Maximizing Non-Monotone Submodular Functions over Small Subsets: Beyond 1/2-Approximation

Aviad Rubinstein
Computer Science Department, Stanford University, CA, USA

Junyao Zhao
Computer Science Department, Stanford University, CA, USA

Abstract

In this work we give two new algorithms that use similar techniques for (non-monotone) submodular function maximization subject to a cardinality constraint. The first is an offline fixed-parameter tractable algorithm that guarantees a 0.539-approximation for all non-negative submodular functions. The second algorithm works in the random-order streaming model. It guarantees a $(1/2 + c)$-approximation for *symmetric* functions, and we complement it by showing that no space-efficient algorithm can beat 1/2 for asymmetric functions. To the best of our knowledge this is the first provable separation between symmetric and asymmetric submodular function maximization.

2012 ACM Subject Classification Theory of computation → Submodular optimization and polymatroids

Keywords and phrases Submodular optimization, Fixed-parameter tractability, Random-order streaming

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.106

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: https://arxiv.org/abs/2204.11149

Funding *Aviad Rubinstein*: Supported by NSF CCF-1954927, and a David and Lucile Packard Fellowship.
Junyao Zhao: Supported by NSF CCF-1954927.

1 Introduction

We study the algorithmic problem of selecting a small subset of k elements out of a (very) large ground set of n elements. In particular, we want the small subset to consist of k elements that are valuable *together*, as is captured by an objective function $f : 2^E \to \mathbb{R}_{\geq 0}$. Without any assumptions on f it is hopeless to get efficient algorithms; we make the assumption that f is *submodular*[1], one of the most fundamental and well-studied assumptions in combinatorial optimization.

Alas, even for submodular functions, strong impossibility results are known. Two of the most important frameworks we have for circumventing impossibility results are (i) approximation algorithms – look for solutions that are only approximately optimal; and (ii) paramerterized complexity – look for algorithms of which the runtime is efficient as a function of the large ground set n, but may have a worse dependence[2] on the smaller parameter

[1] I.e., functions where the marginal value of an element is decreasing as the set grows.

[2] Formally, an algorithm is said to be fixed-parameter tractable if it runs in time $h(k) \cdot \text{poly}(n)$ for *any* function h. Here h could be arbitrarily fast growing, e.g. doubly-exponential or Ackermann – this is asymptotically faster than the naive n^k. In this work we will be more ambitious (and closer to practice) and present algorithms that run in time $2^{\tilde{O}(k)} \cdot n$.

k. To appreciate the relevance of parameterized complexity in practice, it's important to note that in many applications of submodular maximization k is indeed quite small, e.g. in data summarization [4], we want an algorithm that, given a large image dataset chooses a representative subset of images that must be small enough to fit our screen.

Submodular optimization has been thoroughly studied under the lens of approximation algorithms; much less work has been done about its parameterized complexity (with the exception of [27], see discussion of related works). In either case, strong, tight hardness results are known. In this work we show that combining both approaches gives surprisingly powerful algorithms.

On the technical level, we develop novel insights to design and analyze fixed-parameter tractable (FPT) algorithms for (non-monotone) submodular function maximization. We instantiate these ideas to give new results in two settings: offline ("classical") algorithms for which we are interested in the running time and query complexity[3], and random-order streaming algorithms for which we mostly care about the memory cost.

Main result I: offline algorithms

Our first result is an (offline) FPT algorithm that guarantees an improved approximation ratio for submodular function maximization.

To compare our result with the approximation factors achievable by polynomial-time algorithms, we first briefly survey the existing algorithmic and hardness results.

On the algorithmic side, the current state-of-art polynomial-time algorithm for general non-monotone submodular functions achieves 0.385-approximation [5]. For sub-classes of submodular functions, better polynomial-time approximation algorithms are known: notable examples include monotone functions (the greedy algorithm achieves $(1 - 1/e)$-approximation [25]), and symmetric functions[4] (the state-of-the-art approximation factor is 0.432 [11]).

From the hardness perspective, known results rule out *polynomial* query complexity algorithms with approximation factors better than 1/2 or 0.491 for symmetric [10, 28] or asymmetric functions [14], respectively. It is also known that even FPT algorithms cannot beat $(1 - 1/e)$-approximation, and this holds even for monotone submodular functions [24].

In FPT time, the streaming algorithm of [2] implies a 1/2-approximation algorithm for general non-monotone functions (although it is not explicitly stated as an FPT algorithm in their paper), which slightly beats the aforementioned 0.491 bound for asymmetric functions. However, this result does not tell us whether FPT algorithms can beat the 1/2 bound for symmetric functions. A-priori, it was plausible that 1/2-approximation is the best achievable approximation by FPT algorithms for symmetric functions and hence general non-monotone functions. (In fact, prior to discovering our new algorithms, we had expected that the 1/2-approximation would indeed be the best that FPT algorithms can achieve.)

We are thus excited to report that we were able to design an FPT algorithm (Algorithm 4) that not only outperforms all the previous algorithms but also surpasses all the upper limits on the approximation factor in the existing hardness results, *by a significant margin, regardless of whether the function is symmetric or not*:

---

[3] I.e., the function f is given as a value oracle. The query complexity is number of queries made by the algorithm, which is clearly a lower bound of the runtime.

[4] I.e. functions that assign the same value to a set and its complement, which capture some of the most important applications of non-monotone submodular functions, including mutual information and cuts in (undirected) graphs and hypergraphs.

▶ **Theorem 1** (FPT algorithm). *There is a* 0.539*-approximation algorithm for cardinality constrained submodular maximization that has runtime and query complexity* $2^{\widetilde{O}(k)} \cdot n$.

Main result II: random-order streaming

Our FPT algorithm (Algorithm 4) uses a subroutine (Algorithm 1) which can be interpreted as a *random-order streaming algorithm*, and thus, in addition to our FPT result, we also hope to understand the power and the limit of Algorithm 1 in the random-order streaming model. In this model (see the detailed setup in Section 2), a streaming algorithm makes a single pass over a stream of elements arriving in a uniformly random order. An algorithm keeps a carefully chosen subset of the elements it has seen in a buffer of bounded size. At any point in the stream, the algorithm can make unlimited queries of the function values on subsets of the elements in the buffer. The goal is to obtain a good approximation of the offline optimum while keeping the buffer small (ideally, polynomial in k and independent of n).

We show that Algorithm 1 achieves 1/2-approximation for general non-monotone submodular functions[5] and beats 1/2-approximation for symmetric functions using $\widetilde{O}(k^2)$-size buffer[6]:

▶ **Theorem 2** (Random-order streaming algorithm). *For cardinality constrained submodular function maximization in the random-order streaming model, there is an algorithm using* $\widetilde{O}(k^2)$*-size buffer that achieves* 1/2*-approximation for general non-monotone submodular functions and* 0.5029*-approximation for symmetric submodular functions.*

We complement the algorithmic result with a tight 1/2-hardness result in the random-order streaming setting:

▶ **Theorem 3** (1/2-hardness for random-order streaming). *If* $n = 2^{o(k)}$*, any* $(1/2 + \varepsilon)$*-approximation algorithm for cardinality-constrained non-monotone submodular maximization in the random-order streaming model must use an* $\Omega(n/k^2)$*-size buffer. In fact, this hardness result holds against stronger algorithms that are not captured by the standard random-order streaming model for submodular maximization (see Remark 9).*

This hardness result is quite surprising because it shows in contrast to monotone submodular maximization, non-monotone submodular maximization in the random-order setting is *not any easier than* that in the worst-order setting where the elements arrive in the worst-case order – for worst-order streaming model, it is known that $\Omega(n/k^3)$-size buffer is required to beat 1/2-approximation even if the submodular function is monotone [13], but for random-order model, recent work by [1] gives a $(1-1/e-\varepsilon)$-approximate algorithm using $O(k/2^{\mathrm{poly}(\varepsilon)})$-size buffer, which is improved to $O(k/\varepsilon)$ by a simpler algorithm of [20].

Furthermore, notice that our algorithmic result and hardness result together exhibit a separation between symmetric and asymmetric submodular functions in the random-order streaming setting. This separation is interesting because in the literature, tight hardness result for general non-monotone functions often continues to hold for symmetric functions. For example, for unconstrained non-monotone submodular maximization, there is a family of symmetric functions for which $(1/2 + \varepsilon)$-approximation requires $2^{\Omega(n)}$ queries [10, 28], and there are efficient matching 1/2-approximation algorithms even for asymmetric functions [6].

[5] The 1/2-approximation of Algorithm 1 for general non-monotone functions is not interesting by itself – 1/2-approximation was achieved even if the elements arrive in the worst-case order [2]. However, Algorithm 1, which takes advantage of the random order, led us to the discovery of the 1/2-hardness in the random-order setting.

[6] These algorithmic results also hold for the (similar but incomparable) secretary with shortlists model [1].

To the best of our knowledge, our result is the *first provable separation of symmetric and asymmetric submodular maximization in any setting*, let alone a natural setting that gains a lot of interests recently. Although admittedly we would be more excited to see such separation in the more classic offline setting of constrained non-monotone submodular maximization, currently we are still far from figuring out whether there is a separation in this setting (because of the gap between the current best 0.432-approximation algorithm for symmetric functions and the current best 0.491-hardness for general asymmetric functions we mentioned earlier), and we hope our result can provide some insights on how to resolve this problem.

Future directions

Our FPT algorithm achieves significantly better-than-1/2 approximation for general non-monotone submodular functions in the offline setting, and its subrountine Algorithm 1 breaks the 1/2 hardness for symmetric submodular functions in the random-order streaming setting – but what is the best possible approximation ratio in those respective settings? We leave this as an open problem for future work.

We remark that no FPT (small-buffer resp.) algorithm can break the $1 - 1/e$ barrier for symmetric functions in the offline (random-order streaming resp.) setting. For asymmetric functions, this follows from the classic work of [24] for monotone submodular functions which we mentioned earlier. For symmetric submodular maximization, the monotone functions exhibiting $(1-1/e)$-hardness are obviously not symmetric; but in appendix of the full version, we are able to give a simple black-box approximation-preserving reduction from symmetric non-monotone to asymmetric monotone submodular function maximization that works in both the offline and random-order streaming settings:

▶ **Proposition 4.** *For cardinality-constrained symmetric submodular function maximization, any algorithm guaranteeing a $(1 - 1/e + \varepsilon)$-approximation must:*
Offline *use $n^{\Omega(k)}$ queries; or*
Random-order streaming *use $\Omega(n)$-buffer size.*

1.1 Additional related work

FPT submodular optimization

The study of parameterized complexity of submodular maximization was initiated by [27] who focused on monotone submodular functions. [27] gives an FPT approximation scheme for monotone submodular functions that are either p-separable or have a bounded ratio of total singleton contribution ($\sum_{e \in E} f(\{e\})$) to total value ($f(E)$). However, for general monotone submodular functions even FPT algorithms (in terms of query complexity) cannot break the classic $1 - 1/e$ barrier [24]. Furthermore, even for the special case of max-k-cover, no FPT algorithms can beat $1 - 1/e$ assuming gap-ETH [8, 21].

Streaming submodular optimization

Our work is related to recent works on maximizing submodular functions in random order streams [1, 20, 26], but all the latter focus on monotone functions. Submodular optimization in worst-order streaming models has also been extensively studied in recent years, e.g. [4, 7, 19, 9, 12, 23, 3, 17, 18, 22, 2, 15, 16, 13]. In the worst-order literature, most relevant to our work is [2] who gave a 1/2-approximation for general (non-monotone) submodular functions, and [13] who proved a matching inapproximability.

2 Preliminaries

▶ **Definition 5.** *Given a ground set of elements E, a function $f : 2^E \to \mathbb{R}_{\geq 0}$ is submodular if for all $S \subseteq T \subseteq E$ and $i \in E \setminus T$, $f(S \cup \{i\}) - f(S) \geq f(T \cup \{i\}) - f(T)$. Moreover, we denote the marginal gain by $f(X|S) := f(X \cup S) - f(S)$.*

▶ **Definition 6.** *A function $f : 2^E \to \mathbb{R}_{\geq 0}$ is symmetric if for all $S \subseteq E$, $f(S) = f(E \setminus S)$.*

In this paper, we **always consider maximizing non-negative submodular functions** over n elements under a cardinality constraint k, i.e., $\max_{X \in E, |X| \leq k} f(X)$. The following lemma [10] for non-negative submodular functions will be useful.

▶ **Lemma 7.** *Let $f : 2^E \to \mathbb{R}_{\geq 0}$ be a submodular function. Further, let R be a random subset of $T \subseteq E$ in which every element occurs with probability at least p (not necessarily independently). Then, $\mathbf{E}[f(R)] \geq pf(T) + (1-p)f(\emptyset)$.*

Moreover, we are interested in the fixed-parameter tractable algorithms.

▶ **Definition 8.** *For submodular maximization over n elements with cardinality constraint k, we say an algorithm is fixed-parameter tractable (FPT) if it has runtime $h(k) \cdot poly(n)$, where h can be any finite function.*

Besides, we are also interested in studying low-memory algorithms for submodular maximization in the random-order streaming model, and in this setting, we only care about the memory cost but not the runtime. We follow the standard setup of streaming model for submodular maximization in the literature (see e.g., the model in the original work [4, Section 3] and more recent works [2, 16, 20]), and the only additional assumption we make is that the elements arrive in uniformly random order (which was studied in e.g., [1, 20]).

Random-order streaming model

In the random-order streaming model, an algorithm is given a single pass over a dataset in a streaming fashion, where the stream is a uniformly random permutation of the input dataset and each element is seen once. The algorithm is allowed to store the elements or any information in a memory buffer with certain size. To be precise, at any point during the runtime of the algorithm,

$$\text{memory cost} = \text{number of stored elements} + \text{number of bits of stored information}.$$

Note that the elements and information are treated separately. One can think of the elements as physical tokens[7], and the algorithm has a limited number of special slots to store the tokens. Besides these special slots, the algorithm has other limited space to store arbitrary information. The total memory cost should not exceed the algorithm's memory size.

Every time when a new element arrives, the algorithm can decide how to update its memory buffer, i.e., whether to store the new element or remove other elements in its memory, and what information to add or remove. At any time, the algorithm can make any number

[7] The standard streaming model for submodular maximization assumes the elements are stored like physical tokens rather than using arbitrary encoding, because the model eventually wants to restrict the algorithm's access to the value oracle. If we store elements using arbitrary encoding, it is not clear how to restrict oracle access for general submodular functions (although it is possible to define such model for some special applications). There is another model that allows elements to be stored in arbitrary encoding [13, Appendix B] - this model does not restrict oracle access at all, but instead it assumes that the elements appearing in the stream are a small part of the ground set.

of queries to the value oracle of the objective submodular function, but it is only allowed to query the value of any subset of the elements that are stored in its memory. For example, at some point during the stream, suppose the algorithm stores an element e, and it makes a query of the value of set $\{e\}$ and writes the result of the query in its memory in any format it prefers (e.g., "the value of $\{e\}$ is..."), and then it removes element e. After removing e, it will never be allowed to query the value of any set that contains e in the future, but it can still keep the information "the value of $\{e\}$ is...", which it wrote before, in its memory, as long as it wants.

At the end of the stream, the algorithm outputs a subset of elements that are stored in its memory as the solution set.

▶ Remark 9. Our streaming algorithm falls into the above model and has low memory cost (specifically, $\tilde{O}(k^2)$). Our hardness result in fact holds against stronger algorithms that are allowed to (i) store infinite bits of information (i.e., only the number of stored elements counts as memory cost) and (ii) output any size-($\leq k$) subset of elements as the solution set (i.e., during the stream, the algorithm is still only allowed to query any subset of elements stored in its memory, but at the end of the stream, it can output any size-($\leq k$) subset of the ground set as it wants[8]).

Finally, we note that all the missing proofs of this extended abstract can be found in the full version of our paper on arXiv (the URL is provided on the title page).

3 The core algorithm

In this section, we present the core algorithm of this work (Algorithm 1), which is actually our streaming algorithm. Our FPT algorithm will use this core algorithm as a subroutine. The goal of this section is to establish the common setup for the analysis of our FPT algorithm for general submodular functions and the analysis of the streaming algorithm for symmetric submodular functions. In this process, we will also do a warm-up that shows a 1/2-approximation for the core algorithm on general submodular functions.

Algorithm 1 SymmetricStream(f, E, k, ε).

Partition the first ε fraction of the random stream E into windows $w_1, \ldots, w_{3k}$ of equal size.
$S_0 \leftarrow \emptyset$
$H \leftarrow \emptyset$
for $i \leftarrow 1$ to $3k$ **do**
    $e_i \leftarrow \arg\max_{e \in w_i} f(e|S_{i-1})$
    $S_i \leftarrow S_{i-1} \cup \{e_i\}$
for $e \in E \setminus \{w_1, w_2, \ldots, w_{3k}\}$ **do**
    for $i = 1, 2, \ldots, 3k$ **do**
        if $f(e|S_{i-1}) > f(e_i|S_{i-1})$ and $|H| < 18k^2 \log k/\varepsilon$ **then**
            $H \leftarrow H \cup \{e\}$
return $\arg\max_{X \subseteq S_{3k} \cup H, |X| \leq k} f(X)$

[8] I.e., it can output something like "My solution set is {1,3,11,...}" even if elements $1, 3, 11$ are not in its memory.

At a high level, Algorithm 1 divides the first ε fraction of the stream into $3k$ windows[9] and greedily selects the element e_i with the best marginal gain in each window i. (Although we use the notation S_i in the pseudocode for clarity, we only need to keep track of one ordered solution set in the first for loop.) Then it freezes the solution set, and for the rest of the stream, it only selects the first $18k^2 \log k/\varepsilon$ elements that have better marginal gain than e_i conditioned on the $(i-1)$-th partial solution for some $0 \leq i \leq 3k-1$. Finally, it finds the best size-k solution from all the selected elements by brute force. Due to line 9, the memory usage is $O(k^2 \log k/\varepsilon)$. The runtime before the final brute-force search is $O(nk)$ because of the for loops, and the brute-force search takes time $k\binom{|S_{3k} \cup H|}{k} = 2^{\widetilde{O}(k)}$ as $|S_{3k}| = 3k$ and $|H| = O(k^2 \log k/\varepsilon)$, and hence, the total runtime is $O(nk) + 2^{\widetilde{O}(k)}$ (which is not polynomial in k, but in this paper, we focus on the memory bound for streaming setting and FPT algorithms for offline setting).

3.1 Warm-up

As a warm-up, we show the 1/2-approximation of our core algorithm for general submodular functions, which also helps set up the proof of our main algorithmic results. Before getting to the technical proof, we provide the intuition for Algorithm 1 in the following. First, we want to make sure that with high probability $|H|$ never meets the size threshold in the if condition at line 9, and thus the size threshold essentially does not affect our analysis. This follows by a standard argument (see Lemma 11).

Because the stream is in random order, most optimal elements will be visited during the for loop at line 7. A part of them O_H will be picked by the algorithm, and the other part O_L will not be selected. Consider the set $S_{|O_L|}$ defined in the algorithm. Because of the if condition at line 9, the elements in $S_{|O_L|}$ have better marginal contribution than O_L, and we can show that $f(S_{|O_L|}) \geq f(O_L|S_{|O_L|})$, which is somewhat similar to the classic greedy algorithm for monotone submodular maximization. Since $O_H \cup S_{|O_L|}$ and $S_{|O_L|}$ are two candidate solutions under the radar of the algorithm's final brute-force search, the algorithm achieves at least

$$\frac{f(O_H \cup S_{|O_L|}) + f(S_{|O_L|})}{2} \geq \frac{f(O_H \cup S_{|O_L|}) + f(O_L|S_{|O_L|})}{2} \geq \frac{f(O_H \cup O_L \cup S_{|O_L|})}{2}, \quad (1)$$

where the last inequality is by submodularity.

To complete the analysis, we observe that $f(O_H \cup O_L \cup S_{|O_L|})$ is not significantly worse than $f(O_H \cup O_L)$, and hence 1/2-approximation follows from (1). Indeed, because $S_{|O_L|}$ is chosen from a random ε fraction of E, it cannot hurt $O_H \cup O_L$ significantly – otherwise, there should be many other elements similar to $S_{|O_L|}$, and together they would hurt $O_H \cup O_L$ so much that would eventually contradict non-negativity. This is formally shown in Lemma 10. Using Lemma 10 and Lemma 11, we can prove the 1/2-approximation. The proof of Lemma 10 and Lemma 11 can be found in the full version.

▶ **Lemma 10.** *Let O denote the optimal size-k solution. Then, for any constants $\varepsilon \in (0, 1]$ and $\varepsilon' > 0$,*

- *with probability at least $1 - \varepsilon/\varepsilon'$, there does not exist a set Y of elements in the first ε fraction of the stream such that $f(Y|O) \leq -\varepsilon' f(O)$,*
- *and moreover, with probability at least $1 - 3\varepsilon/(\varepsilon')^2$, for all $\ell \in \{\varepsilon' k, 2\varepsilon' k, \ldots, k\}$, for any $S \subseteq S_{3\ell} \setminus S_{2\ell}$, $f(S|O \cup S_{2\ell}) \geq -\varepsilon' f(O)$.*

[9] The choice of $3k$ suffices to beat 1/2 approximation for symmetric submodular function, but it is conceivable that a larger budget might improve the final constant. For our FPT algorithm, dividing into k windows would also work, but that does not improve the runtime asymptotically.

▶ **Lemma 11.** *With probability* $1 - 3/k$, $|H| < 18k^2 \log k/\varepsilon$, *namely, Algorithm 1 never ignores any elements due to the memory threshold (line 9).*

▶ **Theorem 12.** *Algorithm 1 achieves* $(1/2 - O(\sqrt{\varepsilon}) - O(1/k))$*-approximation for non-negative non-monotone submodular functions in the random-order streaming model.*

Proof. Let O be the size-k optimal set, and let $O' \subseteq O$ be the subset of optimal elements that appear in the last $1 - \varepsilon$ fraction of the stream. We partition O' into O_L and O_H, where O_L are the optimal elements that are not selected by the algorithm, and O_H are those selected. Let S_L be short for solution $S_{|O_L|}$ in the algorithm. If the good event in the first bullet point of Lemma 10 with $\varepsilon' = \sqrt{\varepsilon}$ happens (which we denote by A_1), then $f(S_L|O_L \cup O_H) \geq f(S_L|O) \geq -\sqrt{\varepsilon} f(O)$, where the first inequality is by submodularity. If A_1 happens, we have that

$$\begin{aligned} f(O_H \cup S_L) + f(O_L|S_L) &= f(O_H|S_L) + f(S_L \cup O_L) \\ &\geq f(O_H|S_L \cup O_L) + f(S_L \cup O_L) \quad \text{(By submodularity)} \\ &= f(O_L \cup O_H \cup S_L) \\ &= f(O_L \cup O_H) + f(S_L|O_L \cup O_H) \\ &\geq f(O_L \cup O_H) - \sqrt{\varepsilon} f(O). \end{aligned} \tag{2}$$

Let $O_L = \{o_1, o_2, \ldots, o_{|L|}\}$. If the good event in Lemma 11, denoted by A_2, happens, then, we have the following simple observation by design of the algorithm.

▶ **Observation 13.** *If* A_2 *happens, then for any* $o \in O_L$, $f(o|S_{i-1}) \leq f(e_i|S_{i-1})$, *for all* $i \in [3k]$, *because* o *is skipped by the algorithm.*

Given A_2, using the above observation, we have that

$$\begin{aligned} f(O_L|S_L) &\leq \sum_{o \in O_L} f(o|S_L) && \text{(By submodularity)} \\ &\leq \sum_{i=1}^{|O_L|} f(o_i|S_{i-1}) && \text{(By submodularity)} \\ &\leq \sum_{i=1}^{|O_L|} f(e_i|S_{i-1}) && \text{(By Observation 13)} \\ &= f(S_L). && \text{(By telescoping sum)} \end{aligned} \tag{3}$$

Combining Eq. (2) and (3), we get

$$f(O_H \cup S_L) + f(S_L) \geq f(O_L \cup O_H) - \sqrt{\varepsilon} f(O). \tag{4}$$

By Lemma 7, $\mathbf{E}[f(O')] \geq (1 - \varepsilon) f(O)$. By a Markov argument,

$$\mathbf{E}[f(O') \mid A_1, A_2] \geq (1 - \varepsilon - \sqrt{\varepsilon} - 3/k) f(O), \tag{5}$$

and hence, by taking expectation for both sides of Eq. (4), we have that

$$\begin{aligned} \mathbf{E}[f(O_H \cup S_L) + f(S_L) \mid A_1, A_2] &\geq \mathbf{E}[f(O_L \cup O_H) \mid A_1, A_2] - \sqrt{\varepsilon} f(O) \\ &\geq (1 - \varepsilon - 2\sqrt{\varepsilon} - 3/k) f(O). \end{aligned}$$

Finally, $\mathbf{E}[f(O_H \cup S_L) + f(S_L)] \geq \Pr(A_1, A_2) \cdot \mathbf{E}[f(O_H \cup S_L) + f(S_L) \mid A_1, A_2] \geq (1 - \sqrt{\varepsilon} - 3/k)(1 - \varepsilon - 2\sqrt{\varepsilon} - 3/k) f(O) = (1 - O(\sqrt{\varepsilon} + 1/k)) f(O)$. The proof finishes because $O_H \cup S_L$ and S_L are both subsets of $S_{3k} \cup H$, so one of them must achieve at least half of $(1 - O(\sqrt{\varepsilon} + 1/k)) f(O)$. ◀

3.2 Our plan for the algorithmic results

The proof of our main algorithmic results (Theorem 17, Theorem 14 and Theorem 15) is based on factor-revealing convex programs[10]. We know that factor-revealing programs are not intuitive and hence not easy to understand, although they are effective tools for formalizing the proof. Therefore, in the future sections of this extended abstract, instead of formally proving the main results, we will provide the intuition and interpretable (but less formal) analysis for our algorithmic results. **We recommend the readers read the intuition and informal interpretable analysis in this extended abstract, and then check the formal proofs that involve factor-revealing programs starting from Section 3.2 in the full version.**

For convenience, in the informal interpretable analysis in this extended abstract, we will **continue using the notations** O, O_L, O_H that have appeared in this section.

4 FPT algorithms for non-monotone submodular functions

In this section, building on Algorithm 1, we give FPT algorithms that achieves better-than-1/2 approximation for general non-monotone submodular functions. We first present a basic FPT algorithm that achieves 0.512-approximation to show the main ideas, then we discuss how to improve the basic algorithm to get 0.539-approximation.

4.1 The basic FPT algorithm

Essentially, after running Algorithm 1, our basic FPT algorithm (Algorithm 2) searches for O_H by brute force, and then starting with O_H as the initial solution set, it runs classic greedy algorithm to construct a size-k solution set, and it repeats this step many times without replacement, i.e., each time the elements selected by greedy algorithm are removed from ground set, and finally it returns the best size-k solution set among all the repetitions. The pseudocode is given in Algorithm 2.

Algorithm 2 FPT$(f, E, k, \varepsilon, T)$.

Initialize an empty set X^*.
Run Algorithm 1 on the input[11] (f, E, k, ε) and keep S_{3k} and the final version of H in Algorithm 1.
for each size-$(\leq k)$ subset $O_H^{\text{guess}} \subseteq H$ **do**
    Initialize an empty set of elements I.
    for $i = 1, 2, \ldots, T$ **do**
        Run greedy algorithm with O_H^{guess} as the initial solution set[12] to build a size-k solution set $O_H^{\text{guess}} \cup X_i$. Add X_i to I and remove X_i from E.
    Let $X' = \arg\max_{X \subseteq S_{3k} \cup H \cup I, |X| \leq k} f(X)$ and let $X^* = X'$ if $f(X') > f(X^*)$.
    Add I back to E.
return X^*

[10] The certificates for these convex programs can be found in the full version.
[11] Randomly permute E if it is not in random order.
[12] Specifically, the greedy algorithm starts with solution set $X = O_H^{\text{guess}}$ and runs in $k - |X|$ iterations. In each iteration, it selects the element e in E that maximizes $f(e|X)$ and add e to X. (**We assume without loss of generality** that the maximal $f(e|X)$ is always non-negative. Otherwise, we can add dummy elements to E.)

Algorithm 2 runs in fixed-parameter polynomial time. Indeed, because $|H| = \widetilde{O}(k^2)$ by design of Algorithm 1, the outer loop has less than $k\binom{\widetilde{O}(k^2)}{k} = 2^{\widetilde{O}(k)}$ iterations, and for the inner loop, we will only need T to be an arbitrarily large constant. Greedy algorithm runs in $O(kn)$ time. Moreover, since $|I| \leq Tk$ and $|S_{3k} \cup H| = \widetilde{O}(k^2)$ with high probability, the brute-force step (Line 7) in Algorithm 2 takes time $k\binom{\widetilde{O}(k^2)}{k} = 2^{\widetilde{O}(k)}$. Furthermore, the runtime of Algorithm 1 excluding the exhaustive search in its last step is polynomial. Hence, the total runtime is $n \cdot 2^{\widetilde{O}(k)}$.

Instead of formally proving the approximation ratio for Algorithm 2 (which we will do in the full version), we give an interpretable analysis for the better-than-1/2 approximation. At very high level, the intuition is that if Algorithm 1 only gets 1/2 approximation, then it must be the case that $f(O_H) = \frac{f(O_H \cup O_L)}{2}$. Now we consider the candidate solution $O_H \cup X_i$ for $i \in \{1, 2, 3\}$, and we can argue that if $f(X_i|O_H) = 0$, i.e., the candidate solution does not beat 1/2, then X_i must hurt $O_H \cup O_L$ a lot. Moreover, By submodularity, $X_1 \cup X_2 \cup X_3$ hurts $O_H \cup O_L$ by at least the sum of how much each X_i ($i \in \{1, 2, 3\}$) hurts. Together, we show that this would contradict non-negativity of the function. Now we explain this intuition in more details.

Informal interpretable analysis

The starting point is the analysis of Theorem 12. We can show that for the instance to be hard, in the sense that Algorithm 1 is only able to get 1/2 approximation, then it requires $f(O_H) = \frac{f(O_H \cup O_L)}{2}$ (this will be explained with more details in the interpretable analysis provided before Theorem 17, but for now, let us take this as given). Because O_H is selected by Algorithm 1, in the outer iteration when Algorithm 2 guesses O_H correctly, it runs classic greedy algorithm many times based on O_H without replacement. Consider the set X_1 selected in the first run of greedy algorithm. By standard analysis of greedy algorithm, we can derive that $f(X_1|O_H) \geq f(O_L|O_H \cup X_1)$. If $f(X_1|O_H) = 0$ (otherwise $X_1 \cup O_H$ beats 1/2), then $f(O_L|O_H \cup X_1) \leq 0$, which implies $f(O_L \cup O_H \cup X_1) \leq f(O_H \cup X_1) = f(O_H) = \frac{f(O_H \cup O_L)}{2}$. Hence $f(X_1|O_L \cup O_H) \leq -\frac{f(O_H \cup O_L)}{2}$. WLOG, the first run of greedy did not select most of O_L, because otherwise $f(X_1|O_H)$ should be significantly large. Therefore, similarly, we can derive that if $f(X_2|O_H) = 0$, where X_2 is selected in the second run of greedy, then $f(X_2|O_L \cup O_H) \leq -\frac{f(O_H \cup O_L)}{2}$. By submodularity, $f(X_1 \cup X_2|O_L \cup O_H) \leq f(X_1|O_L \cup O_H) + f(X_2|O_L \cup O_H) \leq -f(O_H \cup O_L)$. Notice that this implies $f(X_1 \cup X_2 \cup O_L \cup O_H) \leq 0$. Hence, the third run of greedy algorithm must obtain very large $f(X_3|O_H)$ (and hence $X_3 \cup O_H$ beats 1/2), because otherwise we can repeat above argument and show that $f(X_1 \cup X_2 \cup X_3 \cup O_L \cup O_H) < 0$, which violates non-negativity of the function f. Furthermore, by running greedy many times, we are able to extract even more value from O_L, which is formally formulated by the factor-revealing programs in the proof in the full version.

▶ **Theorem 14.** *For sufficiently large constant T and sufficiently small constant ε, Algorithm 2 achieves* 0.512*-approximation for non-negative non-monotone submodular maximization with a cardinality constraint.*

4.2 Improved FPT algorithm

Algorithm 2 can be improved to achieve better approximation ratio. In this subsection, we present the improved algorithm, the pseudocode of which is given in Algorithm 4. In the full version, we will provide the intuition behind this algorithm (which involves the formal proof of Theorem 14) and the formal proof for the improved approximation factor.

Algorithm 3 RECURSIVE(f, E, k, I, t, T).

Initialize empty sets I' and X^*.
for each size-($\leq k$) subset $O_t^{\text{guess}} \subseteq I$ **do**
  for $i = 1, 2, \ldots, T$ **do**
    Run greedy algorithm on E with O_t^{guess} as the initial solution set to build a size-k solution set $O_t^{\text{guess}} \cup X_i$.
    Add X_i to I' and remove X_i from E.
  Add I' back to E.
  if $t \leq T$ **then**
    Run Algorithm 3 on input $(f, E, k, I \cup I', t+1, T)$, which returns solution set X'.
    Let $X^* = X'$ if $f(X') > f(X^*)$.
  else
    Let $X^* = \arg\max_{X \subseteq I, |X| \leq k} f(X)$.
return X^*

Algorithm 4 FPT$^+$(f, E, k, ε, T).

Initialize an empty set of elements I.
Run Algorithm 1 on input (f, E, k, ε) and keep S_{3k} and the final version of H in Algorithm 1.
Run Algorithm 3 on input $(f, E, k, S_{3k} \cup H, 1, T)$, which returns X^*.
return X^*

▶ **Theorem 15.** *For sufficiently large constant T and sufficiently small constant ε, Algorithm 4 achieves 0.539-approximation for non-negative non-monotone submodular maximization with a cardinality constraint.*

5 $(1/2 + c)$-approximation for random-order streaming symmetric submodular maximization

In this section, we show that Algorithm 1 *beats* 1/2-approximation for symmetric non-monotone submodular functions using $\widetilde{O}(k^2)$ memory. Together with our lower bound result (Theorem 18), this separates the symmetric non-monotone submodular functions from general non-monotone submodular functions in the random-order streaming model. To our best knowledge, this is first such separation.

The following lemma is the key feature of symmetric submodular functions which we will take advantage of, and it basically says for symmetric submodular function, a set can not hurt another set by more than its own value.

▶ **Lemma 16.** *For any non-negative symmetric submodular function $f : V \to \mathbb{R}_{\geq 0}$, for any disjoint $X, Y \subseteq V$, $f(X|Y) \geq -f(X)$.*

Proof. By submodularity, $f(X|Y) \geq f(X|V \setminus X) = f(V) - f(V \setminus X)$, and by symmetry and non-negativity, $f(V) - f(V \setminus X) = f(\emptyset) - f(X) \geq -f(X)$. ◀

Instead of showing the technical proof of the better-than-1/2 approximation (which is provided in the full version), we give the interpretable analysis for why Algorithm 1 can beat 1/2 for symmetric submodular functions. The interpretable analysis is still a little lengthy and technical. At very high level, the idea is if none of $S_{|O_L|}$ and $O_H \cup S_{|O_L|}$ and

$O_H \cup (S_{2|O_L|} \setminus S_{|O_L|})$ beats 1/2, then we can show that (i) $f(O_H) = f(O_L) = \frac{f(O_H \cup O_L)}{2}$, (ii) $f(O_H|S_{2|O_L|}) = -f(O_H)$, and (iii) $f(S_{3|O_L|} \setminus S_{2|O_L|}|S_{2|O_L|}) \geq (1 - 1/e)f(O_L)$. The punchline is that using (ii), we can further show that (iv) $f(S_{3|O_L|} \setminus S_{2|O_L|}|O_H) \geq f(S_{3|O_L|} \setminus S_{2|O_L|}|S_{2|O_L|})$ (basically, the argument is if O_H hurts $S_{2|O_L|}$ a lot, then by Lemma 16, which is due to symmetry, one can argue O_H can not hurt $S_{3|O_L|}$ more than how much it hurts $S_{2|O_L|}$). Therefore, by (i) and (iii) and (iv), we know that $O_H \cup (S_{3|O_L|} \setminus S_{2|O_L|})$ beats 1/2 approximation. Now we explain this idea in more details.

Informal interpretable analysis

The starting point is the analysis for Theorem 12. The reader can first review the intuition given in the beginning of the subsection of Theorem 12. There we argued that the algorithm achieves half of $f(O_H \cup S_{|O_L|}) + f(S_{|O_L|}) \geq f(O_H \cup O_L)$, and hence for the instance to be hard (in the sense that the algorithm only gets 1/2 approximation), it requires $f(O_H \cup S_{|O_L|}) = f(S_{|O_L|}) = \frac{f(O_H \cup O_L)}{2}$. This implies $f(O_H|S_{|O_L|}) = 0$, and by submodularity $f(O_H|O_L \cup S_{|O_L|}) \leq 0$, and hence $f(O_H \cup O_L \cup S_{|O_L|}) \leq f(O_L \cup S_{|O_L|})$. Recall that we argued $S_{|O_L|}$ can not hurt $O_H \cup O_L$ significantly, and thus, $f(O_H \cup O_L) \leq f(O_H \cup O_L \cup S_{|O_L|}) \leq f(O_L \cup S_{|O_L|})$, but since $f(S_{|O_L|}) = \frac{f(O_H \cup O_L)}{2}$, we have that $f(O_L|S_{|O_L|}) = f(S_{|O_L|})$, which implies[13] $f(S_{|O_L|}) \geq f(O_L)$. Moreover, since $f(S_{|O_L|}) = \frac{f(O_H \cup O_L)}{2}$, we have $f(O_L) \leq \frac{f(O_H \cup O_L)}{2}$ and hence $f(O_H) \geq \frac{f(O_H \cup O_L)}{2}$, but we also have $f(O_H) \leq \frac{f(O_H \cup O_L)}{2}$ because otherwise $O_H \cup S_{|O_L|}$ should have beaten 1/2-approximation as $S_{|O_L|}$ does not hurt O_H significantly, and therefore it holds that $f(O_H) = f(O_L) = \frac{f(O_H \cup O_L)}{2}$.

Now consider the set $S_{2|O_L|} \setminus S_{|O_L|}$. If $f(S_{2|O_L|} \setminus S_{|O_L|}|O_H) = 0$ (otherwise $S_{2|O_L|} \setminus S_{|O_L|} \cup O_H$ beats 1/2), then by submodularity and the fact that $S_{2|O_L|} \setminus S_{|O_L|}$ does not hurt anything significantly (which is yet another application of Lemma 10), we have $f(S_{2|O_L|} \setminus S_{|O_L|}|O_H \cup S_{|O_L|}) = 0$ and hence $f(O_H \cup S_{2|O_L|}) = f(O_H \cup S_{|O_L|}) = \frac{f(O_H \cup O_L)}{2}$. Notice that $f(O_L|O_H \cup S_{2|O_L|}) = f(O_L \cup O_H \cup S_{2|O_L|}) - f(O_H \cup S_{2|O_L|}) \geq f(O_L \cup O_H) - f(O_H \cup S_{2|O_L|}) = \frac{f(O_H \cup O_L)}{2}$, where the inequality is again due to the fact that $S_{2|O_L|}$ does not hurt. Therefore, similar to how we argued $f(S_{|O_L|}) \geq f(O_L)$, we can show that $f(S_{2|O_L|} \setminus S_{|O_L|}|S_{|O_L|}) \geq f(O_L)$. Since $f(S_{2|O_L|}) \geq f(S_{2|O_L|} \setminus S_{|O_L|}|S_{|O_L|}) + f(S_{|O_L|}) \geq 2f(O_L) = 2f(O_H)$ and $f(O_H \cup S_{2|O_L|}) = \frac{f(O_H \cup O_L)}{2} = f(O_H)$, we have that $f(O_H|S_{2|O_L|}) \leq -f(O_H)$, and together with Lemma 16, we have that $f(O_H|S_{2|O_L|}) = -f(O_H)$.

Here comes the final punchline – If $S_{3|O_L|}/S_{2|O_L|}$ has significant marginal contribution to $S_{2|O_L|}$, then $S_{3|O_L|}/S_{2|O_L|}$ must have at least the same marginal contribution to O_H (and therefore, $O_H \cup S_{3|O_L|}/S_{2|O_L|}$ will beat 1/2). Specifically, this follows from

$$
\begin{aligned}
f(S_{3|O_L|}/S_{2|O_L|}|O_H) &\geq f(S_{3|O_L|}/S_{2|O_L|}|O_H \cup S_{2|O_L|}) \\
&\qquad \text{(By submodularity)} \\
&= f(O_H|S_{3|O_L|}) - f(O_H|S_{2|O_L|}) + f(S_{3|O_L|}/S_{2|O_L|}|S_{2|O_L|}) \\
&\geq -f(O_H) - f(O_H|S_{2|O_L|}) + f(S_{3|O_L|}/S_{2|O_L|}|S_{2|O_L|}) \\
&\qquad \text{(By Lemma 16)} \\
&= f(S_{3|O_L|}/S_{2|O_L|}|S_{2|O_L|}) \\
&\qquad \text{(By } f(O_H|S_{2|O_L|}) = -f(O_H)\text{)}
\end{aligned}
$$

[13] Intuitively, by the if condition at line 9 of Algorithm 1, we can show that the marginal contribution of each iterate of $S_{|O_L|}$ is at least $\frac{f(O_L|S_{|O_L|})}{|O_L|}$, but because $f(O_L|S_{|O_L|}) = f(S_{|O_L|})$, each iterate actually makes the same marginal contribution. Notice that the first iterate should make contribution more than any element in O_L by the if condition.

(One can check the first equality is true by expanding both sides of the equality.) It remains to show $f(S_{3|O_L|}/S_{2|O_L|}|S_{2|O_L|})$ is indeed significantly large. This is essentially due to $f(O_L|O_H \cup S_{2|O_L|}) \geq \frac{f(O_H \cup O_L)}{2}$, which we argued earlier, and the if condition at line 9. In particular, we can show $f(S_{3|O_L|}/S_{2|O_L|}|S_{2|O_L|})$ is at least $1-1/e$ fraction of $f(O_L|S_{2|O_L|})$ by the standard analysis of the classic greedy algorithm for monotone submodular maximization.

In the full version, we will formally prove the better-than-1/2 approximation of Algorithm 1 using the factor-revealing programs.

▶ **Theorem 17.** *For sufficiently small ε and large k, Algorithm 1 achieves strictly better-than-1/2 approximation for non-negative non-monotone symmetric submodular functions in the random-order streaming model.*

On a side note, the constant we get here is by no means tight. (Indeed, we have an improvement, which may also improve the constant for our FPT algorithm, but it requires numerically solving non-convex programs rather than the convex programs in our proofs.) What is interesting is the separation between symmetric and general submodular functions in the random-order streaming model. Also, it is tempting to conjecture that Algorithm 1 achieves optimal $1-1/e$ approximation for monotone submodular functions, given its success in the non-monotone regime. Nonetheless, we have a hard instance that refutes this conjecture. The details would be made available to the interested reader upon request.

6 Tight 1/2 hardness for random-order streaming non-monotone submodular maximization

In this section, we present the lower bound result for non-monotone submodular maximization in the random-order streaming model (described in Section 2). The approximation factor in the lower bound result is tight because of the upper bound in Theorem 12 for example.

▶ **Theorem 18.** *Assuming $n = 2^{o(k)}$, any $(1/2+\varepsilon)$-approximation algorithm for non-monotone submodular maximization in the random-order streaming model must use $\Omega(n/k^2)$ memory. In fact, this hardness result holds against even stronger algorithms (see Remark 9) that are beyond the scope of the standard random-order streaming model for submodular maximization.*

The formal proof is provided in the full version. Here we give the construction of the hard instance and explain the main idea.

Construction of the hard instance

The function f we construct here is essentially a cut function on an unweighted bipartite directed hypergraph[14] plus a modular function. The ground set $V := A_1 \cup A_2$ for f is the set of n vertices of the graph, where A_1 and A_2 denote the two parts respectively. Specifically, $A_2 := \{u_1, \ldots, u_{\varepsilon k}\}$, and A_1 is partitioned into $\ell := (n-\varepsilon k)/b$ buckets of vertices $B_1, \ldots, B_\ell$, each of size $b := k - \varepsilon k$. Now we describe a random generating procedure that generates the hyperedges in the graph:

1. First, for each $i \in [\ell]$, we sample a random subset of vertices $N_i \subset A_2$ of size $|N_i| = \varepsilon^2 k$, and for each $u_j \in N_i$, we create a directed hyperedge from B_i to u_j.

[14] A directed hyperedge in a directed hypergraph is represented by some (U, v), where U is a subset of vertices, and $v \notin U$ is a vertex. For any subset of vertices S, a hyperedge (U, v) is cut by S iff $|U \cap S| > 0$ and $v \notin S$. It is well-known that such cut function is submodular.

2. Then, we slightly modify the graph generated in step 1 as follows: We sample a uniformly random $g \in [\ell]$. For each $u_j \in N_g$, we remove the hyperedge from B_g to u_j, and instead, for each $v \in B_g$, we create a directed hyperedge from $\{v\}$ to u_j. (That is, for each $u_j \in N_g$, we replace the hyperedge from B_g to u_j with individual edges from each $v \in B_g$ to u_j.)

The final submodular function $f : V \to \mathbb{R}_{\geq 0}$ is the sum of the cut function on the above generated hypergraph plus the modular function $c(S) := (\varepsilon^2 k^2) \cdot \frac{|S \cap A_2|}{|A_2|}$. See Figure 1 for an illustration.

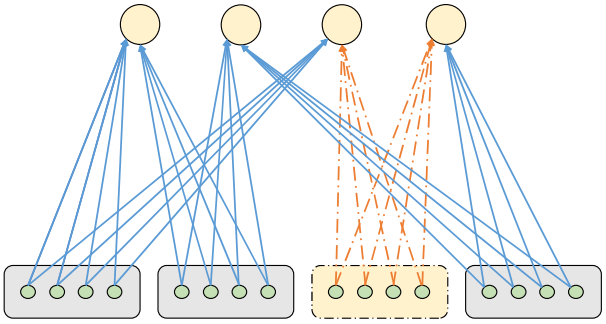

Figure 1 An illustration of our hard instance: At the top we have all εk vertices of A_2, each of which has value εk. At the bottom we have all $(n - \varepsilon k)$ vertices of A_1 that are separated into $\ell = (n - \varepsilon k)/b$ buckets, each of which has $b = k - \varepsilon k$ vertices. We choose a bucket B_g (with yellow filling and dashed outline) uniformly at random. There are εk individual edges (orange and dashed) from each vertex in bucket B_g to B_g's neighborhood N_g, and there are εk hyperedges (blue and solid) from every other bucket B_i (with gray filling and solid outline) to its neighborhood N_i.

Basically, in the above hard instance, we constructed a single good bucket B_g (which is incident to independent individual edges and hence can contribute high value in a cut) and lots of bad buckets B_i (which is incident to hyperedges and hence can only contribute low value in a cut) for all $i \in [\ell] \setminus \{g\}$.

The optimal solution for this hard instance is $B_g \cup (A_2 \setminus N_g)$, i.e., the union of the vertices in the good bucket B_g and the vertices in A_2 except B_g's neighborhood N_g, and specifically, the edges incident to B_g contribute half of the optimal value due to the cut function, and $A_2 \setminus N_g$ contributes the other half due to the modular function c.

However, it is hard for an $o(n/k^2)$-memory algorithm to find out which bucket is the good bucket. Intuitively, to tell whether a bucket B_i is the good bucket, the algorithm needs to query the value of a set that contains at least two vertices of B_i, because otherwise, the value of the set does not depend on whether B_i is incident to individual edges or hyperedges. In order to query the value of a set that contains at least two vertices of B_i, the algorithm has to store at least two vertices of B_i in its memory, because of the restriction in the random-order streaming model (see Section 2). Since the algorithm has low memory, it cannot store two elements for many B_i's at the same time. Using this observation, we can prove w.h.p. the algorithm cannot find the good bucket B_g during the entire stream. Furthermore, we can show by standard concentration inequality that w.h.p. any size-$(\leq k)$ solution set that does not contain enough vertices of B_g has at most half of the optimal value.

Although it was easy to describe the basic idea above, formalizing it requires nontrivial efforts, and we believe that the techniques in our formal proof, which we provide in the full version, are interesting and could be applied for proving hardness of other problems in the random-order streaming setting.

References

1 Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular Secretary Problem with Shortlists. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 1:1–1:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ITCS.2019.1.

2 Naor Alaluf, Alina Ene, Moran Feldman, Huy L Nguyen, and Andrew Suh. Optimal streaming algorithms for submodular maximization with cardinality constraints. In *ICALP*, 2020.

3 Naor Alaluf and Moran Feldman. Making a Sieve Random: Improved Semi-Streaming Algorithm for Submodular Maximization under a Cardinality Constraint. *arXiv:1906.11237 [cs]*, June 2019. arXiv:1906.11237.

4 Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming Submodular Maximization: Massive Data Summarization on the Fly. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 671–680. ACM, 2014. doi:10.1145/2623330.2623637.

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

6 Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. A Tight Linear Time (1/2)-Approximation for Unconstrained Submodular Maximization. *SIAM J. Comput.*, 44(5):1384–1402, 2015. doi:10.1137/130929205.

7 Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 318–330. Springer, 2015. doi:10.1007/978-3-662-47672-7_26.

8 Vincent Cohen-Addad, Anupam Gupta, Amit Kumar, Euiwoong Lee, and Jason Li. Tight FPT approximations for k-median and k-means. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 42:1–42:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.42.

9 Ethan R. Elenberg, Alexandros G. Dimakis, Moran Feldman, and Amin Karbasi. Streaming weak submodularity: Interpreting neural networks on the fly. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4044–4054, 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/c182f930a06317057d31c73bb2fedd4f-Abstract.html.

10 Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing Non-monotone Submodular Functions. *SIAM J. Comput.*, 40(4):1133–1153, 2011. doi:10.1137/090779346.

11 Moran Feldman. Maximizing symmetric submodular functions. *ACM Trans. Algorithms*, 13(3):39:1–39:36, 2017. doi:10.1145/3070685.

12 Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 730–740, 2018. URL: http://papers.nips.cc/paper/7353-do-less-get-more-streaming-submodular-maximization-with-subsampling.

13 Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The One-way Communication Complexity of Submodular Maximization with Applications to Streaming and Robustness. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1363–1374. ACM, 2020. doi:10.1145/3357713.3384286.

14 Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1098–1116. SIAM, 2011.

15 Ran Haba, Ehsan Kazemi, Moran Feldman, and Amin Karbasi. Streaming submodular maximization under a k-set system constraint. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3939–3949. PMLR, 2020. URL: http://proceedings.mlr.press/v119/haba20a.html.

16 Chien-Chung Huang, Naonori Kakimura, Simon Mauras, and Yuichi Yoshida. Approximability of Monotone Submodular Function Maximization under Cardinality and Matroid Constraints in the Streaming Model. *arXiv:2002.05477 [cs]*, February 2020. arXiv:2002.05477.

17 Piotr Indyk and Ali Vakilian. Tight Trade-offs for the Maximum k-Coverage Problem in the General Streaming Model. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 200–217. ACM, 2019. doi:10.1145/3294052.3319691.

18 Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular Streaming in All its Glory: Tight Approximation, Minimum Memory and Low Adaptive Complexity. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3311–3320. PMLR, 2019.

19 Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast Greedy Algorithms in MapReduce and Streaming. *TOPC*, 2(3):14:1–14:22, 2015. doi:10.1145/2809814.

20 Paul Liu, Aviad Rubinstein, Jan Vondrák, and Junyao Zhao. Cardinality constrained submodular maximization for random streams. *Advances in Neural Information Processing Systems*, 34, 2021.

21 Pasin Manurangsi. Tight running time lower bounds for strong inapproximability of maximum k-coverage, unique set cover and related problems (via t-wise agreement testing theorem). In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 62–81. SIAM, 2020. doi:10.1137/1.9781611975994.5.

22 Andrew McGregor and Hoa T. Vu. Better Streaming Algorithms for the Maximum Coverage Problem. *Theory Comput. Syst.*, 63(7):1595–1619, 2019. doi:10.1007/s00224-018-9878-x.

23 Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1379–1386. AAAI Press, 2018. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17014.

24 George L Nemhauser and Laurence A Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of operations research*, 3(3):177–188, 1978.

25 George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An Analysis of Approximations for Maximizing Submodular Set Functions - I. *Math. Program.*, 14(1):265–294, 1978. doi:10.1007/BF01588971.

26 Mohammad Shadravan. Improved submodular secretary problem with shortlists. *CoRR*, abs/2010.01901, 2020. arXiv:2010.01901.

27 Piotr Skowron. FPT approximation schemes for maximizing submodular functions. *Inf. Comput.*, 257:65–78, 2017. doi:10.1016/j.ic.2017.10.002.

28 Jan Vondrák. Symmetry and Approximability of Submodular Maximization Problems. *SIAM J. Comput.*, 42(1):265–304, 2013. doi:10.1137/110832318.