Improved Parallel Algorithm for Non-Monotone Submodular Maximization under Knapsack Constraint

Tan D. Tran1, Canh V. Pham1∗, Dung T. K. Ha2, Phuong N.H. Pham3
1ORLab, Faculty of Computer Science, Phenikaa University, Hanoi, Vietnam
2Faculty of Information Technology, VNU University of Engineering and Technology, Hanoi, Vietnam
3Faculty of Information Technology, Ho Chi Minh City University of Industry and Trade, Ho Chi Minh, Vietnam
canh.phamvan@phenikaa-uni.edu.vn, {22027005, 20028008}@vnu.edu.vn, phuongpnh@huit.edu.vn

Abstract

This work proposes an efficient parallel algorithm for non-monotone submodular maximization under a knapsack constraint problem over the ground set of size \( n \). Our algorithm improves the best approximation factor of the existing parallel one from \( 8 + \epsilon \) to \( 7 + \epsilon \) with \( O(\log n) \) adaptive complexity. The key idea of our approach is to create a new alternate threshold algorithmic framework. This strategy alternately constructs two disjoint candidate solutions within a constant number of sequence rounds. Then, the algorithm boosts solution quality without sacrificing the adaptive complexity. Extensive experimental studies on three applications, Revenue Maximization, Image Summarization, and Maximum Weighted Cut, show that our algorithm not only significantly increases solution quality but also requires comparative adaptivity to state-of-the-art algorithms.

1 Introduction

A wide range of instances in artificial intelligence and machine learning have been modeled as a problem of Submodular Maximization under Knapsack constraint (SMK) such as maximum weighted cut [Amanatidis et al., 2020; Han et al., 2021], data summarization [Han et al., 2021; Mirzasoleiman et al., 2016], revenue maximization in social networks [Han et al., 2021; Cui et al., 2023a; Cui et al., 2021], recommendation systems [Amanatidis et al., 2021; Amanatidis et al., 2020]. The attraction of this problem comes from the diversity of submodular utility functions and the generalization of the knapsack constraint. The submodular function has a high ability to gather a vast amount of information from a small subset instead of extracting a whole large set, while the knapsack constraint can represent the budget, the cardinality, or the total time limit for a resource. Hence, people are interested in proposing expensive algorithms for SMK these years [Amanatidis et al., 2021; Han et al., 2021; Cui et al., 2023a; Pham et al., 2023; Amanatidis et al., 2020].

Formally, a SMK problem can be defined such as given a ground set \( V \) of size \( n \), a budget \( B > 0 \), and a non-negative submodular set function (not necessary monotone) \( f : 2^V \rightarrow \mathbb{R}_+ \). Every element \( e \in V \) has its positive cost \( c(e) \). The problem SMK asks to find \( S \subseteq V \) subject to \( c(S) \leq B \) that maximizes \( f(S) \).

One of the main challenges of SMK is addressing big data in which the sizes of applications can grow exponentially. The modern approach is to design approximation algorithms with low query complexity representing the total number of queries to the oracle of \( f \). However, required oracles of \( f \) are often expensive and may take a long time to process on the machine within a single thread. Therefore, people think of designing efficient parallel algorithms that can leverage parallel computer architectures to obtain a good solution promptly. This motivates the adaptive complexity or adaptivity [Balkanski and Singer, 2018] to become an important measurement of parallel algorithms. It is defined as the number of sequential rounds needed if the algorithm can execute polynomial independent queries in parallel. Therefore, the lower the adaptive complexity of an algorithm is, the higher its parallelism is.

In the era of big data now, several algorithms that achieve near-optimal solutions with low adaptive complexities have been developed recently (See Table 1 for an overview of low adaptive algorithms). As can be seen, although recent studies make an outstanding contribution by significantly reducing the adaptive complexity of a constant factor approximation algorithm from \( O(\log^2 n) \) to \( O(\log n) \), there are two drawbacks, including (1) the high query complexities make them become impractical in some instances [Ene and Nguyen, 2020] and (2) there is a huge gap between the high approximation factors of low adaptivity algorithms, e.g. [Amanatidis et al., 2021; Cui et al., 2023a; Cui et al., 2023b], compared to the best one, e.g. [Buchbinder and Feldman, 2019]. This raises to us an interesting question: Is it possible to improve the factor of an approximation algorithm with near-optimal adaptive complexity of \( O(\log n) \)?

Our contributions. In this work, we address the above question by introducing the AST algorithm for the non-monotone SMK problem. AST has an approximation factor of \( 7 + \epsilon \), within a pair of \( O(\log n) \) adaptivity, \( \tilde{O}(nk) \) query complexity, where \( \epsilon \) is a constant input. Therefore, our algorithm improves the best factor of the near-optimal adap-
tive complexity algorithm in [Cui et al., 2023a]. We investigate the performance of our algorithm on three benchmark applications: Revenue Maximization, Image Summarization, and Maximum Weighted Cut. The results show that our algorithm not only significantly improves the solution quality but also requires comparative adaptivity to existing practical algorithms.

**New technical approach.** It is noted that one popular approach to designing parallel algorithms with near-optimal adaptivity of $O(\log n)$ is based on making multiple guesses of the optimal solution in parallel and adapting a threshold sampling method, which selects a batch of elements whose density gains, i.e., the ratio between the marginal gain of an element per its cost, are at least a given threshold within $O(\log n)$ adaptivity [Amanatidis et al., 2021; Cui et al., 2023a]. By making the guesses of the optimal along with calling the threshold sampling multiple times in parallel, the existing algorithms could keep the adaptive complexity of $O(\log n)$ and obtain some approximation ratios.

From another view, we introduce a novel algorithmic framework named “alternate threshold” to improve the approximation factor to $7 + \epsilon$ but keep the same adaptivity and query complexity with the best one [Cui et al., 2023a]. Firstly, we adapt an existing adaptive algorithm to find a near-optimal solution within $O(\log n)$ adaptivity and give a $O(1)$ number of guesses of the optimal solution. Then, the core of our framework consists of a constant number of iterations. It initiates two disjoint candidate sets and then adapts the threshold sampling to upgrade them alternately during iterations: one is updated at odd iterations, and another is updated at even iterations. Thanks to this strategy, we can find the connection between two solutions for supporting each other in evaluating the “utility loss” after each iteration. At the end of this stage, we enhance the solution quality by finding the best element to be added to each candidate’s subsets (pre-fixes of $i$ elements) without violating the budget constraint.

It must be noted that our method differs from the Twin Greedy-based algorithms [Han et al., 2020; Pham et al., 2023; Sun et al., 2022], which update both candidate sets at the same iterations but do not allow the integration of the threshold sampling algorithm for parallelization. Besides, we carefully analyze the role of the highest cost element in the optimal solution to deserve more tightness for the problem.

### 2 Related Works

This section focuses on the related works for the non-monotone case of the SMK problem.

Firstly, regarding the non-adaptive algorithms, the first algorithm for the non-monotone SMK problem was due to [Lee et al., 2010] with the $5 + \epsilon$ factor and polynomial query complexity. Later, several works concentrated on improving both approximation factor and query complexity [Buchbinder and Feldman, 2019; Gupta et al., 2010; Mirzasoleiman et al., 2016; Li, 2018; Sun et al., 2022; Pham et al., 2023; Han et al., 2021]. In this line of works, algorithm of [Buchbinder and Feldman, 2019] archived the best approximation factor of 2.6 but required a high query complexity; the fastest algorithm was proposed by [Pham et al., 2023] with $4 + \epsilon$ factor in linear queries. For the non-monotone Submodular Maximization under Cardinality (SMC) problem, which finds the best solution that does not exceed $k$ elements to maximize a submodular objective value, the best factor of 2.6 of the algorithm in [Buchbinder and Feldman, 2019] still held. Besides, a few algorithmic models have been proposed for improving running time [Badanidiyuru and Vondrák, 2014; Kuhnle, 2021; Li et al., 2022; Buchbinder et al., 2015]. Among them, the fastest algorithm belonged to [Buchbinder et al., 2015] that provided a $\epsilon$ factor within $O(n^{\log(1/\epsilon)}/\epsilon^2)$ queries. However, the above approaches couldn’t be parallelized efficiently by the high adaptive complexity of $\Omega(n)$.

The adaptive complexity was first proposed by [Balkanski and Singer, 2018] for the SMC problem. Regarding adaptivity-based algorithms for non-monotone SMK, the first one belonged to [Ene and Nguyen, 2019] with $\epsilon$ factor and $O(\log^2 n)$ adaptive complexity. However, due to the high query complexity of accessing, the multi-linear extension of a submodular function and its gradient in their method becomes impractical in real applications [Amanatidis et al., 2021; Fahrbach et al., 2019]. After that, [Amanatidis et al., 2021] devised a $(9.465 + \epsilon)$-approximation algorithm within $O(\log n)$, which was optimal up to a $\Theta(\log(\log (n)))$ factor by adopting the lower bound in [Balkanski and Singer, 2018]. It is noted that improving the adaptive complexity of a constant factor algorithm from $O(\log^2 n)$ to $O(\log n)$ made an

<table>
<thead>
<tr>
<th>Reference</th>
<th>Approximation Factor</th>
<th>Adaptive Complexity</th>
<th>Query Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Buchbinder and Feldman, 2019]</td>
<td>2.6</td>
<td>poly(n)</td>
<td>poly(n)</td>
</tr>
<tr>
<td>[Han et al., 2021]</td>
<td>4 + $\epsilon$</td>
<td>$O(n \log k)$</td>
<td>$O(n \log k)$</td>
</tr>
<tr>
<td>[Pham et al., 2023]</td>
<td>4 + $\epsilon$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>[Ene et al., 2019]</td>
<td>$\epsilon$ + $\epsilon$</td>
<td>$O(\log^2 n)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>[Amanatidis et al., 2021]</td>
<td>9.465 + $\epsilon$</td>
<td>$O(\log n)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>[Cui et al., 2023a] (Alg.3)</td>
<td>8 + $\epsilon$</td>
<td>$O(\log n)$</td>
<td>$O(nk)$</td>
</tr>
<tr>
<td></td>
<td>5 + $2\sqrt{2} + \epsilon$</td>
<td>$\approx 7.83 + \epsilon$</td>
<td>$O(\log^2 n)$</td>
</tr>
<tr>
<td>AST (Algorithm 1, this work)</td>
<td>7 + $\epsilon$</td>
<td>$O(\log n)$</td>
<td>$O(nk)$</td>
</tr>
</tbody>
</table>

Table 1: Algorithms for SMK problem. We use the $\hat{O}$ notation throughout the paper to hide $\text{poly}(\log n)$ factors and $k$ is the largest cardinality of any feasible solution. Bold font indicates the best result(s) in each setting.

---

1We refer to threshold sampling methods as ThreshSeq in [Amanatidis et al., 2021] and RandBatch in [Cui et al., 2023a] with $O(\log n)$ adaptivity.
outstanding contribution since it greatly reduced the number of sequential rounds in practical implementation [Cui et al., 2023a; Ene and Nguyen, 2020; Fahrbach et al., 2019]. More recently, [Cui et al., 2023a] created a big step when contributing an efficient parallel one, which resulted in a factor of $8 + \epsilon$ within a pair of $O(\log n)$ adaptivity and $O(nk)$ query complexity. Nevertheless, this factor still has a huge gap with the best factor of $2.6$ [Buchbinder and Feldman, 2019]. They also provided an enhanced version of increasing the approximation factor to $5 + 2\sqrt{2} + \epsilon \approx 7.83 + \epsilon$. However, it required a higher adaptivity of $O(\log^2 n)$. Thus, from this view, the above result of [Cui et al., 2023a] is the best one until now.

People have also focused on developing parallel algorithms for non-monotone SMC these years [Kuhnle, 2021a; Ene and Nguyen, 2020; Fahrbach et al., 2019], etc. The aforementioned contributions of [Ene and Nguyen, 2020] was also applied for SMC to get the best approximation factor of $e + \epsilon$, however it used multi-linear extension and thus had a high query complexity. Next, [Kuhnle, 2021a] and [Fahrbach et al., 2019] tried to reduce the adaptive complexity to $O((\log n)$ with $25.64 + e$ and $6 + \epsilon$ factors. However, [Chen and Kuhnle, 2022] claimed that both [Kuhnle, 2021a] and [Fahrbach et al., 2019] had a non-trivial error because they used the same threshold sampling subroutine which did not work for the non-monotone objective function. [Chen and Kuhnle, 2022] further tried to fixed the previous work and recovered the $6 + \epsilon$ factor in $O((\log n)$ recently. [Amanatidis et al., 2021] improved the factor to $5.83 + e$ in $O((\log n)$ adaptivity. Later, the work of [Cui et al., 2023a] archived the factor of $8 + \epsilon$ in $O((\log n)$ adaptivity or $4 + \epsilon$ factor in $O(\log^2 n)$.

After all, our algorithm overcome the existing drawbacks by an improved parallel version with the approximation factor increasing to $7 + \epsilon$ within $O((\log n)$ rounds to parallel $O(nk)$ queries.

### 3 Preliminaries

Given a ground set $V = \{e_1, \ldots, e_n\}$ and an utility set function $f: 2^V \mapsto \mathbb{R}_+$ to measure the quality of a subset $S \subseteq V$, we use the definition of submodularity based on the diminishing return property: $f: 2^V \mapsto \mathbb{R}_+$ is submodular iff for any $A \subseteq B \subseteq V$ and $e \in V \setminus B$, we have

$$f(e|A) \geq f(e|B).$$

Each element $e \in V$ is assigned a positive cost $c(e) > 0$. Let $c: 2^V \mapsto \mathbb{R}_+$ be a cost function. Assume that $c$ is modular, i.e., $c(S) = \sum_{e \in S} c(e)$ such that $c(S) = 0$ iff $S = \emptyset$.

The problem SLMK asks to find $S \subseteq V$ subject to $c(S) = \sum_{e \in S} c(e) \leq B$ that maximizes $f(S)$. We denote by a tuple $(f, V, B)$ an instance of SLMK. Without loss of generality, $f$ is assumed non-negative, i.e., $f(X) \geq 0$ for all $X \subseteq V$ and normalized, i.e., $f(\emptyset) = 0$. We also assume there exists an oracle query, which, when queried with the set $S$ returns the value $f(S)$.

For convenience, we denote by $S \cup e$ as $S \cup \{e\}$. Next, we denote by $O$ an optimal solution with the optimal value $\text{OPT} = f(O)$ and $r = \arg \max_{e \in O} c(e)$. We also define the contribution gain of a set $T$ to a set $S$ as $f(T|S) = f(T \cup S) - f(S)$. Also, the contribution gain of an element $e$ to a set $S \subseteq V$ is defined as $f(e|S) = f(S \cup \{e\}) - f(S)$ and $f(\emptyset|S)$ is written as $f(e)$ for any $e \in V$.

In this paper, we design a parallel algorithm based on Adaptive complexity or Adaptivity, which is defined as follows:

**Definition 1** (Adaptive complexity or Adaptivity [Balkanski and Singer, 2018]). Given a value of oracle of $f$, the adaptivity or adaptive complexity of an algorithm is the minimum number of rounds needed such that in each round, the algorithm makes $O(\text{poly}(n))$ independent queries to the evaluation oracle.

In the following, we recap two sub-problems which our algorithm need to solve: Unconstrained Submodular Maximization and Density Threshold.

**Unconstrained Submodular Maximization (UnSubMax)** This problem requires to find a subset $S \subseteq V$ that maximizes $f(S)$ without any constraint. The problem was shown NP-hard [Feige et al., 2011a].

To obtain mentioned approximation factor, our algorithm adapts the low adaptivity algorithm in [Chen et al., 2019] that achieves an approximation factor of $(2 + \epsilon)$ in constant adaptive rounds of $O(\log(1/\epsilon)/\epsilon)$ and linear queries of $O(n\log^2(1/\epsilon)/\epsilon^4)$.

**Density Threshold (DS).** The problem receives an instance $(f, V, B)$, a fixed threshold $\tau$ and a parameter $\epsilon > 0$ as inputs, it asks to find a subset $S \subseteq V$ satisfies two conditions: (1) $f(S) \geq c(S) \cdot \tau$; (2) $\sum_{e \in V \setminus S} f(e|S) \leq \epsilon \cdot \text{OPT}$.

Two algorithms in the literature satisfy the above conditions, including those in [Amanatidis et al., 2021] and [Cui et al., 2023a]. In this work, we adapt the RandBatch algorithm in [Cui et al., 2023a]. RandBatch requires the set $I$, a submodular function $f(\cdot)$, and parameters $c, M$ to control the solution’s accuracy and complexities. RandBatch is combined with the aforementioned density thresholds to set up sieves in parallel for SMK. Due to the space limitations, Pseudocode for RandBatch is given in the appendix.

For an instance $(V, f, B)$ of SLMK, two subsets $I, M$ of $V$, a fixed threshold $\theta$ and input parameter $\epsilon$, the performance of RandBatch is provided in the following Lemmas.

**Lemma 1** (Lemma 1 in [Cui et al., 2023a]). The sets $A, L$ output by RandBatch$(\theta, I, M, \epsilon, f(\cdot), c(\cdot))$ satisfy $E[f(A)] \geq (1 - \epsilon)^2 \theta \cdot E[c(A)]$ and $\epsilon \cdot M \cdot \sum_{u \in L} f(u|A) \leq \text{OPT}$.

**Lemma 2** (Lemma 2 in [Cui et al., 2023a]). RandBatch has $O(\frac{1}{\epsilon^2}(\log(|I| \cdot |\theta(\beta(I)) + M|))$ adaptivity, and its query complexity is $O(|I|k)$ times of its adaptive complexity, where $\beta(I) = \max_{u,v} \frac{c(u)}{c(v)}$. If we use binary search in Line 10 of RandBatch, then it has $O(\frac{1}{\epsilon^2}(\log(|I| \cdot |\theta(\beta(I)) + M|) \log(k))$ adaptivity, and its query complexity is $O(|I|)$ times of its adaptivity.

Interestingly, we further explore a useful property of RandBatch when applying it to our algorithm.

**Lemma 3.** The sets $A, L$ output by RandBatch$(\theta, I, M, \epsilon, f(\cdot), c(\cdot))$ satisfy $E[f(A)|A_i - 1] \geq (1 - \epsilon)^2 E[c(A_i)]\theta$, where $A = \{a_1, a_2, \ldots, a_{|A|}\}, A_i = \{a_1, a_2, \ldots, a_i\}$. 

1963
4 Proposed Algorithm

In this section, we introduce AST (Algorithm 1), a $(7 + \epsilon)$-approximation algorithm in $O(\log n)$ adaptivity and $O(n^2 \log^2 n)$ query complexity.

AST receives an instance $(f, V, B)$, constant parameters $\delta, \epsilon, \alpha$ as inputs. It contains two main phases. At the first phase (Lines 1-14), it first divides the ground set $V$ into two subsets: $V_0$ contains elements with small costs, and $V_1$ contains the rest. AST then calls ParSKP1 (Cui et al., 2023a) as a subroutine which returns a $(1/8 - \delta)$-approximation solution within $O(\log n)$ adaptive rounds. Based on that, the algorithm can offer $O(\log(1/e)/\epsilon)$ guesses of the optimal solution for the main loop (Lines 4-14). The main loop consists of $O(\log(1/e)/\epsilon)$ iterations; each corresponds to a guess of the optimal. It sequentially constructs two disjoint solutions, $X$ and $Y$, one at odd iterations and the other at even iterations. The work of the odd and the even is the same. At the odd (or even) ones, it sets the threshold $\theta_X (\theta_Y)$ and calls the RandBatch routine with the ground set $I$ and the function $f(X)$ as inputs to provide the new set $A_i (B_i)$ (Lines 8, 12). It then updates $X (Y)$ and $I$ as the remaining elements (Line 8 or 12).

The second phase (Lines 15-24) is to improve the quality of solutions. If $c(X_i \cup V_0) \leq \epsilon B$, this phase first adapts UnSubMax algorithm [Chen et al., 2019] for unconstrained submodular maximization over $X_i \cup V_0$ to get a candidate solution $S_i$ (Lines 15-16). This step is based on an observation that $X_1$ is important in analyzing the algorithm’s performance. It then selects the sets of the first $i$ elements added into $X$ and $Y$ and finds the best elements without violating the total cost constraint (Lines 19-24). Finally, the algorithm returns the best candidate solution (Lines 25-26). The details of AST are depicted in Algorithm 1.

At the high level, AST works follow a novel framework that combines an alternate threshold greedy algorithm with the boosting phase. The term “alternate” means that candidate solutions are updated alternately with each other in multiple iterations. At each iteration, only one partial solution is updated based on two factors: one guess of the optimal solution and the remaining elements of the ground set that do not belong to the other solution.

It should be emphasized that the alternate threshold greedy differs from recent works [Cui et al., 2023a; Amanatidis et al., 2021] where two candidate solutions for each guess are constructed after only one adaptive round. Alternate threshold greedy also differs from the twin greedy method in [Han et al., 2020], which allows updating both disjoint sets in each iteration. For the theoretical analysis, the key to obtaining a tighter approximation factor lies in aspects: (1) the connections between $X$ and $Y$ after each iteration of the first loop and (2) carefully considering the role of $r$ to eliminate terms that worsen the approximation factor.

We now analyze the performance guarantees of AST. We consider $X$ and $Y$ after ending the first loop. We first introduce some notations regarding AST as follows.

- $X_i^t, Y_i^t$ is the set of first $i$ elements added into $X$ and $Y$, respectively.

Algorithm 1: AST Algorithm

<table>
<thead>
<tr>
<th>Algorithm 1: AST Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> An instance $(f, V, B)$, parameters $\alpha, \epsilon, \delta$</td>
</tr>
<tr>
<td>1. $V_0 \leftarrow {e \in V : c(e) \leq \epsilon B / n}$, $V_1 \leftarrow V \setminus V_0$, $I \leftarrow V_1$, $p \leftarrow 1$</td>
</tr>
<tr>
<td>2. $S_0 \leftarrow \text{ParSKP1}(\frac{1}{7}, \delta, f(\cdot), c(\cdot), \Gamma) \leftarrow \frac{8n f(S_0)}{(1 - \delta) B}$</td>
</tr>
<tr>
<td>3. $X \leftarrow \emptyset$, $Y \leftarrow \emptyset$, $\Delta \leftarrow \lceil \log \frac{1}{\epsilon} \cdot \frac{8n}{c(1 - \delta B)} \rceil + 1$, $M \leftarrow \frac{1}{\epsilon} \left(\frac{\Delta}{2} + 1\right)$</td>
</tr>
<tr>
<td>For $i = 1$ to $\Delta$ do</td>
</tr>
<tr>
<td>if $i$ is odd then</td>
</tr>
<tr>
<td>6. $\theta_X \leftarrow \Gamma(1 - \epsilon)^i$</td>
</tr>
<tr>
<td>7. $(A_1, U_1, L_1) \leftarrow \text{RandBatch}(\theta_X, I, M, p, \epsilon, f(\cdot), c(\cdot))$</td>
</tr>
<tr>
<td>8. $X \leftarrow X \cup A_i$, $I \leftarrow I \setminus X$</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>9. $\theta_Y \leftarrow \Gamma(1 - \epsilon)^i$</td>
</tr>
<tr>
<td>10. $(B_1, U_1, L_1) \leftarrow \text{RandBatch}(\theta_Y, I, M, p, \epsilon, f(\cdot), c(\cdot))$</td>
</tr>
<tr>
<td>11. $Y \leftarrow Y \cup B_i$, $I \leftarrow I \setminus Y$</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>For $T \in {X, Y}$, define: $T_i$ is $T$ after the iteration $i$ of the first loop, $T^i$ is the set of first $i$ elements added into $T$.</td>
</tr>
<tr>
<td>if $c(X_i \cup V_0) \leq \epsilon B$ then</td>
</tr>
<tr>
<td>16. $S_i \leftarrow \text{UnSubMax}(X_i \cup V_0)$</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>for $i = 1$ to $</td>
</tr>
<tr>
<td>17. $a_i \leftarrow \arg \max_{e \in V \setminus {X_i \cup U_e} \leq B} f(X_i^t \cup {e})$</td>
</tr>
<tr>
<td>18. $X_i^t \leftarrow X_i^t \cup {a_i}$</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>for $i = 1$ to $</td>
</tr>
<tr>
<td>19. $b_i \leftarrow \arg \max_{e \in V \setminus {Y_i \cup U_e} \leq B} f(Y_i^t \cup {e})$</td>
</tr>
<tr>
<td>20. $Y_i^t \leftarrow Y_i^t \cup {a_i}$</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>$S \leftarrow \arg \max_{T \in {X^t, Y^t} \cup {S_i}} f(T)$</td>
</tr>
<tr>
<td>return $S$</td>
</tr>
</tbody>
</table>

- $X_i$ and $Y_i$ are $X$ and $Y$ after the iteration $i$ of the first loop (Lines 4-14) and $X_0 = Y_0 = \emptyset$.

$O_1$ is an optimal solution of SMK over instance $(f, V_1, B)$.

$O' = O_1 \setminus X_1, O'' = O_1 \setminus \{r\}$ and $O''' = O' \setminus \{r\}$.

For an element $e \in X \cup Y$, we denote: $X_X, Y_Y, \theta_X$ and $\theta_Y$ as $X$, $Y$, $\theta_X$ and $\theta_Y$ right before $e$ is selected into $X$ or $Y$, respectively; $l(e)$ is the iteration when $e$ is added to in $X$ or $Y$.

Lemma 4 makes a connection between $X$ and $Y$ after each iteration.

**Lemma 4.** After any iteration $i$ of the first loop (Lines 4-14) of AST, we have:

a) If $i \geq 1$, $i$ is odd and $c(X_i) \leq B - c(r)$. Let $T \subseteq Y_{i-1} \cap O_1$, we have $\sum_{e \in T} f(e | X_i) < \sum_{e \in T} \frac{g(f(e | Y_{i-1}))}{(1 - \epsilon B)} + \epsilon$.
OPT.

b) If \( i \geq 2 \), \( i \) is even and \( c(Y_i) \leq B - c(r) \). Let \( T \subseteq X_{i-1} \cap O' \), we have \( \sum_{e \in T} f(e|Y_i) < \sum_{e \in T} \frac{E[f(e|X_{i-1})]}{1-\epsilon^3} + \epsilon \cdot OPT \).

Proof. Prove a) If \( i = 1 \), \( Y_0 = \emptyset \), the Lemma holds. We consider the other case. We divide \( T \) into several subsets including \( T = T_2 \cup T_3 \cup \ldots \cup T_{i-1} \), where \( T_j \) is a set of all elements in \( T \) that are added into \( Y_j \) at iteration \( j \leq i - 1 \). Since \( T_j \subseteq Y_{j-1} \cup O_1 \) and \( c(X_i) \leq B - c(r) \) so \( c(X_{i-1}) + c(e) \leq c(X_i) + c(e) \leq B \) for all \( e \in T_j \). We therefore classify the elements in \( T_j \) into two disjoint sets \( T_j = T_j^1 \cup T_j^2 \), where

\[
T_j^1 = \{ e \in T_j : \frac{f(e|X_{i-1})}{c(e)} < \theta_e^X, c(X_{i-1}) + c(e) \leq B \} \quad \text{and} \quad \quad T_j^2 = \{ e \in T_j : \frac{f(e|X_{i-1})}{c(e)} \geq \theta_e^X, c(X_{i-1}) + c(e) \leq B \}. \]

Since \( (1 - \epsilon)\theta_e^X = \theta_{i-1}^X \forall e \in T_1 \), we have:

\[
\sum_{e \in T_j} f(e|X_{i-1}) = \sum_{e \in T_j^1} f(e|X_{i-1}) + \sum_{e \in T_j^2} f(e|X_{i-1}) \leq \sum_{e \in T_j^1} c(e)\theta_e^X + \frac{OPT}{\epsilon M} \]

\[
= \sum_{e \in T_j^1} c(e)\frac{\theta_e^X}{1-\epsilon} + \frac{OPT}{\epsilon M} \]

\[
\leq \sum_{e \in T_j^1} \frac{E[f(e|X_{i-1})]}{(1-\epsilon)^3} + \frac{OPT}{\epsilon M} \]  

which implies that

\[
\sum_{e \in T} f(e|Y_{i-1}) = \sum_{j=3,\ldots,i-1} \left( \sum_{e \in T_j} f(e|Y_{i-1}) \right) \leq \sum_{j=3,\ldots,i-1} \left( \sum_{e \in T_j^1} f(e|Y_{i-1}) \right) \leq \sum_{j=3,\ldots,i-1} \left( \sum_{e \in T_j^1} \frac{E[f(e|X_{i-1})]}{(1-\epsilon)^3} + \frac{OPT}{\epsilon M} \right) \]

The proof is completed.

By using Lemma 4, we further provide the bound of \( f(O' \cup T) \) for \( T \) is a subset of \( X \) or \( Y \) in Lemma 5 when \( c(r) \) is very large, i.e., \( c(r) \geq (1-\epsilon)B \).

Lemma 5. If \( c(r) \geq (1-\epsilon)B \), one of the following propositions happens

a) \( \mathbb{E}[f(S)] \geq (1-\epsilon)^5\alpha OPT \).

b) There exists a subset \( X' \subseteq X \) so that

\[ f(X' \cup O') < (1 + \frac{1}{(1-\epsilon)^3})\mathbb{E}[f(S)] + (2\epsilon + \frac{1}{\epsilon M})OPT. \]

Similarly, one of the following propositions happens:

c) \( \mathbb{E}[f(S)] \geq (1-\epsilon)^5\alpha OPT \).

d) There exists a subset \( Y' \subseteq Y \) so that

\[ f(Y' \cup O') < (1 + \frac{1}{(1-\epsilon)^3})\mathbb{E}[f(S)] + (2\epsilon + \frac{1}{\epsilon M})OPT. \]

When \( c(X_1) < cB \) and \( c(Y_2) < cB \), it’s easy to obtain the approximation factor due to \( f(S) \geq \max \{ f(X_1), f(Y_2) \} \geq cB\alpha(1-\epsilon)^2 \). Otherwise, we combine Lemma 5 and the fact that \( f(O') \leq (f(O' \cup X) + f(O' \cup Y)) \) to get the bound of \( f(O') \) in Lemma 6. The proofs of them can be found in the Appendix.
Lemma 6. If \( c(X_1) < \epsilon B \) and \( c(Y_2) < \epsilon B \), we have:

- If \( c(r) < (1 - \epsilon)B \), then \( f(O') \leq \frac{5\mathbb{E}[f(S)]}{(1 - \epsilon)^2} + \epsilon OPT \)
- If \( c(r) \geq (1 - \epsilon)B \), one of two things happens:
  1. \( \mathbb{E}[f(S)] \geq (1 - \epsilon)\alpha OPT \)
  2. \( f(O') \leq \frac{5\mathbb{E}[f(S)]}{(1 - \epsilon)^2} + 2(2\epsilon + \frac{1}{4M})OPT \).

Finally, put Lemmata 4, 5, 6 together and divide \( O \) into appropriate subsets, we state the performance of our algorithm in Theorem 4.1.

Theorem 4.1. For \( \alpha = \frac{1}{9}, \epsilon \in (0, \frac{1}{2}), \delta \in (0, \frac{1}{8}) \), Algorithm 1 needs \( O(\log n) \) adaptive complexity and \( O(nk log^2 n) \) query complexity [Cui et al., 2023b]. For the first loop, it calls ThreshSeq \( \Delta = O(\frac{1}{8} \log(\frac{1}{\epsilon})) \times \) times. Each time, RandBatch needs \( O(\frac{1}{8} \log(\frac{1}{\epsilon}) + M) = O(\frac{1}{8} \log(\frac{1}{\epsilon}) + \frac{n}{2} \log(\frac{1}{\epsilon})) = O(log(n)) \) adaptive complexity and \( O(\frac{nk}{2} \log(\frac{1}{\epsilon}) + \frac{n}{2} \log(\frac{1}{\epsilon})) = O(nk log(n)) \) query complexity. In the second phase, the algorithm may need \( O(\frac{1}{8} \log(\frac{1}{\epsilon})) \) adaptivity and \( O(\frac{n}{2} \log(\frac{1}{\epsilon})) \) query complexity to call UnSubMax algorithm [Chen et al., 2019] (Lines 16-17). Then, it only has two adaptive rounds and takes \( O(kn) \) query complexity to find \( X^m \) and \( Y^m \) (Lines 19-24). Therefore, the adaptive complexity of the algorithm is \( O(\frac{1}{8} \log(\frac{1}{\epsilon}) + \frac{n}{2} \log(\frac{1}{\epsilon}) + \frac{1}{2} \log(\frac{1}{\epsilon})) = 2 = O(\log(n)) \) and its query complexity is \( O(nk log^2 n) + O(nk log(n)) + \frac{1}{2} \log(\frac{1}{\epsilon}) + O(nk log(n)) \).

For the approximation factor, we consider the following cases:

Case 1. If \( c(X_1) > \epsilon B \) or \( c(Y_2) > \epsilon B \), we have

\[
\begin{align*}
\mathbb{E}[f(S)] &\geq \max\{f(X_1), f(Y_2)\} \\
&\geq \epsilon B \alpha \Gamma(1 - \epsilon)^2 \\
&\geq \frac{(1 - \epsilon)^2}{7} OPT > \frac{(1 - \epsilon)}{7} OPT.
\end{align*}
\]

Case 2. If \( c(X_1) < \epsilon B \) and \( c(Y_2) < \epsilon B \), we have

\[
\begin{align*}
f(O) &\leq f(O') + f(O \cap (V_1 \cup X_1)) \quad (17) \\
&\leq f(O') + f(O \cap (V_2 \cup X_1)) \quad (18) \\
&\leq f(O') + (2 + \epsilon)\mathbb{E}[f(S)] \quad (19)
\end{align*}
\]

where inequality (17) is due to the submodularity of \( f \) and \( (V_1 \setminus X_1) \cap (V_0 \cap X_1) = \emptyset \), equality (18) is due to the definition of \( \phi \) and inequality (19) is due to applying Algorithm in [Chen et al., 2019]. We now apply Lemma 6 to bound \( f(O') \). If \( c(r) < (1 - \epsilon)B \), then

\[
\begin{align*}
f(O) &\leq \frac{5\mathbb{E}[f(S)]}{(1 - \epsilon)^2} + \epsilon OPT \quad (20) \\
&\leq \frac{7\mathbb{E}[f(S)]}{(1 - \epsilon)^2} + \epsilon OPT. \quad (21)
\end{align*}
\]

Therefore

\[
\mathbb{E}[f(S)] \geq \frac{(1 - \epsilon)^5}{7} OPT > \frac{1 - 5\epsilon}{7} OPT > \frac{(1 - \epsilon)}{7} OPT.
\]

If \( c(r) \geq (1 - \epsilon)B \), we consider two cases:

- If \( a \) in Lemma 6 happens, then

\[
\mathbb{E}[f(S)] \geq \frac{(1 - \epsilon)^5}{7} OPT > \frac{(1 - \epsilon)}{7} OPT.
\]

- If \( b \) in Lemma 6 happens, then

\[
\begin{align*}
f(O) &\leq \frac{5\mathbb{E}[f(S)]}{(1 - \epsilon)^2} + (4\epsilon + \frac{2}{eM})OPT + (2 + \epsilon)\mathbb{E}[f(S)] \\
&\leq \frac{7\mathbb{E}[f(S)]}{(1 - \epsilon)^2} + \frac{29}{7} OPT. \quad (22)
\end{align*}
\]

where the inequality (22) is due to \( \epsilon M = \frac{1}{2}(\frac{4}{7} + 1) > \frac{14}{7} \) for \( \epsilon \in (0, \frac{1}{7}), \delta \in (0, \frac{1}{8}) \). It follows that

\[
\mathbb{E}[f(S)] \geq \frac{1}{7}(1 - \epsilon)^2(1 - 29\epsilon)OPT > \frac{(1 - \epsilon)}{7} OPT
\]

which completes the proof.

\[\square\]

5 Experimental Evaluation

This section evaluates our AST’s performance by comparing our algorithm with state-of-the-art algorithms for non-monotone SMK including:

- ParSKP1: The parallel algorithm in [Cui et al., 2023a] that runs in \( O(\log n) \) adaptivity and returns a solution \( S \) satisfying \( \mathbb{E}[f(S)] \geq (1/8 - \epsilon)OPT \).
- ParSKP2: The algorithm in [Cui et al., 2023a] that runs in \( O(\log^2 n) \) adaptivity and returns a solution of \( \mathbb{E}[f(S)] \geq (1/5 + 2\sqrt{2}) - \epsilon OPT \).
- ParKnapsack: The parallel algorithm in [Amanatidis et al., 2021] achieves an approximation factor of \( (9.465 + \epsilon) \) within \( O(\log n) \).
- SmkRanAcc: The non-adaptive algorithm in [Han et al., 2021] that achieves an approximation factor of \( 4 + \epsilon \) in query complexity of \( O(n \log(k)/\epsilon) \).
- RL+: The non-adaptive algorithm in [Pham et al., 2023] with a factor of \( 4 + \epsilon \) in linear query complexity of \( O(n \log(1/\epsilon)/\epsilon) \).

We experimented with the following three applications:

Revenue Maximization (RM). Given a network \( G = (V, E) \) where \( V \) is a set of nodes and \( E \) is a set of edges. Each edge \((u, v) \in E \) is assigned a positive weight \( w(u, v) \) sampled uniformly in \([0, 1]\) and each node is assigned a positive cost \( c(u) \) defined as \( c(u) = 1 - \epsilon \sqrt{\sum_{(u, v) \in E} w(u, v)} \). The revenue of any subset \( S \subseteq V \) is defined as \( f(S) = \sum_{v \in E} \sqrt{\sum_{u \in S} w(u, v)} \). Given a budget of \( B \), the goal of the problem is to select a set \( S \) with the cost at most \( B \) to maximize \( f(S) \). As in the prior work, [Amanatidis et al., 2021], we can construct the graph \( G \) using a YouTube community network dataset [Han et al., 2021] with 39,841 nodes and 224,235 edges.
Maximum Weighted Cut (MWC). Consider a graph $G = (V, E)$ where each edge $(u, v) \in E$ has a non-negative weight $w(u, v)$. For a node subset $S \subseteq V$, define the weighted cut function $f(S) = \sum_{u \in V \setminus S} \sum_{v \in S} w(u, v)$. The maximum weighted cut problem seeks a subset $S \subseteq V$ that maximizes $f(S)$. As in recent work [Amanatidis et al., 2020], we generate an Erdős–Rényi random graph with 5,000 nodes and an edge probability of 0.2. The node costs $c(u)$ are randomly uniformly sampled from $(0, 1)$.

Image Summarization (IS). Consider a graph $G = (V, E)$ where each node $u \in V$ represents an image, and each edge $(u, v) \in E$ has a weight $w(u, v)$ showing the similarity between images $u$ and $v$. Let $c(u)$ be the cost to acquire image $u$. The goal is to find a representative subset $S \subseteq V$ under the budget $B$ that maximizes a value $f(S) = \sum_{u \in V} \max_{v \in S} w_{u,v} - \frac{1}{|V|} \sum_{u \in V} \sum_{v \in S} w_{u,v}$ [Mirzasoleiman et al., 2016; Han et al., 2021]. As in recent works [Han et al., 2021; Mirzasoleiman et al., 2016], we create an instance as follows: First, randomly sample 500 images from the CIFAR dataset [Krizhevsky, 2019] of 10,000 images, then measure the similarity between images $u$ and $v$ using the cosine similarity of their 3,072-dimensional pixel vectors.

Experiment setting. In our experiments, we set the accuracy parameter $\epsilon = 0.1$ for all algorithms evaluated, and for AST, we set $\delta = 0.12$. We used OpenMP to program with C++ language. Besides, we experimented on a high-performance computing (HPC) server cluster with the following parameters: partition=large, #threads(CPU)=128, node=4, max memory = 3,073 GB. For UnSubMax, we use setting of previous works [Amanatidis et al., 2021; Cui et al., 2023a], i.e, we adapt Algorithm in [Feige et al., 2011b] returning $1/4 - \epsilon$ ratio in one adaptive round and $O(n)$ query complexity.

Experimental Result. In Figures 1(a), (c), and (e), we compare the objective values between different algorithms. The results show that our AST achieves the best objective values for both the RM and MWC applications. In RM, the objective values achieved by RLA, SmkRanAcc, and ParSKP1 are the same. ParKnapsack attains lower objective values while ParSKP2 hits the lowest objective values among the algorithms. Especially, our one marks the highest value when $B = 0.015$, about 1.3 times higher than the others in RM. In IS, ParKnapsack reaches the highest values while ParSKP2 hits the lowest. Our algorithm results in best values at some points and drops at others. As shown in Figure 1 (c), most algorithms fluctuate widely. The variation in the quality of these algorithms might be due to the characteristics of this dataset.

In Figures 1(b), (d), and (f), we make the comparisons about the number of adaptive rounds. The results show that SmkRanAcc and RLA always require the highest number of adaptive rounds across all three applications. For the AST, the number of adaptive rounds is equivalent to ParSKP1 for RM and MWC. Besides, for IS, the adaptive number of rounds for AST is higher than that of ParSKP1, which has the lowest number of rounds. However, the higher number in this case is insignificant. Overall, our algorithm outperforms the others in both solution performance and the quantities of adaptivity.

Figure 1: Performance of algorithms for non-monotone SMK on three instances: (a), (b) Revenue Maximization; (c), (d) Image Summarization and (e), (f) Maximum Weighted Cut. The budget values represent fractions of the total cost of all elements.

6 Conclusions

Motivated by the challenge of the large scale of input data, in this work, we focus on parallel approximation algorithms based on the concept of adaptive complexity. Moreover, the requirement of improving the approximation factor while decreasing the adaptivity down to $\log(n)$ motivates us to propose a competitive new algorithm. We have proposed an efficient parallel algorithm AST based on a novel alternate threshold greedy strategy. To our knowledge, our AST algorithm is the first to achieve a constant factor approximation of $7 + \epsilon$ for the above problem in the aforementioned adaptivity. Our algorithm also expresses the superiority in solution quality and computation complexity compared to state-of-the-art algorithms via some illustrations in the experiment in three real-world applications. In the future, we will address another valuable question: can we reduce the query complexity of parallelized algorithms for the SMK problem?
Acknowledgments

The first author (Tan D. Tran) was funded by the Master, PhD Scholarship Programme of Vingroup Innovation Foundation (VINIF), code VINIF2023.TS.105. This work has been carried out partly at the Vietnam Institute for Advanced Study in Mathematics (VIASM). The second author (Canh V. Pham) would like to thank VIASM for its hospitality and financial support.

References


[Cui et al., 2023b] Shuang Cui, Kai Han, Jing Tang, He Huang, Xueying Li, Aakas Zhiyuli, and Hanxiao Li. Practical parallel algorithms for non-monotone submodular maximization. CoRR, abs/2308.10656, 2023.


[Han et al., 2020] Kai Han, Zongmai Cao, Shuang Cui, and Benwei Wu. Deterministic approximation for submodu-


