Practical Parallel Algorithms for Submodular Maximization Subject to a Knapsack Constraint with Nearly Optimal Adaptivity

Shuang Cui[1], Kai Han[2] *, Jing Tang[3], He Huang[2] *, Xueying Li[4], Aakas Zhiyuli[4]

[1] School of Computer Science and Technology / Suzhou Research Institute, University of Science and Technology of China
[2] School of Computer Science and Technology, Soochow University
[3] The Hong Kong University of Science and Technology (Guangzhou)
The Hong Kong University of Science and Technology
[4] Alibaba Group
lakers@mail.ustc.edu.cn, hankai@suda.edu.cn, jingtang@ust.hk, huangh@suda.edu.cn,
xiaoming.lxy@alibaba-inc.com, aakas.lzy@alibaba-inc.com

Abstract

Submodular maximization has wide applications in machine learning and data mining, where massive datasets have brought the great need for designing efficient and parallelizable algorithms. One measure of the parallelizability of a submodular maximization algorithm is its adaptivity 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 the *first combinatorial* algorithm achieving an $(8+\epsilon)$-approximation under $\mathcal{O}(\log n)$ adaptive complexity, which is *optimal* up to a factor of $\mathcal{O}(\log\log n)$. Moreover, under slightly larger adaptivity, we also propose approximation algorithms with *nearly optimal* query complexity of $\tilde{\mathcal{O}}(n)$, while achieving better approximation ratios. We show that our 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 approach is demonstrated by extensive experiments on real-world applications.

1 Introduction

Submodular maximization has extensive applications such as influence maximization (Kempe, Kleinberg, and Tardos 2003; Chen et al. 2021), exemplar-based clustering (Gomes and Krause 2010), crowdsourcing (Singla, Tschiatschek, and Krause 2016) and sensor placement (Iyer and Bilmes 2013; Sallam et al. 2020), so it has been widely studied under various constraints such as cardinality, knapsack, matroid constraint. Many algorithms in this area adopt greedy search strategies (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 and Frey 2007; Das and Kempe 2008; Kazemi, Zadimoghaddam, and Karbasi 2018), 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, Mirrokni, and Zadimoghaddam 2017; Kazemi et al. 2021), but these algorithms can still be highly adaptive, as they usually run sequential greedy algorithms on each machine. Recently, Balkanski and 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, Mirrokni, and Zadimoghaddam 2019b; Balkanski, Rubinstein, and Singer 2019a)).

Besides the SMC problem, one of the most fundamental problems in submodular optimizations is the problem of submodular maximization subject to a knapsack constraint (**SKP**), which has applications both for monotone and non-monotone submodular functions (Kulik, Shachnai, and Tamir 2009; Lee et al. 2010; Badanidiyuru and Vondrák 2014). Especially the non-monotone SKP problem has many real-world applications such as revenue maximation (Han et al. 2021; Amanatidis et al. 2020, 2021), movie recommendation (Amanatidis et al. 2020, 2021) and image summarization (Han et al. 2021). Surprisingly, although the SKP problem has been extensively studied since the 1980s (Wolsey 1982), there exist only few studies on designing low-adaptivity algorithms for it. In particular, Chekuri and Quanrud (2019b) provide a $(\frac{e}{e-1}+\epsilon)$-approximation in $\mathcal{O}(\log n)$ adaptive rounds for monotone SKP, while (Ene,

*Corresponding author: Kai Han <hankai@suda.edu.cn> and He Huang <huangh@suda.edu.cn>.

Nguyen, and Vladu 2019) present a $(e+\epsilon)$-approximation in $\mathcal{O}(\log^2 n)$ adaptive rounds for non-monotone SKP. However, both Chekuri and Quanrud (2019b) and Ene, Nguyen, and Vladu (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. 2020, 2021). Very recently, Amanatidis et al. (2021) study the non-monotone SKP problem and present the first parallelizable algorithm dubbed ParKnapsack with $\mathcal{O}(\log n)$ adaptivity or $\tilde{\mathcal{O}}(n)$[1] query complexity, but their approximation ratio of $9.465+\epsilon$ might not hold. Therefore, it still remains as an open problem to find a parallelizable algorithm achieving provable approximation ratios for the *non-monotone* SKP problem with nearly optimal adaptivity of $\mathcal{O}(\log n)$, or with nearly optimal query complexity of $\tilde{\mathcal{O}}(n)$. Note that reducing adaptivity from $\mathcal{O}(\log^2 n)$ to $\mathcal{O}(\log n)$ has been considered significant on parallel submodular maximization (Fahrbach, Mirrokni, and Zadimoghaddam 2019a; Ene and Nguyen 2020; Amanatidis et al. 2021), as this greatly reduces the number of parallel rounds in practical implementation.

Contributions. In this paper, we close the gap mentioned above by presenting two algorithms dubbed ParSKP1 and ParSKP2 for the non-monotone SKP, achieving the following performance bounds:

- ParSKP1 gives an $(8+\epsilon)$-approximation, while achieving $\mathcal{O}(\log n)$ adaptivity under $\mathcal{O}(n^2\log^2 n)$ query complexity, or achieving $\mathcal{O}(\log^2 n)$ adaptivity under $\mathcal{O}(n\log^3 n)$ query complexity.
- ParSKP2 gives a $(5+2\sqrt{2}+\epsilon)$-approximation, while achieving $\mathcal{O}(\log^2 n)$ adaptivity under $\mathcal{O}(n^2\log^2 n)$ query complexity, or achieving $\mathcal{O}(\log^3 n)$ adaptivity under $\mathcal{O}(n\log^3 n)$ query complexity.

Moreover, as the SMC problem is a special case of SKP, our algorithms can be directly used to address the non-monotone SMC problem, for which the approximation ratio of ParSKP2 can be tightened to $4+\epsilon$ under the same complexities listed above. Therefore, ParSKP1 and ParSKP2 also achieve performance bounds comparable to those of the existing parallelizable algorithms for non-monotone SMC, as discussed in the next section.

We conduct extensive experiments using several applications including revenue maximization, movie recommendation and image summarization. The experimental results show that our algorithms can achieve better utility using significantly fewer adaptive rounds than the existing studies.

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. To overcome this challenge, our ParSKP1 algorithm adopts a very different framework by 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 ParSKP2 algorithm adopts another framework 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.

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. The traditional SKP problem has been extensively studied, both for monotone and non-monotone submodular functions (e.g., (Sviridenko 2004; Kulik, Shachnai, and Tamir 2013; Gupta et al. 2010; Ene and Nguyen 2019a; Yaroslavtsev, Zhou, and Avdiukhin 2020)). For non-monotone SKP, Buchbinder and Feldman (2019) achieve the best-known approximation ratio of 2.60, while some other studies (Mirzasoleiman, Badanidiyuru, and Karbasi 2016; Amanatidis et al. 2020; Han et al. 2021) provide faster algorithms with weaker approximation ratios. However, all these algorithms have super-linear adaptive complexity unsuitable for parallelization. Balkanski and Singer (2018) initiate the study on designing low-adaptivity algorithms for the monotone SMC problem, and also prove that no algorithms can achieve a constant approximation ratio in $\mathcal{O}(\log n/\log\log n)$ adaptive rounds. Subsequently, a lot of parallelizable algorithms have appeared for monotone SMC (Ene and Nguyen 2019b; Balkanski, Rubinstein, and Singer 2019b; Chekuri and Quanrud 2019a; Breuer, Balkanski, and Singer 2020; Chen, Dey, and Kuhnle 2021). For the non-monotone SMC problem, Chekuri and Quanrud (2019a) and Balkanski, Breuer, and Singer (2018) propose parallelizable algorithms with $5.83+\epsilon$ and $2e+\epsilon$ approximation ratios, respectively, both under $\mathcal{O}(\log^2 n)$ adaptivity, while Ene and Nguyen (2020) achieve an improved ratio of $e+\epsilon$ under $\mathcal{O}(\log n)$ adaptivity. However, all these studies use multilinear extensions and have high query complexity (larger than $\Omega(nk^2)$). Fahrbach, Mirrokni, and Zadimoghaddam (2019a) aim to reduce both adaptivity and query complexity, but achieving a relatively large ratio of 25.64. Kuhnle (2021) claims a $(5.18+\epsilon)$-approximation with $\mathcal{O}(\log^2 n)$ adaptivity. However, Chen and Kuhnle (2022) claim that both (Fahrbach, Mirrokni, and Zadimoghaddam 2019a) and (Kuhnle 2021) have non-trivial errors and they propose a new adaptive algorithm with an approximation ratio of $(5.18+\epsilon)$. Even so, our ParSKP2 algorithm still achieves a better approximation ratio than (Chen and Kuhnle 2022) for the SMC problem. Compared to SMC, there are relatively few studies on designing low-adaptivity algorithms for SKP. Chekuri and Quanrud (2019b) provide a $(1-1/e-\epsilon)$-approximation in $\mathcal{O}(\log n)$ adaptive rounds for monotone SKP. For non-monotone SKP, Ene, Nguyen, and Vladu (2019) provide an $(e+\epsilon)$-approximation with $\mathcal{O}(\log^2 n)$ adaptivity, based on a continuous optimization approach using multi-linear extension. However, a large

[1]Throughout the paper, we use the $\tilde{O}$ notation to suppress polylogarithmic factors.

Constraint	Reference	Ratio	Adaptivity	Queries
Knapsack	(Ene, Nguyen, and Vladu 2019)	$\boldsymbol{e+\epsilon}$	$\mathcal{O}(\log^2 n)$	$\tilde{\mathcal{O}}(nk^2)$
	(Amanatidis et al. 2021)	$9.465+\epsilon$	$\boldsymbol{\mathcal{O}(\log n)}$ || $\mathcal{O}(\log^2 n)$	$\tilde{\mathcal{O}}(n^2)$ || $\boldsymbol{\tilde{\mathcal{O}}(n)}$
	ParSKP1(ALG. 3)	$8+\epsilon$	$\boldsymbol{\mathcal{O}(\log n)}$ || $\mathcal{O}(\log n \log k)$	$\tilde{\mathcal{O}}(nk)$ || $\boldsymbol{\tilde{\mathcal{O}}(n)}$
	ParSKP2(ALG. 5)	$5+2\sqrt{2}+\epsilon$	$\mathcal{O}(\log^2 n)$ || $\mathcal{O}(\log^2 n \log k)$	$\tilde{\mathcal{O}}(nk)$ || $\boldsymbol{\tilde{\mathcal{O}}(n)}$
Cardinality	(Chekuri and Quanrud 2019a)	$3+2\sqrt{2}+\epsilon$	$\mathcal{O}(\log^2 n)$	$\tilde{\mathcal{O}}(nk^4)$
	(Balkanski, Breuer, and Singer 2018)	$2e+\epsilon$	$\mathcal{O}(\log^2 n)$	$\tilde{\mathcal{O}}(nk^2)$
	(Ene and Nguyen 2020)	$\boldsymbol{e+\epsilon}$	$\boldsymbol{\mathcal{O}(\log n)}$	$\tilde{\mathcal{O}}(nk^2)$
	(Fahrbach, Mirrokni, and Zadimoghaddam 2019a)	$25.641+\epsilon$	$\boldsymbol{\mathcal{O}(\log n)}$	$\boldsymbol{\tilde{\mathcal{O}}(n)}$
	ParSKP1(ALG. 3)	$8+\epsilon$	$\boldsymbol{\mathcal{O}(\log n)}$ || $\mathcal{O}(\log n \log k)$	$\tilde{\mathcal{O}}(nk)$ || $\boldsymbol{\tilde{\mathcal{O}}(n)}$
	ParSKP2(ALG. 5)	$4+\epsilon$	$\mathcal{O}(\log^2 n)$ || $\mathcal{O}(\log^2 n \log k)$	$\tilde{\mathcal{O}}(nk)$ || $\boldsymbol{\tilde{\mathcal{O}}(n)}$

[1] Bold font indicates the best result(s) in each setting, k is the largest cardinality of any feasible solution.

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

number of $\Omega(nk^2 \log^2 n)$ function valuations are needed in (Ene, Nguyen, and Vladu 2019) for simulating a query to the multilinear extension of a submodular function or its gradient with sufficient accuracy, as described in (Fahrbach, Mirrokni, and Zadimoghaddam 2019a). Observing this, Amanatidis et al. (2021) provide the first combinatorial algorithm with nearly optimal adaptivity of $\mathcal{O}(\log n)$, or with $\mathcal{O}(\log^2 n)$ adaptivity under nearly optimal query complexity of $\tilde{\mathcal{O}}(n)$. Unfortunately, their approximation ratio might not hold due to subtle issues in their performance analysis.

3 Preliminaries

Given a ground set $\mathcal{N}$ with $|\mathcal{N}| = n$, a function $f\colon 2^{\mathcal{N}} \mapsto \mathbb{R}$ is called submodular if it satisfies: $\forall X, Y \subseteq \mathcal{N}\colon f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$. In this paper, we consider a non-monotone and non-negative submodular function that satisfies $\forall X \subseteq \mathcal{N}\colon f(X) \geq 0$ but may not satisfy $\forall X \subseteq Y \subseteq \mathcal{N}\colon f(X) \leq f(Y)$. 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 $\forall u \in \mathcal{N}\colon c(u) \leq B$. The submodular maximization with a knapsack constraint problem (SKP) aims to find an optimal solution O to the problem: $\max\{f(S)\colon S \subseteq \mathcal{N} \wedge c(S) \leq B\}$, where $c(S) \triangleq \sum_{u \in S} c(u)$. For convenience, we define $\mathrm{OPT} = f(O)$, and use k to denote the maximum cardinality of any feasible solution to the SKP problem, and use w to denote an element with maximum cost in O.

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, Feldman, and Karbasi 2019)). *For every constant $\epsilon > 0$, there is an algorithm without using multi-linear extension that achieves a $(2+\epsilon)$-approximation for USM 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.*

For convenience, 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.

4 Approximation Algorithms

In this section, we propose our ParSKP1 and ParSKP2 algorithms. Both of them call a procedure RandBatch inspired by (Balkanski, Rubinstein, and Singer 2019b; Amanatidis et al. 2021) using the "adaptive sequencing" method, but they achieve better performance bounds than (Amanatidis et al. 2021) by introducing a "random batch selection" method in adaptive sequencing, and by adopting very different methods for generating candidate solutions. For clarity, we first introduce RandBatch (Sec. 4.1), then introduce ParSKP1 (Sec. 4.2) and ParSKP2 (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 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 budget 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 if $count = M$.

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

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

Algorithm 2: GetSEQ$(A, I, c(\cdot))$

```
1  $V \leftarrow \emptyset$;
2  while $I \neq \emptyset$ do
3      Randomly permute $I$ into $\{v_1, \ldots, v_{|I|}\}$;
4      $s \leftarrow \max_{i \leq |I|}\{c(A \cup V \cup \{v_1, \ldots, v_i\}) \leq B\}$;
5      $V \leftarrow V \cup \{v_1, \ldots, v_s\}$;
6      $I \leftarrow \{u : u \in I \setminus V \wedge c(A \cup V \cup \{u\}) \leq B\}$;
7  return $V$;
```

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^+, E_i^- and D_i 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 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* $\mathbb{E}[f(A)] \geq (1-\epsilon)^2\rho \cdot \mathbb{E}[c(A)]$ *and* $\epsilon \cdot M \cdot \sum_{u \in L} f(u \mid A) \leq \mathrm{OPT}$ *for any* $I \subseteq \mathcal{N}$.

The complexity of RandBatch (shown in Lemma 2) 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 2. *RandBatch has $\mathcal{O}((\frac{1}{\epsilon}\log(|I| \cdot \beta(I)) + M)/p)$ adaptivity, and its query complexity is $\mathcal{O}(|I| \cdot k)$ 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 $\mathcal{O}((\frac{1}{\epsilon}\log(|I| \cdot \beta(I)) + M) \cdot (\log k)/p)$ adaptivity, and its query complexity is $\mathcal{O}(|I|)$ times of its adaptivity.*

4.2 The ParSKP1 Algorithm

Our ParSKP1 algorithm is shown in Algorithm 3. In ParSKP1, 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 $c(\mathcal{N}_2) \leq \epsilon \cdot B$. A major building block of ParSKP1 is the function Probe (shown in Algorithm 4). For clarity, we first elaborate Probe in the following.

Algorithm 3: ParSKP1$(\alpha, \epsilon, f(\cdot), c(\cdot))$

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

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

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

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 2). 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 3, we show Probe can achieve a provable approximation ratio under some special cases:

Lemma 3. *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 $\forall i \in \{1,2\}$: $c(A_i) < B - \max\{\epsilon B, c(w)\}$, where w is the element in O with the maximum cost, then Algorithm 4 returns a solution T satisfying $f(T) \geq (1-2\epsilon)(1-2\alpha-2\epsilon)\mathrm{OPT}/(4-4\epsilon)$.*

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:

$$\begin{aligned} f(O) &\leq f(O \cap \mathcal{N}_1 \setminus A_1) + f((O \cap A_1) \cup (O \cap \mathcal{N}_2)) \\ &\leq f(A_2 \cup (O \cap \mathcal{N}_1 \setminus A_1)) + f(A_1 \cup (O \cap \mathcal{N}_1)) \\ &\quad + f((O \cap A_1) \cup (O \cap \mathcal{N}_2)). \end{aligned} \tag{1}$$

Next, we try to bound the three additive factors in the RHS of Eqn. (1). For the third additive factor, we have

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

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

$$\begin{aligned} &f(A_1 \cup (O \cap \mathcal{N}_1)) \\ &\leq f(A_1 \cup \{w\}) + \textstyle\sum_{u \in Q} f(u \mid A_1 \cup \{w\}) \\ &\leq f(T) + \textstyle\sum_{u \in Q} f(u \mid A_1) \\ &\leq f(T) + \sum_{u \in L_1} f(u \mid A_1) + \sum_{u \in Q \setminus L_1} f(u \mid A_1), \end{aligned} \tag{3}$$

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

$$\textstyle\sum_{u \in Q \setminus L_1} f(u \mid A_1) \leq \rho \cdot c(Q) \leq \rho^* c(Q) \leq \alpha \cdot f(O). \tag{4}$$

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. (3) and Eqn. (4) yields

$$f(A_1 \cup (O \cap \mathcal{N}_1)) \leq f(T) + (\alpha + \epsilon) \cdot f(O). \tag{5}$$

Using similar reasoning as above, we can also get

$$f(A_2 \cup (O \cap \mathcal{N}_1 \setminus A_1)) \leq f(T) + (\alpha + \epsilon) \cdot f(O). \tag{6}$$

The lemma then follows by combining Eqns. (1)–(6). □

There are still two obstacles for using Lemma 3 to find the approximation ratio of ParSKP1: 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 3. In the following, we roughly explain how ParSKP1 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 ParSKP1. Therefore, ParSKP1 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 USM($\mathcal{N}_2$) in Line 2 of ParSKP1 can be used to find a ratio. To address the second problem mentioned above, ParSKP1 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 3 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 combining all these ideas and choosing an appropriate α, we get:

Theorem 2. *ParSKP1 can return a solution S satisfying $\mathbb{E}[f(S)] \geq (1/8 - \epsilon)\mathrm{OPT}$ by setting $\alpha = 1/4$.*

Note that the complexity of ParSKP1 is dominated by Lines 6–9, where Probe is run for multiple times in parallel. Therefore, leveraging Lemma 2, we can also get:

Theorem 3. *The adaptive complexity and query complexity of ParSKP1 are $\mathcal{O}(\log n)$ and $\mathcal{O}(nk \log^2 n)$ respectively, or $\mathcal{O}(\log n \log k)$ and $\mathcal{O}(n \log^2 n \log k)$ respectively.*

4.3 The ParSKP2 Algorithm

In this section, we introduce ParSKP2 (as shown in Algorithm 5), which achieves a better approximation ratio than ParSKP1 under logarithmically larger adaptive complexity, but still can achieve nearly linear query complexity.

Algorithm 5: ParSKP2($p, \epsilon, f(\cdot), c(\cdot)$)

```
1 $\mathcal{N}'_1 \leftarrow \{u \in \mathcal{N}: c(u) > \frac{B}{n}\}$; $\mathcal{N}'_2 \leftarrow \mathcal{N} \setminus \mathcal{N}'_1$;
2 $u_1^* \leftarrow \arg\max_{u \in \mathcal{N}'_1} \frac{f(u)}{c(u)}$; $u_2^* \leftarrow \arg\max_{u \in \mathcal{N}} f(u)$;
3 $T \leftarrow \emptyset$; $I \leftarrow \mathcal{N}'_1$; $M \leftarrow \lceil \frac{\log_{1-\epsilon} \frac{\epsilon}{n} + 2}{\epsilon^2} \rceil$; $\rho_{max} \leftarrow \frac{f(u_1^*)}{c(u_1^*)}$;
4 $H \leftarrow$ USM($\mathcal{N}'_2$); $\ell \leftarrow \lceil \log_{1-\epsilon} \frac{\epsilon \cdot c(u_1^*)}{B} \rceil + 1$;
5 for $i \leftarrow 1$ to $\ell$ do
6     $\rho_i \leftarrow \rho_{max} \cdot (1-\epsilon)^{i-1}$;
7     $(A_i, U_i, L_i) \leftarrow$
        RandBatch($\rho_i, I, M, p, \epsilon, f_T(\cdot), c_T(\cdot)$);
8     $T \leftarrow T \cup A_i$; $I \leftarrow I \setminus (U_i \cup L_i)$;
9 return $S \leftarrow \arg\max_{X \in \{T, H, \{u_2^*\}\}} f(X)$;
```

Similar to ParSKP1, ParSKP2 also partitions the ground set $\mathcal{N}$ into two subsets $\mathcal{N}'_1$ and $\mathcal{N}'_2$ (Line 1), where $\mathcal{N}'_1$ contains elements with sufficiently large costs, and then calls RandBatch to find an approximate solution T using $\mathcal{N}'_1$ as the ground set. However, instead of creating several candidate solutions using a single threshold (as that in ParSKP1), ParSKP2 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 5–8). 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 8), which is useful for the performance analysis presented shortly. The final solution S returned by ParSKP2 is the best one among T, USM($\mathcal{N}'_2$), and the single element in $\mathcal{N}$ with maximum objective function value (Line 9).

Now we begin to analyze the performance of ParSKP2. Note that we have $f(S) \geq (1/2 - \epsilon)f(O \cap \mathcal{N}'_2)$ due to Line 4 of ParSKP2. So we will concentrate on how to bound $f(O \cap \mathcal{N}'_1)$. For notational simplicity, we assume $O \subseteq \mathcal{N}'_1$ in the sequel and remove this assumption in Theorem 4, which provides the approximation ratio of ParSKP2.

Compared to ParSKP1, a key difference of ParSKP2 is that it calls RandBatch with $p < 1$ to introduce additional randomness, which makes its performance analysis more involved. We first introduce some definitions useful in our analysis. When ParSKP2 finishes, let $\mathcal{U} = \cup_{i=1}^{\ell} U_i, \mathcal{L} = \cup_{i=1}^{\ell} L_i, O_{small} = \{u \colon u \in O \setminus (\mathcal{U} \cup \mathcal{L}) \wedge f(u \mid T)/c(u) \leq \rho_\ell\}$ and $O_{big} = O \setminus (O_{small} \cup \mathcal{U} \cup \mathcal{L} \cup \{w\})$. So each element $u \in O_{big}$ must satisfy $\frac{f(u|T)}{c(u)} \geq \rho_\ell$ and $c(T \cup \{u\}) > B$.

With the above definitions, we introduce a random mapping $\Upsilon(\cdot)$ for performance analysis, whose intuition is to map the elements in O_{big} to those in T, such that the utility loss caused by excluding O_{big} from T can be bounded. Suppose that the elements sequentially added into $\mathcal{U}$ are $\{u_1, \ldots, u_h\}$. The mapping $\Upsilon(\cdot)$ is constructed by the following process. Initially we set $\Upsilon(u) = \emptyset$ for all $u \in \mathcal{N}$. Then we set $\Upsilon(u) = \{u\}$ for all $u \in \mathcal{U} \cap O \setminus T$. After that, we check $u_1, u_2, \ldots, u_h$ sequentially. For each $i \in [h]$, if $u_i \in T \setminus O$ and $c(\{u_1, \ldots, u_{i-1}\} \cap T) \leq B - c(w)$, then we remove several elements with smallest costs in O_{big} until their total costs reaches $c(u_i)$ or O_{big} becomes empty, and then add these elements into $\Upsilon(u_i)$. As such, $\Upsilon(u_i)$ may contain "fractional elements" with their costs being only partial of their original ones, but their densities remain unchanged. With the mapping $\Upsilon(\cdot)$ constructed as above, we can get the following lemma:

Lemma 4. *For any $u \in \cup_{i=1}^{\ell} U_i$, let $\rho(u)$ denote the threshold used by ParSKP2 when u is considered to be added into T, and define $\rho(u) = 0$ for all other $u \in \mathcal{N}$. Given any $u \in \mathcal{N}$ with $\Upsilon(u) \neq \emptyset$, we have $\forall v \in \Upsilon(u) \colon \frac{f(v|T)}{c(v)} \leq \frac{\rho(u)}{(1-\epsilon)}$.*

Lemma 4 can be roughly explained as follows. According to the construction rule of $\Upsilon(\cdot)$, each element $v \in \Upsilon(u)$ can also be added into T without violating the budget B when u is added into T, so the reason for $v \notin T$ is that its marginal density is relatively small compared to that of u. Using Lemma 4, we can further get the following lemma:

Lemma 5. *When Algorithm 5 finishes, we have $f(T \cup O) \leq \sum_{u \in \mathcal{N}} \frac{\rho(u)c(\Upsilon(u))}{1-\epsilon} + (2+\epsilon)f(S) + \epsilon f(O)$.*

Proof. When Algorithm 5 finishes, the set $O \setminus T$ can be partitioned into several disjoint subsets: O_{small}, $\mathcal{L} \cap O$, $O_{big} \cup R$ (where $R \triangleq O \cap \mathcal{U} \setminus T$), and $\{w\}$ (if $w \notin O_{small} \cup \mathcal{L} \cup R \cup T$). By submodularity, we have

$$\begin{aligned} f(O \cup T) - f(T) \leq\ & f(O_{small} \mid T) + f(O_{big} \cup R \mid T) \\ & + f(\mathcal{L} \cap O \mid T) + f(w). \end{aligned} \tag{7}$$

Besides, using Lemma 1 and submodularity, we have

$$\begin{aligned} f(\mathcal{L} \cap O \mid T) &\leq \sum\nolimits_{i=1}^{\ell} f(L_i \cap O \mid \cup_{j=1}^{i-1} A_i) \\ &\leq \epsilon^{-1} \cdot \ell f(O)/M \leq \epsilon f(O), \end{aligned} \tag{8}$$

where the last inequality is due to $\ell \leq \log_{1-\epsilon} \frac{\epsilon}{n} + 2$ and $M \geq \frac{\log_{1-\epsilon} \frac{\epsilon}{n} + 2}{\epsilon^2}$ according to Lines 3–4 of Algorithm 5. According to the definition of O_{small}, we have

$$f(O_{small} \mid T) \leq \rho_\ell B \leq \epsilon f(u_1^*) \leq \epsilon f(S). \tag{9}$$

Recall that each element $u \in O_{big}$ must satisfy $c(T \cup \{u\}) > B$. So when $c(T) \leq B - c(w)$, we must have $O_{big} = \emptyset$ and hence $\forall u \in T \setminus O \colon \Upsilon(u) = \emptyset$. If $c(T) > B - c(w)$, then the construction rule of $\Upsilon(\cdot)$ guarantees $\cup_{u \in T \setminus O} \Upsilon(u) = O_{big}$. Besides, we always have $\forall u \in R \colon \Upsilon(u) = \{u\}$ and $\forall u \notin (T \setminus O) \cup R \colon \Upsilon(u) = \emptyset$ according to the construction of $\Upsilon(\cdot)$. Therefore, we always have $O_{big} \cup R = \cup_{u \in \mathcal{N}} \Upsilon(u)$ and hence

$$\begin{aligned} & f(O_{big} \cup R \mid T) \leq f(\cup_{u \in \mathcal{N}} \Upsilon(u) \mid T) \\ \leq\ & \sum_{u \in \mathcal{N}} \sum_{v \in \Upsilon(u)} f(v \mid T) \leq \sum_{u \in \mathcal{N}} \frac{\rho(u) \cdot c(\Upsilon(u))}{1-\epsilon}, \end{aligned} \tag{10}$$

where the second and third inequalities are due to submodularity and Lemma 4, respectively. Finally, note that $f(T) + f(w) \leq 2f(S)$ due to Line 9 of Algorithm 5. Combining this with equations (7)–(10) completes the proof. $\square$

Note that both $\rho(u)$ and $\Upsilon(u)$ 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 6. *By the construction rule of $\Upsilon(\cdot)$, we have $\sum_{u \in \mathcal{N}} \mathbb{E}[\rho(u) \cdot c(\Upsilon(u))] \leq \max\{1, \frac{1-p}{p}\}/(1-\epsilon)^2 \cdot \mathbb{E}[f(S)]$.*

Using Lemmas 4–6, we get the performance bounds of ParSKP2 as follows:

Theorem 4. *The ParSKP2 algorithm can return a solution S satisfying $\mathbb{E}[f(S)] \geq (\frac{1}{5+2\sqrt{2}} - \epsilon)\mathrm{OPT}$ by setting $p = \sqrt{2} - 1$. The adaptive complexity and query complexity of ParSKP2 are $\mathcal{O}(\log^2 n)$ and $\mathcal{O}(nk \log^2 n)$ respectively, or $\mathcal{O}(\log^2 n \log k)$ and $\mathcal{O}(n \log^2 n \log k)$ respectively.*

5 Extensions for Cardinality Constraint

As cardinality constraint is a special case of knapsack constraint, our ParSKP1 and ParSKP2 algorithms can be directly applied to the non-monotone SMC problem, for which the performance bounds shown in Theorem 2 and Theorem 4 still hold. Interestingly, by a more careful analysis, we find that ParSKP2 actually achieves a better approximation ratio for the SMC problem, while its complexities remain the same. This result is shown in Theorem 5. We roughly explain the reason as follows. In the SKP problem, it is possible that an element u with large marginal density cannot be added into the candidate solution T even if $c(T) < B$, due to $c(T \cup \{u\}) > B$. Therefore, we have to handle this case in our analysis for ParSKP2 by considering w as a special element. However, such a case never happens in the SMC problem due to the uniform costs, so the approximation ratio of ParSKP2 can be tightened.

Theorem 5. *For the non-monotone SMC problem, ParSKP2 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.*

6 Performance Evaluation

In this section, we compare our algorithms with several state-of-the-art *practical* algorithms for non-monotone SKP on the objective function value (i.e., utility) and adaptivity. Specifically, we implement five algorithms: (1) ParSKP1 (Algorithm 3); (2) ParSKP2 (Algorithm 5); (3) ParKnapsack (Amanatidis et al. 2021);(4) SampleGreedy (Amanatidis et al. 2020), implemented using *lazy evaluation* (Minoux 1978); (5) SmkRanAcc (Han et al. 2021). 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. 2021) to use the algorithm in (Feige, Mirrokni, and Vondrak 2011) achieving 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 memorory. The implemented algorithms are evaluated in the following three real-world applications.

Revenue Maximization. The goal of revenue maximization (Amanatidis et al. 2020, 2021; Han et al. 2021) is to select a subset of users (i.e., nodes) in a social network to advertise a product in order to maximize the revenue. Given a network $G = (\mathcal{N}, E)$ where each edge $(u, v) \in E$ is assigned a weight $w_{u,v}$ randomly sampled from the continuous uniform distribution $\mathcal{U}(0, 1)$, the revenue of any subset $S \subseteq \mathcal{N}$ is defined as $f(S) = \sum_{v\in\mathcal{N}\setminus S} \sqrt{\sum_{u\in S} w_{u,v}}$, and the cost of any node $u \in \mathcal{N}$ is defined as $c(u) = h\left(\sqrt{\sum_{(u,v)\in E} w_{u,v}}\right)$, where $h(x) = 1 - e^{-\mu x}$ is the exponential cumulative distribution function and μ is set to 0.2. Following (Amanatidis et al. 2020, 2021; Han et al. 2021), we consider the top 5,000 communities of the YouTube network (Leskovec and Krevl 2014) to construct G, and the resultant graph contains 39,841 nodes and 224,235 edges.

Movie Recommendation. Personalized movie recommendation (Mirzasoleiman, Badanidiyuru, and Karbasi 2016; Feldman, Harshaw, and Karbasi 2017; Haba et al. 2020; Amanatidis et al. 2020, 2021) aims to recommend a list of high-quality and diverse movies to a user according to the ratings from similar users. We use the popular MovieLens dataset containing 1,793 movies with adventure, animation and fantasy genres (Haba et al. 2020). Given a set of $\mathcal{N}$ movies, each movie $u \in \mathcal{N}$ is associated with a 25 dimensional feature vector q_u calculated from user ratings. Following (Mirzasoleiman, Badanidiyuru, and Karbasi 2016; Haba et al. 2020), the utility of any $S \subseteq \mathcal{N}$ is defined as $f(S) = \sum_{u\in S}\sum_{v\in\mathcal{N}} s_{u,v} - \sum_{u\in S}\sum_{v\in S} s_{u,v}$, where we use $s_{u,v} = e^{-\lambda \mathrm{dist}(q_u,q_v)}$ to measure the similarity between movies u and v; $\mathrm{dist}(q_u, q_v)$ is the euclidean distance between q_u and q_v; and λ is set to 2. Following (Haba et al. 2020), we also 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.

Image Summarization. This application is also used in (Mirzasoleiman, Badanidiyuru, and Karbasi 2016; Han et al. 2021). The experimental settings are similar to those of these work and hence are omitted due to the space limit.

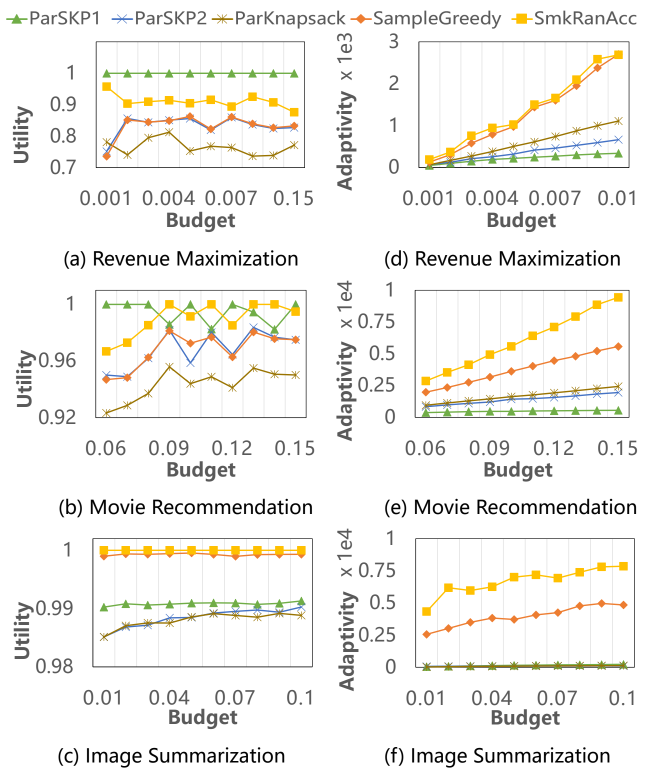

Figure 1: The plots in the left column compare the objective function value of the solutions returned by different algorithms, where the objective function value is normalized by the best utility achieved by implemented algorithms. The plots in the right column compare the adaptive rounds used by different algorithms. The *budget* values are given as a fraction of the total cost of all elements in the groundset.

Experimental Results. In Fig. 1(a)–(c), we compare the implemented algorithms on utility, and the results show ParSKP1 can even achieve better utility compared to non-parallel algorithms SampleGreedy and SmkRanAcc, with the average performance gains of 5% and 3%, respectively. Besides, the utility performance of ParSKP2 is slightly weaker than SampleGreedy and SmkRanAcc (with an average performance loss of 4%). Fig. 1(a)–(c) also shows that ParSKP1 and ParSKP2 achieve significantly (up to 35.74% times) better utility than ParKnapsack, which is the only existing adaptive algorithm for non-monotone SKP problem 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 ParSKP1 and ParSKP2 generally outperform all the baselines. Specifically, ParSKP1 and ParSKP2 incur 3–92 times fewer adaptive rounds than SmkRanAcc, and 2–54 times fewer adaptive rounds than SampleGreedy, which demonstrates the effectiveness of our approach.

Acknowledgments

Kai Han's 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. He Huang's work is partially supported by National Natural Science Foundation of China (NSFC) under Grant No. U20A20182 and Grant No. 61873177. The work of Shuang Cui is done under the guidance of his supervisor: Kai Han.

References

Amanatidis, G.; Fusco, F.; Lazos, P.; Leonardi, S.; and 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.; Spaccamela, A. M.; and Reiffenhäuser, R. 2021. Submodular Maximization subject to a Knapsack Constraint: Combinatorial Algorithms with Near-optimal Adaptive Complexity. In *International Conference on Machine Learning (ICML)*, 231–242.

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

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

Balkanski, E.; Rubinstein, A.; and 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)*, 283–302.

Balkanski, E.; Rubinstein, A.; and 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)*, 66–77.

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

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

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

Buchbinder, N.; and Feldman, M. 2019. Constrained submodular maximization via a nonsymmetric technique. *Mathematics of Operations Research*, 44(3): 988–1005.

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

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

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

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

Chen, W.; Sun, X.; Zhang, J.; and Zhang, Z. 2021. Network Inference and Influence Maximization from Samples. In *International Conference on Machine Learning (ICML)*, 1707–1716.

Chen, Y.; Dey, T.; and Kuhnle, A. 2021. Best of Both Worlds: Practical and Theoretically Optimal Submodular Maximization in Parallel. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34.

Chen, Y.; and Kuhnle, A. 2022. Practical and Parallelizable Algorithms for Non-Monotone Submodular Maximization with Size Constraint. *arXiv preprint arXiv:2009.01947v4*.

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

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

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

Ene, A.; and 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)*, volume 132, 53.

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

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

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

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

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

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

Feldman, M.; Harshaw, C.; and Karbasi, A. 2017. Greed Is Good: Near-Optimal Submodular Maximization via Greedy Optimization. In *Conference on Learning Theory (COLT)*, 758–784.

Gomes, R.; and Krause, A. 2010. Budgeted nonparametric learning from data streams. In *International Conference on Machine Learning (ICML)*, 391–398.

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

Haba, R.; Kazemi, E.; Feldman, M.; and Karbasi, A. 2020. Streaming Submodular Maximization under a k-Set System Constraint. In *International Conference on Machine Learning (ICML)*.

Han, K.; Cui, S.; Zhu, T.; Zhang, E.; Wu, B.; Yin, Z.; Xu, T.; Tang, S.; and 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.

Iyer, R. K.; and Bilmes, J. A. 2013. Submodular Optimization with Submodular Cover and Submodular Knapsack Constraints. In *NIPS*, volume 26.

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

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

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

Kempe, D.; Kleinberg, J.; and Tardos, É. 2003. Maximizing the spread of influence through a social network. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 137–146.

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

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

Kulik, A.; Shachnai, H.; and 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.; and Vattani, A. 2013. Fast greedy algorithms in mapreduce and streaming. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 1–10.

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

Leskovec, J.; and Krevl, A. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. https://snap.stanford.edu/data. Accessed: 2021-12-14.

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

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

Mirzasoleiman, B.; Karbasi, A.; Sarkar, R.; and Krause, A. 2013. Distributed Submodular Maximization: Identifying Representative Elements in Massive Data. In *NIPS*, 2049–2057.

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

Sallam, G.; Zheng, Z.; Wu, J.; and Ji, B. 2020. Robust Sequence Submodular Maximization. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Singla, A.; Tschiatschek, S.; and Krause, A. 2016. Noisy submodular maximization via adaptive sampling with applications to crowdsourced image collection summarization. In *AAAI Conference on Artificial Intelligence (AAAI)*.

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

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.; and 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)*, 3263–3274.