mathcal{N}_2, \epsilon, B, f(\cdot), c(\cdot))$;
9 $S \leftarrow \arg\max_{X \in \{S,T\}} f(X)$;
10 **return** S;

Algorithm 4: Probe$(\rho, \mathcal{N}_1, \mathcal{N}_2, \epsilon, B, f(\cdot), c(\cdot))$

Input: density threshold ρ, two disjoint partitions of the ground set $\mathcal{N}_1$ and $\mathcal{N}_2$, precision $\epsilon \in (0,1)$, budget B, submodular function $f(\cdot)$, and cost function $c(\cdot)$;

1 $T \leftarrow \emptyset$; $M \leftarrow \lceil \epsilon^{-2} \rceil$; $p \leftarrow 1$; $I \leftarrow \mathcal{N}_1$;
2 $(A_1, U_1, L_1) \leftarrow \mathsf{RandBatch}(\rho, I, M, p, \epsilon, f(\cdot), c(\cdot))$;
3 $I \leftarrow \mathcal{N}_1 \setminus A_1$;
4 $(A_2, U_2, L_2) \leftarrow \mathsf{RandBatch}(\rho, I, M, p, \epsilon, f(\cdot), c(\cdot))$;
5 **for** $i \leftarrow 1$ **to** 2 **do**
6 $e_i \leftarrow \arg\max_{u \in \mathcal{N}_1 \wedge c(A_i \cup \{u\}) \le B} f(A_i \cup \{u\})$;
7 $T \leftarrow \arg\max_{X \in \{T, A_i, A_i \cup \{e_i\}\}} f(X)$;
8 **if** $c(\mathcal{N}_2 \cup A_1) \le B$ **then**
9 $A_3 \leftarrow \mathsf{USM}(\mathcal{N}_2 \cup A_1)$;
10 $T \leftarrow \arg\max_{X \in \{T, A_3\}} f(X)$;
11 **return** T;

4.2 The Algorithm for the SKP

Our ParSKP algorithm is shown in Algorithm 3. In ParSKP, the ground set $\mathcal{N}$ is partitioned into two disjoint subsets $\mathcal{N}_1$ and $\mathcal{N}_2$, where $\mathcal{N}_1$ contains every element in $\mathcal{N}$ with a sufficiently large cost (i.e., larger than $\epsilon \cdot B/n$). So we have $c(\mathcal{N}_2) \le \epsilon \cdot B$. A major building block of ParSKP is the function Probe (shown in Algorithm 4). For clarity, we first elaborate Probe in the following.

Given an input threshold ρ and $\mathcal{N}_1, \mathcal{N}_2$, Probe first calls RandBatch with $p = 1$ using $\mathcal{N}_1$ as the ground set to find a candidate solution A_1 (Line 2), and then calls RandBatch again using $\mathcal{N}_1 \setminus A_1$ as the ground set to find another candidate solution A_2 (Line 4). So A_1 and A_2 are disjoint subsets of $\mathcal{N}_1$. The reason for calling RandBatch with only the elements in $\mathcal{N}_1$ is that the adaptive complexity of RandBatch can be bounded only when the costs of

considered elements have a lower bound (due to Lemma 3). Then, Probe tries to "boost" the utilities of A_1 and A_2 by augmenting them with a single element in $\mathcal{N}_1$, neglecting the threshold ρ (Line 6). After that, another candidate solution set A_3 is found by calling an unconstrained submodular maximization algorithm if $c(\mathcal{N}_2 \cup A_1) \leq B$ (Line 9). Finally, Probe returns the candidate solution with maximum function value found so far.

In Lemma 4, we show Probe can achieve a provable approximation ratio under some special cases:

Lemma 4. *If the threshold ρ input into Algorithm 4 is no more than $\rho^* \triangleq \frac{\alpha f(O)}{B-c(w)}$ and Algorithm 4 finds A_1 and A_2 satisfying $\forall i \in \{1,2\}\colon c(A_i) < B - \max\{\epsilon B, c(w)\}$, where w is the element in O with the maximum cost, then Algorithm 4 returns a solution T satisfying $f(T) \geq \frac{(1-2\epsilon)(1-2\alpha-2\epsilon)}{4-4\epsilon} \cdot \mathrm{OPT}$.*

Proof. According to the assumption of the lemma, we must have $c(A_1 \cup \mathcal{N}_2) \leq B$ due to $c(\mathcal{N}_2) \leq \epsilon \cdot B$, so Line 9 of Probe must be executed. If $w \notin \mathcal{N}_1$, then we must have $O \subseteq \mathcal{N}_2$ and hence $2f(T) \geq 2f(A_3) \geq (1-2\epsilon)f(O)$ due to Line 9 of Algorithm 4 (Theorem 1), which completes the proof. Therefore, we assume $w \in \mathcal{N}_1$ in the following.

Note that $\mathcal{N}_1 \cap \mathcal{N}_2 = \emptyset, A_1 \cap A_2 = \emptyset$ and $A_1, A_2 \subseteq \mathcal{N}_1$. So we can use submodularity of $f(\cdot)$ to get:

$$\begin{aligned} f(O) &\leq f(O \cap \mathcal{N}_1 \setminus A_1) + f((O \cap A_1) \cup (O \cap \mathcal{N}_2)) \\ &\leq f(A_2 \cup (O \cap \mathcal{N}_1 \setminus A_1)) + f(A_1 \cup (O \cap \mathcal{N}_1)) + f((O \cap A_1) \cup (O \cap \mathcal{N}_2)). \end{aligned} \tag{10}$$

Next, we try to bound the three additive factors in the RHS of Eqn. (10). For the third additive factor, we have

$$(1-2\epsilon)f((O \cap A_1) \cup (O \cap \mathcal{N}_2)) \leq 2f(A_3) \leq 2f(T), \tag{11}$$

due to Line 9 of Algorithm 4. Besides, we can get

$$\begin{aligned} &f(A_1 \cup (O \cap \mathcal{N}_1)) \leq f(A_1 \cup \{w\}) + \sum\nolimits_{u \in Q} f(u \mid A_1 \cup \{w\}) \leq f(T) + \sum\nolimits_{u \in Q} f(u \mid A_1) \\ &\leq f(T) + \sum\nolimits_{u \in L_1} f(u \mid A_1) + \sum\nolimits_{u \in Q \setminus L_1} f(u \mid A_1), \end{aligned} \tag{12}$$

where $Q = O \cap \mathcal{N}_1 \setminus (A_1 \cup \{w\})$, and L_1 is the set returned by RandBatch in Line 2 of Probe, and the second inequality is due to the the submodularity of $f(\cdot)$ and Lines 5–7 of Algorithm 4. Furthermore, note that each $u \in Q \setminus L_1$ satisfies $c(A_1 \cup \{u\}) \leq B$ according to the assumption of current lemma, so we should have $\frac{f(u|A_1)}{c(u)} < \rho$, because otherwise u should be in either A_1 or L_1 due to the design of RandBatch. Using this and $c(Q) \leq B - c(w)$, we get

$$\sum\nolimits_{u \in Q \setminus L_1} f(u \mid A_1) \leq \rho \cdot c(Q) \leq \rho^* \cdot c(Q) \leq \alpha \cdot f(O). \tag{13}$$

Besides, we can use Lemma 1 to get $\sum_{u \in L_1} f(u \mid A_1) \leq \epsilon \cdot f(O)$ due to $M = \lceil \epsilon^{-2} \rceil$. Combining this with Eqn. (12) and Eqn. (13) yields

$$f(A_1 \cup (O \cap \mathcal{N}_1)) \leq f(T) + (\alpha + \epsilon) \cdot f(O). \tag{14}$$

Using similar reasoning as above, we can also get

$$f(A_2 \cup (O \cap \mathcal{N}_1 \setminus A_1)) \leq f(T) + (\alpha + \epsilon) \cdot f(O). \tag{15}$$

The lemma then follows by combining Eqns. (10)–(15). □

There are still two obstacles for using Lemma 4 to find the approximation ratio of ParSKP: the first problem is that ρ^* is unknown, and the second problem is that $c(A_1)$ and $c(A_2)$ may not satisfy the condition in Lemma 4. In the following, we roughly explain how ParSKP is designed to overcome these hurdles.

For the first problem mentioned above, it can be proved that $\rho^* \in [\rho_{min}, \rho_{max}]$ if $B - c(w) > \epsilon B/n$, where ρ_{min} and ρ_{max} are defined in Line 4 of ParSKP. Therefore, ParSKP tests multiple values of ρ in Z (Line 5) to ensure that one of them lies in $[(1-\epsilon)\rho^*, \rho^*]$. One the other side, if $B - c(w) \leq \epsilon B/n$, then we have $O \setminus \{w\} \subseteq \mathcal{N}_2$ and hence $\mathsf{USM}(\mathcal{N}_2)$ in Line 2 of ParSKP can be used to find a ratio. To address the second problem mentioned above, ParSKP repeatedly runs Probe for a sufficiently large number of times (Lines 7–9). Therefore, if both $\mathbb{E}[c(A_1)]$ and $\mathbb{E}[c(A_2)]$ are sufficiently small, then it can be proved that at least one run of Probe satisfies the condition in Lemma 4 with high probability. On the other side, if either $\mathbb{E}[c(A_1)]$ or $\mathbb{E}[c(A_2)]$ is sufficiently large, then we can directly use Lemma 1 to prove that Probe also satisfies a desired approximation ratio (in expectation). By choosing an appropriate α that can maximize our approximation ratio and combining all above ideas, we get:

Theorem 2. *For the non-monotone SKP, ParSKP can return a solution S satisfying $\mathbb{E}[f(S)] \geq (\frac{1}{8} - \epsilon)\mathrm{OPT}$ by setting $\alpha = \frac{1}{4}$.*

Proof. If $c(O \setminus \{w\}) \leq \epsilon\frac{B}{n}$, then it follows that $O \setminus \{w\} \subseteq \mathcal{N}_2$. As a result, Line 2 of Algorithm 3 guarantees $f(S) \geq (1/2 - \epsilon)f(O \setminus \{w\})$, which implies

$$f(O) \leq f(w) + f(O \setminus \{w\}) \leq f(\{u^*\}) + \frac{2}{1-2\epsilon}f(S) \leq \left(1 + \frac{2}{1-2\epsilon}\right)f(S)$$

where the second inequality is due to Line 3 of Algorithm 3. Thus, we have $f(S) \geq \left(1 + \frac{2}{1-2\epsilon}\right)^{-1} f(O) \geq (1/3 - \epsilon)f(O)$ when $\epsilon < 5/6$, which implies the theorem holds in this case.

Then we only need to consider the case that $c(O \setminus \{w\}) > \epsilon\frac{B}{n}$ in the follwing analysis. In this case, we have $B - c(w) \geq c(O) - c(w) > \epsilon\frac{B}{n}$ and hence

$$\rho_{min} = \frac{\alpha f(u^*)}{B} \leq \frac{\alpha f(O)}{B - c(w)} = \rho^* \leq \frac{n \cdot \alpha f(u^*)}{B - c(w)} \leq \frac{n \cdot \alpha f(u^*)}{\epsilon\frac{B}{n}} \leq \frac{n^2 \cdot \alpha f(u^*)}{\epsilon B} = \rho_{max}$$

which implies that there exists a threshold $\rho \in Z$ for the Probe algorithm such that $(1 - \epsilon)\rho^* \leq \rho \leq \rho^*$, as per the definition of Z. With this threshold established, we can proceed with the following discussion:

- Case 1: The Probe algorithms finds $\{A_1, A_2\}$ such that there exists $A \in \{A_1, A_2\}$ satisfying $\mathbb{E}\big[c(A)\big] \geq (1 - \epsilon)(B - \max\{\epsilon B, c(w)\})/2$. Under this case, we can use

Lemma 1 to get

$$\mathbb{E}\big[f(S)\big] \geq \mathbb{E}\big[f(A)\big] \geq (1-\epsilon)^2\rho \cdot \mathbb{E}\big[c(A)\big] \geq (1-\epsilon)^4 \frac{\alpha f(O)}{B-c(w)} \cdot \frac{B-\max\{\epsilon B, c(w)\}}{2}$$
$$= (1-\epsilon)^4 \frac{\alpha f(O)}{2} \cdot \frac{B-\max\{\epsilon B, c(w)\}}{B-c(w)}. \tag{16}$$

When $\epsilon B \geq c(w)$, we have

$$\frac{B-\max\{\epsilon B, c(w)\}}{B-c(w)} = \frac{B-\epsilon B}{B-c(w)} \geq 1 - \frac{\epsilon B}{B-c(w)} \geq 1 - \frac{\epsilon B}{B-\epsilon B} = \frac{1-2\epsilon}{1-\epsilon}, \tag{17}$$

and hence $\mathbb{E}[f(S)] \geq (1-\epsilon)^3(1-2\epsilon)\frac{\alpha}{2}f(O) \geq (1/8-\epsilon)f(O)$ when $\alpha = 1/4$. Similarly, we can prove $\mathbb{E}[f(S)] \geq (1/8-\epsilon)f(O)$ when $\epsilon B < c(w)$.

- Case 2: The Probe algorithms finds A_1 and A_2 satisfying $\mathbb{E}\big[c(A_i)\big] < (1-\epsilon)(B - \max\{\epsilon B, c(w)\})/2$ for all $i \in \{1,2\}$. Under this case, define an event $\mathcal{E} = \{\max\{c(A_1), c(A_2)\} \geq B - \max\{\epsilon B, c(w)\}\}$, then we have

$$(1-\epsilon)(B-\max\{\epsilon B, c(w)\}) > \mathbb{E}\big[c(A_1)+c(A_2)\big]$$
$$\geq \mathbb{E}[c(A_1)+c(A_2) \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] \geq (B-\max\{\epsilon B, c(w)\}) \cdot \Pr[\mathcal{E}]. \tag{18}$$

and hence $\Pr[\mathcal{E}] < 1-\epsilon$. Recall that ParSKP calls Probe for $\lceil \log_{1-\epsilon} \epsilon \rceil$ times in parallel, so the probability of at least one run of Probe finds A_1 and A_2 satisfying $\max\{c(A_1), c(A_2)\} < B - \max\{\epsilon B, c(w)\}$ is no less than $1-(1-\epsilon)^{\log_{1-\epsilon}\epsilon} = 1-\epsilon$. Using Lemma 4, we have $\mathbb{E}[f(S)] \geq \frac{(1-\epsilon)(1-2\epsilon)(1-2\alpha-2\epsilon)}{4(1-\epsilon)} f(O) \geq (1/8-\epsilon)f(O)$ when $\alpha = 1/4$.

According to the above discussion, the theorem follows. □

Note that the complexity of ParSKP is dominated by Lines 6–9, where Probe is run for multiple times in parallel. Therefore, leveraging Lemma 3, we can also get:

Theorem 3. *The adaptive complexity and query complexity of ParSKP are $\mathcal{O}(\log n)$ and $\mathcal{O}(nr\log^2 n)$ respectively, or $\mathcal{O}(\log n \log r)$ and $\mathcal{O}(n\log^2 n \log r)$ respectively.*

Proof. Note that Probe calls RandBatch using $M = \lceil \epsilon^{-2} \rceil$ and $I \subseteq \mathcal{N}_1$, and we have $\max_{u,v\in\mathcal{N}_1} \frac{c(u)}{c(v)} \leq \frac{n}{\epsilon}$ due to the definition of $\mathcal{N}_1$. Therefore, according to Lemma 3, each call of RandBatch incurs adaptive complexity of $\mathcal{O}(\frac{1}{\epsilon}\log\frac{n}{\epsilon} + \frac{1}{\epsilon^2})$ and query complexity of $\mathcal{O}(\frac{nr}{\epsilon}\log\frac{n}{\epsilon} + \frac{nr}{\epsilon^2})$ if binary search is not used in RandBatch, or incurs adaptive complexity of $\mathcal{O}((\frac{1}{\epsilon}\log\frac{n}{\epsilon} + \frac{1}{\epsilon^2})\cdot\log r)$ and query complexity of $\mathcal{O}((\frac{n}{\epsilon}\log\frac{n}{\epsilon} + \frac{n}{\epsilon^2})\cdot\log r)$ if binary search is used. Besides, note that ParSKP calls Probe for $\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ times for every $\rho \in Z$ in parallel, and we have $|Z| = \mathcal{O}(\frac{1}{\epsilon}\log\frac{n}{\epsilon})$. The USM algorithm called in Line 2 of ParSKP or Line 9 of Probe incurs adaptive complexity of $\mathcal{O}(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ and query complexity of $\mathcal{O}(\frac{n}{\epsilon^4}\log^3\frac{1}{\epsilon})$. Combining all the above results, we know that the adaptive complexity and query complexity of ParSKP are $\mathcal{O}(\frac{1}{\epsilon}\log\frac{n}{\epsilon} + \frac{1}{\epsilon^2}) = \mathcal{O}(\log n)$ and $\mathcal{O}(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon}\log\frac{n}{\epsilon}(\frac{nr}{\epsilon}\log\frac{n}{\epsilon} + \frac{nr}{\epsilon^2} + \frac{n}{\epsilon^4}\log^3\frac{1}{\epsilon})) = \mathcal{O}(nr\log^2 n)$ respectively, or $\mathcal{O}((\frac{1}{\epsilon}\log\frac{n}{\epsilon} + \frac{1}{\epsilon^2})\cdot\log r)$ and $\mathcal{O}(\frac{1}{\epsilon^2}\log\frac{1}{\epsilon}\log\frac{n}{\epsilon}((\frac{n}{\epsilon}\log\frac{n}{\epsilon} + \frac{n}{\epsilon^2})\cdot\log r + \frac{n}{\epsilon^4}\log^3\frac{1}{\epsilon})) = \mathcal{O}(n\log^2 n\log r)$ respectively. □

Algorithm 5: ParSSP$(p, \epsilon, f(\cdot))$

Input: probability p, precision $\epsilon \in (0,1)$, and submodular function $f(\cdot)$;

1 $T \leftarrow \emptyset$; $I \leftarrow \mathcal{N}$; $M \leftarrow \left\lceil \frac{\log_{1-\epsilon} \frac{\epsilon}{r} + 2}{\epsilon^2} \right\rceil$; $\ell \leftarrow \left\lceil \log_{1-\epsilon} \frac{\epsilon}{r} \right\rceil + 1$;
2 $u^* \leftarrow \arg\max_{u \in \mathcal{N}} f(u)$; $\rho_{max} \leftarrow f(u^*)$;
3 **for** $i \leftarrow 1$ **to** ℓ **do**
4 $\quad \rho_i \leftarrow \rho_{max} \cdot (1-\epsilon)^{i-1}$;
5 $\quad (A_i, U_i, L_i) \leftarrow$ RandBatch$(\rho_i, I, M, p, \epsilon, f_T(\cdot))$;
6 $\quad T \leftarrow T \cup A_i$; $I \leftarrow I \setminus (U_i \cup L_i)$;
7 **return** $S \leftarrow \arg\max_{X \in \{T, \{u^*\}\}} f(X)$;

4.3 The Algorithm for the SSP

In this section, we introduce ParSSP (as shown in Algorithm 5), which calls RandBatch using ℓ non-increasing thresholds $\rho_1, \ldots, \rho_\ell$ to find ℓ sequences of elements (i.e., $A_1, \ldots, A_\ell$), and then splices them together to get T (Lines 3–6). Note that the elements in U_j and L_j returned by RandBatch (for every $j \in [i]$) are all neglected when seeking for A_{i+1} (Line 6), which is useful for the performance analysis presented shortly. The final solution S returned by ParSSP is the best one among T and the single element in $\mathcal{N}$ with maximum objective function value (Line 7).

Now we begin to show the performance analysis of ParSSP, which is more involved than those of ParSKP, as ParSSP calls RandBatch with $p < 1$ to introduce additional randomness. We first introduce some definitions useful in our analysis. When ParSSP finishes, let $\mathcal{U} = \cup_{i=1}^{\ell} U_i, \mathcal{L} = \cup_{i=1}^{\ell} L_i$, $O_{small} = \{u \colon u \in O \setminus (\mathcal{U} \cup \mathcal{L}) \wedge f(u \mid T) \leq \rho_\ell\}$ and $O_{big} = O \setminus (O_{small} \cup \mathcal{U} \cup \mathcal{L})$. So each element $u \in O_{big}$ must satisfy $f(u \mid T) > \rho_\ell$ and $T \cup \{u\} \notin \mathcal{I}$.

Based on the definitions given above, we introduce a mapping $\Upsilon(\cdot)$ for performance analysis in Lemma 5. The intuition of this mapping is to map the elements in O_{big} to those in T, so that the utility loss resulting from excluding O_{big} from T can be bounded (as shown by Lemma 6).

Lemma 5. *There exists a mapping $\Upsilon : O_{big} \mapsto T$ satisfying:*

1. *Each $v \in O_{big}$ can be added into T without violating the k-system constraint at the moment that $\Upsilon(v)$ is added into T.*
2. *Let $\Upsilon^{-1}(u) = \{v \in O_{big} : \Upsilon(v) = y\}$ for any $u \in T$. Then we have $|\Upsilon^{-1}(u)| \leq k$.*

Lemma 6. *For any $u \in \mathcal{U}$, let $\rho(u)$ denote the threshold used by ParSSP when u is considered to be added into T and define $\rho(u) = 0$ for other $u \in \mathcal{N}$. For any $u \in \mathcal{U}$, we have*

1. *$f(u \mid T) \leq \rho(u)/(1-\epsilon)$;*
2. *and if $u \in T \wedge \Upsilon^{-1}(u) \neq \emptyset$, we can get $\forall v \in \Upsilon^{-1}(u) \colon f(v \mid T) \leq \rho(u)/(1-\epsilon)$.*

Proof of Lemma 5. Let $T = \{u_1, \cdots, u_s\}$ be a set where elements are ordered according to the order that they are added into T. Let $Z_s = O_{big}$ and construct the mapping by executing the following iterations from $t = s$ to $t = 0$. In iteration t, we first compute a set

$Q_t = \{x \in Z_t \setminus \{u_1, \cdots, u_{t-1}\} : \{u_1, \cdots, u_{t-1}, x\} \in \mathcal{I}\}$. If $|Q_t| \leq k$, then we set $R_t = Q_t$; otherwise, we select a subset $R_t \subseteq Q_t$ such that $|R_t| = k$. Next, we assign $\Upsilon(u) = u_t$ for each $u \in R_t$ and update $Z_{t-1} = Z_t \setminus R_t$. Then we proceed to iteration $(t-1)$.

It is clear that the $\Upsilon(\cdot)$ constructed by the above process satisfies Conditions 1-2. Therefore, we only need to show that every $u \in O_{big}$ is mapped to an element in T, which is equivalent to showing $Z_0 = \emptyset$ since each $u \in Z_s \setminus Z_0$ is mapped to an element in T by the above process. To prove $Z_0 = \emptyset$, we use induction and show that $|Z_t| \leq k \cdot t$ for all $0 \leq t \leq s$.

1. For $t = s$, let $M = T \cup O_{big}$. It is obvious that every element $u \in O_{big}$ satisfies $T \cup \{u\} \notin \mathcal{I}$ by the definition of O_{big}. Hence, we infer that T is a base of M. Since $O_{big} \in \mathcal{I}$, we obtain $|Z_s| = |O_{big}| \leq k|T| = k \cdot s$ by the definition of k-system.

2. For $0 \leq t < s$, assume that $|Z_t| \leq k \cdot t$ for some t. If $|Q_t| > k$, then we set $|R_t| = k$ and thus $|Z_{t-1}| = |Z_t| - k \leq k(t-1)$. If $|Q_t| \leq k$, then we observe that no element $u \in Z_{t-1} \setminus \{u_1, \cdots, u_{t-1}\}$ satisfies $\{u_1, \cdots, u_{t-1}\} \cup \{u\} \in \mathcal{I}$ due to the above process for constructing $\Upsilon(\cdot)$. Now consider the set $M' = \{u_1, \cdots, u_{t-1}\} \cup Z_{t-1}$, we see that $\{u_1, \cdots, u_{t-1}\}$ is a base of M' and $Z_{t-1} \in \mathcal{I}$, which implies $|Z_{t-1}| \leq k(t-1)$ by the definition of k-system.

By induction, we have shown that $Z_t \leq k \cdot t$ for all $0 \leq t \leq s$, which implies $Z_0 = \emptyset$. Therefore, the lemma is proved. □

Proof of Lemma 6. The lemma is trivial for $u \in U_1$ since $\rho(u) = \rho_{max}$. Next, suppose that $u \in U_i$ $(i > 1)$ and there exists $v \in \Upsilon^{-1}(u)$ such that $f(v \mid T) > \rho(u)/(1-\epsilon)$ (for a contradiction). Let T' be the set of elements in T when u is considered to be added to T. Then we get $T \setminus \cup_{t=i}^{\ell} U_t \subseteq T' \subseteq T$, which implies that v can be added to $T \setminus \cup_{t=i}^{\ell} U_t$ without violating the k-system constraint by Lemma 5 and the hereditary property of k-system. By submodularity, we obtain $f(v \mid T \setminus \cup_{t=i}^{\ell} U_t) > \rho(u)/(1-\epsilon)$, and thus $v \in \mathcal{L}$. But this contradicts the fact that $v \in O_{big}$. A similar line of reasoning can demonstrate that $f(u \mid T) \leq \rho(u)/(1-\epsilon)$. Combining all of the above completes the proof. □

By applying submodularity and Lemma 6, we obtain the following lemma:

Lemma 7. *For any $u \in \mathcal{N}$, let $X_u = 1$ if $u \in T$ and $X_u = 0$ otherwise; let $Y_u = 1$ if $u \in O \cap \mathcal{U} \setminus T$ and $Y_u = 0$ otherwise. When ParSSP finishes, we have*

$$f(T \cup O) \leq f(T) + f(O_{small} \mid T) + f(O_{big} \mid T) + f(\mathcal{L} \cap O \mid T)$$
$$+ f(O \cap \mathcal{U} \setminus T \mid T) \leq (1+\epsilon) f(S) + \epsilon f(O) + k \sum_{u \in \mathcal{N}} \frac{X_u \cdot \rho(u)}{1-\epsilon} + \sum_{u \in \mathcal{N}} \frac{Y_u \cdot \rho(u)}{1-\epsilon}.$$

Proof. When ParSSP finishes, the set $O \setminus T$ can be partitioned into several disjoint subsets: O_{small}, $\mathcal{L} \cap O$, O_{big} and $O \cap \mathcal{U} \setminus T$. By submodularity, we have

$$f(O \cup T) - f(T) \leq f(O_{small} \mid T) + f(O_{big} \mid T) + f(\mathcal{L} \cap O \mid T) + f(O \cap \mathcal{U} \setminus T \mid T). \quad (19)$$

Besides, using Lemma 1 and submodularity, we have

$$f(\mathcal{L} \cap O \mid T) \leq \sum\nolimits_{i=1}^{\ell} f(L_i \cap O \mid \cup_{j=1}^{i-1} A_i) \leq \epsilon^{-1} \cdot \ell f(O)/M \leq \epsilon f(O), \quad (20)$$

where the last inequality is due to $\ell \leq \log_{1-\epsilon} \frac{\epsilon}{r} + 2$ and $M \geq \frac{\log_{1-\epsilon} \frac{\epsilon}{r}+2}{\epsilon^2}$ according to Line 1 of Algorithm 5. According to the definition of O_{small}, we have

$$f(O_{small} \mid T) \leq \sum_{u \in O_{small}} f(u \mid T) \leq r \cdot \rho_\ell \leq \epsilon \cdot \rho_{max} \leq \epsilon f(S). \tag{21}$$

By Lemma 6 and submodularity, we have

$$f(O \cap \mathcal{U} \setminus T \mid T) \leq \sum_{u \in O \cap \mathcal{U} \setminus T} f(u \mid T) = \sum_{u \in O \cap \mathcal{U} \setminus T} \rho(u)/(1-\epsilon) = \sum_{u \in \mathcal{N}} Y_u \cdot \rho(u)/(1-\epsilon). \tag{22}$$

and

$$\begin{aligned} f(O_{big} \mid T) &\leq \sum_{u \in O_{big}} f(u \mid T) = \sum_{u \in T} \sum_{v \in \Upsilon^{-1}(u)} f(v \mid T) \\ &\leq \sum_{u \in T} \sum_{v \in \Upsilon^{-1}(u)} \rho(u)/(1-\epsilon) \leq \sum_{u \in T} k \cdot \rho(u)/(1-\epsilon) = k \sum_{u \in \mathcal{N}} X_u \cdot \rho(u)/(1-\epsilon). \end{aligned}$$

Combining all of the above completes the proof. □

Note that both $\rho(u)$ and $\Upsilon^{-1}(u)$ are random for any $u \in \mathcal{N}$, and the randomness is caused by both Line 12 of Algorithm 1 and the random selection in the GetSEQ function. So we study their expectation and get the following lemma:

Lemma 8. *We have*

$$\mathbb{E}[k \sum_{u \in \mathcal{N}} X_u \cdot \rho(u) + \sum_{u \in \mathcal{N}} Y_u \cdot \rho(u)] \leq (k + \frac{1-p}{p})(1-\epsilon)^{-2} \cdot \mathbb{E}[f(T)]$$

Proof. Recall that $\mathcal{U} = \cup_{i=1}^{\ell} U_i$, where U_i is generated in Line 5 of Algorithm 5. Suppose that $|\mathcal{U}| = h$, so we can create a random sequence $\{u_1, \ldots, u_h, u_{h+1}, \ldots, u_n\}$, where $\{u_1, u_2, \ldots, u_h\}$ are the elements in $\mathcal{U}$ listed according to the order that they are selected, and $\{u_{h+1}, \ldots, u_n\}$ are the elements in $\mathcal{N} \setminus \mathcal{U}$ listed in an arbitrary order. For any $i \in [n]$, let $\delta(u_i) = f(u_i \mid \{u_1, \ldots, u_{i-1}\} \cap T \cup \lambda(u_i))$ if $u_i \in T$ and otherwise $\delta(u_i) = 0$, where the function $\lambda(\cdot)$ has been defined in Lemma 2. So $f(T) \geq \sum_{i=1}^{n} \delta(u_i)$, and hence we only need to prove

$$\forall i \in [n] \colon \mathbb{E}[k \cdot X_{u_i} \cdot \rho(u_i) + Y_{u_i} \cdot \rho(u_i)] \leq (k + \frac{1-p}{p})(1-\epsilon)^{-2} \cdot \mathbb{E}[\delta(u_i)],$$

due to the linearity of expectation. Let $\mathcal{F}_{i-1}$ be the filtration capturing all the random choices made by ParSSP until the moment right before selecting u_i. According to the law of total expectation, it is sufficient to prove

$$\begin{aligned} \forall i \in [n], \forall \mathcal{F}_{i-1} \colon \quad & \mathbb{E}[k \cdot X_{u_i} \cdot \rho(u_i) + Y_{u_i} \cdot \rho(u_i) \mid \mathcal{F}_{i-1}] \\ & \leq (k + \frac{1-p}{p})(1-\epsilon)^{-2} \cdot \mathbb{E}[\delta(u_i) \mid \mathcal{F}_{i-1}]. \end{aligned} \tag{23}$$

Note that $\mathcal{F}_{i-1}$ determines whether $u_i \in \mathcal{U}$. Therefore, according to the definitions of $\rho(\cdot)$ and $\delta(\cdot)$, Eqn. (23) trivially holds under the case of $u_i \notin \mathcal{U}$ given $\mathcal{F}_{i-1}$. So in the sequel, we

only consider the case of $u_i \in \mathcal{U}$ given $\mathcal{F}_{i-1}$. By similar reasoning with that in Lemma 1, we can get:

$$\mathbb{E}[\delta(u_i) \mid \mathcal{F}_{i-1}] = p \cdot \mathbb{E}[f(u_i \mid \{u_1, \ldots, u_{i-1}\} \cap T \cup \lambda(u_i)) \mid \mathcal{F}_{i-1}].$$

Note that $\rho(u_i)$ is deterministic given $\mathcal{F}_{i-1}$, then we have

$$\begin{aligned}&\mathbb{E}[k \cdot X_{u_i} \cdot \rho(u_i) + Y_{u_i} \cdot \rho(u_i) \mid \mathcal{F}_{i-1}] = \rho(u_i) \cdot \mathbb{E}[k \cdot X_{u_i} + Y_{u_i} \mid \mathcal{F}_{i-1}] \\ &\leq \rho(u_i)(k \cdot p + 1 - p)\end{aligned}$$

where the inequality is due to the reason that u is accepted with probability of p and discarded with probability of $1 - p$. Besides, by similar reasoning with that in Lemma 2, we can get

$$\mathbb{E}[f(u_i \mid \{u_1, \ldots, u_{i-1}\} \cap T \cup \lambda(u_i)) \mid \mathcal{F}_{i-1}] \geq (1 - \epsilon)^2 \rho(u_i).$$

The proof is now complete by combining the above. □

Using Lemmas 7–8, we get the performance bounds of ParSSP as follows:

Theorem 4. *For the non-monotone SSP, ParSSP algorithm can return a solution T satisfying $\mathbb{E}[f(T)] \geq (1 - \epsilon)^5(\sqrt{k+1} + 1)^{-2} f(O)$ by setting $p = (1 + \sqrt{k+1})^{-1}$. The adaptive complexity and query complexity of ParSSP are $\mathcal{O}(\sqrt{k} \log^2 n)$ and $\mathcal{O}(\sqrt{k} nr \log^2 n)$ respectively, or $\mathcal{O}(\sqrt{k} \log^2 n \log r)$ and $\mathcal{O}(\sqrt{k} n \log^2 n \log r)$ respectively.*

Proof. We first quote the following lemma presented in (Buchbinder et al., 2014):

Lemma 9 ((Buchbinder et al., 2014))**.** *Given a ground set $\mathcal{N}$ and any non-negative submodular function $g(\cdot)$ defined on $2^{\mathcal{N}}$, we have $\mathbb{E}[g(Y)] \geq (1-p)g(\emptyset)$ if Y is a random subset of $\mathcal{N}$ such that each element in $\mathcal{N}$ appears in Y with probability of at most p (not necessarily independently).*

By combining Lemmas 7–8 and using the fact that $f(T) \leq f(S)$, we can get

$$\mathbb{E}[f(T \cup O)] \leq (1 + \epsilon)\mathbb{E}[f(S)] + \epsilon f(O) + (k + \frac{1-p}{p})(1 - \epsilon)^{-3} \cdot \mathbb{E}[f(S)].$$

Let $g(\cdot) = f(\cdot \cup O)$, then we can use Lemma 9 to get $\mathbb{E}[f(T \cup O)] \geq (1-p)f(O)$. Combining this with the above equation, and setting $p = (1 + \sqrt{k+1})^{-1}$, we get

$$\frac{\mathbb{E}[f(S)]}{f(O)} \geq \frac{1 - p - \epsilon}{1 + \epsilon + (k + \frac{1-p}{p})(1 - \epsilon)^{-3}} \geq (1 - \epsilon)^5(\sqrt{k+1} + 1)^{-2}$$

when $\epsilon \in [0, 0.4)$, which completes the proof on the approximation ratio.

In a manner similar to the proof of Theorem 3, we can employ Lemma 3 to analyze and determine the complexity of ParSSP. Then combining all of the above completes the proof. □

5. Extensions for Cardinality Constraint

As cardinality constraint is a special case of knapsack constraint and k-system constraint (where $k = 1$), our ParSKP and ParSSP algorithms can be directly applied to the non-monotone SMC, for which the performance bounds shown in Theorem 2 and Theorem 4 still hold. Interestingly, by a more careful analysis, we find that ParSSP actually achieves a better approximation ratio for the SMC, while its complexities remain the same. This result is shown in Theorem 5. We roughly explain the reason as follows. Since the cardinality constraint is more restrictive than the k-system constraint, we can use a more effective mapping to bound the utility loss caused by the elements in O_{big} by using only the elements in $T \setminus O$ instead of all the elements in T. This leads to stronger versions of Lemma 7 and Lemma 8, as shown by Lemma 10 and Lemma 11.

Lemma 10. *For any $u \in \mathcal{N}$, let $X'_u = 1$ if $u \in T \setminus O$ and $X'_u = 0$ otherwise; let $Y'_u = 1$ if $u \in O \cap \mathcal{U} \setminus T$ and $Y'_u = 0$ otherwise. When ParSSP finishes, we have*

$$f(T \cup O) \leq (1+\epsilon)f(S) + \epsilon f(O) + \sum_{u \in \mathcal{N}} \frac{X'_u \cdot \rho(u)}{1-\epsilon} + \sum_{u \in \mathcal{N}} \frac{Y'_u \cdot \rho(u)}{1-\epsilon}.$$

Proof. Using similar reasoning as Eqn. (19)-(22), we can get

$$f(T \cup O) \leq (1+\epsilon)f(S) + \epsilon f(O) + f(O_{big} \mid T) + \sum_{u \in \mathcal{N}} \frac{Y'_u \cdot \rho(u)}{1-\epsilon}.$$

Based on the cardinality constraint property, any element in O_{big} can be added to the candidate solution T without violating the constraint, provided that it is added prior to u_{last}, where u_{last} denotes the last added element in T. So according to the definition of O_{big} and submodularity, we must have

1. if $|T| < r$, then $|O_{big}| = 0$;
2. if $|T| = r$, then $\forall u \in O_{big} : f(u \mid T) \leq \rho(u_{last})/(1-\epsilon)$ and $|O_{big}| \leq |O \setminus \mathcal{U}| \leq |O| - |O \cap T| \leq r - |O \cap T| = |T| - |T \cap O| = |T \setminus O|$.

Thus, we can get

$$\begin{aligned} f(O_{big} \mid T) &\leq \sum_{u \in O_{big}} f(u \mid T) \leq \sum_{u \in O_{big}} \rho(u_{last})/(1-\epsilon) \\ \leq \sum_{u \in T \setminus O} \rho(u_{last})/(1-\epsilon) &\leq \sum_{u \in T \setminus O} \rho(u)/(1-\epsilon) = \sum_{u \in \mathcal{N}} X'_u \cdot \rho(u)/(1-\epsilon) \end{aligned}$$

The proof now completes by combining the above. □

Lemma 11. *We have*

$$\mathbb{E}[\sum_{u \in \mathcal{N}} X'_u \cdot \rho(u) + \sum_{u \in \mathcal{N}} Y'_u \cdot \rho(u)] \leq (1-\epsilon)^{-2} \cdot \frac{\max\{p, 1-p\}}{p} \cdot \mathbb{E}[f(T)]$$

Proof. By following a similar argument as in the proof of Lemma 8, we only need to prove

$$\forall i \in [n], \forall \mathcal{F}_{i-1}: \qquad \mathbb{E}[X'_{u_i} \cdot \rho(u_i) + Y'_{u_i} \cdot \rho(u_i) \mid \mathcal{F}_{i-1}] \leq (1-\epsilon)^{-2} \cdot \frac{\max\{p, 1-p\}}{p} \cdot \mathbb{E}[\delta(u_i) \mid \mathcal{F}_{i-1}].$$

under the case of $u_i \in \mathcal{U}$ given $\mathcal{F}_{i-1}$ to prove this lemma. Consider the following two scenarios.

1. $u_i \in O$. In this case, we must have $X'_{u_i} = 0$, and $Y'_{u_i} = 1$ if u_i is discarded by the algorithm. Therefore, $\mathbb{E}[X_{u_i} + Y_{u_i} \mid \mathcal{F}_{i-1}] = 1 - p$.
2. $u \notin O$: In this case, we must have $Y'_{u_i} = 0$, and $X'_{u_i} = 1$ if u_i is not discarded by the algorithm. Therefore, $\mathbb{E}[X_{u_i} + Y_{u_i} \mid \mathcal{F}_{i-1}] = p$.

Since $\rho(u_i)$ is deterministic given $\mathcal{F}_{i-1}$, we have

$$\mathbb{E}[X_{u_i} \cdot \rho(u_i) + Y_{u_i} \cdot \rho(u_i) \mid \mathcal{F}_{i-1}] = \rho(u_i) \cdot \mathbb{E}[X_{u_i} + Y_{u_i} \mid \mathcal{F}_{i-1}] \leq \rho(u_i) \cdot \max\{p, 1-p\}.$$

Recall that

$$\mathbb{E}[\delta(u_i) \mid \mathcal{F}_{i-1}] = p \cdot \mathbb{E}[f(u_i \mid \{u_1, \ldots, u_{i-1}\} \cap T \cup \lambda(u_i)) \mid \mathcal{F}_{i-1}] \geq (1-\epsilon)^2 \cdot p \cdot \rho(u_i).$$

The proof now completes by combining the above. □

Using Lemma 10 and Lemma 11, we can improve the approximation ratio of ParSSP for the non-monotone SMC, as stated by Theorem 5.

Theorem 5. *For the non-monotone SMC, ParSSP can return a solution S satisfying $\mathbb{E}[f(S)] \geq (1/4 - \epsilon)\mathrm{OPT}$ by setting $p = 1/2$, under the same adaptivity and query complexity as those shown in Theorem 4 where $k = 1$.*

Proof. By combining Lemmas 10–11 and Lemma 9 and using the fact that $f(T) \leq f(S)$, we get

$$(1-p)f(O) \leq (1+\epsilon)\mathbb{E}[f(S)] + \epsilon f(O) + (1-\epsilon)^{-3} \cdot \frac{\max\{p, 1-p\}}{p} \cdot \mathbb{E}[f(S)],$$

By setting $p = 1/2$ and rearranging the above inequality, we derive the approximation ratio of ParSSP as

$$\frac{\mathbb{E}[f(S)]}{f(O)} \geq \frac{1/2 - \epsilon}{1 + \epsilon + (1-\epsilon)^{-3}} \geq \frac{1}{4} - \epsilon.$$

Finally, the complexity analysis is the same with that in Theorem 4 where $k = 1$. □

6. Performance Evaluation

In this section, we evaluate the performance of our algorithm ParSKP (resp. ParSSP) by comparing it with state-of-the-art algorithms for the non-monotne SKP (resp. SSP). The evaluation metrics include both the objective function value (i.e., utility) and the number of oracle queries to the objective function. We conduct experiments on three real-world applications, which are described in detail below.

6.1 Applications

Revenue Maximization. This application is also considered in (Mirzasoleiman et al., 2016; Balkanski et al., 2018; Fahrbach et al., 2019a; Han et al., 2021; Cui et al., 2021; Amanatidis et al., 2021, 2022). In this problem, we are given a social network $G = (\mathcal{N}, E)$ where each node $u \in \mathcal{N}$ represents a user with a cost $c(u)$, and each edge $(u, v) \in E$ has a weight $w_{u,v}$ denoting the influence of u on v. There are also t advertisers for different products; each advertiser $i \in [t]$ needs to select a subset $S_i \subseteq V$ of seed nodes and provide a sample of product i to each user $u \in S_i$ (and also pay $c(u)$ to u) for advertising product i. Following (Mirzasoleiman et al., 2016; Balkanski et al., 2018; Fahrbach et al., 2019a; Amanatidis et al., 2021, 2020), we define the revenue of advertiser i as $f_i(S_i) = \sum_{u \in \mathcal{N} \setminus S_i} \sqrt{\sum_{v \in S_i} w_{v,u}}$ (which measures the total influence of the seed nodes on the non-seed nodes) and define the objective function of the application as $\sum_{i \in [t]} f_i(S_i)$ (i.e., the total revenue of all advertisers). This function has already been shown by (Mirzasoleiman et al., 2016; Amanatidis et al., 2020; Balkanski et al., 2018; Fahrbach et al., 2019a; Amanatidis et al., 2021) to be a non-monotone submodular function. We also follow the existing work to consider the following constraints to better model the demands of real-world scenarios: 1) each node can serve as a seed node for at most q products; 2) the total number of product samples available for each product is at most m; and 3) the total seed cost is bounded by B (i.e., $\sum_{i \in [t]} \sum_{u \in S_i} c(u) \leq B$). The first two constraints constitute a k-system constraint (where $k = 2$), which has been demonstrated by (Mirzasoleiman et al., 2016; Cui et al., 2021). The third constraint is a conventional knapsack constraint. The edge weights are randomly sampled from the continuous uniform distribution $\mathcal{U}(0, 1)$, and the cost of any node $u \in \mathcal{N}$ is defined as $c(u) = g(\sqrt{\sum_{(u,v) \in E} w_{u,v}})$, where $g(x) = 1 - e^{-\mu}$ is the exponential cumulative distribution function and μ is set to 0.2.

Image Summarization. This application is also considered in (Mirzasoleiman et al., 2016; Balkanski et al., 2018; Fahrbach et al., 2019a; Han et al., 2021; Cui et al., 2024), the goal is to select a representative set S of images from $\mathcal{N}$. Following (Mirzasoleiman et al., 2016; Balkanski et al., 2018; Fahrbach et al., 2019a), we use a non-monotone submodular function $f(\cdot)$ that can capture both coverage and diversity of S to measure its quality:

$$f(S) = \sum_{u \in \mathcal{N}} \max_{v \in S} s_{u,v} - \frac{1}{|\mathcal{N}|} \sum_{u \in S} \sum_{v \in S} s_{u,v},$$

where $s_{u,v}$ denotes the cosine similarity between image u and image v. Thus, this application is in fact a non-monotone submodular maximization problem. Following (Mirzasoleiman et al., 2016; Han et al., 2021), the cost $c(u)$ of any image u is chosen in proportional to the standard deviation of its pixel intensities, such that we assign higher costs to images with higher contrast and lower costs to blurry images. The costs of all images are normalized such that the average cost is 1. In addition, we restrict the number of images in S that belong to each category to no more than q, and we limit the total number of images in S to no more than m. It has been indicated in (Mirzasoleiman et al., 2016) that such a constraint is a matroid (i.e., 1-system) constraint.

Movie Recommendation. This application is also considered in (Mirzasoleiman et al., 2016; Feldman et al., 2017, 2023; Haba et al., 2020; Badanidiyuru et al., 2020; Amanatidis et al., 2022; Cui et al., 2023). In this problem, we consider a set $\mathcal{N}$ of movies, each labeled

by several genres chosen from a predefined set G, and aim to recommend a list of high-quality and diverse movies to a user based on the ratings from similar users. Each movie $u \in \mathcal{N}$ is associated with a 25-dimensional feature vector q_u calculated from user ratings. Following (Mirzasoleiman et al., 2016; Feldman et al., 2017; Haba et al., 2020; Amanatidis et al., 2020; Balkanski et al., 2018; Fahrbach et al., 2019a), we define the utility of any $S \subseteq \mathcal{N}$ as

$$f(S) = \sum\nolimits_{u \in S} \sum\nolimits_{v \in \mathcal{N}} s_{u,v} - \sum\nolimits_{u \in S} \sum\nolimits_{v \in S} s_{u,v},$$

where we use $s_{u,v} = e^{-\lambda \mathrm{dist}(q_u, q_v)}$ to measure the similarity between movies u and v. Thus, this application is in fact a non-monotone submodular maximization problem. Here, $\mathrm{dist}(q_u, q_v)$ is the Euclidean distance between q_u and q_v, and λ is set to 2. Following (Haba et al., 2020), we define the cost $c(u)$ of any movie u to be proportional to $10 - r_u$, where r_u denotes the rating of movie u (ranging from 0 to 10), and the costs of all movies are normalized such that the average movie cost is 1. Thus, movies with higher ratings have smaller costs, and we require $\sum_{u \in S} c(u) \leq B$ to ensure that the movies in S have high ratings. Moreover, we also consider the constraint that the number of movies in S labeled by genre g is no more than m_g for all $g \in G$, and that $|S| \leq m$, where $m_g : g \in G$ and m are all predefined integers. It has been indicated in (Mirzasoleiman et al., 2016; Feldman et al., 2017; Haba et al., 2020) that such a constraint is essentially a k-system constraint with $k = |G|$.

6.2 Experiments for the SKP

In this section, we compare our ParSKP algorithm with several state-of-the-art *practical* algorithms for the non-monotone SKP. Specifically, we implement four algorithms: (1) ParSKP (Algorithm 3) using binary search; (2) ParKnapsack (Amanatidis et al., 2023) using binary search; (3) SampleGreedy (Amanatidis et al., 2022), which gives a $(3 - 2\sqrt{2} - \epsilon)$-approximation using $\mathcal{O}(\frac{n}{\epsilon} \log \frac{n}{\epsilon})$ adaptive rounds and queries, implemented using *lazy evaluation* (Minoux, 1978); (4) SmkRanAcc (Han et al., 2021), which gives a $(0.25 - \epsilon)$-approximation using $\mathcal{O}(\frac{n}{\epsilon} \log \frac{r}{\epsilon})$ adaptive rounds and queries. Note that both SampleGreedy and SmkRanAcc are non-parallel algorithms with super-linear adaptivity, so we use these two baselines only to see how other algorithms can approach them. For all the algorithms tested, the accuracy parameter ϵ is set to 0.1. Each randomized algorithm is executed independently for 10 times, and the average result is reported. For the fairness of comparison, we follow (Amanatidis et al., 2023, 2021) to use the algorithm in (Feige et al., 2011) achieving 1/4-approximation and $\mathcal{O}(1)$ adaptivity for the USM algorithm. All experiments are run on a Linux server with Intel Xeon Gold 6126 @ 2.60GHz CPU and 256GB memory. We transform the three real-world applications introduced in Section 6.1 into SKPs and evaluate all implemented algorithms on these problems. The modifications we made and additional experimental settings are as follows:

- Revenue Maximization. By setting $t = 1$ and $q = m = \infty$, we remove the k-system constraint from this application and transform it into the SKP considered by (Amanatidis et al., 2021, 2022; Pham et al., 2023; Han et al., 2021). Following (Mirzasoleiman et al., 2016), we use the top 5,000 communities of the YouTube network (Leskovec &

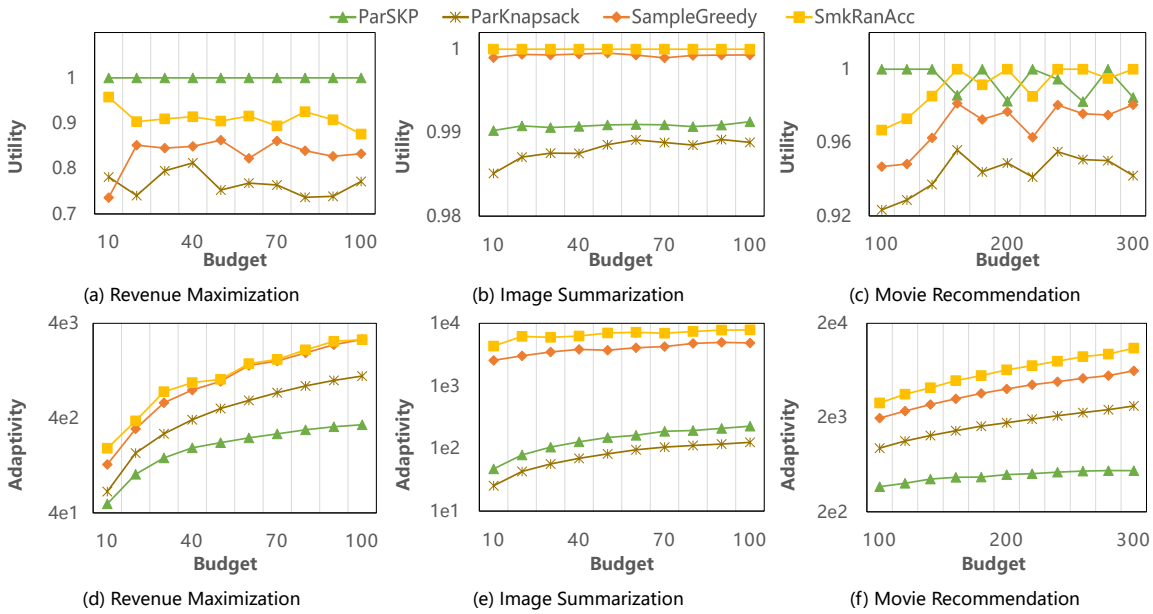


Figure 1: The figure compares the implemented algorithms on utility and adaptivity, where the plotted utilities are normalized by the largest utility achieved by all algorithms.

Krevl, 2014) to construct the network G, which contains 39,841 nodes and 224,235 edges.

- Image Summarization. By setting $q = m = \infty$, we remove the k-system constraint from this application and transform it into the SKP considered by (Pham et al., 2023; Han et al., 2021; Cui et al., 2022). Following (Balkanski et al., 2018; Fahrbach et al., 2019a; Han et al., 2021), we randomly select 1,000 images from the CIFAR-10 dataset (Krizhevsky & Hinton, 2009) to construct $\mathcal{N}$.

- Movie Recommendation. By setting $m = \infty$ and $\forall g \in G : m_g = \infty$, we remove the k-system constraint from this application and transform it into the SKP considered by (Amanatidis et al., 2022). In our experiments, We use the MovieLens dataset (Badanidiyuru et al., 2020; Haba et al., 2020) which contains 1,793 movies from three genres "Adventure", "Animation" and "Fantasy".

Experimental Results. In Fig. 1(a)–(c), we compare the implemented algorithms on utility, and the results show ParSKP can even achieve better utility compared to non-parallel algorithms SampleGreedy and SmkRanAcc, with the average performance gains of 5% and 3%, respectively. Besides, Fig. 1(a)–(c) also show that ParSKP achieves significantly better utility than ParKnapsack (with the performance gains of up to 35.74%), which is the only existing low-adaptivity algorithm for non-monotone SKP with sub-linear adaptivity and practical query complexity. In Fig. 1(d)–(f), we compare the implemented algorithms on adaptivity, and the results show that ParSKP generally outperforms all the baselines.

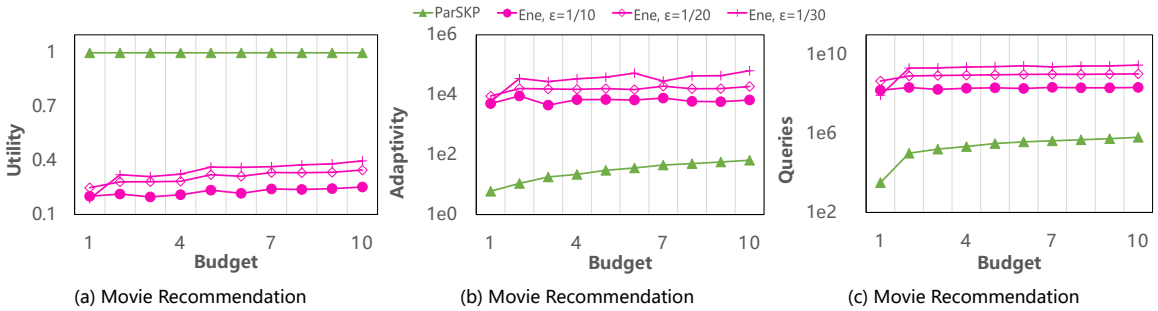


Figure 2: Comparison of our algorithms with the ENE algorithm (Ene et al., 2019) on a small instance of movie recommendation. Similar to Figure 1, the plotted utilities are normalized by the best utility achieved by the implemented algorithms.

Specifically, ParSKP incurs 4–92 times fewer adaptive rounds than SmkRanAcc, and 3–54 times fewer adaptive rounds than SampleGreedy, which demonstrates the effectiveness of our approach.

6.3 Additional Experiments for the SKP

Besides the algorithms mentioned in Section 6.2, we note that (Ene et al., 2019) also propose an algorithm (denoted by ENE for convenience) with $\mathcal{O}(\log^2 n)$ adaptivity. However, as indicated by (Amanatidis et al., 2021; Fahrbach et al., 2019a), the ENE algorithm (based on multi-linear extension) has large query complexity impractical for large datasets, so we compare our ParSKP algorithm with ENE using a small instance of the movie recommendation application, under the same settings as those described in Section 6.2 except that the ground set size is smaller ($n = 80$). For the ENE algorithm, we use 5,000 samples to simulate an oracle for $F(\cdot)$ or $\nabla F(\cdot)$ (i.e., the multi-linear extension of $f(\cdot)$ and its gradient).

As shown by the experimental results in Fig. 2, our ParSKP algorithm outperforms ENE significantly in terms of utility, adaptivity and the number of oracle queries to the objective function, due to the reason that: ENE has a larger theoretical adaptivity of $\mathcal{O}(\log^2 n)$ than ParSKP, and its practical performance is not much better than its worst-case theoretical bound due to its design, while evaluating the multilinear extensions in ENE incurs significantly larger number of oracle queries than our algorithms. Moreover, as the approximation ratio of ENE (i.e., $(1+\epsilon)(e^{1+10\epsilon})$) deteriorates quickly with the increasing of ϵ, we have used smaller values of ϵ (i.e., $\epsilon = 1/20; \epsilon = 1/30$) to further test its utility. The experimental results in Fig. 2 show that our ParSKP algorithm consistently outperforms ENE on all these settings, which demonstrate the effectiveness of our approach again.

6.4 Experiments for the SSP

In this section, we compare our ParSSP algorithm with several state-of-the-art algorithms for the non-monotone SSP. Specifically, we implement four algorithms: (1) ParSSP (Algorithm 5) using binary search; (2) RepSampling (Quinzan et al., 2021), which may not provide a constant approximation as we explained in Appendix B; (3) TwinGreedy (Han

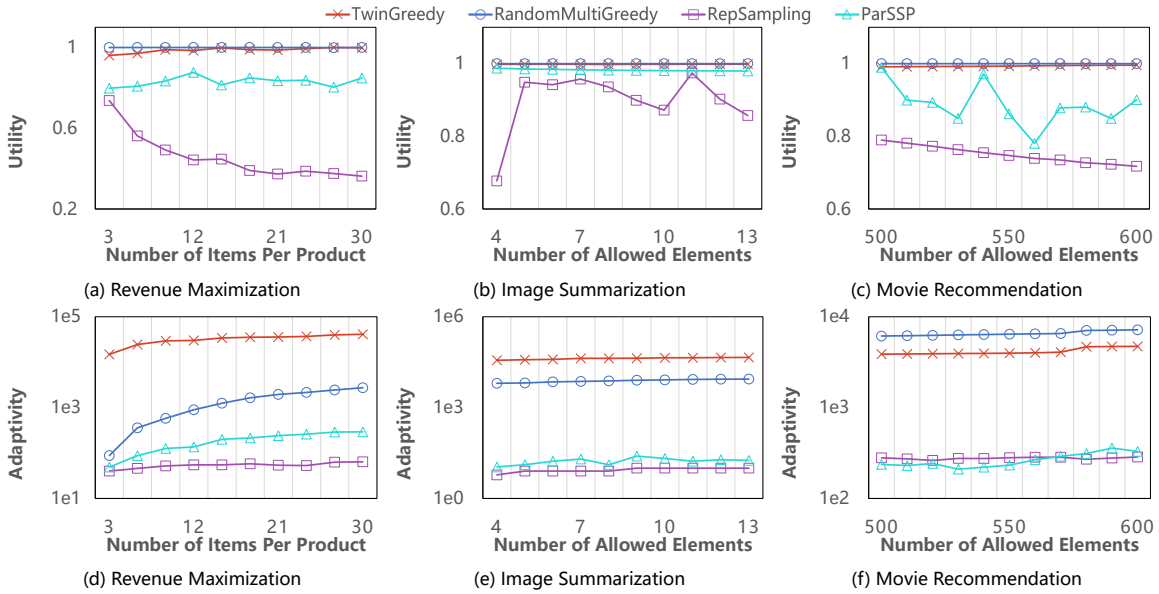


Figure 3: The figure compares the implemented algorithms on utility and adaptivity, where the plotted utilities are normalized by the largest utility achieved by all algorithms

et al., 2020), which gives a $((2k+2)^{-1}-\epsilon)$-approximation using $\mathcal{O}(\frac{n}{\epsilon}\log\frac{r}{\epsilon})$ adaptive rounds and queries; (4) RandomMultiGreedy (Cui et al., 2021), which gives a $(1+\epsilon)^{-1}(\sqrt{k}+1)^{-2}$-approximation using $\mathcal{O}(\frac{n}{\epsilon}\log\frac{r}{\epsilon})$ adaptive rounds and queries. Note that both TwinGreedy and RandomMultiGreedy are non-parallel algorithms with super-linear adaptivity, so we use these two baselines only to see how other algorithms can approach them. For all the algorithms tested, the accuracy parameter ϵ is set to 0.4. Each randomized algorithm is executed independently for 10 times, and the average result is reported. All experiments are run on a Linux server with Intel Xeon Gold 6126 @ 2.60GHz CPU and 256GB memory. We transform the three real-world applications introduced in Section 6.1 into SSPs and evaluated all implemented algorithms on these problems. The modifications we made and additional experimental settings are as follows:

- Revenue Maximization. By setting $\forall u \in \mathcal{N} : c(u) = 1$ and $B = \infty$, we remove the knapsack constraint from this application and transform it into the SSP considered by (Cui et al., 2021). Following (Balkanski et al., 2018; Fahrbach et al., 2019a; Han et al., 2021), we randomly select 25 communities from the top $5,000$ communities in the YouTube social network (Leskovec & Krevl, 2014) to construct the network G, which contains 1,179 nodes and 3,495 edges. We set $t = 5$ and $q = 2$ while scaling the number of items available for seeding (i.e., m) to compare the performance of all algorithms.

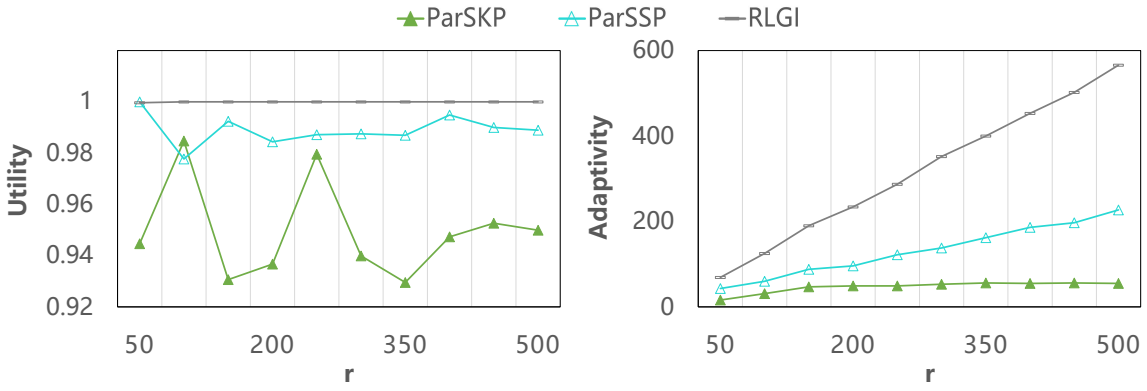
- Image Summarization. By setting $\forall u \in \mathcal{N} : c(u) = 1$ and $B = \infty$, we remove the knapsack constraint from this application and transform it into the SSP considered by (Cui et al., 2021). Following (Mirzasoleiman et al., 2016; Cui et al., 2021), we use

the CIFAR-10 dataset (Krizhevsky & Hinton, 2009) to construct $\mathcal{N}$, and restrict the selection of images from three categories: Airplane, Automobile and Bird. We set $q = 5$ while scaling m to compare the performance of all algorithms.

- Movie Recommendation. By setting $\forall u \in \mathcal{N} : c(u) = 1$ and $B = \infty$, we remove the knapsack constraint from this application and transform it into the SSP considered by (Feldman et al., 2017, 2023; Haba et al., 2020; Cui et al., 2021). We use the MovieLens dataset (Badanidiyuru et al., 2020; Haba et al., 2020) which contains 1,793 movies from three genres "Adventure", "Animation" and "Fantasy", and thus we have $k = 3$ in our experiments. We scale m to compare the performance of all algorithms.

Experimental Results. In Fig. 3(a)–(c), the utility performance of ParSSP is slightly weaker than the two non-parallel algorithms TwinGreedy and RandomMultiGreedy (with the average performance loss of 10%). Besides, Fig. 3(a)–(c) also show that ParSSP achieves significantly better utility than RepSampling (with the performance gains of up to 134%), which is the only existing low-adaptivity algorithm for non-monotone SKP. In Fig. 3(d)–(f), we compare the implemented algorithms on adaptivity, and the results indicate that our algorithm ParSSP performs comparably to RepSampling and significantly better than non-parallel algorithms TwinGreedy and RandomMultiGreedy. Specifically, ParSSP incurs 13–3327 times fewer adaptive rounds than TwinGreedy, and 2–582 times fewer adaptive rounds than RandomMultiGreedy, which demonstrates the effectiveness of our approach.

6.5 Experiments for the SMC



(a) Movie Recommendation (b) Movie Recommendation

Figure 4: The figure compares the implemented algorithms on utility and adaptivity, where the plotted utilities are normalized by the largest utility achieved by all algorithms

In this section, we test the performance of our ParSKP and ParSSP algorithms on SMC problems. The baseline we use is the Random Lazy Greedy Improved algorithm (abbreviated as RLGI) proposed by (Buchbinder et al., 2015), which is the state-of-the-art non-parallel algorithm with optimal approximation ratio and super-linear adaptivity for the non-monotone

SMC. We conduct this comparison to observe how closely our algorithms can match the utility of the algorithm with optimal approximation ratios in practice, while also highlighting the efficiency advantages of our algorithms over non-parallel algorithms. For all the algorithms tested, the accuracy parameter ϵ is set to 0.1. Each randomized algorithm is executed independently for 10 times, and the average result is reported. Following (Amanatidis et al., 2023, 2021), we employ the USM (Unconstrained Submodular Maximization) algorithm from (Feige et al., 2011), which achieves a 1/4-approximation and $\mathcal{O}(1)$ adaptivity, as a subroutine of our algorithms. Besides, our algorithms are implemented using binary search. All experiments are run on a Linux server with Intel Xeon Gold 6126 @ 2.60GHz CPU and 256GB memory. Since our algorithms' contribution to the SMC problem is merely a by-product, we simplify our experiments and test the implemented algorithms on only one application. The specific experimental settings are as follows.

- Movie Recommendation. By setting $m = \infty$, $\forall g \in G : m_g = \infty$ and $\forall u \in \mathcal{N} : c(u) = 1$, we remove the k-system constraint and transform the knapsack constraint as the cardinality constraint. Thus this application now is an SMC. In our experiments, We use the MovieLens dataset (Badanidiyuru et al., 2020; Haba et al., 2020) which contains 1,793 movies from three genres "Adventure", "Animation" and "Fantasy". We scale the maximum cardinality of any feasible solution (i.e., r) to compare the performance of all algorithms.

Experimental Results. In Fig. 4(a), the results show that ParSKP (resp. ParSSP) achieves 93%-98% (resp. 98%-100%) of the utility of RLGI. In Fig. 4(b), we compare the implemented algorithms on adaptivity, and the results show that ParSKP incurs 2-4 times fewer rounds than ParSSP, and 4-10 times fewer rounds than RLGI. The above experimental results demonstrate that our algorithms can achieve utility performance comparable to that of the algorithm with optimal approximation ratio using much fewer adaptive rounds.

7. Conclusions

In this paper, we propose ParSKP, a low-adaptivity algorithm that provides an $(1/8 - \epsilon)$ approximation ratio for the problem of non-monotone submodular maximization subject to a knapsack constraint (SKP). To the best of our knowledge, our ParSKP algorithm is the first of its kind to achieve either near-optimal $\mathcal{O}(\log n)$ adaptive complexity or near-optimal $\tilde{\mathcal{O}}(n)$ query complexity for the non-monotone SKP. Furthermore, we propose ParSSP, a low-adaptivity algorithm that provides an $(1-\epsilon)^5(\sqrt{k+1}+1)^{-2}$ approximation ratio for the problem of non-monotone submodular maximization subject to a k-system constraint (SSP). Our ParSSP algorithm is the first of its kind to achieve sublinear adaptive complexity for the non-monotone SSP. Additionally, we demonstrate that our two algorithms can be extended to solve the problem of submodular maximization subject to a cardinality constraint, achieving performance bounds that are comparable to those of existing state-of-the-art algorithms. During our literature review, we identified and discussed theoretical analysis errors presented in several related studies. Finally, we have validated the effectiveness of our algorithms through extensive experimentation on real-world applications, including revenue maximization, movie recommendation, and image summarization.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China (NSFC) under Grant No. 62172384, and the Alibaba Group through Alibaba Innovative Research Program.

Appendix A. A Subtle Issue in (Amanatidis et al., 2021, Theorem 1)

In Theorem 1 of (Amanatidis et al., 2021), an approximation ratio of $(3-\sqrt{3})/12-\Theta(\epsilon)$ is proved for their Algorithm 3 (i.e., ParKnapsack). In their Algorithm 3, they first randomly delete each element from the ground set $\mathcal{N}$ with probability of $1-p$ to get a new ground set H, and then use H to run a procedure ThreshSeq for multiple times, and finally return S, which is the result of one run of ThreshSeq. Note that there are two randomnesses in their algorithm: the randomness for generating H, and the randomness for generating S by ThreshSeq given a fixed H. The derivation of their approximation ratio is based on the analysis of two events:

- $\mathbb{E}[c(S)] < (1-\hat{\varepsilon})\frac{B}{2}$ given a fixed H. For convenience, let us denote this event by $\mathcal{H}_<$ and put it in a more clear form, i.e., $\mathcal{H}_< = \{\mathbb{E}_S[c(S) \mid H] < (1-\hat{\varepsilon})\frac{B}{2}\}$.
- $\mathbb{E}[c(S)] \geq (1-\hat{\varepsilon})\frac{B}{2}$ given a fixed H. For convenience, let us denote this event by $\mathcal{H}_\geq$ and put it in a more clear form, i.e., $\mathcal{H}_\geq = \{\mathbb{E}_S[c(S) \mid H] \geq (1-\hat{\varepsilon})\frac{B}{2}\}$.

Note that the distribution of H making the event $\mathcal{H}_<$ (or $\mathcal{H}_\geq$) happen is **different** from the original distribution of H where each element in $\mathcal{N}$ appears in H with a probability of p. In a nutshell, the $(3-\sqrt{3})/12-\Theta(\epsilon)$-ratio of (Amanatidis et al., 2021) is derived by: (Step I) Deriving $\mathbb{E}[f(S\cup O_H)] \geq p(1-p)f(O)$; (Step II) Deriving an upper bound of $\mathbb{E}[f(S\cup O_H)]$ using ALG; and (Step III) Using Step I and Step II to bridge ALG and $f(O)$, where $O_H = O\cap H$. The subtle problem lying in their analysis is that, the two $\mathbb{E}[f(S\cup O_H)]$'s they use in Step I and Step II are actually different: the one in Step I considers all randomness and hence the original distribution of H, while the one in Step II is actually conditioned on $\mathcal{H}_<$ and hence only considers a biased distribution of H. Therefore, Step III cannot be done. In the sequel, we explain this in more detail.

In the proof of Theorem 1 by (Amanatidis et al., 2021, pp. 6–7) (see their Section 3), $f(S\cup O_H)$ is used to build the connection between OPT and ALG. When analyzing the case $\mathbb{E}[c(S)] < (1-\hat{\varepsilon})\frac{B}{2}$ given a fixed H (i.e., event $\mathcal{H}_<$ occurs), a more careful analysis, via their Lemmata 5 and 6, is conducted (starting from the line just below Eq. (6) in the right column of page 6). Then, it shows that keeping fixed H,

$$\mathbb{E}[f(S\cup O_H) \mid \mathcal{G}] \leq (1+\hat{\varepsilon}+q)ALG + \tau c(O_H) - q\tau\tfrac{B}{2}, \text{ (first equation in their page 7; (a))}$$

where the event $\mathcal{G}$ is defined such that at least one of the parallel runs of ThreshSeq outputs S with $c(S) < \frac{B}{2}$ and that solution is considered. Subsequently, it shows that, "move on to the expectation with respect to H",

$$\begin{aligned}\mathbb{E}[f(S\cup O_H)] &= \mathbb{E}[f(S\cup O_H) \mid \mathcal{G}]\mathbb{P}(\mathcal{G}) + \mathbb{E}[f(S\cup O_H) \mid \mathcal{G}^C]\mathbb{P}(\mathcal{G}^C)\\ &\leq (1+\hat{\varepsilon}+q)ALG + \tau pc(O) - q\tau\tfrac{B}{2} + 2\hat{\varepsilon}f(O).\end{aligned}$$
(second equation in their page 7; (b))

Combing that with their Equation (5), which is copied as follows

$$p(1-p)f(O) \leq \mathbb{E}[f(S \cup O_H)], \qquad \text{(their Equation (5) in page 6; (c))}$$

they finally obtain

$$f(O) \leq \frac{1+q+\hat{\varepsilon}}{p(1-p)-\alpha p+\frac{\alpha q}{2}-2\hat{\varepsilon}} ALG. \qquad \text{(their Equation (7) in page 7; (d))}$$

Unfortunately, there is a gap in this claim by abusing the expectations. Because Eqn. (a) and Eqn. (b) listed above only hold when event $\mathcal{H}_<$ occurs according to their reasoning, while from Eqn.(a) to Eqn. (b) they never "move on to the expectation with respect to H" as they never bound $\mathbb{E}[f(S \cup O_H)]$ under the event $\mathcal{H}_\geq$, nor $\mathbb{P}(\mathcal{H}_<)$ and $\mathbb{P}(\mathcal{H}_\geq)$. To explain this more clearly, we give the full expression of the above equations. Recall that Eqn. (a) and Eqn. (b) only hold when $\mathcal{H}_<$ occurs. Then,

$$\begin{aligned}
&\mathbb{E}_H\Big[\mathbb{E}_S[f(S \cup O_H) \mid H] \;\Big|\; \mathcal{H}_<\Big] \\
&= \mathbb{E}_H\Big[\mathbb{E}_S[f(S \cup O_H) \,|\, H, \mathcal{G}]\mathbb{P}(\mathcal{G} \,|\, H) + \mathbb{E}_S[f(S \cup O_H) \,|\, H, \mathcal{G}^C]\mathbb{P}(\mathcal{G}^C \,|\, H) \;\Big|\; \mathcal{H}_<\Big] \\
&\leq \mathbb{E}_H\Big[(1+\hat{\varepsilon}+q)ALG + \tau c(O_H) - q\tau\tfrac{B}{2} + 2\hat{\varepsilon} f(O) \;\Big|\; \mathcal{H}_<\Big]. \qquad (24)
\end{aligned}$$

On the other hand, their Equation (5) (i.e. Equation (c) listed above) can be fully expressed as

$$p(1-p)f(O) \leq \mathbb{E}_H\Big[\mathbb{E}_S[f(S \cup O_H) \mid H]\Big] = \mathbb{E}_H\Big[\mathbb{E}_S[f(S \cup O_H \mid H)] \;\Big|\; \mathcal{H}_\geq \cup \mathcal{H}_<\Big]. \qquad (25)$$

Note that the correctness of Equation (25) is ensured by the fact that elements belong to H with probability p. However, when $\mathcal{H}_<$ occurs, it is not guaranteed that the *conditional* probability of every element belonging to H is p (or upper bounded by p). As a result, it is invalid to claim

$$p(1-p)f(O) \overset{?}{\leq} \mathbb{E}_H\Big[\mathbb{E}_S[f(S \cup O_H) \mid H] \;\Big|\; \mathcal{H}_<\Big]. \qquad \text{(invalid claim)}$$

Putting it together, we cannot directly establish the relation between $p(1-p)f(O)$ and $\mathbb{E}_H[(1+\hat{\varepsilon}+q)ALG+\tau c(O_H)-q\tau\frac{B}{2}+2\hat{\varepsilon}f(O) \mid \mathcal{H}_<]$ using Equations (24) and (25). Therefore, the claim of their Equation (7) is invalid.

Another similar issue applies to the use of $\mathbb{E}[O_H] = pc(O) \leq pB$ (in their page 7). Note that in Equation (24), what we actually need is $\mathbb{E}_H[c(O_H) \mid \mathcal{H}_<]$. However, in general, it is trivial to see that

$$\mathbb{E}_H[c(O_H) \mid \mathcal{H}_<] \neq \mathbb{E}_H[c(O_H)] = pc(O),$$

since when $\mathcal{H}_<$ occurs, it is not guaranteed that the conditional probability of every element belonging to H is p.

The same issues as explained above also exist in (Amanatidis et al., 2021, Theorem 4), where they claim a $(3-2\sqrt{2}-\epsilon)$-approximation under $\mathcal{O}(\log n)$ adaptivity for the problem of non-monotone submodular maximization with a cardinality constraint. The reason is that their Theorem 4 uses almost identical methods and analysis as those in their Theorem 1 (see their Appendix D).

Appendix B. Abuse of Markov's Inequality for Non-monotone Submodular Optimization Problems

(Quinzan et al., 2021) recently studied the problem of non-monotone submodular maximization subject to a k-system constraint. However, their analysis contains flaws, as explained below. In the proof of (Quinzan et al., 2021, Lemma 7) (see page 13 in Appendix of their full version at arXiv:2102.06486v1), the following claim is incorrect:

$$\mathbb{E}_{a_i}[f(a_i \mid \{a_1, \ldots, a_{i-1}\})] \geq \Pr[f(a_i \mid \{a_1, \ldots, a_{i-1}\}) > \delta]\delta.$$

The issue arises because $f(a_i \mid \{a_1, \ldots, a_{i-1}\})$ can be negative since $f(\cdot)$ is non-monotone, which violates the non-negative requirement of Markov's inequality.

References

Amanatidis, G., Fusco, F., Lazos, P., Leonardi, S., & Reiffenhäuser, R. (2020). Fast adaptive non-monotone submodular maximization subject to a knapsack constraint. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Amanatidis, G., Fusco, F., Lazos, P., Leonardi, S., & Reiffenhäuser, R. (2022). Fast adaptive non-monotone submodular maximization subject to a knapsack constraint. *Journal of Artificial Intelligence Research (JAIR)*, *74*, 661–690.

Amanatidis, G., Fusco, F., Lazos, P., Leonardi, S., Spaccamela, A. M., & Reiffenhäuser, R. (2021). Submodular maximization subject to a knapsack constraint: Combinatorial algorithms with near-optimal adaptive complexity. In *International Conference on Machine Learning (ICML)*, pp. 231–242.

Amanatidis, G., Fusco, F., Lazos, P., Leonardi, S., Spaccamela, A. M., & Reiffenhäuser, R. (2023). Submodular maximization subject to a knapsack constraint: Combinatorial algorithms with near-optimal adaptive complexity. *CoRR*, *abs/2102.08327v2*.

Angelova, A., & Zhu, S. (2013). Efficient object detection and segmentation for fine-grained recognition. In *IEEE/CVF Computer Vision and Pattern Recognition Conference (CVPR)*, pp. 811–818.

Badanidiyuru, A., Karbasi, A., Kazemi, E., & Vondrák, J. (2020). Submodular maximization through barrier functions. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 33, pp. 524–534.

Badanidiyuru, A., & Vondrák, J. (2014). Fast algorithms for maximizing submodular functions. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 1497–1514.

Balkanski, E., Breuer, A., & Singer, Y. (2018). Non-monotone submodular maximization in exponentially fewer iterations. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2359–2370.

Balkanski, E., Rubinstein, A., & Singer, Y. (2019a). An exponential speedup in parallel running time for submodular maximization without loss in approximation. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 283–302.

Balkanski, E., Rubinstein, A., & Singer, Y. (2019b). An optimal approximation for submodular maximization under a matroid constraint in the adaptive complexity model. In *ACM Symposium on the Theory of Computing (STOC)*, pp. 66–77.

Balkanski, E., & Singer, Y. (2018). The adaptive complexity of maximizing a submodular function. In *ACM Symposium on the Theory of Computing (STOC)*, pp. 1138–1151.

Barbosa, R., Ene, A., Nguyen, H., & Ward, J. (2015). The power of randomization: Distributed submodular maximization on massive datasets. In *International Conference on Machine Learning (ICML)*, pp. 1236–1244.

Breuer, A., Balkanski, E., & Singer, Y. (2020). The fast algorithm for submodular maximization. In *International Conference on Machine Learning (ICML)*, pp. 1134–1143.

Buchbinder, N., & Feldman, M. (2019). Constrained submodular maximization via a non-symmetric technique. *Mathematics of Operations Research*, *44*(3), 988–1005.

Buchbinder, N., Feldman, M., Naor, J., & Schwartz, R. (2014). Submodular maximization with cardinality constraints. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 1433–1452.

Buchbinder, N., Feldman, M., & Schwartz, R. (2015). Comparing apples and oranges: query tradeoff in submodular maximization. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 1149–1168.

Calinescu, G., Chekuri, C., Pal, M., & Vondrák, J. (2011). Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing (SICOMP)*, *40*(6), 1740–1766.

Chekuri, C., & Quanrud, K. (2019a). Parallelizing greedy for submodular set function maximization in matroids and beyond. In *ACM Symposium on the Theory of Computing (STOC)*, pp. 78–89.

Chekuri, C., & Quanrud, K. (2019b). Submodular function maximization in parallel via the multilinear relaxation. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 303–322.

Chen, L., Feldman, M., & Karbasi, A. (2019). Unconstrained submodular maximization with constant adaptive complexity. In *ACM Symposium on the Theory of Computing (STOC)*, pp. 102–113.

Chen, Y., Dey, T., & Kuhnle, A. (2021). Best of both worlds: Practical and theoretically optimal submodular maximization in parallel. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 34.

Chen, Y., & Kuhnle, A. (2022). Practical and parallelizable algorithms for non-monotone submodular maximization with size constraint. *CoRR*, *abs/2009.01947v4*.

Cui, S., Han, K., & Huang, H. (2024). Deletion-robust submodular maximization with knapsack constraints. In *AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 38, pp. 11695–11703.

Cui, S., Han, K., Tang, J., & Huang, H. (2023). Constrained subset selection from data streams for profit maximization. In *International World Wide Web Conferences (WWW)*, pp. 1822–1831.

Cui, S., Han, K., Tang, J., Huang, H., Li, X., & Li, Z. (2022). Streaming algorithms for constrained submodular maximization. In *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*.

Cui, S., Han, K., Tang, J., Huang, H., Li, X., & Zhiyuli, A. (2023a). Practical parallel algorithms for submodular maximization subject to a knapsack constraint with nearly optimal adaptivity. In *AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 37, pp. 7261–7269.

Cui, S., Han, K., Tang, J., Huang, H., Li, X., Zhiyuli, A., & Li, H. (2023b). Practical parallel algorithms for non-monotone submodular maximization. *CoRR*, *abs/2308.10656*.

Cui, S., Han, K., Tang, S., Li, F., & Luo, J. (2024). Fairness in streaming submodular maximization subject to a knapsack constraint. In *ACM Knowledge Discovery and Data Mining (SIGKDD)*, pp. 514–525.

Cui, S., Han, K., Zhu, T., Tang, J., Wu, B., & Huang, H. (2021). Randomized algorithms for submodular function maximization with a k-system constraint. In *International Conference on Machine Learning (ICML)*, pp. 2222–2232.

Das, A., & Kempe, D. (2008). Algorithms for subset selection in linear regression. In *ACM Symposium on the Theory of Computing (STOC)*, pp. 45–54.

Dueck, D., & Frey, B. J. (2007). Non-metric affinity propagation for unsupervised image categorization. In *International Conference on Computer Vision (ICCV)*, pp. 1–8.

Ene, A., & Nguyen, H. (2020). Parallel algorithm for non-monotone dr-submodular maximization. In *International Conference on Machine Learning (ICML)*, pp. 2902–2911.

Ene, A., & Nguyen, H. L. (2019a). A nearly-linear time algorithm for submodular maximization with a knapsack constraint. In *International Colloquium on Automata, Languages and Programming (ICALP)*, Vol. 132, p. 53.

Ene, A., & Nguyen, H. L. (2019b). Submodular maximization with nearly-optimal approximation and adaptivity in nearly-linear time. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 274–282.

Ene, A., Nguyen, H. L., & Vladu, A. (2019). Submodular maximization with matroid and packing constraints in parallel. In *ACM Symposium on the Theory of Computing (STOC)*, pp. 90–101.

Epasto, A., Mirrokni, V., & Zadimoghaddam, M. (2017). Bicriteria distributed submodular maximization in a few rounds. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pp. 25–33.

Fahrbach, M., Mirrokni, V., & Zadimoghaddam, M. (2019a). Non-monotone submodular maximization with nearly optimal adaptivity and query complexity. In *International Conference on Machine Learning (ICML)*, pp. 1833–1842.

Fahrbach, M., Mirrokni, V., & Zadimoghaddam, M. (2019b). Submodular maximization with nearly optimal approximation, adaptivity and query complexity. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 255–273.

Feige, U., Mirrokni, V. S., & Vondrak, J. (2011). Maximizing non-monotone submodular functions. *SIAM Journal on Computing (SICOMP)*, *40*(4), 1133–1153.

Feldman, M., Harshaw, C., & Karbasi, A. (2017). Greed is good: Near-optimal submodular maximization via greedy optimization. In *Conference on Learning Theory (COLT)*, pp. 758–784.

Feldman, M., Harshaw, C., & Karbasi, A. (2023). How do you want your greedy: Simultaneous or repeated?. *Journal of Machine Learning Research (JMLR)*, *24*, 72–1.

Fisher, M., Nemhauser, G., & Wolsey, L. (1978). An analysis of approximations for maximizing submodular set functions—ii. *Mathematical Programming Study*, *8*, 73–87.

Golovin, D., & Krause, A. (2010). Adaptive submodularity: A new approach to active learning and stochastic optimization.. In *Annual Conference on Computational Learning Theory (COLT)*, pp. 333–345.

Golovin, D., & Krause, A. (2011). Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research (JAIR)*, *42*, 427–486.

Gong, S., Nong, Q., Fang, J., & Du, D.-Z. (2024). Algorithms for cardinality-constrained monotone dr-submodular maximization with low adaptivity and query complexity. *Journal of Optimization Theory and Applications*, *200*(1), 194–214.

Gupta, A., Roth, A., Schoenebeck, G., & Talwar, K. (2010). Constrained non-monotone submodular maximization: Offline and secretary algorithms. In *International Workshop on Internet and Network Economics (WINE)*, pp. 246–257.

Haba, R., Kazemi, E., Feldman, M., & Karbasi, A. (2020). Streaming submodular maximization under a k-set system constraint. In *International Conference on Machine Learning (ICML)*.

Han, K., Cao, Z., Cui, S., & Wu, B. (2020). Deterministic approximation for submodular maximization over a matroid in nearly linear time. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Han, K., Cui, S., Zhu, T., Zhang, E., Wu, B., Yin, Z., Xu, T., Tang, S., & Huang, H. (2021). Approximation algorithms for submodular data summarization with a knapsack constraint. *Proceedings of the ACM on Measurement and Analysis of Computing Systems (SIGMETRICS)*, *5*(1), 1–31.

Kazemi, E., Minaee, S., Feldman, M., & Karbasi, A. (2021). Regularized submodular maximization at scale. In *International Conference on Machine Learning (ICML)*, pp. 5356–5366.

Kazemi, E., Mitrovic, M., Zadimoghaddam, M., Lattanzi, S., & Karbasi, A. (2019). Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In *International Conference on Machine Learning (ICML)*, pp. 3311–3320.

Kazemi, E., Zadimoghaddam, M., & Karbasi, A. (2018). Scalable deletion-robust submodular maximization: Data summarization with privacy and fairness constraints. In *International Conference on Machine Learning (ICML)*, pp. 2544–2553.

Kim, G., Xing, E. P., Fei-Fei, L., & Kanade, T. (2011). Distributed cosegmentation via submodular optimization on anisotropic diffusion. In *International Conference on Computer Vision (ICCV)*, pp. 169–176.

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images. *Toronto, ON, Canada*, *1*, 1–60.

Kuhnle, A. (2021). Nearly linear-time, parallelizable algorithms for non-monotone submodular maximization. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Kulesza, A., & Taskar, B. (2012). Determinantal point processes for machine learning. *Foundations and Trends in Machine Learning*, *5*(2–3), 123–286.

Kulik, A., Shachnai, H., & Tamir, T. (2009). Maximizing submodular set functions subject to multiple linear constraints. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 545–554.

Kulik, A., Shachnai, H., & Tamir, T. (2013). Approximations for monotone and nonmonotone submodular maximization with knapsack constraints. *Mathematics of Operations Research*, *38*(4), 729–739.

Kumar, R., Moseley, B., Vassilvitskii, S., & Vattani, A. (2013). Fast greedy algorithms in mapreduce and streaming. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pp. 1–10.

Lawrence, N., Seeger, M., & Herbrich, R. (2002). Fast sparse gaussian process methods: The informative vector machine. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Lee, J., Mirrokni, V. S., Nagarajan, V., & Sviridenko, M. (2010). Maximizing nonmonotone submodular functions under matroid or knapsack constraints. *SIAM Journal on Discrete Mathematics (SIDMA)*, *23*(4), 2053–2078.

Lei, Q., Wu, L., Chen, P.-Y., Dimakis, A., Dhillon, I. S., & Witbrock, M. J. (2019). Discrete adversarial attacks and submodular optimization with applications to text classification. *Systems and Machine Learning (SysML)*, *1*, 146–165.

Leskovec, J., & Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection..

Lin, H., & Bilmes, J. (2012). Learning mixtures of submodular shells with application to document summarization. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 479–490.

Minoux, M. (1978). Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization techniques*, pp. 234–243.

Mirzasoleiman, B., Badanidiyuru, A., & Karbasi, A. (2016). Fast constrained submodular maximization: Personalized data summarization. In *International Conference on Machine Learning (ICML)*, pp. 1358–1367.

Mirzasoleiman, B., Jegelka, S., & Krause, A. (2018). Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1379–1386.

Mirzasoleiman, B., Karbasi, A., Sarkar, R., & Krause, A. (2013). Distributed submodular maximization: Identifying representative elements in massive data.. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2049–2057.

Mirzasoleiman, B., Karbasi, A., Sarkar, R., & Krause, A. (2016). Distributed submodular maximization. *Journal of Machine Learning Research (JMLR)*, *17*(1), 8330–8373.

Mitrovic, M., Kazemi, E., Zadimoghaddam, M., & Karbasi, A. (2018). Data summarization at scale: A two-stage submodular approach. In *International Conference on Machine Learning (ICML)*, pp. 3596–3605.

Pham, C. V., Tran, T. D., Ha, D. T. K., & Thai, M. T. (2023). Linear query approximation algorithms for non-monotone submodular maximization under knapsack constraint. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4127–4135.

Qian, C., Yu, Y., Tang, K., Yao, X., & Zhou, Z.-H. (2019). Maximizing submodular or monotone approximately submodular functions by multi-objective evolutionary algorithms. *Artificial Intelligence*, *275*, 279–294.

Quinzan, F., Doskoc, V., Göbel, A., & Friedrich, T. (2021). Adaptive sampling for fast constrained maximization of submodular functions. In *Artificial Intelligence and Statistics (AISTATS)*, pp. 964–972.

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

Wei, K., Iyer, R., & Bilmes, J. (2015). Submodularity in data subset selection and active learning. In *International Conference on Machine Learning (ICML)*, pp. 1954–1963.

Wolsey, L. A. (1982). Maximising real-valued submodular functions: Primal and dual heuristics for location problems. *Mathematics of Operations Research*, *7*(3), 410–425.

Yaroslavtsev, G., Zhou, S., & Avdiukhin, D. (2020). "bring your own greedy"+ max: Near-optimal 1/2-approximations for submodular knapsack. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 3263–3274.

Zhu, F., Jiang, Z., & Shao, L. (2014). Submodular object recognition. In *IEEE/CVF Computer Vision and Pattern Recognition Conference (CVPR)*, pp. 2457–2464.