Improved Parallel Algorithm for Minimum Cost Submodular Cover Problem

Yingli Ran RANYINGLI@ZJNU.EDU.CN and **Zhao Zhang**[*] HXHZZ@SINA.COM
College of Mathematics and Computer Sciences, Zhejiang Normal University

Shaojie Tang SHAOJIE.TANG@UTDALLAS.EDU
Naveen Jindal School of Management, University of Texas at Dallas

Editors: Po-Ling Loh and Maxim Raginsky

Abstract

In the minimum cost submodular cover problem (MinSMC), we are given a monotone nondecreasing submodular function $f\colon 2^V \to \mathbb{Z}^+$, a linear cost function $c : V \to \mathbb{R}^+$, and an integer $k \leq f(V)$, the goal is to find a subset $A \subseteq V$ with the minimum cost such that $f(A) \geq k$. The MinSMC can be found at the heart of many machine learning and data mining applications. In this paper, we design a parallel algorithm for the MinSMC that takes at most $O(\frac{\log(km)\log k(\log m+\log\log(mk))}{\varepsilon^4})$ adaptive rounds, and it achieves an approximation ratio of $\frac{H(\min\{\Delta,k\})}{1-5\varepsilon}$ with probability at least $1-3\varepsilon$, where $\Delta = \max_{v\in V} f(v)$, $H(\cdot)$ is the Harmonic number, $m = |V|$, and ε is a constant in $(0, \frac{1}{5})$.

Keywords: minimum cost submodular cover; approximation algorithm; parallel algorithm.

1. Introduction

Recently, submodular optimization has attracted considerable interest in machine learning and data mining, and optimizing a submodular function can be found in a variety of applications, including viral marketing (Kempe et al., 2003), information gathering (Krause and Guestrin, 2009), and active learning (Golovin and Krause, 2011). In this paper, we design parallel approximation algorithms for the fundamental problem of *minimum cost submodular cover* (MinSMC). The input of MinSMC is a set V of m elements. Given a monotonically nondecreasing submodular function $f\colon 2^V \to \mathbb{Z}^+$, a linear cost function $c : V \to \mathbb{R}^+$, and an integer $k \leq f(V)$, the goal of the MinSMC is to find a subset $A \subseteq V$ with the minimum cost such that $f(A) \geq k$, where the cost of A is $c(A) = \sum\limits_{v\in A} c(v)$. The MinSMC has numerous applications, including data summarization (Tschiatschek et al., 2014) and recommender systems (El-Arini and Guestrin, 2011). In the example of data summarization, we are given a set of data, our goal is to select a cheapest set of data, whose representativeness meets some minimum requirement. Many commonly used utility functions exhibit submodularity, a natural diminishing returns property, leading to the formulation of a MinSMC (Mirzasoleiman et al., 2015). To solve MinSMC, Wolsey (1982) developed a centralized sequential greedy algorithm which achieves an approximation ratio of $H(\Delta)$, where $H(\Delta) = \sum_{i=1}^{\Delta} 1/i$ is the Δ-th Harmonic number and $\Delta = \max_{v\in V} f(v)$.

Unfortunately, the aforementioned centralized sequential greedy method requires $\Omega(n)$ adaptive rounds. A formal definition of *adaptive round* is presented in Definition 4. To this end, we

[*] Corresponding author

are interested designing effective parallel algorithms for MinSMC. The state-of-the-art parallel algorithm for the *unweighted* MinSMC was presented in (Fahrbach et al., 2019); it takes at most $O(\log(m \log k) \log k)$ adaptive rounds to produce a solution sized at most $O(\log k|OPT|)$, where OPT is the optimal solution. Note that their approximation ratio is dependent on k which might be as large as $\Theta(n)$, whereas Δ might be much smaller than k. In this work we consider a general *weighted* MinSMC and we develop an effective and efficient parallel algorithm, whose approximation ratio is arbitrarily close to $H(\Delta)$.

1.1. Related Works

Recently, Balkanski and Singer (2018) introduced the concept of "adaptive complexity" which is defined as the number of parallel rounds required to achieve a constant factor approximation ratio. We use the same notation to measure the running time of a parallel algorithm. Chekuri and Quanrud (2019) describe parallel algorithms for approximately maximizing the multilinear relaxation of a monotone submodular function subject to packing constraints. Both aforementioned studies focus on developing effective parallel algorithms for constrained submodular maximization problem, whereas we study the MinSMC. For the MinSMC, Wolsey (Wolsey, 1982) presented a greedy algorithm that achieves an approximation ratio of $H(\Delta)$. For the *unweighted* version of the MinSMC, Fahrbach et al. (2019) developed a parallel algorithm whose approximation ratio is at most of $O(\log k)$ and it takes $O(\log(m \log k) \log k)$ adaptive rounds. For the set cover problem (finding the smallest subcollection of sets that covers all elements), a special case of the MinSMC, Berger et al. (1989) provided the first parallel algorithm whose approximation guarantee is similar to that of the centralized greedy algorithm. They used the bucketing technique to obtain a $(1+\varepsilon)H(n)$-approximation in $O(\log^5 M)$ rounds, where M is the total sum of the sets' sizes. Rajagopalan and Vazirani (1998) improved the number of rounds to $O(\log^3(Mn))$ at the cost of a larger approximation ratio of $2(1+\varepsilon)H(n)$. Blelloch et al. (2011) further enhanced these results by obtaining a $(1+\varepsilon)H(n)$-approximation algorithm in $O(\log^2 M)$ rounds. We list the performance bounds of the closely related studies in Table 1.

Source	**Approximation ratio**	**# of adaptive rounds**
Minimum submodular cover		
Our algorithm	$\frac{H(\min\{\Delta,k\})}{1-5\varepsilon}$	$O(\frac{\log(km)\log k(\log m+\log\log(mk))}{\varepsilon^4})$
(Wolsey, 1982)(sequential)	$H(\Delta)$	$\Omega(m)$
(Fahrbach et al., 2019)(unweighted MinSMC)	$O(\log k)$	$O(\log(m \log k) \log k)$
Minimum set cover		
(Berger et al., 1989)	$(1+\varepsilon)H(n)$	$O(\log^5 M)$
(Rajagopalan and Vazirani, 1998)	$2(1+\varepsilon)H(n)$	$O(\log^3(Mn))$
(Blelloch et al., 2011)	$(1+\varepsilon)H(n)$	$O(\log^2 M)$

Table 1: Performance bounds of the closely related studies.

1.2. Our contributions and technical overview

In this paper, we design a parallel algorithm for the MinSMC. Our algorithm achieves a near-optimal $\frac{H(\min\{\Delta,k\})}{1-5\varepsilon}$-approximation with probability of at least $1-3\varepsilon$, and it takes poly-logarithmic $O(\frac{\log(km)\log k(\log m+\log\log(mk))}{\varepsilon^4})$ adaptive rounds, where ε is a constant in $(0, \frac{1}{5})$. Note that the aforementioned $O(\log k)$-approximation algorithm (Fahrbach et al., 2019) only works for the unweighted MinSMC, and Δ might be much smaller than k. One naive approach to solve our problem is to iteratively call a parallel algorithm for the submodular maximization problem with a knapsack constraint (Chekuri and Quanrud, 2019) until we find a feasible solution to the MinSMC. Unfortunately, the approximation ratio of this method is $O(\log k)$. Nonetheless, further effort is needed to improve this ratio to $O(\log \Delta)$. We build our algorithm through novel combinations of the ideas of multilayer bucket (Berger et al., 1989), maximal nearly independent set (Blelloch et al., 2011), and random sample (Fahrbach et al., 2019). Note that both Berger et al. (1989) and Blelloch et al. (2011) focus on the set cover problem, which is a special case of the MinSMC. Their approach can not be applied to solving the MinSMC directly. When applied separately, both of them encounter some structural difficulties.

High-Level Intuition of Our Proposed Approach: Inspired by the parallel algorithm (Blelloch et al., 2011) for the minimum set cover problem, we design a parallel mechanism to imitate the sequential greedy algorithm. In each iteration of our algorithm, we only consider those elements with similar marginal profit-to-cost ratios. We find a nearly independent set from those elements such that the profit-to-cost ratio of the nearly independent set is almost the same as that of a best single element. Unfortunately, it is not clear how to adapt the solution from (Blelloch et al., 2011) to find a nearly independent set in the context of MinSMC. To overcome this challenge, we are inspired by (Fahrbach et al., 2019) to use a randomized selection and guessing technique. Note that the original solution developed in (Fahrbach et al., 2019) only works for the unweighted MinSMC. To deal with the weighted case, we further group elements into buckets such that all elements from the same bucket have similar profit-to-cost ratios and marginal profits. In this way, all elements from the same bucket have similar costs. This idea of a multi-layer-bucket originated in (Berger et al., 1989) for solving the minimum set cover problem, we extend their idea to the general MinSMC.

The remaining part of this paper is organized as follows. The design of our parallel algorithm and its analysis are presented in Section 2 and 3. Section 4 concludes the paper with some discussions on future work. All missing proofs can be found at (Ran et al., 2021).

2. Parallel Algorithm and Analysis for MinSMC

2.1. Preliminaries

Definition 1 (Submodular and monotone nondecreasing function) *Given a set V consisting of elements $v_1, v_2 \ldots, v_m$, and a function $f\colon 2^V \to \mathbb{R}^+$, f is* submodular *if $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ for any $A, B \subseteq V$; f is* monotone non-decreasing *if $f(A) \geq f(B)$ for any $B \subseteq A \subseteq V$.*

For any two sets $A, B \subseteq V$, denote $f_A(B) = f(A \cup B) - f(A)$ to be the *marginal profit* of B over A. Assume $f(V) = n$. In this paper, f is always assumed to be an integer-valued, monotone nondecreasing, submodular function. It can be verified that for any $A \subseteq V$, the marginal profit function $f_A(\cdot)$ is also a monotone nondecreasing, submodular function.

Definition 2 (Minimum Submodular Cover Problem (MinSMC)) Given a monotone nondecreasing integer submodular function $f\colon 2^V \to \mathbb{Z}^+$, a cost function $c : V \to \mathbb{R}^+$, an integer $k \leq n$, MinSMC can be formulated as follows:

$$\min\{c(A)\colon A \subseteq V, f(A) \geq k\}, \tag{1}$$

where $c(A) = \sum_{v \in A} c(v)$.

Define a function g as $g(A) = \min\{f(A), k\}$ for any subset $A \subseteq V$. When f is a monotone nondecreasing submodular function, it can be verified that g is also a monotone nondecreasing submodular function. Note that $\max\{g(A)\colon A \subseteq V\} = k = g(V)$, and for the modified MinSMC

$$\min\{c(A)\colon g(A) = g(V)\}, \tag{2}$$

a set A is feasible to (2) if and only if A is feasible to (1). Hence, problems (1) and (2) are equivalent in terms of approximability, that is, A is an α-approximate solution to problem (2) if and only if A is an α-approximate solution to problem (1). In the following, we concentrate on the modified MinSMC (2).

We next introduce the notation of *ε-nearly independent set*, which is adapted from the concept of *ε-maximal nearly independent set* (ε-MaxNIS) (Blelloch et al., 2011).

Definition 3 (ε-nearly independent set (ε-NIS)) For a real number $\varepsilon > 0$ and a set $S \subseteq V$, we say that a set $J \subseteq V \setminus S$ is an ε-NIS with respect to S and ε if J satisfies the following nearly independent property:

$$g_S(J) \geq (1-\varepsilon)^2 \sum_{v \in J} g_S(v). \tag{3}$$

At last, we introduce the notation of *adaptive round* from (Balkanski and Singer, 2018). For simplicity, we use *adaptive round* and *round* interchangeably in the rest of this paper.

Definition 4 (Adaptive round) *Given a value oracle f, which receives a set $S \subseteq N$ and returns its value $f(S)$, we say an algorithm takes γ adaptive rounds if (1) every query to f in round $i \in [\gamma]$ depends only on the answers to queries in rounds 1 to $i-1$, and (2) it performs polynomially-many parallel queries in each adaptive round.*

2.2. Outline of Our Algorithm

We first present an outline of our algorithm. The main algorithm is presented in Algorithm 1, which can be viewed as a parallel implementation of the sequential greedy algorithm (Wolsey, 1982). The overall design of our algorithm follows the framework of adaptive greedy cover algorithm presented in (Fahrbach et al., 2019) for the unweighted MinSMC. However, we need to design a new algorithm and perform more sophisticated analysis to achieve a better approximation ratio for the general MinSMC. Our basic idea is to process elements in batches in accordance with their profit-to-cost ratio and profits. Specifically, in each iteration of our algorithm, we first collect a group of elements with similar profit-to-cost ratio and profits, then we pick an ε-NIS from them using Algorithm 2 and add it to the solution. The main idea of Algorithm 2 is to guess the size of a largest ε-NIS, and for each guess, we use a mean function (Algorithm 3) to verify whether it is an ε-NIS or not. Let $c_{\max}$ and $c_{\min}$ be the maximum and the minimum cost of elements, respectively. Unfortunately, the running time of Algorithm 1 is dependent on $c_{\max}/c_{\min}$, whose value could be arbitrarily large. In order to achieve the running time of $O(\frac{\log(km)\log k(\log m+\log\log(mk))}{\varepsilon^4})$, we add a preprocessing stage to Algorithm 1 to ensure that $c_{\max}/c_{\min}$ is upper bounded by $O(\log(mk))$. Our final algorithm is presented in Algorithm 4.

Algorithm 1 MinSMC-Par$(V, g, c, k, \varepsilon)$

```
Input: MinSMC-Par instance $(V, g, c, k, \varepsilon)$.
Output: A subset $B \subseteq V$ with $g(B) \geq k$.
 1: $t \leftarrow 1$
 2: $\beta = \max_{v \in V} g(v)/c(v)$
 3: $\tau = \max_{v \in V} g(v)$
 4: $T = \log_{1/(1-\varepsilon)}(kc_{\max}/c_{\min})$
 5: $\ell = \log_{1/(1-\varepsilon)} k$
 6: $t' \leftarrow 1$
 7: $B_1^1 \leftarrow \emptyset$
 8: while $t \leq T$ do
 9:     while $t' \leq \ell$ do
10:         $A_t^{t'} = \left\{ v \in V : \begin{array}{l} (1-\varepsilon)^t \beta \leq g_{B_t^{t'}}(v)/c(v) \leq (1-\varepsilon)^{t-1}\beta, \\ (1-\varepsilon)^{t'} \tau \leq g_{B_t^{t'}}(v) \leq (1-\varepsilon)^{t'-1}\tau \end{array} \right\}$
11:         if $A_t^{t'} = \emptyset$ then
12:             break and go to line 18
13:         end if
14:         $J_t^{t'} \leftarrow \text{NIS}(A_t^{t'}, B_t^{t'}, \varepsilon, \ell, (1-\varepsilon)^{t-1}\beta, (1-\varepsilon)^{t'-1}\tau)$
15:         $B_t^{t'+1} \leftarrow B_t^{t'} \cup J_t^{t'}$
16:         $t' \leftarrow t' + 1$
17:     end while
18:     if $g(B_t^{t'}) \geq k$ then
19:         break and go to line 24
20:     end if
21:     $B_{t+1}^1 \leftarrow B_t^{t'}$
22:     $t \leftarrow t + 1$
23: end while
24: return $B \leftarrow B_t^{t'}$
```

2.3. Design of Algorithm 1

Now we are ready to present the details of Algorithm 1. We defer the description of the final Algorithm 4 to the next section. Let $\beta = \max_{v \in V} g(v)/c(v)$ denote the largest profit-to-cost ratio of a single element. In each round (line 10 of Algorithm 1), we construct a bucket $A_t^{t'}$ such that all elements in $A_t^{t'}$ have similar marginal profits and marginal profit-to-cost ratio, e.g.,

$$A_t^{t'} = \left\{ v \in V : \begin{array}{l} (1-\varepsilon)^t \beta \leq g_{B_t^{t'}}(v)/c(v) \leq (1-\varepsilon)^{t-1}\beta, \\ (1-\varepsilon)^{t'} \tau \leq g_{B_t^{t'}}(v) \leq (1-\varepsilon)^{t'-1}\tau \end{array} \right\},$$

where $B_t^{t'}$ denotes the set of already selected elements before this round, and $\tau = \max_{v \in V} g(v)$ is the largest profit of a single element; then it picks an ε-NIS with respect to $B_t^{t'}$ from $A_t^{t'}$ using Algorithm 2 (see line 14 and line 15 of Algorithm 1) and adds it to the solution. A detailed description of Algorithm 2 will be provided in the next paragraph. In the process of selecting elements, we give higher priority to those elements with higher profit-to-cost ratio. For those elements with the

Algorithm 2 NIS$(A, B, \varepsilon, \ell, \beta, \tau)$

Input: Two sets A and B, two threshold values β and τ, a constant $0 < \varepsilon < 1/4$, a parameter ℓ.
Output: An ε-nearly independent set $J \subseteq A$ with respect to B.

```
1: $J_1 \leftarrow \emptyset$
2: $p \leftarrow 1$
3: $i \leftarrow -1$
4: $A_0 \leftarrow A$
5: $B_1 \leftarrow B$
6: $\bar{\varepsilon} \leftarrow \frac{1}{3}(1 - \frac{1}{2T\ell})\varepsilon$
7: $r \leftarrow \log_{\frac{1}{1-\bar{\varepsilon}}}(2mT\ell)/\varepsilon$
8: $\delta \leftarrow \varepsilon/(2rkT^2\ell)$
9: while $p \leq r$ do
10:    $A_p \leftarrow \{v \in A_{p-1} \colon (1-\varepsilon)\beta \leq g_{B_p}(v)/c(v) \leq \beta, (1-\varepsilon)\tau \leq g_{B_p}(v) \leq \tau\}$
11:    if $A_p \leftarrow \emptyset$ then
12:        break (exit the while loop)
13:    end if
14:    for $i \leq \log_{1+\bar{\varepsilon}} m$ do
15:        $t_p \leftarrow \min\{\lfloor (1+\bar{\varepsilon})^i \rfloor, |A_p|\}$
16:        $\bar{\mu}_p \leftarrow$Mean$(B_p, A_p, t_p, \tau, \bar{\varepsilon}, \delta)$
17:        if $\bar{\mu}_p \leq 1 - 1.5\bar{\varepsilon}$ then
18:            break (exit the for loop)
19:        end if
20:        $i \leftarrow i + 1$
21:    end for
22:    select a $t_p$-set $T_p$ from $A_p$ uniformly at random, and let $J_{p+1} \leftarrow J_p \cup T_p$
23:    $B_{p+1} \leftarrow B_p \cup J_{p+1}$
24:    if $g(B_{p+1}) \geq k$ then
25:        break (exit the while loop)
26:    end if
27:    $p \leftarrow p + 1$
28: end while
29: Return $J_p$
```

same profit-to-cost ratio, we give higher priority to those with higher marginal profit. We can prove that when the algorithm terminates, with high probability, it outputs a feasible solution with a good approximation.

We next explain Algorithm 2 in details. Given two sets A and B, a constant $0 < \varepsilon < 1/4$, the goal of Algorithm 2 is to compute an ε-NIS with respect to B from A. For ease of presentation, we call a set consisting of t elements *t-set*. Starting with $p = 1$, $B_1 = B$ and $A_0 = A$. In the p-th round of the while loop (line 9), we compute a "good" ε-NIS with respect to B_p from A_p, where B_p is the set of selected elements before round p and A_p is defined in line 10. Then we add this ε-NIS to B_p to obtain B_{p+1}. This process takes at most r rounds, where $r = \log_{\frac{1}{1-\bar{\varepsilon}}}(2mT\ell)/\varepsilon$. To compute the ε-NIS in each round p, we use a for loop (line 14) to guess its size t_p. For each guess, we use Algorithm 3 to measure the expected quality of a t_p-set that is sampled from A_p uniformly at random. Note that we can try all $\log_{1+\bar{\varepsilon}} m$ guesses in parallel.

We next introduce the design of Algorithm 3. We first define a function $I_{t,B,A,\tau,\epsilon}$ as follows. Given two sets A, B, a parameter τ, and a real number $0 < \varepsilon < 1$, for a t-set X, and an element x from $A \setminus X$, define

$$I_{t,B,A,\tau,\epsilon}(X, x) = I[g_{B\cup X}(x) \geq (1-\varepsilon)\tau], \text{ where } I[\cdot] \text{ is an indicator function,}$$

that is, $I_{t,B,A,\tau,\epsilon}(X, x) = 1$ if $g_{B\cup X}(x) \geq (1-\varepsilon)\tau$, and $I_{t,B,A,\tau,\epsilon}(X, x) = 0$ otherwise. As a convention,

$$\text{if } A \setminus X = \emptyset, \text{ define } I_{t,B,A,\tau,\epsilon}(X, x) = 0. \tag{4}$$

With the above function, Algorithm 3 computes $\bar{\mu}_p$, which is the estimated expectation of $I_{t_p,B_p,A_p,\tau,\epsilon}(X, x)$ assuming that X is a t_p-set that is selected from A_p uniformly at random, and x is an element that is drawn uniformly at random from $A_p \setminus X$. It will become clear later that there exists a t_p which ensures that (1) $\bar{\mu}_p \leq 1 - 1.5\bar{\varepsilon}$, and (2) the random set T_p returned from line 22 of Algorithm 2 is an ε-NIS with respect to B_p, with high probability.

Algorithm 3 Mean$(B, A, t, \tau, \bar{\varepsilon}, \delta)$

Input: $(B, A, t, \tau, \bar{\varepsilon}, \delta)$.

Output: $\bar{\mu}$.

```
1: set the number of samples $m' \leftarrow 8\lceil \log(2/\delta)/\bar{\varepsilon}^2 \rceil$
2: sample $m'$ sets $X_1, \dots, X_{m'}$ and $m'$ elements $x_1, \dots, x_{m'}$, where each $X_i$ is a $t$-set selected
   from $A$ uniformly at random, and $x_i$ is an element sampled from $A \setminus X_i$ uniformly at random
3: return $\bar{\mu} \leftarrow \frac{1}{m'} \sum_{i=1}^{m'} I_{t,B,A,\tau,\epsilon}(X_i, x_i)$
```

Unless specified otherwise, we assume X (resp. X') is a t-set (resp. t'-set) that is selected from A uniformly at random, and x (resp. x') is an element that is drawn uniformly at random from $A \setminus X$ (resp. $A \setminus X'$). We next present a useful lemma to show that $\mathbb{E}_{X,x}[I_{t,B,A,\tau,\epsilon}(X, x)]$ is monotone non-increasing with respect to the sample size t. In the rest of this paper, we will omit the subscript from $\mathbb{E}_{X,x}[\cdot]$ if it is clear from the context, and use a shorthand notation $I_t(X, x)$ to denote $I_{t,B,A,\tau,\epsilon}(X, x)$.

Lemma 5 *Given B, A, τ, ϵ, suppose t and t' are two integers with $t < t'$. Then*

$$\mathbb{E}[I_t(X, x)] \geq \mathbb{E}[I_{t'}(X', x')].$$

The following lemma reveals the relation between $\bar{\mu}_p$ and $\mathbb{E}[I_{t_p,B_p,A_p,\tau,\epsilon}(X, x)]$.

Lemma 6 *With probability at least $1 - \delta$, $\mathbb{E}[I_{t_p,B_p,A_p,\tau,\epsilon}(X, x)] \leq 1 - \bar{\varepsilon}$ if $\bar{\mu}_p \leq 1 - 1.5\bar{\varepsilon}$, and $\mathbb{E}[I_{t_p,B_p,A_p,\tau,\epsilon}(X, x)] \geq 1 - 2\bar{\varepsilon}$ if $\bar{\mu}_p > 1 - 1.5\bar{\varepsilon}$.*

2.4. Performance analysis

In this section, we analyze the running time and the approximation ratio of Algorithm 1. We first provide some technical lemmas. The first lemma shows that the expected size of A_p in Algorithm 2 decreases exponentially as p grows, which implies that A_p will become empty in at most $\log_{1+\bar{\varepsilon}} m$ rounds. Note that in line 14 of Algorithm 2, if $i = \log_{1+\bar{\varepsilon}} m$, then $t_p = |A_p|$, which implies $\bar{\mu}_p = 0$ and thus $\bar{\mu}_p \leq 1 - 1.5\bar{\varepsilon}$. This indicates that line 18 in Algorithm 2 is guaranteed to be triggered.

Lemma 7 *If line 18 in Algorithm 2 is triggered (the for loop is exited), then* $\mathbb{E}[|A_{p+1}|] \leq (1-\varepsilon)|A_p|$ *with probability at least* $1-\delta$*, where* $\mathbb{E}[|A_{p+1}|]$ *denotes the expected size of* A_{p+1} *conditioned on a fixed* A_p*.*

Proof The inequality is obvious if $A_{p+1} = \emptyset$. In the following, assume $A_{p+1} \neq \emptyset$.

By the assumption of this lemma, we have $\bar{\mu}_p \leq 1 - 1.5\bar{\varepsilon}$. Then by Lemma 6, with probability at least $1-\delta$,

$$\mathbb{E}[I_{t_p,B_p,A_p,\tau,\epsilon}(X,x)] \leq 1 - \bar{\varepsilon}. \tag{5}$$

Note that once T_p is picked, for any element $x \in A_p$, we move x to A_{p+1} only if $I[g_{B_p \cup T_p}(x) \geq (1-\varepsilon)\tau, g_{B_p \cup T_p}(x)/c(x) \geq (1-\varepsilon)\beta] = 1$; also note that $I[g_{B_p \cup T_p}(x) \geq (1-\varepsilon)\tau, g_{B_p \cup T_p}(x)/c(x) \geq (1-\varepsilon)\beta] = 0$ if $x \in T_p$. It follows that

$$\begin{aligned}\mathbb{E}[|A_{p+1}|] &= \sum_{x \in A_p \setminus T_p} I[g_{B_p \cup T_p}(x) \geq (1-\varepsilon)\tau, g_{B_p \cup T_p}(x)/c(x) \geq (1-\varepsilon)\beta] \\ &\leq \sum_{x \in A_p \setminus T_p} I[g_{B_p \cup T_p}(x) \geq (1-\varepsilon)\tau].\end{aligned}$$

It follows that

$$\begin{aligned}\mathbb{E}\left[\frac{|A_{p+1}|}{|A_p \setminus T_p|}\right] &= \sum_{T_p} \Pr[T_p \text{ is picked}]\mathbb{E}\left[\frac{|A_{p+1}|}{|A_p \setminus T_p|} | T_p\right] \\ &\leq \sum_{T_p} \Pr[T_p \text{ is picked}] \left(\sum_{x \in A_p \setminus T_p} \Pr[x \text{ is picked} | T_p] \frac{I[g_{B_p \cup T_p}(x) \geq (1-\varepsilon)\tau]}{|A_p \setminus T_p|} \right) \\ &= \sum_{T_p, x \in A_p \setminus T_p} \Pr[T_p, x \text{ are picked}] \frac{I[g_{B_p \cup T_p}(x) \geq (1-\varepsilon)\tau]}{|A_p \setminus T_p|} \\ &\leq \sum_{T_p, x \in A_p \setminus T_p} \Pr[T_p, x \text{ are picked}] I[g_{B_p \cup T_p}(x) \geq (1-\varepsilon)\tau] \\ &= \mathbb{E}[I_{t_p,B_p,A_p,\tau,\epsilon}(T,x)],\end{aligned}$$

where the second inequality uses the observation that $|A_p \setminus T_p| \geq 1$ (since $A_{p+1} \neq \emptyset$). Combining this with inequality (5), we have $\mathbb{E}\left[\frac{|A_{p+1}|}{|A_p \setminus T_p|}\right] \leq 1 - \bar{\varepsilon}$. Thus $\mathbb{E}[|A_{p+1}|] \leq (1-\varepsilon)\mathbb{E}[|A_p \setminus T_p|] \leq (1-\bar{\varepsilon})|A_p|$. The lemma is proved. ■

For ease of presentation, we call $A_t^{t'}$ (line 10 of Algorithm 2) as a *subordinate bucket* and $A_t = \{v \in V\colon (1-\varepsilon)^t\beta \leq g_{B_t^{t'}}(v)/c(v) \leq (1-\varepsilon)^{t-1}\beta\}$ as a *primary bucket*. The following lemma says that for any $t \leq T$ and $t' \leq \ell$, when line 14 of Algorithm 1 returns a set $J_t^{t'}$, the subordinate bucket $A_t^{t'}$ becomes empty with probability at least $1 - \varepsilon/(T^2\ell)$.

Lemma 8 *When Algorithm 2 reaches line 29, the subordinate bucket* A_p *(line 10) becomes empty with probability at least* $1 - \varepsilon/(T^2\ell)$*.*

The following corollary shows that when the inner while loop of Algorithm 1 halts, the primary bucket A_t becomes empty with probability at least $1 - \varepsilon/T^2$.

Corollary 9 *After J_t^ℓ is computed (line 14 of Algorithm 1), the primary bucket A_t becomes empty with probability at least $1-\varepsilon/T^2$.*

The next lemma shows that with probability at least $1-\delta kr$, the set $J_t^{t'}$ computed in line 14 of Algorithm 1 satisfies the nearly independent property defined in (3).

Lemma 10 $\mathbb{E}[g_{B_t^{t'}}(J_t^{t'})] \geq (1-\varepsilon)^2 \sum_{v\in J_t^{t'}} g_{B_t^{t'}}(v)$ *with probability at least* $1-\delta kr$.

Proof Consider the case when the input B of Algorithm 2 is $B_t^{t'}$. That is, $B_1 = B_t^{t'}$ (line 5 of Algorithm 2). We first prove that for any round p, with probability at least $1-n\delta$, the random set T_p (line 22 of Algorithm 2) satisfies

$$\mathbb{E}[g_{B_p}(T_p)] \geq (1-\varepsilon)^2 \sum_{v\in T_p} g_{B_1}(v), \tag{6}$$

where B_p is computed in line 23 of Algorithm 2. We consider a fixed round p in the rest of this proof. For ease of presentation, denote the size of T_p as t^*. Inequality (6) is obviously true if $t^*=0$ or 1. Next, suppose $t^* \geq 2$. Note that line 22 of Algorithm 2 is executed after we jumped out of the for loop. Further note that this jump out is always due to line 17. In fact, if the number of iterations the for loop takes has reached $\log_{1+\bar{\varepsilon}} m$, then $t_p = |A_p|$, and every X_i in Algorithm 3 is A_p, resulting in $\bar{\mu}_p = 0$ (see (4)), in which case the condition of line 17 is satisfied. In the previous round of the for loop, that is, when t_p tries the value $\bar{t} = t^*/(1+\bar{\varepsilon})$, we must have $\bar{\mu}_p > 1-1.5\bar{\varepsilon}$, and thus

$$\mathbb{E}[I_{\bar{t},B_p,A_p,\tau,\epsilon}(X,x)] \geq 1-2\bar{\varepsilon} \tag{7}$$

by Lemma 6. Assume that $T_p = \{v_1,\dots,v_{t^*}\}$, and for any $i \leq t^*$, denote $T_p^i = \{v_1,\dots,v_i\}$. By the monotonicity of g, we have

$$\mathbb{E}[g_{B_p}(T_p)] \geq \mathbb{E}[g_{B_p}(T_p^{\bar{t}})] = \sum_{i=1}^{\bar{t}} \mathbb{E}[g_{B_p\cup T_p^{i-1}}(v_i)]. \tag{8}$$

By the definition of $I_{i,B_p,A_p,\tau,\epsilon}(X,x)$ and Markov's inequality,

$$\mathbb{E}[I_{i,B_p,A_p,\tau,\epsilon}(T_p^i, v_{i+1})] = \Pr[g_{B_p\cup T_p^i}(v_{i+1}) \geq (1-\varepsilon)\tau] \leq \frac{\mathbb{E}[g_{B_p\cup T_p^i}(v_{i+1})]}{(1-\varepsilon)\tau}. \tag{9}$$

Combining inequalities (8) and (9), we have

$$\mathbb{E}[g_{B_p}(T_p)] \geq (1-\varepsilon)\tau \cdot \sum_{i=1}^{\bar{t}} \mathbb{E}[I_{i,B_p,A_p,\tau,\epsilon}(T_p^i, v_{i+1})]. \tag{10}$$

For any $i \leq \bar{t}$, by Lemma 5 and inequality (7), with probability at least $1-\delta$,

$$\mathbb{E}[I_{i,B_p,A_p,\tau,\epsilon}(T_p^i, v_{i+1})] \geq 1-2\bar{\varepsilon}. \tag{11}$$

Combining inequalities (10), (11), and the union bound, with probability at least $1 - n\delta$,

$$\begin{aligned}\mathbb{E}[g_{B_p}(T_p)] \geq\ & (1 - 2\bar{\varepsilon})\bar{t}(1 - \varepsilon)\tau \\ =\ & \frac{t^*}{1 + \bar{\varepsilon}}(1 - 2\bar{\varepsilon})(1 - \varepsilon)\tau \\ \geq\ & (1 - \varepsilon)^2 t^* \tau,\end{aligned} \tag{12}$$

where the last inequality is due to the choice of $\bar{\varepsilon}$. According to line 10 and line 14 of Algorithm 1, when Algorithm 2 is triggered, we have $g_{B_1}(v) \leq \tau$ for any $v \in A$, with respect to the input parameter τ. It follows that $g_{B_1}(v) \leq \tau$ holds for every $v \in T_p$. Combining this with (12), inequality (6) is proved.

Then, by the union bound, and a proof similar to the proof of Corollary 9, with probability at least $1 - \delta k r$,

$$\sum_{p=1}^{r} \mathbb{E}[g_{B_p}(T_p)] \geq (1 - \varepsilon)^2 \sum_{i=p}^{r} \sum_{v \in T_p} g_{B_1}(v) \tag{13}$$

Combining this with $J_t^{t'} = \bigcup_{p=1}^{r} T_p$, with probability at least $1 - \delta k r$,

$$\begin{aligned}\mathbb{E}[g_{B_t^{t'}}(J_t^{t'})] =\ & \sum_{p=1}^{r} \mathbb{E}[g_{B_p}(T_p)] \\ \geq\ & (1 - \varepsilon)^2 \sum_{p=1}^{r} \sum_{v \in T_p} g_{B_1}(v) \\ =\ & (1 - \varepsilon)^2 \sum_{v \in J_t^{t'}} g_{B_t^{t'}}(v),\end{aligned}$$

where the last inequality is due to $B_1 = B_t^{t'}$ and $J_t^{t'} = \bigcup_{p=1}^{r} T_p$. ■

Without loss of generality, we assume that every inner while loop of Algorithm 1 is executed ℓ times. Denote by $D_t = J_t^1 \cup J_t^2 \cup \ldots \cup J_t^\ell$ for any $t \leq T$. The following corollary shows that the expected cost effectiveness of D_t decreases geometrically as t grows.

Corollary 11 *For any $t \leq T$, $\frac{\mathbb{E}[g_{B_t^1}(D_t)]}{c(D_t)} \geq (1 - \varepsilon)^{t+2}\beta$ with probability at least $1 - \delta k r \ell$.*

Now, we are ready to analyze the expected performance of Algorithm 1.

Theorem 12 *For any constant $0 < \varepsilon < 1/4$, with probability at least $1 - 3\varepsilon$, Algorithm 1 achieves an approximation ratio of $\frac{H(\min\{\Delta, k\})}{1 - 4\varepsilon}$, where $\Delta = \max_{v \in V} f(v)$. It takes at most $O(\frac{T \log k (\log m + \log(T \log k))}{\varepsilon^3})$ rounds.*

Proof We first analyze the running time of Algorithm 1. The two layers of while loops takes at most $T\ell$ iterations, where $T = \log_{1/(1-\varepsilon)}(kc_{\max}/c_{\min})$ and $\ell = \log_{1/(1-\varepsilon)} k$. In each iteration, it calls Algorithm 2 to find an ε-NIS. Recall that the for loop in Algorithm 2 can be processed in parallel. Moreover, Algorithm 3 can also be parallelized using m' parallel queries. It follows that Algorithm 2 takes at most r rounds, where $r = \log_{\frac{1}{1-\bar{\varepsilon}}}(2mT\ell)/\varepsilon$. Hence, the running time of Algorithm 1 is

Algorithm	**# of adaptive rounds**
Algorithm 1	$O(T\ell \times$ number of rounds of Algorithm 2$)$
Algorithm 2	$O(r \times$ number of rounds of Algorithm 3$)$
Algorithm 3	$O(1)$

Table 2: Summary of Running Time Analysis

$O(T\ell r) = O(\frac{T\log k(\log m + \log(T\log k))}{\varepsilon^3})$. A summary of running time analysis is presented in Table 2.

Next we analyze the approximation ratio of Algorithm 1. Let B be the output of Algorithm 1, then $B = D_1 \cup \ldots \cup D_T$, $B_t^1 = D_1 \cup \cdots D_{t-1}$ for $t \geq 2$ and $B_1^1 = \emptyset$, where $D_t = J_t^1 \cup J_t^2 \cup \ldots \cup J_t^\ell$ for $t \leq T$. The following claim shows that we can group $B = D_1 \cup \ldots \cup D_T$ into a sequence of sets whose expected cost-effectiveness is monotonically increasing.

Claim 1. We can group $B = D_1 \cup \ldots \cup D_T$ into a sequence of sets $D'_1, D'_2, \ldots, D'_p$ with $p \leq T$ such that with probability at least $1 - 3\varepsilon/2$,

$$\frac{\mathbb{E}[g_{B'_i}(D'_{i+1})]}{c(D'_{i+1})} \leq \frac{\mathbb{E}[g_{B'_{i-1}}(D'_i)]}{c(D'_i)} \tag{14}$$

holds for any $i \leq p$, where $B'_i = D'_1 \cup \ldots \cup D'_i$ for $i \leq p$.

For a set S, denote by $\beta(S) = \max_{v\in V} \frac{g_S(v)}{c(v)}$ the maximum marginal profit-to-cost ratio with respect to S. The next claim shows that the expected cost-effectiveness of D'_i is close to $\beta(B'_{i-1})$.

Claim 2. For any $1 \leq i \leq p-1$, with probability at least $1 - 3\varepsilon/2$,

$$\frac{\mathbb{E}[g_{B'_i}(D'_{i+1})]}{c(D'_{i+1})} \geq (1-\varepsilon)^4 \beta(B'_i).$$

To prove the approximation ratio, we consider an optimal solution A^*, and construct an auxiliary weight w as follows. Denote $r_i = \mathbb{E}[g_{B'_{i-1}}(D'_i)]$ and $z_{v,i} = \mathbb{E}[g_{B'_{i-1}}(v)]$ for $1 \leq i \leq p$ and $v \in A^*$. For any $v \in A^*$, define

$$w(v) = \sum_{i=1}^{p} (z_{v,i} - z_{v,i+1}) \frac{c(D'_i)}{r_i},$$

where $z_{v,p+1} = 0$.

Claim 3. With probability at least $1 - 3\varepsilon/2$, $c(B'_p) \leq \sum_{v\in A^*} w(v)$.

Claim 4. With probability at least $1 - 3\varepsilon/2$, $w(v) \leq c(v) \cdot \frac{H(\min\{\Delta,k\})}{1-4\varepsilon}$.

Combining Claim 3, Claim 4, and the union bound, with probability at least $1 - 3\varepsilon$, $c(B'_p) \leq \frac{H(\min\{\Delta,k\})}{1-4\varepsilon} c(A^*)$. The approximation ratio is proved. ■

3. Completing the Last Piece of the Puzzle: Bounding $c_{\max}/c_{\min}$

Note that the running time in Theorem 12 depends on $T = \log_{1/(1-\varepsilon)}(kc_{\max}/c_{\min})$, where $c_{\max}/c_{\min}$ could be arbitrarily large. To this end, we add a preprocessing step to Algorithm 1 in order to create a modified instance with bounded $c_{\max}/c_{\min}$. The complete algorithm is presented in Algorithm 4. We first sort all elements in non-decreasing order of their costs such that $c(v_1) \leq c(v_2) \leq \ldots \leq c(v_m)$. Then we compute the the minimum j such that $g(\{v_1, \ldots, v_j\}) \geq k$. Notice that $\{v_1, \ldots, v_j\}$ must be a feasible solution to our problem. Let $V_0 \leftarrow \{v \in V : c(v) < \frac{\varepsilon}{mk} c(v_j)\}$ and $V_1 \leftarrow \{v \in V : c(v) > jc(v_j)\}$. That is, V_0 contains all elements with low cost and V_1 contains all elements with high cost. Let $V^{mod} \leftarrow V - (V_0 \cup V_1)$ denote the set of elements with "moderate" cost. Then we apply Algorithm 1 to V^{mod} to obtain an output B^{mod}. Because V^{mod} contains all elements with moderate cost, we can bound the ratio $c_{\max}/c_{\min}$ as $c_{\max}/c_{\min} \leq kmj/\varepsilon$, where we abuse the notations to use $c_{\max}$ and $c_{\min}$ to denote the highest and lowest cost in V^{mod} respectively. At last, $B^{mod} \cup V_0$ is returned as the final solution. We next present the main theorem of this paper.

Theorem 13 *With probability at least $1 - 3\varepsilon$ for any $0 < \varepsilon < 1/5$, Algorithm 4 achieves an approximation ratio of at most $\frac{H(\min\{\Delta,k\})}{1-5\varepsilon}$. It takes $O(\frac{\log(km)\log k(\log m + \log\log(mk))}{\varepsilon^4})$ rounds.*

Algorithm 4 MinSMC-Main

Input: MinSMC instance $\mathcal{I} = (V, g, c, k)$ and a constant $0 < \varepsilon < 1/4$.
Output: A subset $V' \subseteq V$ such that $g(V') \geq k$.

```
1: index all elements in non-decreasing order of their costs
2: $j \leftarrow \arg\min\{i : g(\{v_1, \ldots, v_i\}) \geq k\}$
3: $V_0 \leftarrow \{v \in V : c(v) < \frac{\varepsilon}{mk} c(v_j)\}$
4: $V_1 \leftarrow \{v \in V : c(v) > jc(v_j)\}$
5: $V^{mod} \leftarrow V - (V_0 \cup V_1)$
6: $g^{mod} \leftarrow g_{V_0}$ where $g_{V_0}$ is the marginal profit function of the set over $V_0$
7: $k^{mod} \leftarrow \max\{0, k - g(V_0)\}$
8: Let $\mathcal{I}^{mod} = (V^{mod}, g^{mod}, c, k^{mod}, \varepsilon)$
9: $B^{mod} \leftarrow$MinSMC-Par($\mathcal{I}^{mod}$)
10: $V' \leftarrow B^{mod} \cup V_0$
```

4. Conclusion and Discussion

In this paper, we present a parallel algorithm for the MinSMC to achieve an approximation ratio of $\frac{H(\min\{\Delta,k\})}{1-5\varepsilon}$, with probability at least $1 - 3\varepsilon$, in $O(\frac{\log(km)\log k(\log m + \log\log(mk))}{\varepsilon^4})$ rounds, where $0 < \varepsilon < 1/5$ is a constant. How to obtain a near $H(\min\{\Delta, k\})$-approximation parallel algorithm using less number of rounds is a topic deserving further exploration.

Acknowledgments

This research is supported in part by National Natural Science Foundation of China (11901533, U20A2068, 11771013), Zhejiang Provincial Natural Science Foundation of China (LD19A010001).

References

Eric Balkanski and Yaron Singer. The adaptive complexity of maximizing a submodular function. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1138–1151. ACM, 2018.

B. Berger, J. Rompel, and P.W. Shor. Efficient nc algorithms for set cover with applications to learning and geometry. In *30th Annual Symposium on Foundations of Computer Science*, pages 54–59, 1989. doi: 10.1109/SFCS.1989.63455.

Guy E. Blelloch, Richard Peng, and Kanat Tangwongsan. Linear-work greedy parallel approximate set cover and variants. In *SPAA'11: Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, pages 23–32, 2011.

C. Chekuri and K. Quanrud. Submodular function maximization in parallel via the multilinear relaxation. In *ACM-SIAM Symposium on Discrete Algorithms (SODA19)*, pages 303–322. SIAM, 2019.

K. El-Arini and C. Guestrin. Beyond keyword search: discovering relevant scientific literature. In *KDD'11: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 439–447, 2011.

M. Fahrbach, V. Mirrokni, and M. Zadimoghaddam. Submodular maximization with nearly optimal approximation, adaptivity and query complexity. In *SODA'19*, pages 255–273. SIAM, 2019.

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

D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD'03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, 2003.

A. Krause and C. Guestrin. Intelligent information gathering and submodular function optimization. In *Tutorial at the International Joint Cgoonference in Artificial Intelligence*, 2009.

B. Mirzasoleiman, A. Karbasi, A. Badanidiyuru, and A. Krause. Distributed submodular cover: succinctly summarizing massive data. In *NIPS'15: Proceedings of the 28th International Conference on Neural Information Processing Systems*, 2015.

Sridhar Rajagopalan and Vijay V. Vazirani. Primal-dual rnc approximation algorithms for set cover and covering integer programs. *SIAM Journal on Computing*, 28(2):525–540, 1998.

Yingli Ran, Zhao Zhang, and Shaojie Tang. Improved parallel algorithm for minimum cost submodular cover problem, 2021. URL https://arxiv.org/abs/2108.04416.

S. Tschiatschek, R. Iyer, H. Wei, and J. Bilmes. Learning mixtures of submodular functions for image collection summarization. In *NIPS*, 2014.

L. A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.