Journal of Artificial Intelligence Research 79 (2024) 599-637 Submitted 10/2022; published 02/2024

Practical and Parallelizable Algorithms for Non-Monotone Submodular Maximization with Size Constraint

Yixin Chen CHEN777@TAMU.EDU
Department of Computer Science & Engineering
Texas A&M University
College Station, TX

Alan Kuhnle KUHNLE@TAMU.EDU
Department of Computer Science & Engineering
Texas A&M University
College Station, TX

Abstract

We present combinatorial and parallelizable algorithms for the maximization of a submodular function, not necessarily monotone, with respect to a size constraint. We improve the best approximation factor achieved by an algorithm that has optimal adaptivity and nearly optimal query complexity to $1/6 - \varepsilon$, and even further to $0.193 - \varepsilon$ by increasing the adaptivity by a factor of $\mathcal{O}(\log(k))$. The conference version of this work mistakenly employed a subroutine that does not work for non-monotone, submodular functions. In this version, we propose a fixed and improved subroutine to add a set with high average marginal gain, ThreshSeq, which returns a solution in $\mathcal{O}(\log(n))$ adaptive rounds with high probability. Moreover, we provide two approximation algorithms. The first has approximation ratio $1/6 - \varepsilon$, adaptivity $\mathcal{O}(\log(n))$, and query complexity $\mathcal{O}(n \log(k))$, while the second has approximation ratio $0.193 - \varepsilon$, adaptivity $\mathcal{O}(\log(n)\log(k))$, and query complexity $\mathcal{O}(n \log(k))$. Our algorithms are empirically validated to use a low number of adaptive rounds and total queries while obtaining solutions with high objective value in comparison with state-of-the-art approximation algorithms, including continuous algorithms that use the multilinear extension.

1. Introduction

A nonnegative set function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, defined on all subsets of a ground set $\mathcal{N}$ of size n, is *submodular* if for all $A, B \subseteq \mathcal{N}$, $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. A set function is monotone if $A \subseteq B$ implies $f(A) \leq f(B)$. Submodular set functions naturally arise in many learning applications, including data summarization (Simon, Snavely, & Seitz, 2007; Sipos, Swaminathan, Shivaswamy, & Joachims, 2012; Tschiatschek, Iyer, Wei, & Bilmes, 2014; Libbrecht, Bilmes, & Noble, 2018), viral marketing (Kempe, Kleinberg, & Tardos, 2003; Hartline, Mirrokni, & Sundararajan, 2008), and recommendation systems (El-Arini & Guestrin, 2011). Some applications yield submodular functions that are not monotone: for example, image summarization with diversity (Mirzasoleiman, Badanidiyuru, & Karbasi, 2016) or revenue maximization on a social network (Hartline et al., 2008). In this work, we study the maximization of a (not necessarily monotone) submodular function subject to a cardinality constraint; that is, given submodular function f and integer k, determine

$\arg\max_{|S|\leq k} f(S)$ (**SMCC**). Access to f is provided through a value query oracle, which when queried with the set S returns the value $f(S)$.

As the amount of data in applications has exhibited exponential growth in recent years (*e.g.* the growth of social networks (Mislove, Koppula, Gummadi, Druschel, & Bhattacharjee, 2008) or genomic data (Libbrecht et al., 2018)), it is necessary to design algorithms for **SMCC** that can scale to these large datasets. One aspect of algorithmic efficiency is the *query complexity*, the total number of queries to the oracle for f. Since evaluation of f is often expensive, the queries to f often dominate the runtime of an algorithm. In addition to low query complexity, it is necessary to design algorithms that parallelize well to take advantage of modern computer architectures. To quantify the degree of parallelizability of an algorithm, the *adaptivity* or *adaptivive complexity* of an algorithm is the minimum number of sequential rounds such that in each round the algorithm makes $\mathcal{O}(\text{poly}(n))$ independent queries to the evaluation oracle. The lower the adaptivive complexity of an algorithm, the more suited the algorithm is to parallelization, as within each adaptive round, the queries to f are independent and may be easily parallelized.

The design of algorithms with nontrivial adaptivity for **SMCC** when f is monotone was initiated by Balkanski and Singer (2018), who also prove a lower bound of $\Omega(\log(n)/\log\log(n))$ adaptive rounds to achieve a constant approximation ratio. Recently, much work has focused on the design of adaptive algorithms for **SMCC** with (not necessarily monotone) submodular functions, as summarized in Table 1. However, although many algorithms with low adaptivity have been proposed, most of these algorithms exhibit at least a quadratic dependence of the query complexity on the size n of the ground set, for $k = \Omega(n)$. For many applications, instances have grown too large for quadratic query complexity to be practical. Therefore, it is necessary to design adaptive algorithms that also have nearly linear query complexity. An algorithm in prior literature that meets this requirement is the algorithm developed by Fahrbach, Mirrokni, and Zadimoghaddam (2019), which has $\mathcal{O}(n\log(k))$ query complexity and $\mathcal{O}(\log(n))$ adaptivity. However, the approximation ratio stated in Fahrbach et al. (2019) for this algorithm does not hold, as discussed in Section 1.1 and Appendix B. During our revision of this paper, Fahrbach, Mirrokni, and Zadimoghaddam (2023) fixed it, ensuring that the approximation ratio holds now.

Contributions. In this work, we propose two fast, combinatorial algorithms for **SMCC**: the $(1/6-\varepsilon)$-approximation algorithm AdaptiveSimpleThreshold (AST) with adaptivity $\mathcal{O}(\log(n))$ and query complexity $\mathcal{O}(n\log(k))$; and the $(0.193-\varepsilon)$-approximation algorithm AdaptiveThresholdGreedy (ATG) with adaptivity $\mathcal{O}(\log(n)\log(k))$ and query complexity $\mathcal{O}(n\log(k))$.

The above algorithms both employ a lowly-adaptive subroutine to add multiple elements that satisfy a given marginal gain, on average. The conference version (Kuhnle, 2021) of this paper used the Threshold-Sampling subroutine of Fahrbach, Mirrokni, and Zadimoghaddam (2019), Fahrbach et al. (2019) for this purpose. However, the theoretical guarantee (Lemma 2.3 of Fahrbach et al. (2019)) for non-monotone functions does not hold due to a bug that has since been fixed in Fahrbach et al. (2023). In Appendix B, we give a counterexample to the performance guarantee of Threshold-Sampling. In this version, we introduce a new threshold subroutine ThreshSeq, which not only fixes the problem that

Table 1: Adaptive algorithms for **SMCC** where objective f is not necessarily monotone. We consider three metrics here. "Approximation Ratio" reflects the accuracy of the algorithm, where a higher value signifies greater accuracy. "Adaptivity" measures the algorithm's parallelizability, with a lower value indicating higher parallelizability. "Queries" represents the total number of queries to the oracle for f, dominating the algorithm's runtime. A lower query count implies faster algorithmic performance. Both the adaptivity and query complexity values presented in this table are asymptotic.

Reference	Approximation Ratio	Adaptivity	Queries
Buchbinder et al. (2015)	$1/e - \varepsilon \approx 0.367 - \varepsilon$	k	n
Balkanski et al. (2018)	$1/(2e) - \varepsilon \approx 0.183 - \varepsilon$	$\log^2(n)$	$OPT^2 n \log^2(n) \log(k)$
Chekuri and Quanrud (2019)	$3 - 2\sqrt{2} - \varepsilon \approx 0.171 - \varepsilon$	$\log^2(n)$	$nk^4 \log^2(n)$
Ene and Nguyen (2020)	$1/e - \varepsilon \approx 0.367 - \varepsilon$	$\log(n)$	$nk^2 \log^2(n)$
Fahrbach et al. (2023)	$0.039 - \varepsilon$	$\log(n)$	$n \log(k)$
Amanatidis et al. (2021)	$0.172 - \varepsilon$	$\log(n)$ $\log(n) \log(k)$	$nk \log(n) \log(k)$ $n \log(n) \log^2(k)$
Theorem 7 (AST) Theorem 8 (ATG)	$1/6 - \varepsilon \approx 0.166 - \varepsilon$ $0.193 - \varepsilon$	$\log(n)$ $\log(n) \log(k)$	$n \log(k)$ $n \log(k)$

Threshold-Sampling faced, but achieves its guarantees with high probability as opposed to in expectation; the high probability guarantees simplify the analysis of our approximation algorithms that rely upon the ThreshSeq subroutine.

Our algorithm AST uses a double-threshold procedure to obtain its ratio of $1/6 - \varepsilon$. Our second algorithm ATG is a low-adaptivity modification of the algorithm of Gupta, Roth, Schoenebeck, and Talwar (2010), for which we improve the ratio from 1/6 to 0.193 through a novel analysis. Both of our algorithms use the low-adaptivity, threshold sampling procedure ThreshSeq and a subroutine for unconstrained maximization of a submodular function (Feige, Mirrokni, & Vondrák, 2011; Chen, Feldman, & Karbasi, 2019) as components. More details are given in the related work discussion below and in Section 4.

The new ThreshSeq does not rely on sampling to achieve concentration bounds, which significantly improves the practical efficiency of our algorithms over the conference version (Kuhnle, 2021). Empirically, we demonstrate that both of our algorithms achieve superior objective value to current state-of-the-art algorithms while using a small number of queries and adaptive rounds on two applications of **SMCC**.

1.1 Related Work

Theshold Procedures. A recurring subproblem of **SMCC** (and other submodular optimization problems) is to add to a candidate solution S those elements x of the ground set $\mathcal{N}$ that give a marginal gain of at least τ, for some constant threshold τ. To solve this subproblem, the algorithm Threshold-Sampling is proposed in Fahrbach et al. (2019)

for monotone submodular functions and applied in Fahrbach et al. (2019) and the conference version of this work (Kuhnle, 2021) as subroutines for non-monotone **SMCC**. However, theoretical guarantee (Lemma 2.3 of Fahrbach et al. (2019)) does not hold when the objective function is non-monotone. Counterexamples and pseudocode for Threshold-Sampling are given in Appendix B. A recent work by Fahrbach et al. (2023) has modified the Threshold-Sampling algorithm and fixed the problem discussed above.

Two alternative solutions to the non-monotone threshold problem were proposed in Amanatidis et al. (2021) for the case of non-monotone, submodular maximization subject to a knapsack constraint. Due to the complexity of the constraints, the thresholding procedures in Amanatidis et al. (2021) have a high time complexity and require $\mathcal{O}\left(n^2\right)$ query calls within one iteration even when restricted to a size constraint. Although a variant with binary search is proposed to get fewer queries, the sequential binary search worsens the adaptivity of the algorithm.

In this work, we propose the ThreshSeq algorithm (Section 2) that fixes the problems of Threshold-Sampling and runs in $\mathcal{O}(n)$ queries and $\mathcal{O}(\log n)$ adaptive rounds. We solve these problems by introducing two sets found by the algorithm: an auxilliary set A separate from the solution set A' found by ThreshSeq that solves Threshold (Def. 2) separately. The algorithm maintains that $A' \subseteq A$, and the larger set is used for filtering from the ground set, while the smaller set maintains desired bounds on the average marginal gain.

Algorithms with Low Adaptivive Complexity. Since the study of parallelizable algorithms for submodular optimization was initiated by Balkanski and Singer (2018), there have been a number of $\mathcal{O}(\log n)$-adaptive algorithms designed for **SMCC**. When f is monotone, adaptive algorithms that obtain the optimal ratio (Nemhauser & Wolsey, 1978) of $1-1/e-\varepsilon$ have been designed by Balkanski, Rubinstein, and Singer (2019a), Fahrbach et al. (2019), Ene and Nguyen (2019), Chen, Dey, and Kuhnle (2021). Of these, the algorithm of Chen et al. (2021) also has the state-of-the-art sublinear adapativity and linear query complexity.

However, when the function f is not monotone, the best approximation ratio with polynomial query complexity for **SMCC** is unknown, but falls within the range $[0.385, 0.491]$ (Buchbinder & Feldman, 2019; Gharan & Vondrák, 2011). For **SMCC**, algorithms with nearly optimal adaptivity have been designed by Balkanski et al. (2018), Chekuri and Quanrud (2019), Ene, Nguyen, and Vladu (2019), Fahrbach et al. (2019), Amanatidis et al. (2021); for the query complexity and approximation factors of these algorithms, see Table 1. Of these, the best approximation ratio of $(1/e-\varepsilon) \approx 0.368$ is obtained by the algorithm of Ene and Nguyen (2020). However, this algorithm requires access to an oracle for the gradient of the continuous extension of a submodular set function, which requires $\Omega(nk^2 \log^2(n))$ queries to sufficiently approximate. The practical performance of the algorithm of Ene and Nguyen (2020) is investigated in our empirical evaluation of Section 5. Other than the algorithms of Fahrbach et al. (2019) and Amanatidis et al. (2021), all parallelizable algorithms exhibit a runtime of at least quadratic dependence on n. In contrast, our algorithms have query complexity of $\mathcal{O}(n \log k)$ and have $\mathcal{O}(\log n)$ or $\mathcal{O}\left(\log^2 n\right)$ adaptivity.

After the conference version (Kuhnle, 2021) of this paper, Amanatidis et al. (2021) proposed a parallelizable algorithm, ParCardinal, for knapsack constraints, which is the first

constant factor approximation with optimal adaptivive complexity. In the paper, ParCardinal is directly applied to cardinality constraints. It achieves a $0.172 - \varepsilon$ ratio with two different variants: one has $\mathcal{O}(\log(n))$ adaptive rounds and $\mathcal{O}(nk\log(n)\log(k))$ queries; another one has $\mathcal{O}(\log(n)\log(k))$ adaptive rounds and $\mathcal{O}\left(n\log(n)\log^2(k)\right)$ queries. Compared to our nearly linear algorithms, the first variant of ParCardinal requires total queries with more than quadratic dependence on n; and the second variant gets a worse approximation ratio and worse number of queries than our algorithm (ATG) with the same adaptivity.

The IteratedGreedy Algorithm. Although the standard greedy algorithm performs arbitrarily badly for **SMCC**, Gupta et al. (2010) showed that multiple repetitions of the greedy algorithm, combined with an approximation for the unconstrained maximization problem, yields an approximation for **SMCC**. Specifically, Gupta et al. (2010) provided the IteratedGreedy algorithm, which achieves an approximation ratio of 1/6 for **SMCC** when the 1/2-approximation of Buchbinder, Feldman, Naor, and Schwartz (2012) is used for the unconstrained maximization subproblems. Our algorithm AdaptiveThreshold-Greedy uses ThreshSeq combined with the descending thresholds technique of Badanidiyuru and Vondrák (2014) to obtain an adaptive version of IteratedGreedy, as described in Section 4. Pseudocode for IteratedGreedy is given in Appendix E, where an improved ratio of ≈0.193 is proven for this algorithm; we also prove the ratio of nearly 0.193 for our adaptive algorithm ATG in Section 4.

1.2 Preliminaries

A submodular set function defined on all subsets of ground set $\mathcal{N}$ is denoted by f. The marginal gain of adding an element x to a set S is denoted by $\Delta(x \mid S) = f(S \cup \{x\}) - f(S)$. Let $\mathrm{OPT} = \max_{|S| \leq k} f(S)$, the optimal value of the SMCC problem for ground set $\mathcal{N}$ and size constraint k. The restriction of f to all subsets of a set $S \subseteq \mathcal{N}$ is denoted by $f\restriction_S$. Next, we describe two subproblems both of our algorithms need to solve: namely, unconstrained maximization subproblems and a threshold sampling subproblem. For both of these subproblems, procedures with low adaptivity are needed.

The Unconstrained Maximization Problem. The first subproblem is unconstrained maximization of a submodular function. When the function f is non-monotone, the problem of maximizing f without any constraints is NP-hard (Feige et al., 2011). Recently, Chen et al. (2019) developed an algorithm that achieves nearly the optimal ratio of 1/2 with constant adaptivity, as summarized in the following theorem.

Theorem 1 (Chen et al. (2019))**.** *For each $\varepsilon > 0$, there is an algorithm that achieves a $(1/2 - \varepsilon)$-approximation for unconstrained submodular maximization using $\mathcal{O}(\log(1/\varepsilon)/\varepsilon)$ adaptive rounds and $\mathcal{O}\left(n\log^3(1/\varepsilon)/\varepsilon^4\right)$ evaluation oracle queries.*

To achieve the approximation factor listed for our algorithms in Table 1, the algorithm of Chen et al. (2019) is employed for unconstrained maximization subproblems.

The Threshold Problem. The second subproblem is the following:

Definition 2 (Threshold)**.** *Given a threshold $\tau \in \mathbb{R}$ and integer k, choose a set S such that 1) $f(S) \geq \tau|S|$; 2) if $|S| < k$, then for any $x \notin S$, $\Delta(x \mid S) < \tau$.*

Algorithms that can use a solution to this subproblem occur frequently, and so multiple algorithms in the literature for this subproblem have been formulated (Fahrbach et al., 2019; Balkanski, Rubinstein, & Singer, 2019b; Kazemi, Mitrovic, Zadimoghaddam, Lattanzi, & Karbasi, 2019; Amanatidis et al., 2021; Chen et al., 2021). We want a procedure that can solve Threshold with the following three properties: 1) using $\mathcal{O}(n)$ queries of the submodular function; 2) in $\mathcal{O}(\log n)$ adaptive rounds; 3) the function f is non-monotone.

None of the prior algorithms satisfy our requirements, since the procedures in Fahrbach et al. (2019), Kazemi et al. (2019), Chen et al. (2021) only work when the submodular function is monotone; and the two procedures in Amanatidis et al. (2021) have either $\mathcal{O}\left(n^2 \log(n)\right)$ queries or $\mathcal{O}\left(\log^2(n)\right)$ adaptivity. Moreover, in both Fahrbach et al. (2019) and Amanatidis et al. (2021), the procedures for Threshold only guarantee $\mathbb{E}\left[f(S)\right] \geq \tau|S|$.

In this paper, we propose ThreshSeq, an algorithm that makes $\mathcal{O}(n)$ query calls and has $\mathcal{O}(\log n)$ adaptive rounds to solve Threshold. However, this algorithm does not exactly solve Threshold. Instead, it returns two sets that solve each of the questions in Threshold, which is enough for our algorithms.

Organization. In Section 2, we introduce our threshold sampling algorithm: ThreshSeq. Then, in Sections 3 and 4, we analyze our algorithms using the ThreshSeq and UnconstrainedMax procedures. Our empirical evaluation is reported in Section 5 with more discussions in Appendix G.1.

2. The ThreshSeq Algorithm

In this section, we introduce the linear and highly parallelizable threshold sampling algorithm ThreshSeq (Alg. 2). ThreshSeq takes as input oracle f, constraint k, error rate ε, threshold τ, and failure probability parameter δ which reflects the success probability. This algorithm has logarithmic adaptive rounds and linear query calls with high probability. Rather than directly solving Threshold (Def. 2) with one solution set, it returns two relevant sets that deal with the two properties separately.

Algorithm 1 A general framework of threshold sampling algorithms

procedure $(f, \mathcal{N}, k)$
  Input: evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, constraint k, error ε, threshold τ
  Initialize $V \leftarrow \mathcal{N}$, $A \leftarrow \emptyset$
  while $|V| > 0$ **do**
    $V \leftarrow \{x \in V : \Delta(x \,|\, A) \geq \tau$ and $A \cup \{x\}$ feasible$\}$
    $T \leftarrow$ a subset of V ▷ make a decision on selecting a good subset from V
    $A \leftarrow A \cup T$
  return A

2.1 Algorithm Overview

The state-of-the-art threshold sampling algorithms, whether for monotone or non-monotone functions, share a common structure (Alg. 1) that works as follows: 1) The algorithm initial-

izes a candidate set V with the whole ground set $\mathcal{N}$ and an empty solution set A (Line 3); 2) During each iteration, it filters out elements in the candidate set V that either make negligible contributions to A or violate the given constraint, and then selects a prefix of V to add to A (Line 5-7); 3) Then, the algorithm repeats the last step until the candidate set V is empty. The difference between those algorithms lies in how they select the prefix in Step (2) on Line 6. Threshold-Sampling in Fahrbach et al. (2019) applies a random sampling procedure for each prefix considered at that iteration. Threshold sampling algorithms in Balkanski et al. (2019b) and Amanatidis et al. (2021) explicitly check all the candidate elements for a given prefix. Later, Kazemi et al. (2019) and Chen et al. (2021) proposed threshold sampling algorithms that work by performing a uniformly random permutation of elements and making the decision after querying once for each prefix. This makes them comparably much more practical in performance and demonstrates that multiple query calls of a given prefix are redundant. Subsequently, we are able to keep a solution with the same threshold and fewer query calls.

To efficiently obtain large sequences of elements with gains above τ, an approach inspired by monotone threshold sampling algorithms in Kazemi et al. (2019) and Chen et al. (2021) is proposed. As discussed above, these algorithms work by adaptively adding sequences of elements to a set A, where the sequence has been checked in parallel to have at most an ε fraction of the sequence failing the marginal gain condition. A uniformly random permutation of elements is considered, where the average marginal gain being below τ is detected by a high proportion of failures in the sequence. This step leads to a constant fraction of elements being filtered out at the next iteration with high probability. When combined with an exponentially decreasing candidate set and a constant number of adaptive rounds for each iteration, these algorithms achieve logarithmic adaptivity and linear query complexity.

The intuitive reason why this does not directly work for non-monotone functions (*i.e.* A is not a solution to Threshold (Def. 2)) is: if one of the elements added fails the marginal gain condition, it may do so arbitrarily badly and have a large negative marginal gain. Moreover, one cannot simply exclude such elements from consideration, because they are needed to ensure that the filtering step at the next iteration will discard a large enough fraction of elements. Deleting such elements requires recalculating the marginal gains with respect to the updated sets, which increases the number of adaptive rounds required in each iteration by a factor of $\mathcal{O}(k)$. Our solution is to keep these elements in the set A which is used for filtering and responsible for Property (2) of Threshold (Def. 2), but only include those elements with a nonnegative marginal gain in the candidate solution set A', which is responsible for Property (1) of Threshold (Def. 2). The membership of A' is known since the gain of every element was computed in parallel. Moreover, $|A'| \geq (1-\varepsilon)|A|$ gives the needed relationship on the average marginal gain of each element of A'. Due to submodularity, the objective value does not decrease when we exclude elements with negative marginal gains.

Discussion of δ. Different from other threshold sampling algorithms, ThreshSeq incorporates an additional input parameter, δ. This parameter reflects the number of iterations in the outer for loop, or specifically the adaptive rounds achieved by the algorithm. As

Algorithm 2 A parallelizable threshold algorithm for threshold τ

procedure ThreshSeq($f, \mathcal{N}, k, \delta, \varepsilon, \tau$)
  Input: evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, constraint k, failure probability parameter δ, error ε, threshold τ
  Initialize $A \leftarrow \emptyset$, $A' \leftarrow \emptyset$, $V \leftarrow \mathcal{N}$, $\ell = \lceil 4\left(\frac{2}{\varepsilon}\log(n) + \log\left(\frac{n}{\delta}\right)\right)\rceil$
  for $j \leftarrow 1$ to ℓ **do** ▷ Sequential **for** loop
    Update $V \leftarrow \{x \in V : \Delta(x \mid A) \geq \tau\}$ ▷ Filtering step w.r.t. A
    if $|V| = 0$ **then**
      return A, A'
    $V \leftarrow$ **random-permutation**(V)
    $s \leftarrow \min\{k - |A|, |V|\}$
    $B[1:s] \leftarrow [\textbf{none}, \cdots, \textbf{none}]$
    for $i \leftarrow 1$ to s in parallel **do** ▷ Parallel gain computation
      $T_{i-1} \leftarrow \{v_1, v_2, \ldots, v_{i-1}\}$
      if $\Delta(v_i \mid A \cup T_{i-1}) \geq \tau$ **then** $B[i] \leftarrow$ **true**
      elif $\Delta(v_i \mid A \cup T_{i-1}) < 0$ **then** $B[i] \leftarrow$ **false**
    $i^* \leftarrow \max\{i : \#\textbf{true}\text{s in } B[1:i] \geq (1-\varepsilon)i\}$ ▷ Detection of good filtering next iteration
    $A \leftarrow A \cup T_{i^*}$ ▷ A gets all elements
    $A' \leftarrow A' \cup \{V[i] : 1 \leq i \leq i^*, B[i] \neq \textbf{false}\}$ ▷ A' only gets nonnegative-gain elements
    if $|A| = k$ **then**
      return A, A'
  return *failure*

the algorithm progresses and more elements are added to the solution set, the size of A increases while the size of V decreases. Then, the algorithm stops successfully once $|A| = k$ or $|V| = 0$. The more iterations, the more likely it is to succeed. Intuitively, the higher δ is, the lower is the probability of ThreshSeq choosing a subset that improves on costs and satisfies the constraint. As stated in Theorem 3 Property (1), ThreshSeq succeeds with a probability greater than $1 - \delta/n$. For downstream approximation algorithms that use ThreshSeq as a subroutine with a specific δ value, the more calls made to ThreshSeq, the lower success probability it achieves. The adoption of δ makes such probability manageable.

2.2 Theoretical Guarantees

Theorem 3. *Let (f, k) be an instance of **SMCC**. For any constant ε, the algorithm* ThreshSeq *outputs $A' \subseteq A \subseteq \mathcal{N}$ such that the following properties hold:*

1) The algorithm succeeds with probability at least $1 - \delta/n$.

2) There are $\mathcal{O}(n/\varepsilon)$ oracle queries in expectation and $\mathcal{O}(\log(n/\delta)/\varepsilon)$ adaptive rounds.

3) It holds that $f(A') \geq (1-\varepsilon)\tau|A|$. If $|A| < k$, then $\Delta(x \mid A) < \tau$ for all $x \in \mathcal{N}$.

4) It also holds that $f(A') \geq f(A)$ *and* $|A'| \geq (1-\varepsilon)|A|$

The performance of ThreshSeq is derived mainly by answering two questions: 1) if a constant fraction of elements can be filtered out at any iteration with a high probability; 2) if the two sets returned solve Threshold (Def. 2) indirectly. In Lemma 4 below, it is certified that the number of elements being deleted in the next iteration monotonically increases from 0 to $|V|$ as the size of the selected set increases. Then, by probability lemma and concentration bounds (in Appendix A), Lemma 5 answers the first question.

Lemma 4. *Given* V *after* ***random-permutation*** *on Line 8, let* $S_i = \{x \in V : \Delta(x \mid A \cup T_i) < \tau\}$. *It holds that* $|S_0| = 0$, $|S_{|V|}| = |V|$, *and* $|S_{i-1}| \leq |S_i|$.

Lemma 5. *It holds that* $Pr(i^* < \min\{s,t\}) \leq 1/2$.

Furthermore, with enough iterations, the candidate set V becomes empty at some point with a high probability. Also, since the size of the candidate set $|V|$ exponentially decreases, intuitively, the total queries is linear in expectation.

A downside of this bifurcated approach is that a downstream algorithm receives two sets A, A' instead of one from ThreshSeq. It is obvious that the second property of Threshold (Def. 2) holds naturally with set A. Lemma 6 below shows how we can relate set A' with set A. Therefore, by discarding the elements with negative gains in A, the gains of the rest elements, denoted by A', increase and follow the first property of Threshold (Def. 2).

Lemma 6. *Say an element added to the solution set is good if its gain is greater than* τ. *Suppose that Algorithm 2 terminates successfully.* A *and* A' *returned by Algorithm 2 hold the following properties:*

1) There are at least $(1-\varepsilon)$*-fraction of* A *that is good.*

2) A good element in A *is always a good element in* A'.

3) And, any element in A' *has non-negative marginal gain when added.*

The proofs of the lemmas above can be found in Appendix C. Now, we provide the proof concerning the performance of ThreshSeq.

Proof of Success Probability (Property 1). The algorithm succeeds if $|V| = 0$ or $|A| = k$ at termination. If we can filter out a constant fraction of V or select a subset with $k - |A|$ elements at any iteration with a constant probability, then, with enough iterations, the algorithm successfully terminates with a high probability.

From Lemma 4, there exists a point t such that $t = \min\{i : |S_i| \geq \varepsilon|V|/2\}$, where the next iteration filters out more than $\varepsilon/2$-fraction of elements if $i^* \geq t$. Intuitively, when $i \leq t$, there is a constant probability that the fraction of **true**s in $B[1:i]$ exceeds $1-\varepsilon$. According to Lemma 4, Lemma 5 is provided to give the probability that whether $|A| = k$ or $\varepsilon/2-$fraction of V are filtered out at the next iteration.

For the purposes of the analysis, consider a version of the algorithm that does not break on Line 7 when $|V| = 0$. If so, in subsequent iterations following $|V| = 0$, it is always the case

that $s = 0$ and $T_{i^*} = \emptyset$. Lemma 5 still holds in this case. As a result, the algorithm returns the same solution set as the original one.

When the algorithm fails to terminate, at each iteration, it always holds that $i^* < s$; and there are no more than $m = \lceil \log_{1-\varepsilon/2}(1/n) \rceil$ iterations that $i^* \geq t$. Therefore, there are no more than m iterations that $i^* \geq \min\{s, t\}$. Otherwise, with more than m iterations that $i^* \geq \min\{s, t\}$, if there is an iteration that $s \leq t$, the algorithm terminates with $|A| = k$. Otherwise, with more than m iterations that $i^* \geq t$, the algorithm terminates with $|V| = 0$. Define a *successful iteration* as an iteration that $i^* \geq \min\{s, t\}$, which means it successfully filters out $\varepsilon/2$-fraction of V or the algorithm stops here. Let X be the number of successes in the ℓ iterations. Then, X can be regarded as a sum of dependent Bernoulli trails, where the success probability is larger than $1/2$ from Lemma 5. Let Y be a sum of independent Bernoulli trials, where the success probability is equal to $1/2$. Then, the probability of failure can be bounded as follows,

$$\begin{aligned} Pr\left(\text{failure}\right) \leq Pr\left(X \leq m\right) &\overset{(a)}{\leq} Pr\left(Y \leq m\right) \leq Pr\left(Y \leq 2\log(n)/\varepsilon\right) \\ &\overset{(b)}{\leq} e^{-\left(\frac{\log(n)}{2\log(n)+\varepsilon\log\left(\frac{n}{\delta}\right)}-1\right)^2 \cdot \left(\frac{2}{\varepsilon}\log(n)+\log\left(\frac{n}{\delta}\right)\right)} \leq \frac{\delta}{n}, \end{aligned}$$

where Inequality (a) follows from Lemma 13, and Inequality (b) follows from Lemma 12. □

Proof of Adaptivity and Query Complexity (Property 2). In Alg. 2, the oracle queries occur on Line 5 and 13. Since filtering and inner **for** loop can be done in parallel, there are constant adaptive rounds in an iteration. Therefore, the adaptivity is $\mathcal{O}\left(\ell\right) = \mathcal{O}\left(\log(n/\delta)/\varepsilon\right)$.

As for the query complexity, let V_j be the set V after filtering on Line 5 in iteration j. Let j_i be the i-th successful iterations, $Y_i = j_i - j_{i-1}$. By Lemma 13 in Appendix A, it holds that $\mathbb{E}\left[Y_i\right] \leq 2$. For any iteration j that $j_{i-1} + 1 \leq j \leq j_i$, there are $i - 1$ successes before it. Thus, it holds that $|V_j| \leq n(1 - \varepsilon/2)^{i-1}$.

At any iteration j, there are $|V_{j-1}| + 1$ oracle queries on Line 5. As for the inner **for** loop, there are no more than $|V_j| + 1$ oracle queries. The expected number of total queries can be bounded as follows:

$$\begin{aligned} \mathbb{E}\left[\text{Queries}\right] &\leq \sum_{j=1}^{\ell} \mathbb{E}\left[|V_{j-1}| + |V_j| + 2\right] \leq n + 2\ell + \sum_{j=1}^{\ell} 2\mathbb{E}\left[|V_j|\right] \\ &\leq n + 2\ell + \sum_{i \geq 1} 2\mathbb{E}\left[Y_i \cdot n(1 - \varepsilon/2)^{i-1}\right] \leq n + 2\ell + 4n/\varepsilon. \end{aligned}$$

Therefore, the total queries are $\mathcal{O}\left(n/\varepsilon\right)$, where $\varepsilon \in (0, 1)$. □

Proof of Marginal Gains (Property 3 and 4). The algorithm terminates successfully if either $|V| = 0$ or $|A| = k$ during its execution. As proved above, this happens with a probability of as least $1 - \delta/n$. In the proof below, we condition on the event that the algorithm terminates successfully and returns A, A'.

If the algorithm returns A such that $|A| < k$, then it must be the case that the algorithm terminates with $|V| = 0$. So, for any $x \in \mathcal{N}$, there exists an iteration $j_{(x)} + 1$ such that x is filtered out at iteration $j_{(x)} + 1$. Let $A_{j_{(x)}}$ be A after iteration $j_{(x)}$. Then, due to submodularity and $A_{j_{(x)}} \subseteq A$, it holds that

$$\Delta\left(x \,|\, A\right) \leq \Delta\left(x \,|\, A_{j_{(x)}}\right) < \tau.$$

Lemma 6 applies to any case in which the algorithm terminates successfully. As a reminder, an added element is considered good if its gain is greater than τ with respect to the solution prior to its inclusion. As per Line 17, A' includes all such good elements that are in A. Based on Property 1 of Lemma 6, it is guaranteed that the number of good elements in A' is more than $(1-\varepsilon)|A|$. Hence, we have $|A'| \geq (1-\varepsilon)|A|$.

Furthermore, due to the diminishing returns property of submodular functions, removing an element from a set in a sequence will result in non-increasing marginal gains for the remaining elements. For any $x \in A$, let $A_{(x)}$ be a subsequence of A before x is added into A. Define $A'_{(x)}$ analogously. Then, consider any $x \in A$, if $x \notin A'$, it implies that $\Delta\left(x \,|\, A_{(x)}\right) < 0$; if $x \in A'$, it holds that $\Delta\left(x \,|\, A'_{(x)}\right) \geq \Delta\left(x \,|\, A_{(x)}\right)$. Therefore,

$$f(A') = \sum_{x \in A'} \Delta\left(x \,|\, A'_{(x)}\right) \geq \sum_{x \in A'} \Delta\left(x \,|\, A_{(x)}\right) + \sum_{x \in A \setminus A'} \Delta\left(x \,|\, A_{(x)}\right) \geq f(A).$$

By Property 2 and 3 of Lemma 6, if an element x in A is good, it holds that $\Delta\left(x \,|\, A'_{(x)}\right) \geq \tau$; if not, it holds that $\Delta\left(x \,|\, A'_{(x)}\right) \geq 0$. Then,

$$f(A') = \sum_{x \in A'} \Delta\left(x \,|\, A'_{(x)}\right) \geq \sum_{x \in A', x \text{ is good}} \Delta\left(x \,|\, A'_{(x)}\right) \geq (1-\varepsilon)\tau|A|.$$

□

3. The AdaptiveSimpleThreshold Algorithm

In this section, we present the simple algorithm AdaptiveSimpleThreshold (AST, Alg. 3) and show it obtains an approximation ratio of $1/6-\varepsilon$ with nearly optimal query and adaptivive complexity. This algorithm relies on running ThreshSeq for a suitably chosen threshold value. A procedure for unconstrained maximization is also required.

Overview of Algorithm. Algorithm AST works as follows. First, the **for** loop guesses a value of τ close to $\frac{\mathrm{OPT}}{(4+\alpha)k}$, where $1/\alpha$ is the ratio of the algorithm used for the unconstrained maximization subproblem. Next, ThreshSeq is called with parameter τ to yield set A and A'; followed by a second call to ThreshSeq with f restricted to $\mathcal{N} \setminus A$ to yield set B and B'. Next, an unconstrained maximization is performed with f restricted to A to yield set A''. Finally, the best of the three candidate sets A', B', A'' is returned.

We prove the following theorem concerning the performance of AST.

Algorithm 3 The AdaptiveSimpleThreshold Algorithm

procedure AST($f, \mathcal{N}, k, \varepsilon$)
  Input: evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, constraint k, accuracy parameter $\varepsilon > 0$
  Initialize $M \leftarrow \max_{x \in \mathcal{N}} f(x)$; $c \leftarrow 4 + \alpha$, where α^{-1} is ratio of UnconstrainedMax; $\ell \leftarrow \lceil \log_{1-\varepsilon}(1/(ck)) \rceil$
  for $i \leftarrow 0$ to ℓ in parallel **do**
    $\tau_i \leftarrow M(1-\varepsilon)^i$
    $A_i, A_i' \leftarrow \text{ThreshSeq}(f, k, \tau_i, \varepsilon, 1/2)$
    $B_i, B_i' \leftarrow \text{ThreshSeq}\left(f\restriction_{\mathcal{N} \setminus A_i}, k, \tau_i, \varepsilon, 1/2\right)$
    $A_i'' \leftarrow \text{UnconstrainedMax}(A_i)$
    $C_i \leftarrow \arg\max\{f(A_i'), f(B_i'), f(A_i'')\}$
  return $C \leftarrow \arg\max_i\{f(C_i)\}$

Theorem 7. *Suppose there exists an $(1/\alpha)$-approximation for* UnconstrainedMax *with adaptivity Φ and query complexity Ξ, and let $\varepsilon > 0$. Then there exists an algorithm for* ***SMCC*** *with expected approximation ratio $\frac{1}{4+\alpha} - \varepsilon$ with probability at least $1 - 1/n$, expected query complexity $\mathcal{O}\left(\log_{1-\varepsilon}(1/k) \cdot (n/\varepsilon + \Xi)\right)$, and adaptivity $\mathcal{O}(\log(n)/\varepsilon + \Phi)$.*

If the algorithm of Chen et al. (2019) is used for UnconstrainedMax, AST achieves ratio $1/6 - \varepsilon$ with adaptivive complexity $\mathcal{O}(\log(n)/\varepsilon + \log(1/\varepsilon)/\varepsilon)$ and query complexity $\mathcal{O}\left(\log_{1-\varepsilon}(1/k) \cdot \left(n/\varepsilon + n\log^3(1/\varepsilon)/\varepsilon^4\right)\right)$. In the experiment, UnconstrainedMax is implemented to use a random subset which gives an expected $(1/4)-$approximation ratio (Feige et al., 2011). In this case, AST achieves ratio $1/8-\varepsilon$ with adaptivity $\mathcal{O}(\log(n)/\varepsilon)$ and query complexity $\mathcal{O}\left(\log_{1-\varepsilon}(1/k)n/\varepsilon\right)$.

Overview of Proof. The proof uses the following strategy: either ThreshSeq finds a set A' or B' with value approximately τk, which is sufficient to achieve the ratio, or we have two disjoint sets A, B of size less than k, such that for any $x \notin A \cup B$, $\Delta(x \mid A) < \tau$ and $\Delta(x \mid B) < \tau$. In this case, for any set O, we have by submodularity and nonnegativity, $f(O) \leq f(O \cap A) + f(O \setminus A)$. The first term is bounded by the unconstrained maximization, and the second term is bounded by an application of submodularity and the fact that the maximum marginal gain of adding an element into A or B is below τ. The choice of constant c balances the trade-off between the two cases of the proof.

Proof of Theorem 7. Let (f, k) be an instance of **SMCC**, and let $\varepsilon > 0$. Suppose algorithm AST uses a procedure for UnconstrainedMax with expected ratio $1/\alpha$. We will show that, with some events that happen with probability of at least $(1 - 1/n)$, the set C returned by algorithm AST(f, k, ε) satisfies $\mathbb{E}[f(C)] \geq \left(c^{-1} - \varepsilon\right)\text{OPT}$, where OPT is the optimal solution value on the instance (f, k).

Observe that $\tau_0 = M = \max_{x \in \mathcal{N}} f(x) \geq \text{OPT}/k$ by submodularity of f; $\tau_\ell = M(1-\varepsilon)^\ell \leq \text{OPT}/(ck)$ since $M \leq \text{OPT}$. To better explain it, we attach Fig. 1 above. Because τ_i decreases by a factor of $1 - \varepsilon$, there exists i_0 such that $\frac{(1-\varepsilon)\text{OPT}}{ck} \leq \tau_{i_0} \leq \frac{\text{OPT}}{ck}$. Let A, A', B, B', A'' denote $A_{i_0}, A_{i_0}', B_{i_0}, B_{i_0}', A_{i_0}''$, respectively. For the rest of the proof, we

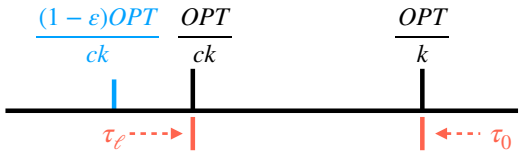


Figure 1: Value of τ_0 and τ_ℓ. It is satisfied that $\tau_0 \geq \mathrm{OPT}/k$ and $\tau_\ell \leq \mathrm{OPT}/(ck)$.

assume that the properties of Theorem 3 hold for the calls to ThreshSeq with threshold τ_{i_0}, which happens with at least probability $1 - 1/n$ by the union bound.

Case $|A| = k$ or $|B| = k$. We suppose that $|A| = k$, the proof for the case $|B| = k$ is directly analogous. By Theorem 3 and the value of τ_{i_0}, it holds that,

$$f(A') \geq (1-\varepsilon)\tau_{i_0}|A| \geq \frac{(1-\varepsilon)^2\mathrm{OPT}}{c} \geq (1/c - \varepsilon)\mathrm{OPT}.$$

Then $f(C) \geq f(A') \geq (1/c - \varepsilon)\mathrm{OPT}$.

Case $|A| < k$ and $|B| < k$. Let O be a set such that $f(O) = \mathrm{OPT}$ and $|O| \leq k$. Since $|A| < k$, by Theorem 3, it holds that for any $x \in \mathcal{N}$, $\Delta(x \mid A) < \tau_{i_0}$. Similarly, for any $x \in \mathcal{N} \setminus A$, $\Delta(x \mid B) < \tau_{i_0}$. Hence, by submodularity,

$$f(O \cup A) - f(A) \leq \sum_{o \in O} \Delta(o \mid A) < k\tau_{i_0} \leq \mathrm{OPT}/c, \tag{1}$$

$$f((O \setminus A) \cup B) - f(B) \leq \sum_{o \in O \setminus A} \Delta(o \mid B) < k\tau_{i_0} \leq \mathrm{OPT}/c. \tag{2}$$

Next, from (1), (2), submodularity, nonnegativity, Theorem 3, and the fact that $A \cap B = \emptyset$, it holds that,

$$\begin{aligned} f(A') + f(B') &\geq f(A) + f(B) \\ &\geq f(O \cup A) + f((O \setminus A) \cup B) - 2\mathrm{OPT}/c \\ &\geq f(O \setminus A) + f(O \cup A \cup B) - 2\mathrm{OPT}/c \\ &\geq f(O \setminus A) - 2\mathrm{OPT}/c. \end{aligned} \tag{3}$$

Since UnconstrainedMax is an α-approximation, we have

$$\alpha\mathbb{E}\left[f(A'')\right] \geq f(O \cap A). \tag{4}$$

From Inequalities (3), (4), and submodularity, we have

$$\begin{aligned} \mathrm{OPT} = f(O) &\leq f(O \cap A) + f(O \setminus A) \\ &\leq \alpha\mathbb{E}\left[f(C)\right] + 2f(C) + 2\mathrm{OPT}/c, \end{aligned}$$

from which it follows that $\mathbb{E}\left[f(C)\right] \geq \mathrm{OPT}/c$.

Algorithm 4 The AdaptiveThresholdGreedy Algorithm

procedure ATG($f, \mathcal{N}, k, \varepsilon$)
Input: evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, constraint k, accuracy parameter $\varepsilon > 0$, failure probability $\delta > 0$
  Initialize $c \leftarrow 8/\varepsilon$, $\varepsilon' \leftarrow (1 - 1/e)\varepsilon/8$, $\ell = \lceil \log_{1-\varepsilon'}(1/(ck)) \rceil + 1$, $\delta \leftarrow 1/(2\ell)$, $M \leftarrow \max_{x \in \mathcal{N}} f(x)$, $A \leftarrow \emptyset$, $A' \leftarrow \emptyset$, $B \leftarrow \emptyset$, $B' \leftarrow \emptyset$
  for $i \leftarrow 1$ to ℓ **do**
    $\tau \leftarrow M\,(1 - \varepsilon')^{i-1}$
    $S, S' \leftarrow$ ThreshSeq$(f_A, k - |A|, \tau, \varepsilon', \delta)$
    $A \leftarrow A \cup S$
    $A' \leftarrow A' \cup S'$
    if $|A| = k$ **then** break
  for $i \leftarrow 1$ to ℓ **do**
    $\tau \leftarrow M\,(1 - \varepsilon')^{i-1}$
    $S, S' \leftarrow$ ThreshSeq$(f_B\restriction_{\mathcal{N} \setminus A}, k - |B|, \tau, \varepsilon', \delta)$
    $B \leftarrow B \cup S$
    $B' \leftarrow B' \cup S'$
    if $|B| = k$ **then** break
  $A'' \leftarrow$ UnconstrainedMax(A, ε')
  $C \leftarrow \arg\max\{f(A'), f(B'), f(A'')\}$
  return C

Adaptivity and Query Complexities. The adaptivity of AST is twice the adaptivity of ThreshSeq plus the adaptivity of UnconstrainedMax plus a constant. Further, the total query complexity is $\log_{1-\varepsilon}(1/(ck))$ times the sum of twice the query complexity of ThreshSeq and the query complexity of UnconstrainedMax. □

4. The AdaptiveThresholdGreedy Algorithm

In this section, we present the algorithm AdaptiveThresholdGreedy (ATG, Alg. 4), which achieves ratio $\approx 0.193 - \varepsilon$ in nearly optimal query and adaptivive complexity. The price of improving the ratio of the preceding section is an extra $\log(k)$ factor in the adaptivity.

Overview of Algorithm. Our algorithm (pseudocode in Alg. 4) works as follows. Each **for** loop corresponds to a low-adaptivity greedy procedure using ThreshSeq with descending thresholds. Thus, the algorithm is structured as two iterated calls to a greedy algorithm, where the second greedy call is restricted to select elements outside the auxiliary set A returned by the first. Finally, an unconstrained maximization procedure is used within the first greedily-selected auxiliary set A. Then, the best of three candidate sets is returned. In the pseudocode for ATG, Alg. 4, ThreshSeq is called with functions of the form f_S, which is defined to be the submodular function $f_S(\cdot) = f(S \cup \cdot)$.

At a high level, our approach is the following: the IteratedGreedy framework of Gupta et al. (2010) runs two standard greedy algorithms followed by an unconstrained maximization, which yields an algorithm with $\mathcal{O}(nk)$ query complexity and $\mathcal{O}(k)$ adaptivity. We adopt this framework but replace the standard greedy algorithm with a novel greedy approach with low adaptivity and query complexity. To design this novel greedy approach, we modify the descending thresholds algorithm of Badanidiyuru and Vondrák (2014), which has query complexity $\mathcal{O}(n \log k)$ but very high adaptivity of $\Omega(n \log k)$. We use ThreshSeq to lower the adaptivity of the descending thresholds greedy algorithm (see Appendix D for pseudocode and a detailed discussion).

For the resulting algorithm ATG, we prove a ratio of $0.193 - \varepsilon$ (Theorem 8), which improves the 1/6 ratio for IteratedGreedy proven in Gupta et al. (2010). Also, by adopting ThreshSeq proposed in this paper, the analysis of approximation ratio is simplified. Thanks to the fact that the contribution of each element added to the solution set A' is determined, at least $(1-\varepsilon)|A|$ elements in the solution set A' have marginal gains which exactly exceed the threshold τ, while the rest of it have non-negative marginal gains. Therefore, it is not needed to analyze the marginal gain in expectation anymore. An exact lower bound is given by the analysis of the two greedy procedures.

A simpler form of our arguments shows that the improved ratio also holds for the original IteratedGreedy of Gupta et al. (2010); this analysis is given in Appendix E. We prove the following theorem concerning the performance of ATG.

Theorem 8. *Suppose there exists an $(1/\alpha)$-approximation for* UnconstrainedMax *with adaptivity Φ and query complexity Ξ, and let $\varepsilon > 0$. Then the algorithm* AdaptiveThresholdGreedy *for* ***SMCC*** *has expected approximation ratio $\frac{e-1}{e(2+\alpha)-\alpha} - \varepsilon$ with probability at least $(1 - 1/n)$, adaptivive complexity of $\mathcal{O}\left(\log_{1-\varepsilon}(1/k)\log(n)/\varepsilon + \Phi\right)$ and expected query complexity of $\mathcal{O}\left(\log_{1-\varepsilon}(1/k) \cdot (n/\varepsilon) + \Xi\right)$.*

If the algorithm of Chen et al. (2019) is used for UnconstrainedMax, ATG achieves approximation ratio $\approx 0.193 - \varepsilon$ with adaptivive complexity $\mathcal{O}(\log(n)\log(k))$ and query complexity $\mathcal{O}(n \log(k))$, wherein the ε dependence has been suppressed. Same as the AST algorithm, UnconstrainedMax is implemented to use a random subset, and ATG achieves approximation ratio $\approx 0.139 - \varepsilon$ with adaptivive complexity $\mathcal{O}(\log(n)\log(k))$ and query complexity $\mathcal{O}(n \log(k))$.

Proof of Theorem 8. In this proof, we assume that the guarantees of Theorem 3 hold for each call to ThreshSeq made by ATG; this occurs with probability at least $(1 - 1/n)$ by the union bound and the choice of δ.

Overview of Proof. For the proof, a substantial amount of machinery is necessary to lower bound the marginal gain. The necessary notations are made first; then, in Lemmas 9 – 10, we formulate the necessary lower bounds on the marginal gains for the first and second greedy procedures. For each respective greedy procedure, this is accomplished by considering the good elements in the selected set returned by ThreshSeq, or the dummy element if the size of selected set is limited. This allows us to formulate a recurrence on the sum of the marginal gains (Lemma 11). Finally, the recurrence allows us to proceed similarly

to our proof in Appendix E after a careful analysis of the error introduced (Lemma 18 in Appendix F).

Notations. Followed by the notations in the pseudocode of Alg. 4, A and A' are returned by the first greedy procedure, while B and B' are returned by the second one. Let A_i be the set A after iteration i, a'_j be the j-th element in A', and $i(j)$ be the iteration that returns a'_j. If $j > |A'|$, let a'_j be a dummy element, and $i(j) = \ell + 1$. Furthermore, define $\mathcal{A}'_j = \{a'_1, \ldots, a'_j\}$. Then, we define $B_{i(j)}$ and $\mathcal{B}'_j$ analogously.

Lemma 9. *For $1 \leq j \leq k$, there are at least $\lceil (1-\varepsilon')k \rceil$ inequalities where*

$$f(\mathcal{A}'_j) - f(\mathcal{A}'_{j-1}) + \frac{M}{ck} \geq \frac{1-\varepsilon'}{k} \left(f(O \cup A_{i(j)-1}) - f(\mathcal{A}'_{j-1}) \right).$$

And for any j,

$$f(\mathcal{A}'_j) \geq f(\mathcal{A}'_{j-1}).$$

The proof of the above lemma can be found in Appendix F. Following the notations and the proof of Lemma 9, we can get an analogous result for the gain of $\mathcal{B}'$ as follows.

Lemma 10. *For $1 \leq j \leq k$, there are at least $\lceil (1-\varepsilon')k \rceil$ of j such that*

$$f(\mathcal{B}'_j) - f(\mathcal{B}'_{j-1}) + \frac{M}{ck} \geq \frac{1-\varepsilon'}{k} \left(f((O \backslash A) \cup B_{i(j)-1}) - f(\mathcal{B}'_{j-1}) \right).$$

And for any j,

$$f(\mathcal{B}'_j) \geq f(\mathcal{B}'_{j-1}).$$

The next lemma proved in Appendix F establishes the main recurrence.

Lemma 11. *Let $\Gamma_u = f(\mathcal{A}'_{j(u)}) + f(\mathcal{B}'_{j(u)})$, where $j(u)$ is the u-th j which satisfies Lemma 9 or Lemma 10. Then, there are at least $\lceil (1-\varepsilon')k \rceil$ of u follow that*

$$f(O \backslash A) - \Gamma_u - \frac{2M}{c(1-\varepsilon')} \leq \left(1 - \frac{1-\varepsilon'}{k}\right) \left(f(O \backslash A) - \Gamma_{u-1} - \frac{2M}{c(1-\varepsilon')} \right).$$

Lemma 11 yields a recurrence of the form $(b - u_{i+1}) \leq a\,(b - u_i)$, $u_0 = 0$, and has the solution $u_i \geq b(1 - a^i)$. Consequently, we have

$$\begin{aligned} f(A') + f(B') &\geq \left(1 - \left(1 - \frac{1-\varepsilon'}{k}\right)^{(1-\varepsilon')k}\right) \left(f(O \backslash A) - \frac{2M}{c(1-\varepsilon')} \right) \\ &\geq \left(1 - e^{-(1-\varepsilon')^2}\right) \left(f(O \backslash A) - \frac{2M}{c(1-\varepsilon')} \right) \end{aligned} \tag{5}$$

Let $\beta = 1 - e^{-(1-\varepsilon')^2}$. From the choice of C on line 16, we have $2f(C) \geq f(A') + f(B')$ and so from (5), we have

$$\begin{aligned} f(O \setminus A) &\leq \frac{2}{\beta} f(C) + \frac{2M}{c(1-\varepsilon')} \\ &\leq \frac{2}{\beta} f(C) + \frac{2f(O)}{c(1-\varepsilon')}. \end{aligned} \tag{6}$$

Since an $(1/\alpha)$-approximation is used for UnconstrainedMax, for any A, $f(O \cap A)/\alpha \leq \mathbb{E}[f(A'')|A]$; therefore,

$$\mathbb{E}[f(O \cap A)] \leq \alpha \mathbb{E}[f(C)]. \tag{7}$$

For any set A, $f(O) \leq f(O \cap A) + f(O \setminus A)$ by submodularity and nonnegativity. Therefore, by Inequalities 6 and 7,

$$\begin{aligned} f(O) &\leq \mathbb{E}[f(O \cap A) + f(O \setminus A)] \\ &\leq \alpha \mathbb{E}[f(C)] + \frac{2}{\beta} \mathbb{E}[f(C)] + \frac{2f(O)}{c(1-\varepsilon')}. \end{aligned}$$

Therefore, we have from Lemma 18 in Appendix F,

$$\mathbb{E}[f(C)] \geq \frac{1 - \frac{2}{c(1-\varepsilon')}}{\alpha + \frac{2}{\beta}} f(O) \geq \left(\frac{e-1}{\alpha(e-1) + 2e} - \varepsilon \right) f(O). \qquad \square$$

5. Empirical Evaluation

In this section, we evaluate our algorithm in comparison with the state-of-the-art parallelizable algorithms: AdaptiveNonmonotoneMax of Fahrbach et al. (2019), the algorithm of Ene and Nguyen (2020), and two versions of ParCardinal in Amanatidis et al. (2021) (ParCardinal(v1) represents the one without binary search and ParCardinal(v2) is the one with binary search). Also, we compare four versions of our algorithms with different threshold procedures: Threshold-Sampling of Fahrbach et al. (2019), two versions of threshold sampling algorithms of Amanatidis et al. (2021), and ThreshSeq proposed in this paper. Our results are summarized as follows. [1]

- Our algorithm ATG obtains the best objective value of any of the parallelizable algorithms; obtaining an improvement of up to 19% over the next algorithm, our AST. Both Fahrbach et al. (2019) and Ene and Nguyen (2020) exhibit a large loss of objective value at both small and large k values.
- Both our algorithm AST, ParCardinal(v1), and AdaptiveNonmonotoneMax use a very small number of adaptive rounds. Both ATG and the algorithm of Ene and Nguyen (2020) use roughly an order of magnitude more adaptive rounds.

1. Our code is available at *https://gitlab.com/luciacyx/nm-adaptive-code.git*.

- The algorithm of Ene and Nguyen (2020) is the most query efficient if access is provided to an exact oracle for the multilinear extension of a submodular function and its gradient [2]. However, if these oracles must be approximated with the set function, their algorithm becomes very inefficient and does not scale beyond small instances ($n \leq 100$).
- Our algorithms used fewer queries to the submodular set function than the linear-time algorithm FastRandomGreedy in Buchbinder et al. (2015). Both versions of ParCardinal are the most query inefficient.
- Comparing AST with four threshold sampling algorithms, our ThreshSeq proposed in this paper is the most query and round efficient without loss of objective values. If running Threshold-Sampling theoretically, with a large amount of sampling in ReducedMean, we empirically establish that the query complexity of algorithms using Threshold-Sampling can be three to four orders of magnitude worse than other algorithms over the SMCC instances in our benchmark.

5.1 Algorithm Setup for AST and ATG

In the pseudocodes for AST and ATG, M is used as the upper bound of OPT/k which is set to $\max_{x \in \mathcal{N}} f(x)$. In the experiment, we used a sharper upper bound, the average of the top k singleton values, maintaining the analysis of approximation ratio. Additionally, the $(1/2)-$approximation UnconstrainedMax algorithm is substituted with a random set, which is a $(1/4)$-approximation by Feige et al. (2011). Consequently, the obtained approximation ratios for AST and ATG in the actual experiment are $1/8-\varepsilon$ and $0.139-\varepsilon$, respectively.

5.2 Comparison Algorithms and Other Settings

In addition to the algorithms discussed in the preceding paragraphs, we evaluate the following baselines: the IteratedGreedy algorithm of Gupta et al. (2010), and the linear-time $(1/e-\varepsilon)$-approximation algorithm FastRandomGreedy of Buchbinder et al. (2015). These algorithms are both $\mathcal{O}(k)$-adaptive, where k is the cardinality constraint.

The algorithm of Ene and Nguyen (2020) requires access to an oracle for the multilinear extension and its gradient. In the case of maximum cut, the multilinear extension and its gradient can be computed in closed form in time linear in the size of the graph, as described in Appendix G. This fact enables us to evaluate the algorithm of Ene and Nguyen (2020) using direct oracle access to the multilinear extension and its gradient on the maximum cut application. However, no closed form exists for the multilinear extension of the revenue maximization objective. In this case, we found (see Appendix G.1) that sampling to approximate the multilinear extension is exorbitant in terms of runtime; hence, we were unable to evaluate Ene and Nguyen (2020) on revenue maximization.

For all algorithms, the accuracy parameter ε was set to 0.1; the failure probability parameter δ was set to 0.1; 100 samples were used to evaluate expectations for Threshold-Sampling

2. The definition of the multilinear extension is given in Appendix G.

in AdaptiveNonmonotoneMax (thus, this algorithm was run as heuristics with no performance guarantee). Further, in the algorithms AdaptiveThresholdGreedy, ParCardinal, and AdaptiveNonmonotoneMax, we ignored the smaller values of ε, δ in each algorithm, and simply used the input values of ε and δ. For AdaptiveThresholdGreedy and ParCardinal, by using the best solution value found so far as a lower bound on OPT, we used an early termination condition to check if the threshold value $\tau < \alpha \mathrm{OPT}(1-\varepsilon)/k$, where α is the approximation ratio for each algorithm. This early termination condition is responsible for the high variance in total queries. We attempted to use the same sharper upper bound on OPT/k as our algorithms in AdaptiveNonmonotoneMax, but it resulted in significantly worse objective values, so we simply used the maximum singleton as described in Fahrbach et al. (2019). ParCardinal is generalized by an algorithm that deals with knapsack constraints. By calling threshold sampling algorithm a large number of times, ParCardinal is able to achieve a constant probability on certain events. Hence, it is comparatively less efficient than algorithms dealing with cardinality constraints. For our experiments, we ran only ParCardinal(v1) and ParCardinal(v2) on BA and ca-GrQc datasets.

Randomized algorithms are averaged over 20 independent repetitions, and the mean is reported. The standard deviation is indicated by a shaded region in the plots. Any algorithm that requires a subroutine for UnconstrainedMax is implemented to use a random set, following the setting used for AST and ATG.

5.3 Applications and Datasets

Maxcut. The cardinality-constrained maximum cut function is defined as follows. Given graph $G = (V, E)$, and nonnegative edge weight w_{ij} on each edge $(i, j) \in E$. For $S \subseteq V$, let

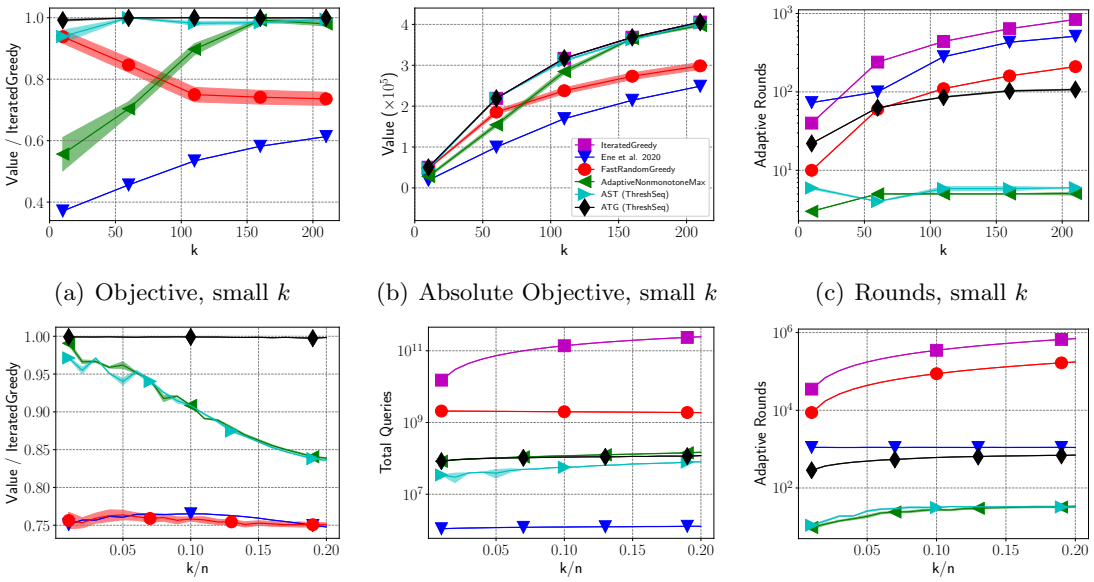
$$f(S) = \sum_{i \in V \setminus S} \sum_{j \in S} w_{ij}.$$

In general, this is a non-monotone, submodular function. In our implementation, all edges have a weight of 1.

Revmax. The revenue maximization objective is defined as follows. Let graph $G = (V, E)$ represent a social network, with nonnegative edge weight w_{ij} on each edge $(i, j) \in E$. We use the concave graph model introduced by Hartline et al. (2008). In this model, each user $i \in V$ is associated with a non-negative, concave function $f_i : \mathbb{R}^+ \to \mathbb{R}^+$. The value $v_i(S) = f_i(\sum_{j \in S} w_{ij})$ encodes how likely the user i is to buy a product if the set S has adopted it. Then the total revenue for seeding a set S is

$$f(S) = \sum_{i \in V \setminus S} f_i \left(\sum_{j \in S} w_{ij} \right).$$

This is a non-monotone, submodular function. In our implementation, each edge weight $w_{ij} \in (0, 1)$ is chosen uniformly randomly; further, $f_i(\cdot) = (\cdot)^{\alpha_i}$, where $\alpha_i \in (0, 1)$ is chosen uniformly randomly for each user $i \in V$.



(a) Objective, small k (b) Absolute Objective, small k (c) Rounds, small k

(d) Objective, large k (e) Queries, large k (f) Rounds, large k

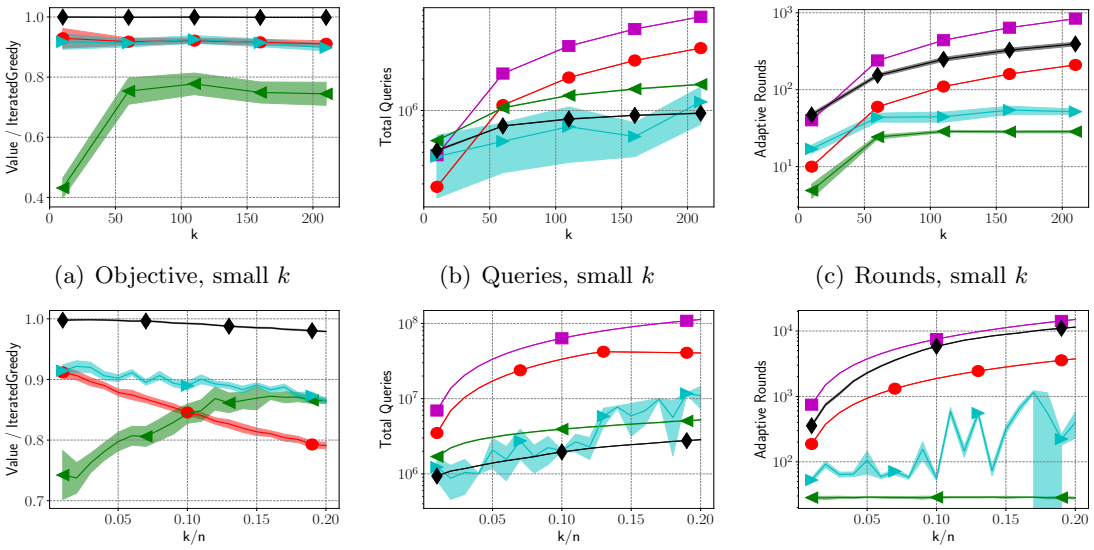
Figure 2: Comparison of objective value (normalized by the IteratedGreedy objective value), total queries, and adaptive rounds on web-Google for the maxcut application for both small and large k values. The large k values are given as a fraction of the number of nodes in the network. The algorithm of Ene and Nguyen (2020) is run with oracle access to the multilinear extension and its gradient; total queries reported for this algorithm are queries to these oracles, rather than the original set function. The legend in Fig. 2(b) applies to all other subfigures.

Dataset. Network topologies from SNAP were used; specifically, web-Google ($n = 875713$, $m = 5105039$), a web graph from Google, ca-GrQc ($n = 5242, m = 14496$), a collaboration network from Arxiv General Relativity and Quantum Cosmology, and ca-Astro ($n = 18772, m = 198110$), a collaboration network of Arxiv Astro Physics. In addition, a Barabási-Albert random graph was used (BA), with $n = 968$, $m = 5708$.

5.4 Main Results

In Fig. 2, we show representative results for cardinality-constrained maximum cut on web-Google ($n = 875713$) for both small and large k values. Results on other datasets and revenue maximization are given in Fig. 4 and 3. In addition, results for Ene and Nguyen (2020) when the multilinear extension is approximated via sampling are given in Appendix G.1. The algorithms are evaluated by objective value of solution, total queries made to the oracle, and the number of adaptive rounds (lower is better). Objective value is normalized by that of IteratedGreedy.

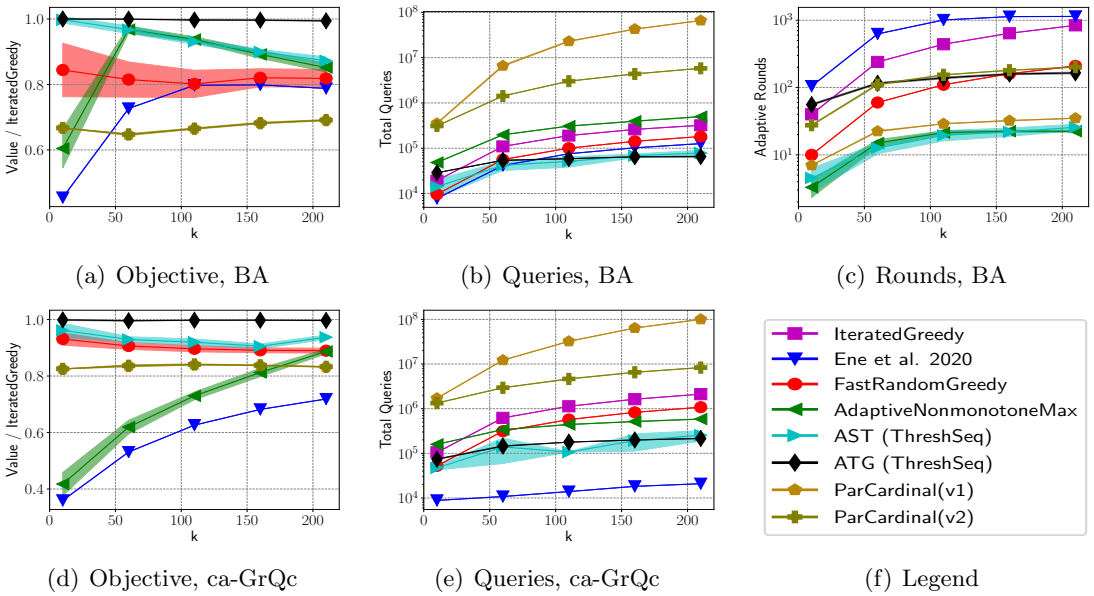
In terms of objective value (Figs. 2(a) and 2(d)), our algorithm ATG maintained better than 0.99 of the IteratedGreedy value, while all other algorithms fell below 0.95 of the IteratedGreedy value on some instances. Our algorithm AST obtained similar objective



(a) Objective, small k (b) Queries, small k (c) Rounds, small k

(d) Objective, large k (e) Queries, large k (f) Rounds, large k

Figure 3: Results for revenue maximization on ca-Astro, for both small and large k values. Large k values are indicated by a fraction of the total number n of nodes. The legends in Fig. 2 and 4 apply.



(a) Objective, BA (b) Queries, BA (c) Rounds, BA

(d) Objective, ca-GrQc (e) Queries, ca-GrQc (f) Legend

Figure 4: Additional results for maximum cut on BA and ca-GrQc with ParCardinal algorithms.

value to AdaptiveNonmonotoneMax on larger k values, but performed much better on small k values. Finally, the algorithm of Ene and Nguyen (2020) obtained poor objective value for $k \leq 100$ and about 0.75 of the IteratedGreedy value on larger k values. It

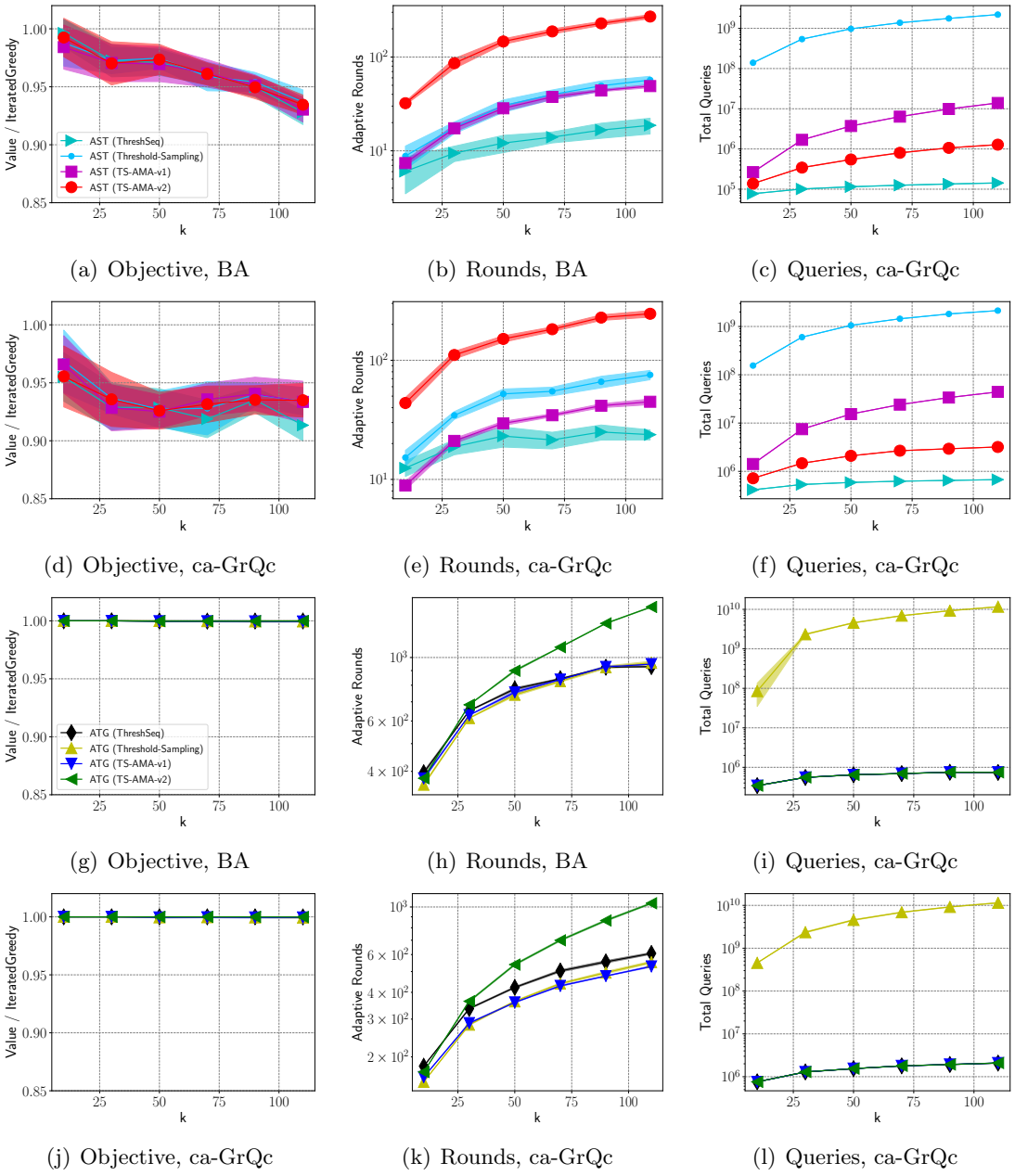
is interesting to observe that the two algorithms with the best approximation ratio of $1/e$, Ene and Nguyen (2020) and FastRandomGreedy, returned the worst objective values on larger k (Fig. 2(d)). For total queries (Fig. 2(e)), the most efficient is Ene and Nguyen (2020), although it does not query the set function directly, but the multilinear extension and its gradient. The most efficient of the combinatorial algorithms was AST, followed by ATG. Finally, with respect to the number of adaptive rounds (Fig. 2(f)), the best was AdaptiveNonmonotoneMax, closely followed by AST; the next lowest was ATG, followed by Ene and Nguyen (2020).

The results in Fig. 4 and 3 are qualitatively similar. Regarding the ParCardinal algorithms, the results in Fig. 4 demonstrate that ParCardinal(v2) is highly parallelizable. However, despite achieving a 0.172 approximation ratio, the objective values of ParCardinal(v1) and ParCardinal(v2) fell below 0.85 of the IteratedGreedy. Because of a constant number of call repetitions to ThreshSeq in ParCardinal, these two algorithms are the most query inefficient and are roughly two to three orders of magnitude worse than our algorithms.

5.5 Comparison of Different Threshold Sampling Procedures.

Fig. 5 shows the results of AST and ATG with different threshold sampling procedures for cardinality-constrained maximum cut on two datasets, BA ($n = 968$) and ca-GrQc ($n = 5242$). All the algorithms are run according to pseudocode without any modification. TS-AMA-v1 and TS-AMA-v2 represent the ThreshSeq algorithms without and with binary search proposed in Amanatidis et al. (2021).

All four versions of AST return similar results on objective values; see Figs. 5(a) and 5(d). As for adaptive rounds, ThreshSeq, Threshold-Sampling, and TS-AMA-v1 all run in $\mathcal{O}(\log(n))$ rounds, while TS-AMA-v2 runs in $\mathcal{O}\left(\log^2(n)\right)$ rounds. By the results in Figs. 5(b) and 5(e), our ThreshSeq is the most highly parallelizable algorithm, followed by TS-AMA-v1. TS-AMA-v2 is significantly worst as what it is in theory. Perhaps the reason why our algorithm performed better in practical settings can be attributed to the following factors. Theoretically, all algorithms except for TS-AMA-v2 have the same order of adaptivity. However, the adaptivity of each algorithm is associated with different constants, which in turn depend on the design of the algorithm. Our algorithm maintains two sets A, $A' \subseteq A$ during its execution. At the beginning of each iteration, the filtration step is with respect to set A, which contains elements with negative marginal gains; at the end of the algorithm, elements with negative marginal gains are excluded from A to get A', which is used to get the average marginal gains of the solution. From an experimental point of view, this implementation allows us to filter out more elements after one round while maintaining the same average marginal gain. With respect to the query calls, while our ThreshSeq only queries once for each prefix, Threshold-Sampling queries $16\lceil\log(2/\hat{\delta})/\hat{\varepsilon}^2\rceil$ times, and both TS-AMA-v1 and TS-AMA-v2 query $|V|$ times. According to Figs. 5(c) and 5(f), our ThreshSeq is the most query efficient one among all. Also, the total queries do not increase a lot when k increases. With binary search, TS-AMA-v2 is the second best one which has $\mathcal{O}\left(n\log^2(n)\right)$ query complexity. As for



(a) Objective, BA (b) Rounds, BA (c) Queries, ca-GrQc

(d) Objective, ca-GrQc (e) Rounds, ca-GrQc (f) Queries, ca-GrQc

(g) Objective, BA (h) Rounds, BA (i) Queries, ca-GrQc

(j) Objective, ca-GrQc (k) Rounds, ca-GrQc (l) Queries, ca-GrQc

Figure 5: Results of AST and ATG with four threshold sampling procedures on two datasets. The algorithms are run strictly following pseudocode. The legends in Fig. 5(a) and 5(g) apply to all other subfigures.

Threshold-Sampling, with the input values as $n = 968$, $k = 10$, and $\varepsilon = \delta = 0.1$, it queries about 2×10^5 times for each prefix which is significantly large.

Regarding to different ATG algorithms, they all return the competitive solutions compared with IteratedGreedy; see Figs. 5(g) and 5(j). Since each iteration of ATG calls a

threshold sampling subroutine which is based on the solution of previous iterations and a slowly decreasing threshold τ, after the first filtration of the subroutine, the size of the candidate set is limited. Thus, there is no significant difference between different ATGs concerning rounds and queries. However, there are two exceptions. First, since TS-AMA-v2 is the only one who has $\mathcal{O}\left(\log^2(n)\right)$ adaptive rounds, it still runs with more rounds; see Figs. 5(h) and 5(k). Also, the number of queries of ATG with Threshold-Sampling is significantly large with the same reason discussed before.

Among all, ThreshSeq proposed in this paper is not only the best theoretically, but also performs well in experiments compared with the pre-existing threshold sampling algorithms.

6. Discussion and Future Directions

In this paper, we propose a new threshold sampling algorithm, ThreshSeq, which solves Threshold on non-monotone instances with high probability, optimal adaptivity, and query complexity. Different from other state-of-the-art thresholding algorithms, ThreshSeq is based on maintaining two sets that separately solve Threshold. Then, we propose two approximation algorithms AdaptiveSimpleThreshold and AdaptiveThresholdGreedy that are inspired by IteratedGreedy.

Compared to state-of-the-art algorithms, our ThreshSeq exhibits the highest query efficiency with relatively fewer adaptive rounds; ATG produces results that are almost identical to IteratedGreedy in terms of objective value, and, relatively speaking, is the most query efficient combinatorial algorithm; AST is the second most parallelizable algorithm among all algorithms and delivers reasonably good objective values. Despite demonstrating good results, it should be noted that our approximation algorithms rely on the UnconstrainedMax subroutine, which requires access to the multilinear extension and may be impractical in certain settings. So, in the experiment, we substituted it with a random subset sampling approach which provides an expected (1/4)-approximation ratio. However, this substitution may result in a decrease in the objective value during the experiment.

Further investigations are needed in our work and there is still significant room for improvement. For instance, in non-monotone submodular maximization problems, using the objective value of max singleton to guess OPT is a common practice that involves $\mathcal{O}\left(\log(n)\right)$ guesses. If we can reduce the number of guesses to a constant, the query complexity can be improved significantly by a factor of $\mathcal{O}\left(\log(n)\right)$. Additionally, the current theoretical best approximation ratio is 0.385. In our paper, the best we proposed is a $(0.193-\varepsilon)-$approximation algorithm ($(0.139-\varepsilon)-$approximation in our experiment with a random subset unconstraint algorithm). Hence, the question remains interesting: Can we parallelize other algorithms that provide a better approximation ratio?

In our paper, we focus on the number of queries and the query parallelizability assuming that the oracle computation time dominates the overall computation duration. However, in practical scenarios, the function representation significantly influences algorithm performance (*e.g.* the multilinear extension and its gradient have closed forms for maximum cut). Thus, for any particular application, there is a lot of room for improvement based on such specific representation of the submodular function.

Appendix A. Probability Lemma and Concentration Bounds

Lemma 12. *(Chernoff bounds (Mitzenmacher & Upfal, 2017)). Suppose X_1, ... , X_n are independent binary random variables such that $Pr(X_i = 1) = p_i$. Let $\mu = \sum_{i=1}^{n} p_i$, and $X = \sum_{i=1}^{n} X_i$. Then for any $\delta \geq 0$, we have*

$$Pr(X \geq (1+\delta)\mu) \leq e^{-\frac{\delta^2 \mu}{2+\delta}}. \tag{8}$$

Moreover, for any $0 \leq \delta \leq 1$, we have

$$Pr(X \leq (1-\delta)\mu) \leq e^{-\frac{\delta^2 \mu}{2}}. \tag{9}$$

Lemma 13. *(Chen et al., 2021). Suppose there is a sequence of n Bernoulli trials: $X_1, X_2, \ldots, X_n$, where the success probability of X_i depends on the results of the preceding trials $X_1, \ldots, X_{i-1}$. Suppose it holds that*

$$Pr(X_i = 1 | X_1 = x_1, X_2 = x_2, \ldots, X_{i-1} = x_{i-1}) \geq \eta,$$

where $\eta > 0$ is a constant and $x_1, \ldots, x_{i-1}$ are arbitrary.

Then, if $Y_1, \ldots, Y_n$ are independent Bernoulli trials, each with probability η of success, then

$$Pr\left(\sum_{i=1}^{n} X_i \leq b\right) \leq Pr\left(\sum_{i=1}^{n} Y_i \leq b\right),$$

where b is an arbitrary integer.

Moreover, let A be the first occurrence of success in sequence X_i. Then,

$$\mathbb{E}[A] \leq 1/\eta.$$

Lemma 14. *(Chen et al., 2021). Suppose there is a sequence of $n+1$ Bernoulli trials: $X_1, X_2, \ldots, X_{n+1}$, where the success probability of X_i depends on the results of the preceding trials $X_1, \ldots, X_{i-1}$, and it decreases from 1 to 0. Let t be a random variable based on the $n+1$ Bernoulli trials. Suppose it holds that*

$$Pr(X_i = 1 | X_1 = x_1, X_2 = x_2, \ldots, X_{i-1} = x_{i-1}, i \leq t) \geq \eta,$$

where $x_1, \ldots, x_{i-1}$ are arbitrary and $0 < \eta < 1$ is a constant. Then, if $Y_1, \ldots, Y_{n+1}$ are independent Bernoulli trials, each with probability η of success, then

$$Pr\left(\sum_{i=1}^{t} X_i \leq bt\right) \leq Pr\left(\sum_{i=1}^{t} Y_i \leq bt\right),$$

where b is an arbitrary integer.

Appendix B. Counterexample for Threshold-Sampling with Non-monotone Submodular Functions

Fahrbach et al. (2019) proposed a subroutine, Threshold-Sampling, which returns a solution $S \subseteq \mathcal{N}$ that $\mathbb{E}[f(S)/S] \geq (1-\varepsilon)\tau$ within logarithmic rounds and linear time. The full pseudocode for Threshold-Sampling is given in Alg. 6. The notation $\mathcal{U}(S,t)$ represents the uniform distribution over subsets of S of size t. Threshold-Sampling relies upon the procedure ReducedMean, given in Alg. 5. The Bernoulli distribution input to ReducedMean is the distribution $\mathcal{D}_t$, which is defined as follows.

Definition 15. *Conditioned on the current state of the algorithm, consider the process where the set $T \sim \mathcal{U}(A, t-1)$ and then the element $x \sim A \setminus T$ are drawn uniformly at random. Let $\mathcal{D}_t$ denote the probability distribution over the indicator random variable*

$$I_t = \mathbb{I}[f(S \cup T + x) - f(S \cup T) \geq \tau].$$

Below, we state the lemma of Threshold-Sampling in Fahrbach et al. (2019).

Lemma 16 (Fahrbach et al.'s (2019))**.** *The algorithm* Threshold-Sampling *outputs $S \subseteq \mathcal{N}$ with $|S| \leq k$ in $\mathcal{O}(\log(n/\delta)/\varepsilon)$ adaptive rounds such that the following properties hold with probability at least $1-\delta$:*

1. *There are $\mathcal{O}(n/\varepsilon)$ oracle queries in expectation.*
2. *The expected average marginal $\mathbb{E}[f(S)/|S|] \geq (1-\varepsilon)\tau$.*
3. *If $|S| < k$, then $f_x(S) < \tau$ for all $x \in \mathcal{N}$.*

In Fahrbach et al. (2019) and Kuhnle (2021), the above lemma is used with non-monotone submodular functions; however, in the case that f is non-monotone, the lemma does not hold. Alg. 5 only checks (on Line 13) if there is more than a constant fraction of elements whose marginal gains are larger than the threshold τ. If there exist elements with large magnitude, *negative* marginal gains, then the average marginal gain may fail to satisfy the lower bound in Lemma 16. As for the proof in Fahrbach et al. (2019), the following inequality does not hold (needed for the proof of Lemma 3.3 of Fahrbach et al. (2019)):

$$\mathbb{E}[\Delta(T \mid S)] \geq (\mathbb{E}[I_1] + \mathbb{E}[I_2] + \ldots + \mathbb{E}[I_t])\tau,$$

where $|T| = t^*$ and $t \geq t^*/(1+\hat{\varepsilon})$. Next, we give a counterexample for the two versions of Threshold-Sampling used in Fahrbach et al. (2019) and Fahrbach et al. (2019) where the only difference is that the if condition in Alg. 6 on Line 9 changes to $|A| < 3k$ in Fahrbach et al. (2019).

Counterexample 1. Define a set function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$ as follows, where $a \in \mathcal{N}$,

$$f(B) = \begin{cases} n^2 + |B|, & \text{if } a \notin B \\ n^2 + 1 - (|B|-1)n, & \text{if } a \in B \end{cases}.$$

Let $k = n = |\mathcal{N}| > 400$, $\tau = 1$, $\varepsilon = 0.1$, $\delta = 0.1$. Run Threshold-Sampling$(f, k, \tau, \varepsilon, \delta)$.

Algorithm 5 The ReducedMean algorithm of Fahrbach et al. (2019)

Input: access to a Bernoulli distribution $\mathcal{D}$, error ε, failure probability δ
Set number of samples $m \leftarrow 16\lceil \log(2/\delta)/\varepsilon^2 \rceil$
Sample $X_1, X_2, \ldots, X_m \sim \mathcal{D}$
Set $\bar{\mu} \leftarrow \frac{1}{m}\sum_{i=1}^{m} X_i$
if $\bar{\mu} \leq 1 - 1.5\varepsilon$ **then**
    return `true`
return `false`

Algorithm 6 The threshold sampling algorithm of Fahrbach et al. (2019)

1: **procedure** Threshold-Sampling($f, k, \tau, \varepsilon, \delta$)
2: **Input:** evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, constraint k, threshold τ, error ε, failure probability δ
3: Set smaller error $\hat{\varepsilon} \leftarrow \varepsilon/3$
4: Set iteration bounds $r \leftarrow \lceil \log_{(1-\hat{\varepsilon})^{-1}}(2n/\delta) \rceil, m \leftarrow \lceil \log(k)/\hat{\varepsilon} \rceil$
5: Set smaller failure probability $\hat{\delta} \leftarrow \delta/(2r(m+1))$
6: Initialize $S \leftarrow \emptyset, A \leftarrow N$
7: **for** r sequential rounds **do**
8: Filter $A \leftarrow \{x \in A : \Delta(x, S) \geq \tau\}$
9: **if** $|A| = 0$ **then**
10: **break**
11: **for** $i = 0$ to m in parallel **do**
12: Set $t \leftarrow \min\{\lfloor (1+\hat{\varepsilon})^i \rfloor, |A|\}$
13: $rm[t] \leftarrow$ ReducedMean $(\mathcal{D}_t, \hat{\varepsilon}, \hat{\delta})$
14: $t' \leftarrow \min t$ such that $rm[t]$ is `true`
15: Sample $T \sim \mathcal{U}(A, \min\{t', k - |S|\})$
16: Update $S \leftarrow S \cup T$
17: **if** $|S| = k$ **then**
18: **break**
19: **return** S

Proof. For any $B \subseteq \mathcal{N}$ and $x \in \mathcal{N} \backslash B$, the above set function follows that

$$\Delta(x \mid B) = \begin{cases} 1, & \text{if } x \neq a \text{ and } a \notin B \\ -n, & \text{if } x \neq a \text{ and } a \in B \\ 1 - |B|(n+1), & \text{if } x = a \end{cases}.$$

Thus, f is a non-negative, non-monotone submodular function.

Consider the first iteration of the outer for loop, where $S = \emptyset$, and $A = \mathcal{N}$ after Line 8. For any $1 < t \leq |\mathcal{N}|$, $|T| = t - 1$,

$$\mathbb{E}[I_t] = Pr(f(S \cup T + x) - f(S \cup T) \geq \tau) = Pr(x \neq a \text{ and } a \notin T) = 1 - \frac{t}{n}.$$

So, with any value of ε, ReducedMean returns `true` when $t > \varepsilon n/2$. The first round of Threshold-Sampling samples a set T_1 with $t_1' = |T_1| > \varepsilon n/2$. Then update S by $S = T_1$.

For the Threshold-Sampling in Fahrbach et al. (2019) with stop condition $|A| < 3k$, the algorithm stopped here after the first iteration, no matter what is sampled. In this case, the expectation of marginal gains of the set returned by the algorithm would be as follows,

$$\begin{aligned}\mathbb{E}\left[\Delta\left(S \mid \emptyset\right)\right] &= Pr\left(a \in T_1\right) \cdot \mathbb{E}\left[\Delta\left(T_1 \mid \emptyset\right) \mid a \in T_1\right] + Pr\left(a \notin T_1\right) \cdot \mathbb{E}\left[\Delta\left(T_1 \mid \emptyset\right) \mid a \notin T_1\right] \\ &= \frac{t_1'}{n} \cdot \left(1 - (t_1' - 1)n\right) + \frac{n - t_1'}{n} \cdot t_1' \\ &= t_1' \left(2 - t_1' + \frac{1 - t_1'}{n}\right) < 0.\end{aligned}$$

Next, we consider the Threshold-Sampling with stop condition $|A| = 0$. After the first iteration discussed above, if $a \in T_1$, all the elements would be filtered out at the second round. Algorithm stoped here and returned S, say S_1. If $a \notin T_1$, T_1 and a would be filtered out at the second round, which means $A = \mathcal{N} \backslash (S \cup \{a\})$. And for any $T \subseteq A$ and $x \in A \backslash T$,

$$f(S \cup T + x) - f(S \cup T) = 1.$$

Therefore, $\mathbb{E}\left[I_t\right] = 1$ for all t. After several iterations, $S = \mathcal{N} \backslash \{a\}$ would be returned, say S_2.

The expectation of objective value of the set returned would be as follows,

$$\begin{aligned}\mathbb{E}\left[\Delta\left(S \mid \emptyset\right)\right] &= Pr\left(a \in T_1\right) \cdot \mathbb{E}\left[\Delta\left(S_1 \mid \emptyset\right) \mid a \in T_1\right] + Pr\left(a \notin T_1\right) \cdot \mathbb{E}\left[\Delta\left(S_2 \mid \emptyset\right) \mid a \notin T_1\right] \\ &= \frac{t_1'}{n}(1 - (t_1' - 1)n) + \frac{n - t_1'}{n}(n - 1) \\ &= \frac{2t_1'}{n} - 1 - t_1'^2 + n < 0,\end{aligned}$$

since $\varepsilon = 0.1$, $n > 400$, and $\varepsilon n/2 < t_1' < \varepsilon(1 + \varepsilon/3)n/2$. □

Appendix C. Proofs for Section 2

Lemma 4. *Given V after* ***random-permutation*** *on Line 8, let $S_i = \{x \in V : \Delta\left(x \mid A \cup T_i\right) < \tau\}$. It holds that $|S_0| = 0$, $|S_{|V|}| = |V|$, and $|S_{i-1}| \leq |S_i|$.*

proof of Lemma 4. After filtering on Line 5, any element $x \in V$ follows that $\Delta\left(x \mid A\right) \geq \tau$. Therefore, $S_0 = \emptyset$. Also, it is obvious that $\Delta\left(x \mid A \cup V\right) = 0$. So, $S_{|V|} = V$. Next, let's consider any $x \in S_{i-1}$. By submodularity,

$$\Delta\left(x \mid A \cup T_i\right) \leq \Delta\left(x \mid A \cup T_{i-1}\right) < \tau.$$

Thus, for any $x \in S_{i-1}$, it holds that $x \in S_i$, which means $S_{i-1} \subseteq S_i$. □

Lemma 5. *It holds that $Pr\left(i^* < \min\{s, t\}\right) \leq 1/2$.*

proof of Lemma 5. Call an element $v_i \in V$ *bad* iff $\Delta\left(v_i \mid A \cup T_{i-1}\right) < \tau$; and *good*, otherwise. The random permutation of V can be regarded as $|V|$ dependent Bernoulli trials, with

success iff the element is bad and failure otherwise. Observe that, the probability that an element in T_i is bad, when $i \leq t$, is less than $\varepsilon/2$, conditioned on the outcomes of the preceding trials. We know that,

$$Pr\left(i^* < \min\{s,t\}\right) \leq Pr\left(\#\text{ bad elements in } T_{i'} > \varepsilon i', \text{ where } i' = \min\{s,t\}\right).$$

Let $X_i = 1$, if v_i is bad; and $X_i = 0$, otherwise. Then, (X_i) is a sequence of dependent Bernoulli trails. And for any $i \leq i'$, $Pr\left(X_i = 1\right) \leq \varepsilon/2$. Let (Y_i) be a sequence of independent and identically distributed Bernoulli trails, each with success probability $\varepsilon/2$. Then, the probability of $i^* < \min\{s,t\}$ can be bounded as follows:

$$Pr\left(i^* < \min\{s,t\}\right) \leq Pr\left(\sum_{i=1}^{i'} X_i > \varepsilon i'\right) \overset{(a)}{\leq} Pr\left(\sum_{i=1}^{i'} Y_i > \varepsilon i'\right) \overset{(b)}{\leq} 1/2,$$

where Inequality (a) follows from Lemma 14, and Inequality (b) follows from Law of Total Probability and Markov's inequality. □

Lemma 6. *Say an element added to the solution set is good if its gain is greater than τ. Suppose that Algorithm 2 terminates successfully. A and A' returned by Algorithm 2 hold the following properties:*

1) There are at least $(1-\varepsilon)$-fraction of A that is good.

2) A good element in A is always a good element in A'.

3) And, any element in A' has non-negative marginal gain when added.

proof of Lemma 6. Let A_j be the set A after iteration j, $T_{j,i}$ be the first i elements of V_j at j-th iteration. Similarly, define A'_j as the set A' after iteration j, $T'_{j,i} = T_{j,i} \cap A'$.

From Algorithm 2, $A = \sum_{j=1}^{\ell} T_{j,i^*}$. For each T_{j,i^*}, there are at least $(1-\varepsilon)$-fraction of T_{j,i^*} are good. Totally, there are at least $(1-\varepsilon)$-fraction of A are good.

By Line 17, $T'_{j,i}$ only contains the elements with nonnegative marginal gains in $T_{j,i}$. Therefore, any element in A' has nonnegative marginal gain when added. For any good element $v_i \in V_j$, by submodularity, $\Delta\left(v_i \,|\, A'_{j-1} \cup T'_{j,i-1}\right) \geq \Delta\left(v_i \,|\, A_{j-1} \cup T_{j,i-1}\right) \geq \tau$. Thus, a good element in A is always good in A'. □

Appendix D. ThresholdGreedy and Modification

In this section, we describe ThresholdGreedy (Alg. 7) of Badanidiyuru and Vondrák (2014) and how it is modified to have low adaptivity. This algorithm achieves ratio $1-1/e-\varepsilon$ in $\mathcal{O}\left(n \log k\right)$ queries if the function f is monotone but has no constant ratio if f is not monotone.

The ThresholdGreedy algorithm works as follows: a set S is initialized to the empty set. Elements whose marginal gain exceed a threshold value are added to the set in the following way: initially, a threshold of $\tau = M = \arg\max_{a \in \mathcal{N}} f(a)$ is chosen, which is iteratively

Algorithm 7 The ThresholdGreedy Algorithm of Badanidiyuru and Vondrák (2014)

procedure ThresholdGreedy(f, k, ε)
  Input: evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, constraint k, accuracy parameter $\varepsilon > 0$
  $M \leftarrow \arg\max_{x \in \mathcal{N}} f(x)$;
  $S \leftarrow \emptyset$
  for $\tau = M; \tau \geq (1-\varepsilon)M/k; \tau \leftarrow \tau(1-\varepsilon)$ **do**
    for $x \in \mathcal{N}$ **do**
      if $f(S \cup \{x\}) - f(S) \geq \tau$ **then**
        $S \leftarrow S \cup \{x\}$
        if $|S| = k$ **then**
          break from outer **for**
  return S

Algorithm 8 The IteratedGreedy Algorithm of Gupta et al. (2010)

procedure IteratedGreedy(f, k)
  Input: evaluation oracle $f : 2^{\mathcal{N}} \to \mathbb{R}^+$, constraint k
  $A \leftarrow \emptyset$
  for $i \leftarrow 1$ to k **do**
    $a_i \leftarrow \arg\max_{x \in \mathcal{N}} f(A \cup \{x\}) - f(A)$
    $A \leftarrow A \cup \{a_i\}$
  $B \leftarrow \emptyset$
  for $i \leftarrow 1$ to k **do**
    $b_i \leftarrow \arg\max_{x \in \mathcal{N} \setminus A} f(B \cup \{x\}) - f(B)$
    $B \leftarrow B \cup \{b_i\}$
  $A' \leftarrow$ UnconstrainedMax (A)
  return $C \leftarrow \arg\max\{f(A), f(A'), f(B)\}$

decreased by a factor of $(1-\varepsilon)$ until $\tau < M/k$. For each threshold τ, a pass through all elements of $\mathcal{N}$ is made, during which any element x that satisfies $f(S \cup \{x\}) - f(S) \geq \tau$ is added to the set S. While this strategy leads to an efficient $\mathcal{O}(n \log k)$ total number of queries, it also has $\Omega(n \log k)$ adaptivity, as each query depends on the previous ones.

To make this approach less adaptive, we replace the highly adaptive pass through $\mathcal{N}$ (the inner **for** loop) with a single call to Threshold-Sampling, which requires $\mathcal{O}(\log n)$ adaptive rounds and $\mathcal{O}(n/\varepsilon)$ queries in expectation. This modified greedy approach appears twice in ATG (Alg. 4), corresponding to the two **for** loops.

Appendix E. Improved Ratio for IteratedGreedy

In this section, we prove an improved approximation ratio for the algorithm Iterated-Greedy of Gupta et al. (2010), wherein a ratio of $1/(4+\alpha)$ is proven given access to a $1/\alpha$-approximation for UnconstrainedMax. We improve this ratio to $\frac{e-1}{e(2+\alpha)-\alpha} \approx 0.193$ if $\alpha = 2$. Pseudocode for IteratedGreedy is given in Alg. 8.

IteratedGreedy works as follows. First a standard greedy procedure is run which produces set A of size k. Next, a second greedy procedure is run to yield set B; during this second procedure, elements of A are ignored. A subroutine for UnconstrainedMax is used on f restricted to A, which yields set A'. Finally the set of $\{A, A', B\}$ that maximizes f is returned.

Theorem 17. *Suppose there exists an $(1/\alpha)$-approximation for* UnconstrainedMax. *Then by using this procedure as a subroutine, the algorithm* IteratedGreedy *has approximation ratio $\frac{e-1}{e(2+\alpha)-\alpha}$ for **SMCC**.*

Proof. For $1 \leq i \leq k$, let a_i, b_i be as chosen during the run of IteratedGreedy. Define $A_i = \{a_1, \ldots, a_{i-1}\}$, $B_i = \{b_1, \ldots, b_{i-1}\}$. Then for any $1 \leq i \leq k$, we have

$$\begin{aligned} f(A_{i+1}) + f(B_{i+1}) - f(A_i) - f(B_i) &= f_{a_i}(A_i) + f_{b_i}(B_i) \\ &\geq \frac{1}{k} \sum_{o \in O} f_o(A_i) + \frac{1}{k} \sum_{o \in O \setminus A} f_o(B_i) \\ &\geq \frac{1}{k} \left(f\left(O \cup A_i\right) - f\left(A_i\right) + f\left((O \setminus A) \cup B_i\right) - f\left(B_i\right)\right) \\ &\geq \frac{1}{k} \left(f\left(O \setminus A\right) - \left(f\left(A_i\right) + f\left(B_i\right)\right)\right), \end{aligned}$$

where the first inequality follows from the greedy choices, the second follows from submodularity, and the third follows from submodularity and the fact that $A_i \cap B_i = \emptyset$. Hence, from this recurrence and standard arguments,

$$f(A) + f(B) \geq (1 - 1/e) f\left(O \setminus A\right),$$

where A, B have their values at termination of IteratedGreedy. Since $f(A') \geq f(O \cap A)/\alpha$, we have from submodularity

$$\begin{aligned} f(O) &\leq f(O \cap A) + f(O \setminus A) \\ &\leq \alpha f(A') + (1 - 1/e)^{-1}(f(A) + f(B)) \\ &\leq (\alpha + 2(1 - 1/e)^{-1}) f(C). \end{aligned}$$

□

Appendix F. Proofs for Section 4

In this section, we provide the proofs omitted from Section 4.

Lemma 9. *For $1 \leq j \leq k$, there are at least $\lceil (1 - \varepsilon')k \rceil$ inequalities where*

$$f(\mathcal{A}'_j) - f(\mathcal{A}'_{j-1}) + \frac{M}{ck} \geq \frac{1 - \varepsilon'}{k} \left(f(O \cup A_{i(j)-1}) - f(\mathcal{A}'_{j-1}) \right).$$

And for any j,

$$f(\mathcal{A}'_j) \geq f(\mathcal{A}'_{j-1}).$$

Proof of Lemma 9. Since each element in A' has nonnegative marginal gain, it always holds that $f(\mathcal{A}'_j) \geq f(\mathcal{A}'_{j-1})$.

From Lemma 6, there are at least $(1-\varepsilon)$-fraction of A are good elements. Therefore, there are at least $(1-\varepsilon)k$ of a'_j which is good element or dummy element. Next, let's consider the following 3 cases of a'_j.

Case $i(j) = 1$ and a'_j is good. By Theorem 3 and Lemma 6, it holds that

$$f(\mathcal{A}'_j) - f(\mathcal{A}'_{j-1}) \geq \tau_1 = M \geq \frac{1}{k}\sum_{o\in O} f(o) \geq \frac{1}{k} f(O).$$

Case $i(j) > 1$ and a'_j is good. Since a'_j is returned at iteration $i(j)$ and a'_j is good, it holds that: (1) $f(\mathcal{A}'_j) - f(\mathcal{A}'_{j-1}) \geq \tau_{i(j)}$; (2) at previous iteration $i(j)-1$, ThreshSeq returns $S_{i(j)-1}$ that $|S_{i(j)-1}| < k - |A_{i(j)-2}|$. By property (2) and Theorem 3, for any $o \in O\backslash A_{i(j)-1}$, $\Delta\left(o \,|\, A_{i(j)-1}\right) < \tau_{i(j)-1}$. Then,

$$\begin{aligned}
f(\mathcal{A}'_j) - f(\mathcal{A}'_{j-1}) \geq \tau_{i(j)} &= (1-\varepsilon')\tau_{i(j)-1} \\
&> \frac{1-\varepsilon'}{k} \sum_{o\in O\backslash A_{i(j)-1}} \Delta\left(o \,|\, A_{i(j)-1}\right) \\
&\geq \frac{1-\varepsilon'}{k} \left(f(O\cup A_{i(j)-1}) - f(A_{i(j)-1})\right) \\
&\geq \frac{1-\varepsilon'}{k} \left(f(O\cup A_{i(j)-1}) - f(A'_{i(j)-1})\right) \qquad (10) \\
&\geq \frac{1-\varepsilon'}{k} \left(f(O\cup A_{i(j)-1}) - f(\mathcal{A}'_{j-1})\right), \qquad (11)
\end{aligned}$$

where Inequality 10 follows from the proof of Lemma 6, and Inequality 11 follows from $A'_{i(j)-1} \subseteq \mathcal{A}'_{j-1}$.

Case $i(j) = \ell + 1$ (or a'_j is dummy element). In this case, $|A| < k$ when the first **for** loop ends. So, ThreshSeq in the last iteration returns S_ℓ that $|S_\ell| < k - |A_{\ell-1}|$. From Theorem 3, it holds that $\Delta\left(o \,|\, A_\ell\right) < \tau_\ell < \frac{M}{ck}$, for any $o \in O\backslash A_\ell$. Thus,

$$\frac{M}{ck} > \frac{1}{k}\sum_{o\in O\backslash A_\ell} \Delta\left(o \,|\, A_\ell\right) \geq \frac{1}{k}\left(f(O\cup A_\ell) - f(A_\ell)\right) \overset{(a)}{\geq} \frac{1}{k}\left(f(O\cup A_\ell) - f(\mathcal{A}'_j)\right),$$

where Inequality (a) follows from $A_\ell = A$ and $f(\mathcal{A}'_j) = f(A')$.

The first inequality of Lemma 9 holds in those three cases with at least $(1-\varepsilon')k$ of j. □

Lemma 11. *Let $\Gamma_u = f(\mathcal{A}'_{j(u)}) + f(\mathcal{B}'_{j(u)})$, where $j(u)$ is the u-th j which satisfies Lemma 9 or Lemma 10. Then, there are at least $\lceil (1-\varepsilon')k \rceil$ of u follow that*

$$f(O\backslash A) - \Gamma_u - \frac{2M}{c(1-\varepsilon')} \leq \left(1 - \frac{1-\varepsilon'}{k}\right)\left(f(O\backslash A) - \Gamma_{u-1} - \frac{2M}{c(1-\varepsilon')}\right).$$

Proof of Lemma 11. From Lemma 9, $f(\mathcal{A}'_{j(u)-1}) \geq f(\mathcal{A}'_{j(u-1)})$, and

$$\begin{aligned} f(\mathcal{A}'_{j(u)}) &\geq \left(1 - \frac{1-\varepsilon'}{k}\right) f(\mathcal{A}'_{j(u)-1}) + \frac{1-\varepsilon'}{k} f(O \cup A_{i(j(u))-1}) - \frac{M}{ck} \\ &\geq \left(1 - \frac{1-\varepsilon'}{k}\right) f(\mathcal{A}'_{j(u-1)}) + \frac{1-\varepsilon'}{k} f(O \cup A_{i(j(u))-1}) - \frac{M}{ck}. \end{aligned}$$

Similarly,

$$f(\mathcal{B}'_{j(u)}) \geq \left(1 - \frac{1-\varepsilon'}{k}\right) f(\mathcal{B}'_{j(u-1)}) + \frac{1-\varepsilon'}{k} f((O \backslash A) \cup B_{i(j(u))-1}) - \frac{M}{ck}.$$

By adding the above two inequalities and the submodularity, we have,

$$\begin{aligned} \Gamma_u &\geq \left(1 - \frac{1-\varepsilon'}{k}\right) \Gamma_{u-1} + \frac{1-\varepsilon'}{k} \left(f(O \cup A_{i(j(u))-1}) + f((O \backslash A) \cup B_{i(j(u))-1})\right) - \frac{2M}{ck} \\ &\geq \left(1 - \frac{1-\varepsilon'}{k}\right) \Gamma_{u-1} + \frac{1-\varepsilon'}{k} f(O \backslash A) - \frac{2M}{ck}. \end{aligned}$$

□

Lemma 18. *Let $\varepsilon \in (0,1)$, and suppose $c = 8/\varepsilon$, $\varepsilon' = (1-1/e)\varepsilon/8$, and $\beta = 1 - e^{(1-\varepsilon')^2}$. Then*

$$\left(\frac{1 - \frac{2}{c(1-\varepsilon')}}{\alpha + \frac{2}{\beta}}\right) \geq \left(\frac{e-1}{\alpha(e-1)+2e} - \varepsilon\right). \tag{12}$$

Proof of Lemma 18. We start with the following two inequalities, which are verified below.

$$1 - \frac{2}{c(1-\varepsilon')} \geq 1 - \varepsilon, \tag{13}$$

$$\frac{2}{1 - e^{-(1-\varepsilon')^2}} \leq \frac{2}{1-1/e} + \varepsilon/2. \tag{14}$$

Let $A = 1$, $B = 1/\alpha + 2/(1-1/e)$. From the inequalities above, the left-hand side of (12) is at least $\frac{A-\varepsilon}{B+\varepsilon}$ and

$$\begin{aligned} \frac{A-\varepsilon}{B+\varepsilon} \geq \frac{A}{B} - \varepsilon &\iff \varepsilon \geq \frac{A}{B} - \frac{A-\varepsilon}{B+\varepsilon} \\ &\iff 1 \geq \frac{A}{\varepsilon B} - \frac{A}{\varepsilon(B+\varepsilon)} + \frac{1}{B+\varepsilon}. \end{aligned}$$

Next,

$$\frac{A}{\varepsilon B} - \frac{A}{\varepsilon(B+\varepsilon)} + \frac{1}{B+\varepsilon} = \frac{A}{B(B+\varepsilon)} + \frac{1}{B+\varepsilon} \leq 1/4 + 1/2 < 1,$$

since $B \geq 2$ and $A = 1$. Finally, $A/B = \frac{\alpha(e-1)}{e-1+2\alpha e}$.

Proof of (13).

$$c \geq 8/\varepsilon \geq \frac{2}{\varepsilon(1-\varepsilon')},$$

since $\varepsilon' = (1-1/e)\varepsilon/8 < 3/4$.

Proof of (14). Let $\lambda = 1-1/e$, $\kappa = e^{-(1-\varepsilon')^2}$. Inequality (14) is satisfied iff.

$$\begin{aligned}
2\lambda \leq 2(1-\kappa) + \frac{\lambda\varepsilon(1-\kappa)}{2} &\iff 2\lambda \leq 2 - 2\kappa + \lambda\varepsilon/2 - \lambda\varepsilon\kappa/2 \\
&\iff 2\kappa + \lambda\varepsilon\kappa/2 \leq \lambda\varepsilon/2 + 2 - 2\lambda \\
&\iff \kappa = e^{-(1-\varepsilon')^2} \leq \frac{\lambda\varepsilon/2 + 2 - 2\lambda}{2 + \lambda\varepsilon/2} \\
&\iff (1-\varepsilon')^2 \geq \log\left(\frac{2+\lambda\varepsilon/2}{\lambda\varepsilon/2+2-2\lambda}\right),
\end{aligned}$$

which in turn is satisfied if

$$2\varepsilon' \leq 1 - \log\left(\frac{2+\lambda\varepsilon/2}{\lambda\varepsilon/2+2-2\lambda}\right).$$

Then

$$\begin{aligned}
2\varepsilon' = \lambda\varepsilon/4 \leq \frac{2+\lambda\varepsilon/2-4\lambda}{2-\lambda} &\leq \frac{2+\lambda\varepsilon/2-4\lambda}{2+\lambda\varepsilon/2-2\lambda} \\
&= \frac{2(\lambda\varepsilon/2+2-2\lambda)-2-\lambda\varepsilon/2}{\lambda\varepsilon/2+2-2\lambda} \\
&= 2 - \frac{2+\lambda\varepsilon/2}{\lambda\varepsilon/2+2-2\lambda} \\
&= 1 - \left(\frac{2+\lambda\varepsilon/2}{\lambda\varepsilon/2+2-2\lambda} - 1\right) \\
&\leq 1 - \log\left(\frac{2+\lambda\varepsilon/2}{\lambda\varepsilon/2+2-2\lambda}\right),
\end{aligned}$$

where we have used $\log x \leq x-1$, for $x > 0$. □

Appendix G. Multilinear Extension and Implementation of Ene and Nguyen (2020)

In this section, we describe the multilinear extension and implementation of Ene and Nguyen (2020). The multilinear extension F of set function f is defined to be, for $x \in [0,1]^n$:

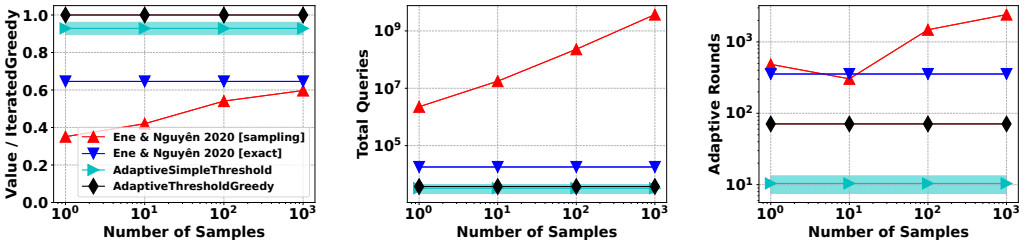
$$F(x) = \mathbb{E}\left[f(S)\right] = \sum_{S \subseteq V} f(S) Pr(S),$$

where

$$Pr(S) = \prod_{i \in S} x_i \cdot \prod_{i \notin S} (1 - x_i).$$

The gradient is approximated by using the central difference in each coordinate

$$\frac{dF}{dx}(x) \approx \frac{F(x+\gamma/2) - F(x-\gamma/2)}{\gamma},$$



(a) Objective value (b) Queries to set function (c) Adaptive Rounds

Figure 6: Comparison of our algorithms with Ene and Nguyen (2020) on a very small random graph ($n = 87$, $k = 10$). In all plots, the x-axis shows the number of samples used to approximate the multilinear extension.

unless using this approximation required evaluations outside the unit cube, in which case the forward or backward difference approximations were used. The parameter γ is set to 0.5.

Finally, for the maximum cut application, closed forms expressions exist for both the multilinear extension and its gradient. These are:

$$F(x) = \sum_{(u,v)\in E} x_u \cdot (1 - x_v) + x_v \cdot (1 - x_u),$$

and

$$(\nabla F)_u = \sum_{v \in N(u)} (1 - 2x_v).$$

Implementation. The algorithm was implemented as specified in the pseudocode on page 19 of the arXiv version of Ene and Nguyen (2020). We followed the same parameter choices as in Ene and Nguyen (2020), although we set $\varepsilon = 0.1$ as setting it to 0.05 did not improve the objective value significantly but caused a large increase in runtime and adaptive rounds. The value of $\delta = \varepsilon^3$ was used after communications with the authors.

G.1 Additional Experiments

In this section, we further investigate the performance of Ene and Nguyen (2020) when closed-form evaluation of the multilinear extension and its gradient are impossible. It is known that sampling to approximate the multilinear extension and its gradient is extremely inefficient or yields poor solution quality with a small number of samples. For this reason, we exclude this algorithm from our revenue maximization experiments. To perform this evaluation, we compared versions of the algorithm of Ene and Nguyen (2020) that use varying number of samples to approximate the multilinear extension.

Results are shown in Fig. 6 on a very small random graph with $n = 87$ and $k = 10$. The figure shows the objective value and total queries to the set function vs. the number of samples used to approximate the multilinear extension. There is a clear tradeoff between the solution quality and the number of queries required; at 10^3 samples per evaluation, the algorithm

matches the objective value of the version with the exact oracle; however, even at roughly 10^{11} queries (corresponding to 10^4 samples for each evaluation of the multilinear extension), the algorithm of Ene and Nguyen (2020) is unable to exceed 0.8 of the IteratedGreedy value. On the other hand, if ≤ 10 samples are used to approximate the multilinear extension, the algorithm is unable to exceed 0.5 of the IteratedGreedy value and still requires on the order of 10^7 queries.

References

Amanatidis, G., Fusco, F., Lazos, P., Leonardi, S., Marchetti-Spaccamela, A., & Reiffenhäuser, R. (2021). Submodular maximization subject to a knapsack constraint: Combinatorial algorithms with near-optimal adaptive complexity. In Meila, M., & Zhang, T. (Eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, Vol. 139 of *Proceedings of Machine Learning Research*, pp. 231–242. PMLR.

Badanidiyuru, A., & Vondrák, J. (2014). Fast algorithms for maximizing submodular functions. In Chekuri, C. (Ed.), *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pp. 1497–1514. SIAM.

Balkanski, E., Breuer, A., & Singer, Y. (2018). Non-monotone submodular maximization in exponentially fewer iterations. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., & Garnett, R. (Eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 2359–2370.

Balkanski, E., Rubinstein, A., & Singer, Y. (2019a). An exponential speedup in parallel running time for submodular maximization without loss in approximation. In Chan, T. M. (Ed.), *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pp. 283–302. SIAM.

Balkanski, E., Rubinstein, A., & Singer, Y. (2019b). An optimal approximation for submodular maximization under a matroid constraint in the adaptive complexity model. In Charikar, M., & Cohen, E. (Eds.), *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pp. 66–77. ACM.

Balkanski, E., & Singer, Y. (2018). The adaptive complexity of maximizing a submodular function. In Diakonikolas, I., Kempe, D., & Henzinger, M. (Eds.), *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pp. 1138–1151. ACM.

Buchbinder, N., & Feldman, M. (2019). Constrained submodular maximization via a non-symmetric technique. *Math. Oper. Res.*, *44*(3), 988–1005.

Buchbinder, N., Feldman, M., Naor, J., & Schwartz, R. (2012). A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pp. 649–658. IEEE Computer Society.

Buchbinder, N., Feldman, M., & Schwartz, R. (2015). Comparing apples and oranges: Query tradeoff in submodular maximization. In Indyk, P. (Ed.), *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pp. 1149–1168. SIAM.

Chekuri, C., & Quanrud, K. (2019). Parallelizing greedy for submodular set function maximization in matroids and beyond. In Charikar, M., & Cohen, E. (Eds.), *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pp. 78–89. ACM.

Chen, L., Feldman, M., & Karbasi, A. (2019). Unconstrained submodular maximization with constant adaptive complexity. In Charikar, M., & Cohen, E. (Eds.), *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pp. 102–113. ACM.

Chen, Y., Dey, T., & Kuhnle, A. (2021). Best of both worlds: Practical and theoretically optimal submodular maximization in parallel. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., & Vaughan, J. W. (Eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 25528–25539.

El-Arini, K., & Guestrin, C. (2011). Beyond keyword search: discovering relevant scientific literature. In Apté, C., Ghosh, J., & Smyth, P. (Eds.), *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pp. 439–447. ACM.

Ene, A., & Nguyen, H. L. (2019). Submodular maximization with nearly-optimal approximation and adaptivity in nearly-linear time. In Chan, T. M. (Ed.), *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pp. 274–282. SIAM.

Ene, A., & Nguyen, H. L. (2020). 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*, Vol. 119 of *Proceedings of Machine Learning Research*, pp. 2902–2911. PMLR.

Ene, A., Nguyen, H. L., & Vladu, A. (2019). Submodular maximization with matroid and packing constraints in parallel. In Charikar, M., & Cohen, E. (Eds.), *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pp. 90–101. ACM.

Fahrbach, M., Mirrokni, V., & Zadimoghaddam, M. (2019). Submodular Maximization with Nearly Optimal Approximation, Adaptivity, and Query Complexity. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 255–273.

Fahrbach, M., Mirrokni, V., & Zadimoghaddam, M. (2023). Non-monotone submodular maximization with nearly optimal adaptivity and query complexity. *arXiv preprint arXiv:1808.06932*.

Fahrbach, M., Mirrokni, V. S., & Zadimoghaddam, M. (2019). Non-monotone submodular maximization with nearly optimal adaptivity and query complexity. In Chaudhuri, K., & Salakhutdinov, R. (Eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, Vol. 97 of *Proceedings of Machine Learning Research*, pp. 1833–1842. PMLR.

Feige, U., Mirrokni, V. S., & Vondrák, J. (2011). Maximizing non-monotone submodular functions. *SIAM J. Comput.*, *40*(4), 1133–1153.

Gharan, S. O., & Vondrák, J. (2011). Submodular maximization by simulated annealing. In Randall, D. (Ed.), *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pp. 1098–1116. SIAM.

Gupta, A., Roth, A., Schoenebeck, G., & Talwar, K. (2010). Constrained non-monotone submodular maximization: Offline and secretary algorithms. In Saberi, A. (Ed.), *Internet and Network Economics - 6th International Workshop, WINE 2010, Stanford, CA, USA, December 13-17, 2010. Proceedings*, Vol. 6484 of *Lecture Notes in Computer Science*, pp. 246–257. Springer.

Hartline, J. D., Mirrokni, V. S., & Sundararajan, M. (2008). Optimal marketing strategies over social networks. In Huai, J., Chen, R., Hon, H., Liu, Y., Ma, W., Tomkins, A., & Zhang, X. (Eds.), *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pp. 189–198. ACM.

Kazemi, E., Mitrovic, M., Zadimoghaddam, M., Lattanzi, S., & Karbasi, A. (2019). Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In Chaudhuri, K., & Salakhutdinov, R. (Eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, Vol. 97 of *Proceedings of Machine Learning Research*, pp. 3311–3320. PMLR.

Kempe, D., Kleinberg, J. M., & Tardos, É. (2003). Maximizing the spread of influence through a social network. In Getoor, L., Senator, T. E., Domingos, P. M., & Faloutsos, C. (Eds.), *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pp. 137–146. ACM.

Kuhnle, A. (2021). Nearly linear-time, parallelizable algorithms for non-monotone submodular maximization. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Event, February 2-9, 2021*, pp. 8200–8208. AAAI Press.

Libbrecht, M. W., Bilmes, J. A., & Noble, W. S. (2018). Choosing non-redundant representative subsets of protein sequence data sets using submodular optimization. In Shehu, A., Wu, C. H., Boucher, C., Li, J., Liu, H., & Pop, M. (Eds.), *Proceedings of*

the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, BCB 2018, Washington, DC, USA, August 29 - September 01, 2018, p. 566. ACM.

Mirzasoleiman, B., Badanidiyuru, A., & Karbasi, A. (2016). Fast constrained submodular maximization: Personalized data summarization. In Balcan, M., & Weinberger, K. Q. (Eds.), *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, Vol. 48 of *JMLR Workshop and Conference Proceedings*, pp. 1358–1367. JMLR.org.

Mislove, A., Koppula, H. S., Gummadi, K. P., Druschel, P., & Bhattacharjee, B. (2008). Growth of the flickr social network. In Faloutsos, C., Karagiannis, T., & Rodriguez, P. (Eds.), *Proceedings of the first Workshop on Online Social Networks, WOSN 2008, Seattle, WA, USA, August 17-22, 2008*, pp. 25–30. ACM.

Mitzenmacher, M., & Upfal, E. (2017). *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press.

Nemhauser, G. L., & Wolsey, L. A. (1978). Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Res.*, *3*(3), 177–188.

Simon, I., Snavely, N., & Seitz, S. M. (2007). Scene summarization for online image collections. In *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, pp. 1–8. IEEE Computer Society.

Sipos, R., Swaminathan, A., Shivaswamy, P., & Joachims, T. (2012). Temporal corpus summarization using submodular word coverage. In Chen, X., Lebanon, G., Wang, H., & Zaki, M. J. (Eds.), *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pp. 754–763. ACM.

Tschiatschek, S., Iyer, R. K., Wei, H., & Bilmes, J. A. (2014). Learning mixtures of submodular functions for image collection summarization. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., & Weinberger, K. Q. (Eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 1413–1421.