Efficient Parallel Algorithms with Linear Queries for Non-Monotone Submodular Maximization

Canh V. Pham[1*], **Tan D. Tran**[1], **Dung T. K. Ha**[1] and **My T. Thai**[2]
[1]ORLab, Faculty of Computer Science
Phenikaa School of Computing, Phenikaa University, Hanoi 12116, Vietnam
[2]Department of Computer and Information Science and Engineering
University of Florida, Gainesville, Florida 32611, USA
{canh.phamvan, tan.trandinh, dung.hathikim}@phenikaa-uni.edu.vn, mythai@cise.ufl.edu

Abstract

In this work, we propose the first constant-approximation algorithms, LinAst and LinAtg, which simultaneously achieve optimal query complexity $O(n)$ and adaptive complexity $O(\log n)$ for non-monotone submodular maximization under a cardinality constraint k over a ground set of size n. Specifically, compared with existing algorithms that attain the best known adaptive complexity of $O(\log n)$, our approach preserves this adaptivity while reducing the query complexity from $O(n \log k)$ to $O(n)$ and improving the approximation ratio from $0.172 - \epsilon$ to $0.193 - \epsilon$. Our algorithms are built upon LinAdapt, which achieves a constant approximation ratio with $O(\log n)$ adaptive rounds and linear query complexity by requiring only $O(1)$ candidate guesses of the optimal value. We further introduce the BoostAdapt algorithm, which improves the approximation guarantee to $0.25 - \epsilon$ with $O(\log n \log k)$ adaptive complexity and $O(n \log k)$ query complexity, based on a novel staggered greedy threshold framework that alternately constructs two disjoint solution sets over $O(\log k)$ sequential rounds. Extensive experiments on standard benchmark datasets demonstrate that our algorithms consistently outperform state-of-the-art methods in terms of solution quality, query complexity, and running time.

1 Introduction

Maximizing a non-negative (not necessarily monotone) submodular set function under a cardinality constraint is a fundamental problem with broad applications in artificial intelligence and machine learning. Notable examples include data summarization [Kuhnle, 2021; Lin and Bilmes, 2011; Fahrbach *et al.*, 2019a; Han *et al.*, 2021; Mirzasoleiman *et al.*, 2016], revenue maximization in social networks [Kuhnle, 2021; Chen and Kuhnle, 2023], recommendation systems [Mirzasoleiman *et al.*, 2016], and weight cut problems [Chen and Kuhnle, 2023; Kuhnle, 2021].

Formally, given a finite ground set V of size n, a submodular set function $f : 2^V \to \mathbb{R}^+$, and a positive integer k representing the cardinality constraint, the goal of the Submodular Maximization under a Cardinality Constraint (SMC) problem is to find a subset $S \subseteq V$ such that $|S| \leq k$ and $f(S)$ is maximized. This problem has attracted extensive attention, with many works focusing on developing approximation algorithms with provable performance guarantees [Buchbinder *et al.*, 2014; Kuhnle, 2019; Chen and Kuhnle, 2023].

However, the exponential growth of the solution space with the input size poses a significant computational challenge, particularly in modern large-scale applications. This has spurred growing interest in designing efficient parallel algorithms capable of leveraging modern parallel computing architectures to compute high-quality solutions rapidly (see Table 1 for a summary of parallelizable methods). In this context, the concept of adaptive complexity (or adaptivity) has emerged as a key metric for assessing the parallel efficiency of such algorithms. A central notion in assessing the parallel efficiency of such algorithms is adaptive complexity (or adaptivity), which measures the number of sequential rounds an algorithm requires, allowing a polynomial number of oracle queries to be processed in parallel during each round [Balkanski and Singer, 2018]. Notably, reducing the adaptive complexity from $O(\log^2 n)$ to $O(\log n)$ has been shown to significantly cut down the number of sequential steps, resulting in notable improvements in practical runtime performance [Fahrbach *et al.*, 2019a; Ene and Nguyen, 2020; Amanatidis *et al.*, 2021; Fahrbach *et al.*, 2023; Kuhnle, 2021; Chen and Kuhnle, 2024]. Although recent advances have successfully brought down the adaptive complexity to $O(\log n)$, several significant challenges remain: (1) The high query complexity made the some parallel algorithms with $O(\log n)$ adaptive complexity impractical for real applications. For instances, [Chen and Kuhnle, 2024] demonstrated that the algorithm in [Ene and Nguyen, 2020] providing the best ratio of $1/e - \epsilon$ within $O(\log n)$ adaptivity is of mostly theoretical interest because it needs $\Omega(nk^2 \log^2(n))$ queries to approximate the multi-linear extension of a submodular function and its gradient.

(2) The approximation ratios of the existing practical parallel algorithms [Fahrbach *et al.*, 2019a; Fahrbach *et al.*, 2023; Kuhnle, 2021; Chen and Kuhnle, 2024; Cui *et al.*, 2023] are

*Corresponding author

Reference	Appr. Ratio	Adaptivity	Query Complexity
[Buchbinder *et al.*, 2015]	$1/e-\epsilon$	$O(k)$	$\boldsymbol{O(n)}$
[Buchbinder and Feldman, 2024]	$0.385-\epsilon$	$O(n)$	$O(n+k^2)$
[Chen *et al.*, 2024b]	$0.385-\epsilon$	$O(n)$	$O(kn)$
[Buchbinder and Feldman, 2024]	$\mathbf{0.401}$	$\mathsf{poly}(n)$	$\mathsf{poly}(n)$
Blits [Balkanski *et al.*, 2018]	$1/(2e)-\epsilon$	$O(\log^2(n))$	$O(\mathsf{opt} n\log^2(n)\log(k))$
[Chekuri and Quanrud, 2019]	$3-2\sqrt{2}-\epsilon$	$O(\log^2(n))$	$O(nk^4\log^2(n))$
[Ene and Nguyen, 2020]	$1/e-\epsilon$	$\boldsymbol{O(\log n)}$	$\Omega(nk^2\log^2(n))$
ANM [Fahrbach *et al.*, 2023]	$0.039-\epsilon$	$\boldsymbol{O(\log n)}$	$O(n\log(k))$
ParKnapSack[Amanatidis *et al.*, 2021]	$0.172-\epsilon$	$\boldsymbol{O(\log n)}$	$O(nk\log(n)\log(k))$
AST [Chen and Kuhnle, 2024]	$1/6-\epsilon$	$\boldsymbol{O(\log n)}$	$O(n\log(k))$
ATG [Chen and Kuhnle, 2024]	$0.193-\epsilon$	$O(\log(n)\log(k))$	$O(n\log(k))$
ParSKP [Cui *et al.*, 2023]	$1/8-\epsilon$	$O(\log(n)\log(k))\|\|\boldsymbol{O(\log n)}$	$O(n\log^2(n)\log(k))\|\|O(nk\log^2(n))$
ParSSP [Cui *et al.*, 2023]	$1/4-\epsilon$	$O(\log^2(n)\log(k))\|\|O(\log^2(n))$	$O(n\log^2(n)\log(k))\|\|O(nk\log^2(n))$
PIG [Chen *et al.*, 2025]	0.25	$O(\log(n)\log(k))$	$O(n\log(n)\log(k))$
PITG [Chen *et al.*, 2025]	$1/e-\epsilon$	$O(\log(n)\log(k))$	$O(n\log(n)\log(k))$
LinAst (this work)	$1/6-\epsilon$	$\boldsymbol{O(\log n)}$	$\boldsymbol{O(n)}$
LinAtg (this work)	$0.193-\epsilon$	$\boldsymbol{O(\log n)}$	$\boldsymbol{O(n)}$
BoostAdapt (this work)	$0.25-\epsilon$	$O(\log(n)\log(k))$	$O(n\log(k))$

Table 1: A comparison of efficient parallel algorithms for SMC is presented across three criteria: approximation ratio, adaptive complexity, and query complexity. The best known results are highlighted in bold.

still low as they have a large gap in the approximation ratios compared to the best non-parallel algorithm for SMC (e.g. the ratio of 0.401 in [Buchbinder and Feldman, 2024]). This raises two interesting questions for us to solve SMC: *Can we improve the query complexity and approximation ratio of a practical parallel algorithm?*

Our contributions and techniques. In this work, we tackle the above questions with the following contributions:

First, we introduce LinAdapt, the first constant approximation ratio $(1/(12+O(\epsilon)))$ within near-optimal adaptivity $O(\log n)$ and linear query complexity $O(n)$, where $\epsilon>0$ is a constant parameter. LinAdapt is used as a building block to reduce the query complexity of our latter algorithms. The key idea to obtain a such improvement lies in: **(1)** constructing two sets S, S' in $O(\log n)$ sequential rounds with an interesting property: $f(X\cup S)\leq O(1)\cdot f(S')$ with high probability for any set $X\subseteq V, |X|\leq k$, and S' with $|S'|\leq k$ is the set of last elements added into S; **(2)** combining **(1)** appropriately with the unconstrained submodular maximization algorithm to get the desired ratio.

Second, we introduce two algorithms LinAst and LinAtg which run $O(\log n)$ adaptivity and $O(n)$ query complexity and return the approximation ratios to $1/6-\epsilon$ and $0.193-\epsilon$, respectively. Therefore, our algorithm LinAtg improves the approximation ratio from $0.172-\epsilon$ to $0.193-\epsilon$ and significantly reduces the query complexity of the current best practical algorithm of [Amanatidis *et al.*, 2021].

Third, we propose a practical algorithm, denoted by BoostAdapt, that achieves an approximation ratio of $1/4-\epsilon$ with $O(\log n\log k)$ adaptive complexity and $O(n\log k)$ query complexity. As a result, BoostAdapt improves the best-known approximation ratio for algorithms with the **same adaptive and query complexity** from $0.193-\epsilon$ to $0.25-\epsilon$. The design of BoostAdapt is inspired by the twin greedy strategy, which incorporates ThreSeq to update two disjoint sets, X and Y, in an alternating manner over only $O(\log k)$ iterations. It must be noted that the staggered threshold is different from the Iterated Greedy [Gupta *et al.*, 2010], which uses two threshold greedy strategies to construct candidate solutions separately.

Finally, to show the consistency between theory and practice, we conducted extensive experiments on two applications: Revenue Maximization and Maximum Weighted Cut. The results show that our algorithms not only produce the better solution quality and query complexity sharply but also provide comparative adaptivity to state-of-the-art algorithms.

Paper Organization. The rest of this work is structured as follows. Section 2 provides the literature review on the studied problem. Notations are presented in Section 3. Sections 4-6 introduce our algorithms and theoretical analysis. Experimental computation is provided in Section 7. Finally, Section 8 concludes this work.

2 Related Work

Approximation Algorithms. The first popular approach to solving the SMC problem in practice is to design approximation algorithms with low query complexity. The greedy algorithm was an effective approach for submodular optimization problems. It sequentially selects elements with the largest marginal gain and explores the diminishing return property to get performance bounds. [Gupta *et al.*, 2010] first developed an iterated greedy algorithm with $1/6$ ratio in $O(nk)$ queries for SMC. The work of [Lee *et al.*, 2010] latter improve the ratio to $1/4-\epsilon$ by developing a local search method but this wasted an expensive query complexity of $O(n^4\log n)$. Significantly, the elegant random greedy in [Buchbinder *et al.*, 2014] could provide the ratio of $1/e$ with $O(nk)$ query complexity. Instead of selecting an element with the best marginal gain as the naive greedy, it chose a uniformly random element from the set of k elements with the largest marginal gain. [Buchbinder *et al.*, 2015] then improved the query complexity to $O(n\log(1/\epsilon)/\epsilon^2)$ but still kept the same $1/e$ ratio.

Recently, independent works [Chen *et al.*, 2025] and [Tukan *et al.*, 2024] improved the ratio to $0.385-\epsilon$ in $O(n^2)$ queries and $O(n)$ adaptive complexity. To the best of our knowledge, the best ratio for SMC was 0.401 by [Buchbinder and Feldman, 2024]. However, this work used the multi-linear extensions method and used a considerably high query complexity of $O(n^5)$ [Chen and Kuhnle, 2023]. Finally, it must be emphasized that the above works weren't well parallelized due to their high adaptive complexities of $\Omega(n)$.

Parallel Algorithms. Research on parallel algorithms was initiated by [Balkanski and Singer, 2018]. In this seminal work, they introduced the concept of *adaptive complexity or adaptivity* that measures the parallelizable of algorithm, showed a lower bound on adaptive complexity of $O(\log(n)/\log(\log n))$ to achieve a constant ratio and devised an $1/3$-approximation algorithm in $O(\log n)$ adaptive rounds for the monotone SMC problem. Since then, many works focused on devising algorithms with low adaptivity with tighter ratio for SMC [Balkanski *et al.*, 2019; Fahrbach *et al.*, 2019b; Ene *et al.*, 2019; Chen *et al.*, 2021]. The best parallel algorithm for monotone SMC had an optimal ratio of $1-1/e-\epsilon$ in $O(\log n)$ adaptivity and $O(n)$ queries was due to [Chen *et al.*, 2021]. However, their performance bounds relied on the monotone objective and did not hold for non-monotone.

For the non-monotone SMC, [Chekuri and Quanrud, 2019; Balkanski *et al.*, 2018] first developed parallelizable algorithms with $3-2\sqrt{2}-\epsilon$ and $1/(2e)-\epsilon$ ratios, respectively; both took $O(\log^2 n)$ adaptivity. [Ene and Nguyen, 2020] improved the ratio to $1/e-\epsilon$ in nearly optimal adaptive complexity of $O(\log n)$. However, due to the high query complexity of $\Omega(nk^2\log^2 n)$ to approximate the multilinear extension of a submodular function and its gradient, the [Ene and Nguyen, 2020]' algorithm and algorithms based on multilinear extension methods in general are still impractical in some real applications [Kuhnle, 2021; Fahrbach *et al.*, 2019a; Chen and Kuhnle, 2024; Cui *et al.*, 2023]. To attack the above issue, [Fahrbach *et al.*, 2019a] first aimed to reduce the query complexity to near-linear of $O(n\log k)$ and still kept the $O(\log n)$ adaptive complexity. However, their algorithm resulted in a small ratio, $0.039-\epsilon$. In subsequent work, [Kuhnle, 2021] tried to boost the ratio to $1/6-\epsilon$ in $O(\log n)$ adaptive rounds with $O(n\log k)$ queries. Their algorithm further improved the ratio to $0.193-\epsilon$ in $O(\log^2 n)$ adaptive rounds by exploring iterated greedy framework in [Gupta *et al.*, 2010]. It must be noted that the ratios of both [Fahrbach *et al.*, 2019a] and [Kuhnle, 2021] may not hold because they adapt the threshold sampling routine in [Fahrbach *et al.*, 2019b], which was pointed out disable with non-monotone functions [Chen and Kuhnle, 2024]. Recently, [Chen and Kuhnle, 2024; Fahrbach *et al.*, 2023] recovered the ratios of [Fahrbach *et al.*, 2019a; Kuhnle, 2021] by some threshold sampling routines with the respective analysis. [Cui *et al.*, 2023] then showed two algorithms with ratios of $1/8-\epsilon$ and $1/4-\epsilon$ with $O(\log n)$ and $O(\log^2 n)$ adaptive complexities, respectively. However, they took at least $\Omega(nk\log^2 n)$ query complexity. More recently, [Chen *et al.*, 2025] proposed two parallel algorithms, PIG and PITG, which achieve approximation ratios of 0.25 and $1/e-\epsilon$, respectively. Both algorithms takes a query complexity of $O(n\log(n)\log(k))$ and an adaptive complexity of $O(\log(n)\log(k))$.

3 Preliminaries

Given a ground set V of size n, and the set function $f:2^V\mapsto\mathbb{R}^+$ is submodular iff it satisfies the diminishing return property, i.e., $f(A\cup\{e\})-f(A)\geq f(B\cup\{e\})-f(B)$ where $A\subseteq B$ and $e\notin B$. The marginal gain (or contribution gain) of a set T to a set S is defined as $f(T|S)=f(T\cup S)-f(S)$. For simplicity, we denote $f(\{e\}|A)$ by $f(e|A)$ and assume f normalized, i.e., $f(\emptyset)=0$. Given a positive number k (cardinality constraint), the SMC problem is to determine $\arg\max_{S\subseteq V,|S|\leq k} f(S)$.

We define $[k]=\{1,2,\ldots,k\}$ for any integer k. We denote an instance of SMC by a tuple (f,V,k) and O is an optimal solution with the optimal value $\text{opt}=f(O)$. In this work, we assume that there exists an oracle query that returns $f(S)$ when queried for the set S with any $S\subseteq V$. The parallelization capacity of an algorithm is evaluated through the following definition.

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

We recap two sub-problems for solving SMC in parallel setting, Batch Selection with Threshold (BST) and unconstrained submodular maximization (USM). Given an instance (f,V,B), a fixed threshold τ and $\epsilon>0$ as inputs, BST asks to find a subset $S\subseteq V$ in $O(\log n)$ adaptive complexity satisfying two conditions: (1) $f(S)\geq(1-\epsilon)\tau|S|$; (2) if $|S|<k$, $f(e|S)<\tau$ for any $e\notin S$.

Although several attempts exist to solve the BST, most only work with a monotone submodular function [Fahrbach *et al.*, 2019b; Kazemi *et al.*, 2019]. In this work, we use the ThreSeq algorithm in [Chen and Kuhnle, 2024] and its theoretical satisfaction to analyze our algorithm's performance in a nonmontone setting.

Theorem 1 (Theorem 3 in [Chen and Kuhnle, 2024])**.** *Let (f,V,k) be an instance of* SMC*. For any constants $\epsilon,\delta>0$, the algorithm* ThreSeq *outputs $A'\subseteq A\subseteq V$ such that the following properties hold: 1) The algorithm succeeds with probability at least $1-\delta/n$. 2) There are $O(n/\epsilon)$ oracle queries in expectation and $O(\log(n/\delta)/\epsilon)$ adaptive complexity. 3) It holds $f(A')\geq(1-\epsilon)\tau|A|$. If $|A|<k$, then $f(e|A)<\tau$ for all $e\in V$. 4) It also holds $f(A')\geq f(A)$ and $|A'|\geq(1-\epsilon)|A|$.*

Regarding USM, which aims to find $\arg\max_{S\subseteq V} f(S)$, there exist several solving methods giving the constant approximation ratios such as the [Fahrbach *et al.*, 2019a]'s algorithm (USM1) provides a ratio of $1/4-\epsilon$ with probability at least $1-\delta$ for the problem in $O(1)$ adaptive round and $O(\frac{1}{\epsilon}\log(\frac{1}{\delta}))$ queries and an essentially optimal algorithm of [Chen *et al.*, 2019] (USM2) slightly improves the approximation ratio to $1/2-\epsilon$ using $O(\log(1/\epsilon)/\epsilon)$ adaptive rounds and $O(n\log^3(1/\epsilon)/\epsilon^4)$ query complexity.

4 A Parallel Algorithm with Linear Queries

In this section, we introduce LinAdapt (Algorithm 1), the first constant ratio approximation algorithm within $O(\log n)$ adaptivity and $O(n)$ query complexity. LinAdapt receives

Algorithm 1: LinAdapt$(f, V, k, \alpha, \epsilon, \delta)$

1: **Input:** f, V, k, α, ϵ, δ.
2: $A, A' \leftarrow$ LinBoundSet$(f, V, k, \alpha, \epsilon, \alpha, \delta/3)$
3: $B, B' \leftarrow$ LinBoundSet$(f, V \setminus A, k, \alpha, \epsilon, \delta/3)$
4: $C \leftarrow$ USM1$(A, \epsilon, \delta/(3n))$
5: **Return** $\arg\max_{S \in \{A', B', C\}} f(S)$

an instance (f, V, k), accuracy parameters $\epsilon, \delta > 0$, a constant $\alpha > 0$ and works in a novel algorithmic framework. LinAdapt first sequentially calls a subroutine LinBoundSet twice to find two candidates: one to get A over the ground set V and the other to get B' over a new ground set $V \setminus A$. Note that A and B in LinAdapt are temporary sets that are useful only for theoretical analysis. For any ground set V, LinBoundSet is a key subroutine with the following nice properties:

1) running in $O(\log(n)/\epsilon^2)$ and $O(n/\epsilon^3)$ query complexity.
2) returning two sets A and A' so that $f(A \cup X) \leq (8 + 2\alpha + \frac{2}{\alpha} + O(\epsilon))f(A')$, for any subset $X \subseteq V, |X| \leq k$ with high probability.

To easily follow the algorithm, we will latter provide a detailed analysis of LinBoundSet in Section 4. Finally, LinAdapt calls USM1 to find another candidate solution C and returns the best among A', B', C. Conceptually, our LinAdapt takes advantage of the properties LinBoundSet (by Theorem 3), and the fact that $f(O) \leq f(O \cup A) + f(O \cup B)$ to give the desired bound of solution. We state the LinAdapt's performance in Theorem 2.

Theorem 2. *For any input (f, V, k), where $\epsilon, \delta \in (0, 1)$, LinAdapt runs in $O(\log(n/\delta)/\epsilon^2)$ adaptive complexity and $O(n/\epsilon^3)$ query complexity, returns a solution S satisfying* $\mathsf{opt} \leq \left(8 + 2\alpha + \frac{2}{\alpha} + (\frac{16}{1-4\epsilon} + 2(1+\frac{1}{\alpha})\frac{2(2-\epsilon)}{(1-\epsilon)(1-2\epsilon)})\epsilon\right) f(S)$ *with probability at least $1 - \delta/n$. If $\alpha = 1$, the algorithm achieves the best ratio of $\frac{1}{12+O(\epsilon)}$.*

Proof. Each subroutine terminates with failure probability $1/(3n)$. By the union bound of probabilities, the failure probability of the algorithm is bounded by $3 \cdot \frac{\delta}{3n} = \frac{\delta}{n}$. Let $a = 2 + \alpha + \frac{1}{\alpha} + (1 + \frac{1}{\alpha})\frac{2(2-\epsilon)}{(1-\epsilon)(1-2\epsilon)}\epsilon$. If the algorithm terminates successfully, we have:

$$
\begin{aligned}
f(O) &\leq f(O \cup A) + f(O \cup B) && (1)\\
&\leq f(O \cup A) + f((O \cup B) \setminus A) + f((O \cup B) \cap A) && (2)\\
&= f(O \cup A) + f((O \setminus A) \cup B) + f(O \cap A) && (3)\\
&\leq a(f(A') + f(B')) + \frac{4}{1-4\epsilon} f(C) && (4)\\
&\leq (2a + \frac{4}{1-4\epsilon}) f(S)
\end{aligned}
$$

where inequality (1) is due to the submodularity of f and the fact that $A \cap B = \emptyset$; the inequality (2) is due to the submodularity of f, and the inequality (4) is due to Theorem 3 and USM1's performance. For the complexities, the algorithm calls USM1 and LinAdapt twice so its adaptive complexity is $2 \cdot O(\frac{\log n}{\epsilon^2}) + 1 = O(\frac{\log n}{\epsilon^2})$ and query complexity is $2 \cdot O(\frac{n}{\epsilon^3}) + O(\frac{1}{\epsilon}\log(\frac{n}{\epsilon})) = O(\frac{n}{\epsilon^3})$. □

Algorithm 2: LinBoundSet$(f, V, k, \alpha, \epsilon, \delta)$

1: **Input:** f, V, k, α, ϵ, δ
2: $e_{max} \leftarrow \max_{e \in V} f(e)$, $S \leftarrow \{e_{max}\}$, $\beta \leftarrow \epsilon \log((1 - e^{\frac{-\epsilon}{2}})/8)/16$, $\ell \leftarrow \lceil (4 + \frac{4}{\beta\epsilon}) \log(\frac{n}{\delta}) \rceil$
3: **for** $j \leftarrow 1$ to ℓ **do**
4: $\quad V \leftarrow \{x \in V : f(x|S) \geq \alpha f(S)/k\}$
5: $\quad$ **If** $V = \emptyset$ **then** break;
6: $\quad V = \{v_1, v_2, \ldots, v_{|V|}\} \leftarrow$ rand-permutation(V)
7: $\quad \Lambda_1 \leftarrow \{\lfloor (1+\epsilon)^l \rfloor : 1 \leq \lfloor (1+\epsilon)^l \rfloor \leq k, l \in \mathbb{N}\}$
8: $\quad \Lambda_2 \leftarrow \{\lfloor k + l\epsilon k \rfloor : \lfloor k + l\epsilon k \rfloor \leq |V|, l \in \mathbb{N}\} \cup \{|V|\}$
9: $\quad \Lambda = \{\lambda_1, \ldots, \lambda_m\} \leftarrow \Lambda_1 \cup \Lambda_2$, $T_i = \{v_1, v_2, \ldots, v_i\}$, $T'_{\lambda_i} \leftarrow T_{\lambda_i} \setminus T_{\lambda_{i-1}}$
10: $\quad$ Calculate $M_{\lambda_i} \leftarrow \max_{\lambda_j \leq \lambda_i} f(S \cup T_{\lambda_j})$ in parallel
11: $\quad b[v_i] \leftarrow$ **none**, $\forall v_i \in V$; $B[\lambda_i] \leftarrow$ **false**, $\forall \lambda_i \in \Lambda$
12: $\quad$ **for** $\lambda_i \in \Lambda$ (in parallel) **do**
13: $\quad\quad$ **for** $v_l \in T'_{\lambda_i}$ (in parallel) **do**
14: $\quad\quad\quad$ **if** $f(v_l|S \cup T_{l-1}) \geq (1-\epsilon)\alpha M_{\lambda_{i-1}}/k$ **then** $b[v_l] \leftarrow$ **true**
15: $\quad\quad\quad$ **else if** $f(v_l|S \cup T_{l-1}) < 0$ **then** $b[v_l] \leftarrow$ **false**
16: $\quad\quad$ **end for**
17: $\quad\quad$ **if** $|\{v \in T'_{\lambda_i} : b[v] =$ **true**$\}| \geq (1-\epsilon)|T'_{\lambda_i}|$ **then** $B[\lambda_i] \leftarrow$ **true**
18: $\quad$ **end for**
19: $\quad \lambda_1^* \leftarrow \max\{\lambda_i \in \Lambda, \lambda_i < k : B[\lambda_i] =$ **false**$, B[\lambda_1] = B[\lambda_2] = \ldots = B[\lambda_{i-1}] =$ **true**$\}$
20: $\quad \lambda_2^* \leftarrow \max\{\lambda_i \in \Lambda, \lambda_i \geq k : \exists u \geq 1\ s.t. |\bigcup_{j=u}^{i-1} T'_{\lambda_j}| \geq k$ and $(B[\lambda_u] = B[\lambda_2] = \ldots = B[\lambda_{i-1}] =$ **true**$\}$
21: $\quad \lambda^* \leftarrow \max\{\lambda_1^*, \lambda_2^*\}$
22: $\quad$ Define new blocks $T''_{\lambda_i} \leftarrow \{v \in T'_{\lambda_i} : b[v] \neq$ **false**$\}$
23: $\quad T^j \leftarrow \bigcup_{\lambda_i \leq \lambda^*} T''_{\lambda_i}$, $S \leftarrow S \cup T^j$, $V \leftarrow V \setminus \bigcup_{\lambda_i \leq \lambda^*} T'_{\lambda_i}$
24: **end for**
25: **If** $|V| > 0$, **then Return** $failure$
26: $S' \leftarrow$ last blocks added into S with the size at most k
27: **Return** S, S'

A key subroutine LinBoundSet **(Algorithm 2).** It receives an instance (f, V, k), accuracy parameters $\epsilon, \delta > 0$ and a positive constant α. It first initiates a set S that contains an element with maximal value e_{max} and then operates in $\ell = O(\log n)$ iterations of the main loop (Lines 3-24). Denote S_j as S after iteration j. At each iteration of main loop, it generates a random permutation of V (Line 6) and divides V into blocks $T'_{\lambda_i}, \forall \lambda_i \in \Lambda$. We define an element $v_t \in T'_{\lambda_i}$ that is **good** if $f(v_t|\{v_1, \ldots, v_{t-1}\}) \geq \alpha M_{\lambda_i}/k$,

where $M_{\lambda_i} = \max_{j=0\dots i} f(S \cup T_{\lambda_j}), i \in [m]$; a block T'_{λ_i} is **good** if it has at least $(1-\epsilon)\alpha|T'_{\lambda_i}|$ good elements.

At a high level, at iteration j, the algorithm selects a segment of elements T_{λ^*} and its subset T^j (Line 23) from set T, a random permutation of V satisfying three following requirements with high probability: (1) $|T^j| \geq (1-\epsilon)|T_{\lambda^*}|$; (2) $f(T^j|S) \geq (1-O(\epsilon))\alpha|T_{\lambda^*}|f(S)/k$; and (3) remove elements in V with the marginal gain is less than $\alpha f(S)/k$, i.e., $f(e|S) < \alpha f(S)/k$ for all $e \in V \setminus S$.

To achieve the requirements, we propose the strategy to select a pair of good subset S and S' in algorithm 2 as follows: It first removes every element whose marginal gain is less than $\alpha f(S)/k$ (Line 4); later, it selects T_{λ^*}, a consecutive sequence of elements T that contains at least $(1-\epsilon)$-fraction number of blocks that are good by finding λ^* in Line 19. Then, it selects new block T''_{λ_i} that only contains elements with non-negative marginal gain from T'_{λ_i} and selects all new blocks from T_{λ^*}. At the end of each iteration, it adds T_j into S and the main loop ends after ℓ iterations or V is empty. Finally, the algorithm returns S' as the union of last T''_{λ_i} blocks added to S so that the size of S' does not exceed k (Line 23).

By combining the our selection strategy for subsets S and S' with the analysis of probability bounds in [Chen *et al.*, 2021; Chen and Kuhnle, 2024], we provide the analysis LinBoundSet's performance in following process. We first focus on the bridge between S and S' in Lemma 1.

Lemma 1. *If* LinBoundSet *ends successfully, then* $f(S) \leq \left(1 + \frac{1+\epsilon}{(1-\epsilon)(1-2\epsilon)\alpha}\right) f(S')$.

We define an *iteration j succeeds* if the algorithm successfully filters out more than $\beta\epsilon$-fraction of V at the next iteration. Otherwise, the iteration j *fails*. The following Lemma provides a bound of probability for the event that iteration j fails.

Lemma 2. $\Pr[\textit{iteration } j \textit{ fails}] \leq 1/2$.

Combine Lemma 1 and Lemma 2, we state the performance of LinBoundSet in Theorem 3.

Theorem 3. *For any $\epsilon, \delta \in (0,1)$, the algorithm* LinBoundSet *runs in $O(\log(n/\delta)/\epsilon^2)$ queries, returns two sets S and S' such that the following holds unconditionally: (1) The algorithm succeeds with a probability of at least $1-\delta/n$; (2) The algorithm takes $O(n/\epsilon^3)$ query complexity in expectation; (3) If the algorithm succeeds, it returns S, S' satisfying: for any subset $X \subseteq V, |X| \leq k$, we have $f(S \cup X) \leq \left(2 + \alpha + \frac{1}{\alpha} + (1+\frac{1}{\alpha})\frac{2(2-\epsilon)}{(1-\epsilon)(1-2\epsilon)}\epsilon\right) f(S')$.*

5 Improved Ratio with Linear Queries

This section introduces two combinatorial algorithms within linear query and near-optimal adaptivity complexities: LinAtg and LinAst. Both share a common framework in which LinAdapt plays a central role and provides $O(1)$ number of guesses of the optimal value. Firstly, they adapt LinAdapt with $\alpha = 1, \delta = 1/3$ to give a candidate solution S_0 with opt $\in [f(S_0), (12+O(\epsilon))f(S_0)]$. It thus provides $O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon}))$ guesses of opt. They then use the Adaptive Simple Threshold (AST) framework [Chen and Kuhnle, 2024] and the Iterated Greedy (IG) algorithm framework [Gupta *et al.*, 2010; Chen and Kuhnle, 2024] to boost the ratio.

LinAst constructs several solutions in the main loop with $O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon}))$ adaptive rounds: each iteration i calls ThreSeq twice sequentially to get candidate solutions A'_i, B'_i. It finally combines the obtained solution and the solution of USM over $\mathcal{A}'_i$ to get the final solution. LinAtg works in a different way. It sequentially constructs two pairs of disjoint sets (A, B) and (A', B') in two main loops which contain at most $\ell = O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon}))$ iterations: each iteration calls ThreSeq to select two batches of elements with high marginal gain (S, S') and adds them into (A, A') (or (B, B')), respectively. It also the solution of USM over $\mathcal{A}'$ and finally returns the best one among the obtained feasible solutions. We provide the performance of LinAtg and LinAst in Theorems 4, 5.

Theorem 4. *For any input (f, V, k, ϵ), where $\epsilon \in (0,1)$,* LinAst *runs in $O(\log(n)/\epsilon^2)$ adaptive complexity and $O(n/\epsilon^3)$ query complexity and returns an approximation ratio of $1/6 - \epsilon$ in expectation with probability of at least $1 - 1/n$.*

Theorem 5. *For any input (f, V, k, ϵ), where $\epsilon \in (0,1)$,* LinAtg *runs in $O(\log(1/\epsilon)\log(n)/\epsilon^2)$ adaptive complexity and $O(n/\epsilon^3)$ query complexity, returns an approximation ratio of $0.193 - \epsilon$ in expectation with probability of at least $1 - 1/n$.*

6 Boost Ratio with Near-Linear Queries

This section introduces our last algorithm, BoostAdapt (Algorithm 3), which further improves the ratio to $1/4 - \epsilon$ in $O(\log(n)\log(k))$ adaptivity and $O(n\log(k))$ query complexity. Different from LinAtg and LinAst, after reusing LinAdapt's solution, BoostAdapt operates in a novel staggered strategy that consists of a main loop with $O(\log(k/\epsilon)/\epsilon)$ iterations. It initiates two pair of disjoint sets: (X, Y) and (X', Y'). At each iteration, it only updates alternatively partial solutions, either X, X' or Y, Y', by calling ThreSeq. The algorithm then carefully selects candidates X'' and Y'' that consist of $\min\{k, |X'|\}$ and $\min\{k, |Y'|\}$ from X' and Y', respectively. Finally, it returns the best among candidates without violating the cardinality constraint.

At a high level, our BoostAdapt algorithm do not adapt USM algorithms (may make the approximation ratio worse) and operates differently from Iterated Greedy [Gupta *et al.*, 2010], which inspired AdaptiveThresholdGreedy of [Kuhnle, 2021; Chen and Kuhnle, 2024]. BoostAdapt is an elegant combination between threshold greedy [Badanidiyuru and Vondrák, 2014] and twin greedy [Han *et al.*, 2020] via using ThreSeq procedures alternatively to update two disjoint solutions X and Y. Interestingly, we found that X and Y can support each other to bound elements in the optimal solution. Besides, by carefully analyzing the role of the set X after the first iteration (i.e., X_1), we can give a better ratio than previous algorithms with the same complexities.

Before analyzing the BoostAdapt's performance, we provide some useful notations as follows: X^i, Y^i are the sets of first i elements added into X and Y, respectively; X_i and Y_i are X and Y after the iteration i of the main loop and $X_0 = Y_0 = \emptyset$; X'_i and Y'_i are defined analogously; A_i^+ (res.

Algorithm 3: BoostAdapt

Input: f, V, k, ϵ.
$S_0 \leftarrow \mathsf{LinAdapt}(f, V, k, 1, \epsilon, 1/3)$,
$a \leftarrow 12 + (\frac{16}{1-4\epsilon} + \frac{8(2-\epsilon)}{(1-\epsilon)(1-2\epsilon)})\epsilon$
$M \leftarrow af(S_0)$, $\Delta \leftarrow \lceil \log_{\frac{1}{1-\epsilon}}(\frac{ak}{4\epsilon}) \rceil + 1$, $\delta \leftarrow 1/(3\Delta)$
$X, X', Y, Y' \leftarrow \emptyset, k' \leftarrow \max\{i \in \mathbb{N} : (1-\epsilon)i \leq k\}$,
$\tau \leftarrow \frac{k'M}{4k}$
for $i \leftarrow 1$ to Δ **do**
  if i is **odd then**
    $\tau_X \leftarrow \tau(1-\epsilon)^i$, $(A_i, A_i') \leftarrow \mathsf{ThreSeq}(f(X \cup \cdot), V \setminus (X \cup Y), k' - |X|, \epsilon, \delta, \tau_X)$,
    $X \leftarrow X \cup A_i$, $X' \leftarrow X' \cup A_i'$
  else
    $\tau_Y \leftarrow \tau(1-\epsilon)^i$,
    $(B_i, B_i') \leftarrow \mathsf{ThreSeq}(f(Y \cup \cdot), V \setminus (X \cup Y), k' - |Y|, \epsilon, \delta, \tau_Y)$,
    $Y \leftarrow Y \cup B_i$, $Y' \leftarrow Y' \cup B_i'$
  end if
end for
Define $X' = \{x_1', x_2', \ldots, x_{|X'|}'\}$,
$Y' = \{y_1', y_2', \ldots, y_{|Y'|}'\}$ contain elements in the order selected.
$X'' \leftarrow$ set of $\min\{k, |X'|\}$ elements $x_i' \in X'$ with largest marginal gain $f(x_i'|X'^{<x_i'})$
$Y'' \leftarrow$ set of $\min\{k, |Y'|\}$ elements $y_i' \in Y'$ with largest marginal gains $f(y_i'|Y'^{<y_i'})$
$S \leftarrow \arg\max_{Z \in \{X', Y', X'', Y'', S_0\}, |Z| \leq k} f(Z)$
Return S

B_i^+) as the set of elements in A_i (res. B_i) having the marginal gain at least τ_X (res. τ_Y) at the iteration i, $X_i^+ = \cup_{j=1}^i A_i^+$, $Y_i^+ = \cup_{j=1}^i B_i^+$ and $X^+ = X_\Delta^+, Y^+ = Y_\Delta^+$.

By the properties of ThreSeq [Chen and Kuhnle, 2024], it is noted that $|A_i^+| \geq (1-\epsilon)|A_i|$, $|B_i^+| \geq (1-\epsilon)|B_i|$. For an element $e \in X \cup Y$, we denote by $X^{<e}, Y^{<e}$ as X, Y right before e is selected into X or Y, respectively.

We now analyze theoretical bounds of BoostAdapt algorithm. Firstly, we explore some basic properties of X, X', X'' and Y, Y', Y'' according to the iteration i of the main loop.

Lemma 3. ***a)*** *For any iteration i of the outer loop (Lines 5-13) of Algorithm 3, we have $f(X_i') \geq f(X_i)$ and $f(Y_i') \geq f(Y_i)$.* ***b)*** *At the end of* BoostAdapt*, we have $f(X'') \geq (1-\epsilon)f(X')$ and $f(Y'') \geq (1-\epsilon)f(Y')$.*

By carefully analyzing the relationship between X and Y through thresholds, we provide the connection between X and Y after each iteration in Lemma 4.

Lemma 4. *If* BoostAdapt *succeeds and $X_1 = \emptyset$, after iteration i the following properties hold:*
(a) *If i is odd and $|X_i| < k'$: for any sets $C \subseteq Y_{i-1}^+$ and $C' \subseteq Y_{i-1} \setminus Y_{i-1}^+$, we have $\sum_{x \in C} f(x|X_i) \leq \sum_{x \in C} \frac{f(e|Y^{<e})}{1-\epsilon}$ and $\sum_{e \in C'} f(e|X_i) = \frac{\epsilon}{1-\epsilon} f(Y_{i-1}^+)$.*
(b) *If i is even and $|Y_i| < k'$, for any set $D \subseteq X_{i-1}^+$ and $D' \subseteq X_{i-1} \setminus X_{i-1}^+$, we have $\sum_{x \in D} f(x|Y_i) \leq \sum_{x \in D} \frac{f(e|X^{<e})}{1-\epsilon}$ and $\sum_{e \in D'} f(e|Y_i) = \frac{\epsilon}{1-\epsilon} f(X_{i-1}^+)$.*

Using Lemma 4 and the fact that if $|T| = k$, then $|T \setminus O| \geq |O \setminus T|$, where $T \in \{X^+, Y^+\}$, we can bound of the optimal solution in some cases related to the size of X and Y in Lemma 5.

Lemma 5. *If* BoostAdapt *succeeds and $X_1 = \emptyset$, at the end* BoostAdapt *we have*
a) *If $|X| < k'$ and $|Y| < k'$, then $f(O) < \frac{4f(S)}{(1-\epsilon)^2(1-2\epsilon)}$.*
b) *If there exists $T \in \{X, Y\}$ such that $|T| = k'$, then $f(S) \geq (1-\epsilon)^4 \frac{\mathsf{opt}}{4}$.*

Finally, putting them together, we give the tighter ratio in Theorem 6.

Theorem 6. *For any $\epsilon \in (0, 1/4)$,* BoostAdapt *algorithm runs in $O(\log(n/\epsilon)\log(k/\epsilon)/\epsilon^2)$ adaptivity, $O(n\log(k/\epsilon)/\epsilon^2)$ query complexity in expectation. The algorithm succeeds with probability at least $1 - \frac{1}{n}$, in which case it achieves a $(\frac{1}{4} - \epsilon)$-approximation ratio.*

Proof. Due to the space limit, we put proof for successful probability and complexities in the appendix. If the algorithm succeeds, we now prove the ratio by considering the following cases: (1) If $X_1 \neq \emptyset$ or $Y_2 \neq \emptyset$, we have $f(S) \geq \max\{f(X_1), f(Y_2)\} \geq \frac{(1-\epsilon)^2 k' M}{4k} \geq (1-\epsilon)^2 \frac{\mathsf{opt}}{4} > (\frac{1}{4} - \epsilon)\mathsf{opt}$. (2) If X_1 and Y_2 are both empty. If $|X| < k'$ and $|Y| < k'$, from Lemma 5, we have $f(S) > \frac{(1-\epsilon)^2(1-2\epsilon)}{4}\mathsf{opt} > (\frac{1}{4} - \epsilon)\mathsf{opt}$. If there exists $T \in \{X, Y\}$ such that $|T| = k$, then $f(S) \geq (1-\epsilon)^4 \frac{\mathsf{opt}}{4} > (\frac{1}{4} - \epsilon)\mathsf{opt}$. □

7 Empirical Study

In this section, we compare our LinAst, LinAtg and BoostAdapt[1] with state-of-the-art for non-monotone SMC, including IteratedGreedy (IG) [Gupta *et al.*, 2010], FastRandomGreedy (FRG) [Buchbinder *et al.*, 2015], PITG[Chen *et al.*, 2025], ParSSP[Cui *et al.*, 2023], FASTLS+GUIDEDRG (FLS) [Chen *et al.*, 2024a], AdaptiveSimpleThreshold (AST) [Chen and Kuhnle, 2024], and AdaptiveThresholdGreedy (ATG) [Chen and Kuhnle, 2024]. The comparison is about **four metrics**: object values, adaptive complexity, number of queries, and running time. We experimented with two applications: Revenue Maximization (RM) and Maximum Cut (MC) [Chen and Kuhnle, 2024; Kuhnle, 2019; Amanatidis *et al.*, 2020]. **Dataset and Setting.** For all algorithms, we've fixed the parameter $\epsilon = 0.1$. For our proposed algorithms. Each algorithm was executed over 20 independent runs, and the reported results are the average over these runs. We used Astro ($n = 18,772, m = 198,110$) for RM and utilized web-Google ($n = 875,713, m = 5,105,039$), GrQc ($n = 5,242, m = 14,496$), and Barabasi-Albert ($n = 968, m = 5,708$) for MC.
Figure 1 displays the results on the Astro and GrQC datasets. More detailed settings and results are presented in the appendix. **Objective value.** Figures 1(a)(e) show the objective values of algorithms. It can be observed that the

[1]The source code of our algorithms is available at https://github.com/tantdhvan/AdaptIJCAI26

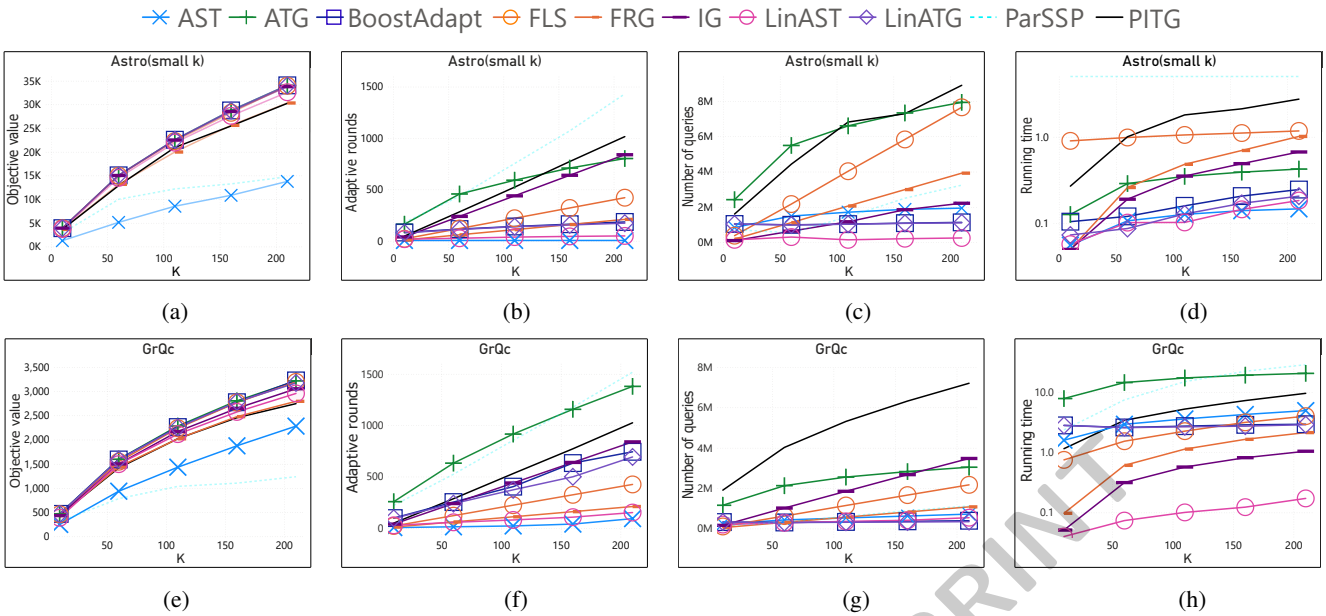

Figure 1: Performance of algorithms on Revenue Maximization (a-d) and Maximum Cut (e-h).

lines of BoostAdapt on the top while they consistently reach the highest points with every k. The top includes FLS, BoostAdapt, LinAtg, IG and AST. Both LinAst, PITG and FRG mark a little lower than BoostAdapt, followed by AST, and ParSSP. However, with small k, the gap between lines on the top seem inconsiderable. Finally, our algorithms significantly contribute to the quality of the solution. **Adaptive rounds.** In the MC application (Fig. 1(f)), the adaptive rounds result three distinct groups. The group with low adaptivity includes AST, LinAst, and FRG while the medium group includes IG, LinAtg, FLS, PITG and BoostAdapt. The high group is the remaining. ParSSP and ATG algorithms waste the highest number of adaptive rounds while AST hits the lowest points. With the RM instance (Fig. 1(b)), ParSSP, ATG, PITG, and IG give high numbers of adaptivities. When k increases, their quantities of the adaptive rounds seem to grow quickly, 4-10 times larger than the others. LinAtg slightly differs from the lowest AST while LinAst and BoostAdapt are higher than AST, but they do not exceed 200. **Number of queries.** In both RM and MC (Figures 1(c) and (g)), our algorithms almost always minimize the number of queries. In RM (Figure 1(c)), LinAst has the lowest number of queries, followed by LinAtg and BoostAdapt algorithms. The number of queries for AST is slightly higher than that of BoostAdapt. The remaining with ATG, FRG, ParSSP, FLS, PITG, and IG belong to the high-query number group. Especially, ATG and PITG mark the highest. For MC (Figure 1(g)), LinAtg reaches the lowest, followed by BoostAdapt and LinAst. The algorithms AST, ParSSP, and FRG again perform at an average level. In contrast, PITG typically requires the highest number of queries-approximately five times more than the lowest, LinAtg. On the whole, all our algorithms save queries more than the others. Moreover, our query numbers remain steady over the range of k. This is consistent with (nearly) linear query complexities of our algorithms. **Time taken.** In MC (Figure 1(h)), LinAst wastes the lowest time, followed by IG and FRG. While BoostAdapt, FLS, and LinAtg have the same running time, which is higher than IG and FRG but lower than AST, PITG, and ParSSP. The gap among them considerably reduces, especially when k increases, except ParSSP. Meanwhile, the time consumption of ParSSP steadily grows up along with k's growth. In RM (Figure 1(d)), LinAst and AST have the lowest running time, closely followed by the group of LinAtg, BoostAdapt, ATG. The rest spend more time than the others, especially ParSSP, which runs the slowest, about 10-20 times higher than the others. **Overall**, these metrics demonstrate that our **algorithms consistently achieve higher-quality solutions while requiring substantially fewer queries and a small number of adaptive rounds.** Compared to PITG, the strongest baseline, BoostAdapt achieves higher objective values with up to 5–7$\times$ fewer iterations and oracle queries (RM, $k = 210$).

8 Conclusions

Motivated by the big data challenge for non-monotone submodular maximization under cardinality constraint, we focus on parallel approximation algorithms based on the concept of adaptive complexity. In particular, we proposed efficient parallel algorithms that significantly improve both approximation ratio and query complexity but keep the near-optimal adaptive complexity of $O(\log n)$. Our algorithm also expresses superior solution quality and computation complexity compared to state-of-the-art algorithms.

Acknowledgments

Tan D. Tran was funded by the Master, PhD Scholarship Programme of Vingroup Innovation Foundation (VINIF), code VINIF.2024.TS.069.

References

[Amanatidis *et al.*, 2020] Georgios Amanatidis, Federico Fusco, Philip Lazos, Stefano Leonardi, and Rebecca Reiffenhäuser. Fast adaptive non-monotone submodular maximization subject to a knapsack constraint. In *Annual Conference on Neural Information Processing Systems*, 2020.

[Amanatidis *et al.*, 2021] Georgios Amanatidis, Federico Fusco, Philip Lazos, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Rebecca Reiffenhäuser. Submodular maximization subject to a knapsack constraint: Combinatorial algorithms with near-optimal adaptive complexity. In *International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 231–242, 2021.

[Badanidiyuru and Vondrák, 2014] Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1497–1514, 2014.

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

[Balkanski *et al.*, 2018] Eric Balkanski, Adam Breuer, and Yaron Singer. Non-monotone submodular maximization in exponentially fewer iterations. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2359–2370, 2018.

[Balkanski *et al.*, 2019] Eric Balkanski, Aviad Rubinstein, and Yaron Singer. An exponential speedup in parallel running time for submodular maximization without loss in approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 283–302. SIAM, 2019.

[Buchbinder and Feldman, 2024] Niv Buchbinder and Moran Feldman. Constrained submodular maximization via new bounds for dr-submodular functions. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 1820–1831. ACM, 2024.

[Buchbinder *et al.*, 2014] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1433–1452. SIAM, 2014.

[Buchbinder *et al.*, 2015] Niv Buchbinder, Moran Feldman, and Roy Schwartz. Comparing apples and oranges: Query tradeoff in submodular maximization. In *Proc. of the 26thl ACM-SIAM SODA 2015*, pages 1149–1168, 2015.

[Chekuri and Quanrud, 2019] Chandra Chekuri and Kent Quanrud. Parallelizing greedy for submodular set function maximization in matroids and beyond. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 78–89. ACM, 2019.

[Chen and Kuhnle, 2023] Yixin Chen and Alan Kuhnle. Approximation algorithms for size-constrained non-monotone submodular maximization in deterministic linear time. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, pages 250–261. ACM, 2023.

[Chen and Kuhnle, 2024] Yixin Chen and Alan Kuhnle. Practical and parallelizable algorithms for non-monotone submodular maximization with size constraint. *Journal of Artificial Intelligence Research*, 79:599–637, 2024.

[Chen *et al.*, 2019] Lin Chen, Moran Feldman, and Amin Karbasi. Unconstrained submodular maximization with constant adaptive complexity. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2019, page 102–113, 2019.

[Chen *et al.*, 2021] Yixin Chen, Tonmoy Dey, and Alan Kuhnle. Best of both worlds: Practical and theoretically optimal submodular maximization in parallel. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 25528–25539, 2021.

[Chen *et al.*, 2024a] Yixin Chen, Ankur Nath, Chunli Peng, and Alan Kuhnle. Discretely beyond 1/e: Guided combinatorial algorithms for submodular maximization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[Chen *et al.*, 2024b] Yixin Chen, Ankur Nath, Chunli Peng, and Alan Kuhnle. Discretely beyond $1/e$: Guided combinatorial algortihms for submodular maximization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[Chen *et al.*, 2025] Yixin Chen, Wenjing Chen, and Alan Kuhnle. Breaking barriers: Combinatorial algorithms for non-monotone submodular maximization with sublinear adaptivity and 1/e approximation. *CoRR*, abs/2502.07062, 2025.

[Cui *et al.*, 2023] Shuang Cui, Kai Han, Jing Tang, He Huang, Xueying Li, and Aakas Zhiyuli. Practical parallel algorithms for submodular maximization subject to a knapsack constraint with nearly optimal adaptivity. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 7261–7269. AAAI Press, 2023.

[Ene and Nguyen, 2020] Alina Ene and Huy L. Nguyen. Parallel algorithm for non-monotone dr-submodular maximization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 2902–2911. PMLR, 2020.

[Ene *et al.*, 2019] Alina Ene, Huy L. Nguyen, and Adrian Vladu. Submodular maximization with matroid and packing constraints in parallel. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 90–101. ACM, 2019.

[Fahrbach *et al.*, 2019a] Matthew Fahrbach, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Non-monotone submodular maximization with nearly optimal adaptivity and query complexity. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1833–1842. PMLR, 2019.

[Fahrbach *et al.*, 2019b] Matthew Fahrbach, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Submodular maximization with nearly optimal approximation, adaptivity and query complexity. In *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 255–273, 2019.

[Fahrbach *et al.*, 2023] Matthew Fahrbach, Vahab Mirrokni, and Morteza Zadimoghaddam. Non-monotone submodular maximization with nearly optimal adaptivity and query complexity. *preprint, arXiv:1808.06932*, 2023.

[Gupta *et al.*, 2010] Anupam Gupta, Aaron Roth, Grant Schoenebeck, and Kunal Talwar. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In *International Workshop on Internet and Network Economics*, 2010.

[Han *et al.*, 2020] Kai Han, Zongmai Cao, Shuang Cui, and Benwei Wu. Deterministic approximation for submodular maximization over a matroid in nearly linear time. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[Han *et al.*, 2021] Kai Han, Shuang Cui, Tianshuai Zhu, Enpei Zhang, Benwei Wu, Zhizhuo Yin, Tong Xu, Shaojie Tang, and He Huang. Approximation algorithms for submodular data summarization with a knapsack constraint. *Proc. ACM Meas. Anal. Comput. Syst.*, 5(1):05:1–05:31, 2021.

[Kazemi *et al.*, 2019] Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3311–3320. PMLR, 2019.

[Kuhnle, 2019] Alan Kuhnle. Interlaced greedy algorithm for maximization of submodular functions in nearly linear time. In *Neural Information Processing Systems*, pages 2371–2381, 2019.

[Kuhnle, 2021] Alan Kuhnle. Nearly linear-time, parallelizable algorithms for non-monotone submodular maximization. In *Proc. of the 30th AAAI 2021*, pages 8200–8208, 2021.

[Lee *et al.*, 2010] Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Maximizing non-monotone submodular functions under matroid or knapsack constraints. *SIAM J. Discret. Math.*, 23(4):2053–2078, 2010.

[Lin and Bilmes, 2011] Hui Lin and Jeff A. Bilmes. A class of submodular functions for document summarization. In *Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 510–520, 2011.

[Mirzasoleiman *et al.*, 2016] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Amin Karbasi. Fast constrained submodular maximization: Personalized data summarization. In *International Conference on Machine Learning*, volume 48 of *JMLR Workshop and Conf. Proc.*, pages 1358–1367, 2016.

[Tukan *et al.*, 2024] Murad Tukan, Loay Mualem, and Moran Feldman. Practical 0.385-approximation for submodular maximization subject to a cardinality constraint. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.