Optimal Streaming Algorithms for Submodular Maximization with Cardinality Constraints

Naor Alaluf
Department of Mathematics and Computer Science, Open University of Israel, Ra'anana, Israel
naoralaluf@gmail.com

Alina Ene
Department of Computer Science, Boston University, MA, USA
aene@bu.edu

Moran Feldman
Department of Computer Science, University of Haifa, Israel
moranfe@cs.haifa.ac.il

Huy L. Nguyen
Khoury College of Computer and Information Science, Northeastern University, Boston, MA, USA
hlnguyen@cs.princeton.edu

Andrew Suh
Department of Computer Science, Boston University, MA, USA
asuh9@bu.edu

Abstract

We study the problem of maximizing a *non-monotone* submodular function subject to a cardinality constraint in the streaming model. Our main contributions are two single-pass (semi-)streaming algorithms that use $\tilde{O}(k) \cdot \mathrm{poly}(1/\varepsilon)$ memory, where k is the size constraint. At the end of the stream, both our algorithms post-process their data structures using *any offline algorithm* for submodular maximization, and obtain a solution whose approximation guarantee is $\frac{\alpha}{1+\alpha} - \varepsilon$, where α is the approximation of the offline algorithm. If we use an exact (exponential time) post-processing algorithm, this leads to $\frac{1}{2} - \varepsilon$ approximation (which is nearly optimal). If we post-process with the algorithm of [5], that achieves the state-of-the-art offline approximation guarantee of $\alpha = 0.385$, we obtain 0.2779-approximation in polynomial time, improving over the previously best polynomial-time approximation of 0.1715 due to [17]. One of our algorithms is combinatorial and enjoys fast update and overall running times. Our other algorithm is based on the multilinear extension, enjoys an improved space complexity, and can be made deterministic in some settings of interest.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Mathematics of computing → Combinatorial optimization; Mathematics of computing → Probabilistic algorithms

Keywords and phrases Submodular maximization, streaming algorithms, cardinality constraint

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.6

Category Track A: Algorithms, Complexity and Games

Related Version `https://arxiv.org/abs/1911.12959`

Funding *Naor Alaluf*: The work of Naor Alaluf was supported in part by ISF grant number 1357/16.
Alina Ene: The work of Alina Ene was supported in part by NSF CAREER grant CCF-1750333, NSF grant CCF-1718342, and NSF grant III-1908510.
Moran Feldman: The work of Moran Feldman was supported in part by ISF grant number 1357/16.
Huy L. Nguyen: The work of Huy L. Nguyen was supported in part by NSF CAREER grant CCF-1750716 and NSF grant CCF-1909314.
Andrew Suh: The work of Andrew Suh was supported in part by NSF CAREER grant CCF-1750333, NSF grant CCF-1718342, and NSF grant III-1908510.

1 Introduction

In this paper, we study the problem of maximizing a *non-monotone* submodular function subject to a cardinality (size) constraint in the streaming model. This problem captures problems of interest in a wide-range of domains, such as machine learning, data mining, combinatorial optimization, algorithmic game theory, social networks, and many others. A representative application is data summarization, where the goal is to select a small subset of the data that captures the salient features of the overall dataset [2]. One can model these problems as submodular maximization with a cardinality constraint: the submodular objective captures how informative the summary is, as well as other considerations such as how diverse the summary is, and the cardinality constraint ensures that the summary is small. Obtaining such a summary is very beneficial when working with massive data sets, that may not even fit into memory, since it makes it possible to analyze the data using algorithms that would be prohibitive to run on the entire dataset.

There have been two main approaches to deal with the large size of modern data sets: the *distributed* computation approach partitions the data across many machines and uses local computation on the machines and communication across the machines in order to perform the analysis, and the *streaming* computation approach processes the data in a stream using only a small amount of memory and (ideally) only a single pass over the data. Classical algorithms for submodular maximization, such as the Greedy algorithm, are not suitable in these settings since they are centralized and require many passes over the data. Motivated by the applications as well as theoretical considerations, there has been a significant interest in studying submodular maximization problems both in the distributed and the streaming setting, leading to many new results and insights [22, 29, 2, 9, 11, 26, 4, 28, 3, 14, 27, 17, 31, 1].

Despite this significant progress, several fundamental questions remain open both in the streaming and distributed setting. In the streaming setting, which is the main focus of this paper, submodular maximization is fairly well understood when the objective function is additionally *monotone* – i.e., we have $f(A) \leq f(B)$ whenever $A \subseteq B$. For example, the Greedy approach, which obtains an optimal $(1 - 1/e)$-approximation in the centralized setting when the function is monotone [30], can be adapted to the streaming model [22, 2]. This yields the single-threshold Greedy algorithm: make a single pass over the data and select an item if its marginal gain exceeds a suitably chosen threshold. If the threshold is chosen to be $\frac{1}{2}\frac{f(\mathrm{OPT})}{k}$, where $f(\mathrm{OPT})$ is the value of the optimal solution and k is the cardinality constraint, then the single-threshold Greedy algorithm is guaranteed to achieve $\frac{1}{2}$-approximation. Although the value of the optimal solution is unknown, it can be estimated based on the largest singleton value even in the streaming setting [2]. Remarkably, this approximation guarantee is optimal in the streaming model even if we allow unbounded computational power: Feldman et al. [19] showed that any algorithm for monotone submodular maximization that achieves an approximation better than $\frac{1}{2}$ requires $\Omega\left(\frac{n}{k^3}\right)$ memory, where n is the length of the stream. Additionally, the single-threshold Greedy algorithm enjoys a fast update time of $O(\varepsilon^{-1} \log k)$ marginal value computations per item, and it uses $O(\varepsilon^{-1} k \log k)$ space.

In contrast, the general problem with a *non-monotone* objective has proved to be considerably more challenging. Even in the centralized setting, the Greedy algorithm fails to achieve any approximation guarantee when the objective is non-monotone. Thus, several approaches have been developed for handling non-monotone objectives in this setting, including local search [15, 24, 23], continuous optimization [18, 13, 5] and sampling [6, 16]. The currently best approximation guarantee is 0.385 [5], and the strongest inapproximability is 0.491 [20], and it remains a long-standing open problem to settle the approximability of submodular maximization subject to a cardinality constraint.

Adapting the above techniques to the streaming setting is challenging, and the approximation guarantees are weaker. The main approach for non-monotone maximization in the streaming setting has been to extend the local search algorithm of Chakrabarti and Kale [9] from monotone to non-monotone objectives. This approach was employed in a sequence of works [11, 17, 27], leading to the currently best approximation of $\frac{1}{3+2\sqrt{2}} \approx 0.1715$.[1] This naturally leads to the following questions.

- *What is the optimal approximation ratio achievable for submodular maximization in the streaming model? Is it possible to achieve $\frac{1}{2} - \varepsilon$ approximation using an algorithm that uses only* $\mathrm{poly}(k, 1/\varepsilon)$ *space?*
- *Is there a good streaming algorithm for non-monotone functions based on the single-threshold Greedy algorithm that works so well for monotone functions?*
- *Can we exploit existing heuristics for the offline problem in the streaming setting?*

Our contributions. In this work, we give an affirmative answer to all of the above questions. Specifically, we give streaming algorithms[2] that perform a single pass over the stream and output a set of size $k \cdot \mathrm{poly}(1/\varepsilon)$ that can be post-processed using *any offline algorithm* for submodular maximization. The post-processing is itself quite straightforward: we simply run the offline algorithm on the output set to obtain a solution of size at most k. We show that, if the offline algorithm achieves α-approximation, then we obtain $\left(\frac{\alpha}{1+\alpha} - \varepsilon\right)$-approximation.

Our main result implies that if we post-process using an exact (exponential time) algorithm, we obtain $(\frac{1}{2} - \varepsilon)$-approximation. This matches the inapproximability result proven by [19] for the special case of a monotone function. Furthermore, we show that in the non-monotone case any streaming algorithm guaranteeing $(\frac{1}{2} + \varepsilon)$-approximation for some positive constant ε must use in fact $\Omega(n)$ space.[3] Thus, we essentially settle the approximability of the problem if exponential-time computation is allowed.

The best (polynomial-time) approximation guarantee that is currently known in the offline setting is $\alpha = 0.385$ [5]. If we post-process using this algorithm, we obtain 0.2779-approximation in polynomial time, improving over the previously best polynomial-time approximation of 0.1715 due to [17]. The offline algorithm of [5] is based on the multilinear extension, and thus is quite slow. One can obtain a more efficient overall algorithm by using the combinatorial random Greedy algorithm of [6] that achieves $\frac{1}{e}$-approximation. Furthermore, any existing heuristic for the offline problem can be used for post-processing, exploiting their effectiveness beyond the worst case.

Our techniques. The two streaming algorithms that we present enjoy the same approximation guarantee, but differ in other properties. Our first algorithm (StreamProcess) is a combinatorial algorithm that achieves very fast update time and overall running time. StreamProcess takes inspiration both from the single-threshold Greedy algorithm for *monotone* maximization and distributed algorithms that randomly partition the data [26, 4, 3]: it randomly partitions the elements into $1/\varepsilon$ parts as they arrive in the stream and runs

[1] Chekuri et al. [11] claimed an improved approximation ratio of $\frac{1}{2+e} - \varepsilon$ for a cardinality constraint, but an error was later found in the proof of this improved ratio [10]. We defer the details to the full version.

[2] Formally, our algorithms are semi-streaming algorithms, i.e., their space complexity is nearly linear in k. Since this is unavoidable for algorithms designed to output an approximate solution (as opposed to just estimating the value of the optimal solution), we ignore the difference between streaming and semi-streaming algorithms in this paper and use the two terms interchangably.

[3] This result is a simple adaptation of a result due to Buchbinder et al. [7]. For completeness, we include the proof in the full version of the paper.

the single-threshold Greedy algorithm on each part; this process is repeated independently and in parallel $O(\ln(1/\varepsilon)/\varepsilon)$ times. Since the main engine behind our algorithm is the very efficient and practical single-threshold Greedy algorithm, our StreamProcess algorithm inherits its very efficient update time and practical potential. Compared to the optimal streaming algorithm for monotone maximization discussed above, our algorithm is quite similar: the monotone algorithm runs $O(\log k/\varepsilon)$ instances of single-threshold Greedy, each of which processes all n items; StreamProcess runs $O(\ln(1/\varepsilon)/\varepsilon^2) \cdot O(\log k/\varepsilon)$ instances of single-threshold Greedy, each of which processes $O(\varepsilon \cdot n)$ items with high probability.

Our second algorithm (StreamProcessExtension) is based on the multilinear extension of the submodular function. This algorithm is similar to the single-threshold Greedy algorithm, but adds fractions of elements rather than whole elements to the solution it maintains. The extension based approach of this algorithm allows us to save on the space usage. Furthermore, when the multilinear extension can be evaluated deterministically, this approach leads to a deterministic algorithm. However, the time complexity of this approach depends on the complexity of evaluating the multilinear extension, which is quite high if we are only given value oracle access to f. Thus, given such restricted access, this approach leads to higher update and overall running time.

We note that combining the single-threshold Greedy with randomization is difficult because it requires delicate care of the event that the single-threshold Greedy algorithm fills up the budget. In particular, this was the source of the subtle error mentioned above in one of the results of [11]. Our approach here for handling this issue is simple in retrospect. In our combinatorial algorithm, we consider two cases depending on the probability that the budget is filled up in a run (this is a good event since the resulting solution has good value). If this probability is sufficiently large (at least ε), we repeat the basic algorithm $O(\ln(1/\varepsilon)/\varepsilon)$ times to boost the probability of this good event to $1 - \varepsilon$. Otherwise, the probability that the budget is not filled up in a run is at least $1 - \varepsilon$, and conditioning on this event changes the probabilities by only a $1 - \varepsilon$ factor.

In our extension based algorithm, the decisions of the algorithm are based on the values taken by derivatives of the extension, which are values of expectations over appropriately chosen distributions. On the one hand, this allows our algorithm to include a random component, which is a component that appears (at least implicitly) in all of the known algorithms for non-monotone submodular maximization. On the other hand, since expectations have deterministic values, the algorithm we get is deterministic enough that it suffices for us to consider at each time only one of two possible cases: the case in which the budget fills up, and the case in which it does not.

Paper structure. In Section 2, defines the notation that we use and presents some known lemmata. Section 3 presents and analyzes our combinatorial algorithm (StreamProcess). Finally, in Section 4, we present and analyze our extension based algorithm.

2 Preliminaries

Basic notation. Let V denote a finite ground set of size $n := |V|$. We occasionally assume without loss of generality that $V = \{1, 2, \ldots, n\}$, and use, e.g., $x = (x_1, x_2, \ldots, x_n)$ to denote a vector in $\mathbb{R}^V$. For two vectors $x, y \in \mathbb{R}^V$, we let $x \vee y$ and $x \wedge y$ be the vectors such that $(x \vee y)_e = \max\{x_e, y_e\}$ and $(x \wedge y)_e = \min\{x_e, y_e\}$ for all $e \in V$. For a set $S \subseteq V$, we let $\mathbf{1}_S$ denote the indicator vector of S, i.e., the vector that has 1 in every coordinate $e \in S$ and 0 in every coordinate $e \in V \setminus S$. Given an element $e \in V$, we use $\mathbf{1}_e$ as a shorthand for $\mathbf{1}_{\{e\}}$. Furthermore, if S is a random subset of V, we use $\mathbb{E}[\mathbf{1}_S]$ to denote the vector p such that $p_e = \Pr[e \in S]$ for all $e \in V$ (i.e., the expectation is applied coordinate-wise).

Submodular functions. In this paper, we consider the problem of maximizing a non-negative submodular function subject to a cardinality constraint. A set function $f\colon 2^V \to \mathbb{R}$ is submodular if $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ for all subsets $A, B \subseteq V$.

Continuous extensions. We make use of two standard continuous extensions of submodular functions. The first of these extensions is known as the *multilinear extension*. To define this extension, we first need to define the random set $\mathtt{R}(x)$. For every vector $x \in [0,1]^V$, $\mathtt{R}(x)$ is defined as a random subset of V that includes every element $e \in V$ with probability x_e, independently. The multilinear extension F of f is now defined for every $x \in [0,1]^V$ by

$$F(x) = \mathbb{E}\big[f\big(\mathtt{R}(x)\big)\big] = \sum_{A \subseteq V} f(A) \cdot \Pr[\mathtt{R}(x) = A] = \sum_{A \subseteq V} \left(f(A) \cdot \prod_{e \in A} x_e \cdot \prod_{e \notin A} (1 - x_e) \right) .$$

One can observe from the definition that F is indeed a multilinear function of the coordinates of x, as suggested by its name. Thus, if we use the shorthand $\partial_e F(x)$ for the first partial derivative $\frac{\partial F(x)}{\partial x_e}$ of the multilinear extension F, then $\partial_e F(x) = F(x \vee \mathbf{1}_e) - F\big(x \wedge \mathbf{1}_{V \setminus \{e\}}\big)$.

In the analysis of our extension based algorithm, we need an upper bound on the possible increase in the value of $F(x)$ when some of the indices of x are zeroed. Corollary 2 provides such an upper bound. It readily follows from the following known lemma by Buchbinder et al. [6].

▶ **Lemma 1** (Lemma 2.2 from [6]). *Let $f\colon 2^V \to \mathbb{R}_{\geq 0}$ be a non-negative submodular function. Denote by $A(p)$ a random subset of $A \subseteq V$ where each element appears with probability at most p (not necessarily independently). Then, $\mathbb{E}[f(A(p))] \geq (1-p) \cdot f(\varnothing)$.*

In the statement of Corollary 2, and in the rest of the paper, we denote by $\operatorname{supp}(x)$ the support of vector x, i.e., the set $\{e \in V \mid x_e > 0\}$.

▶ **Corollary 2.** *Let $f\colon 2^V \to \mathbb{R}_{\geq 0}$ be a non-negative submodular function, let p be a number in the range $[0,1]$ and let $x, y \in [0,1]^V$ be two vectors such that $\operatorname{supp}(x) \cap \operatorname{supp}(y) = \varnothing$ and $y_e \leq p$ for every $e \in V$. Then, the multilinear extension F of f obeys $F(x+y) \geq (1-p) \cdot F(x)$.*

The analyses of both our algorithms make use of the *Lovász extension* $\hat{f}$ of f. The Lovász extension $\hat{f}\colon [0,1]^V \to \mathbb{R}$ is defined as follows. For every $x \in [0,1]^V$, $\hat{f}(x) = \mathbb{E}_{\theta \sim [0,1]}[f(\{e \in V\colon x_e \geq \theta\})]$, where we use the notation $\theta \sim [0,1]$ to denote a value chosen uniformly at random from the interval $[0,1]$. The Lovász extension $\hat{f}$ of a non-negative submodular function has the following properties: (1) convexity: $c\hat{f}(x) + (1-c)\hat{f}(y) \geq \hat{f}(cx + (1-c)y)$ for all $x, y \in [0,1]^V$ and all $c \in [0,1]$ [25]; (2) restricted scale invariance: $\hat{f}(cx) \geq c\hat{f}(x)$ for all $x \in [0,1]^V$ and all $c \in [0,1]$; (3) it lower bounds the multilinear extension, i.e., $F(x) \geq \hat{f}(x)$ for every $x \in [0,1]^V$ [32, Lemma A.4].

3 Combinatorial Algorithm

Our combinatorial streaming algorithm is shown in Algorithm 1. For simplicity, we describe the algorithm assuming the knowledge of an estimate of the value of the optimal solution, $f(\mathrm{OPT})$. To remove this assumption, we use the standard technique introduced by [2]. The basic idea is to use the maximum singleton value $v = \max_e f(\{e\})$ as a k-approximation of $f(\mathrm{OPT})$. Given this approximation, one can guess a $1+\varepsilon$ approximation of $f(\mathrm{OPT})$ from a set of $O(\log(k/\alpha)/\varepsilon)$ values ranging from v to kv/α (α is the approximation guarantee of the offline algorithm OfflineAlg that we use in the post-processing step). The final streaming algorithm is simply $O(\log(k/\alpha)/\varepsilon)$ copies of the basic algorithm running in parallel with different guesses. As new elements appear in the stream, the value $v = \max_e f(\{e\})$ also increases over time and thus, existing copies of the basic algorithm with small guesses are dropped and new copies with higher guesses are added. An important observation is that when we introduce a new copy with a large guess, starting it from mid-stream has exactly the same outcome as if we started it from the beginning of the stream: all previous elements have marginal gain much smaller than the guess and smaller than the threshold so they would have been rejected anyway. We refer to [2] for the full details.

▶ **Theorem 3.** *There is a streaming algorithm* StreamProcess *for non-negative, non-monotone submodular maximization with the following properties ($\varepsilon > 0$ is any desired accuracy and it is given as input to the algorithm):*

- *The algorithm makes a single pass over the stream.*
- *The algorithm uses $O\left(\frac{k\log(k/\alpha)\log(1/\varepsilon)}{\varepsilon^3}\right)$ space.*
- *The update time per item is $O\left(\frac{\log(k/\alpha)\log(1/\varepsilon)}{\varepsilon^2}\right)$ marginal gain computations.*

At the end of the stream, we post-process the output of StreamProcess *using any offline algorithm* OfflineAlg *for submodular maximization. The resulting solution is a $\frac{\alpha}{1+\alpha} - \varepsilon$ approximation, where α is the approximation of* OfflineAlg.

Algorithm 1 Streaming algorithm for $\max_{|S|\le k} f(S)$. PostProcess uses any offline algorithm OfflineAlg with approximation α. Lines shown in blue are comments. The algorithm does **not** store the sets $V_{i,j}$, they are defined for analysis purposes only.

StreamProcess$(f, k, \varepsilon, \kappa)$
$r \leftarrow \Theta(\ln(1/\varepsilon)/\varepsilon)$
$m \leftarrow 1/\varepsilon$
$S_{i,j} \leftarrow \varnothing$ for all $i \in [r], j \in [m]$
$V_{i,j} \leftarrow \varnothing$ for all $i \in [r], j \in [m]$ // **not stored**, defined for analysis purposes only
for *each arriving element e* **do**
    for $i = 1$ *to* r **do**
        choose an index $j \in [m]$ uniformly and independently at random
        $V_{i,j} \leftarrow V_{i,j} \cup \{e\}$ // **not stored**, defined for analysis purposes only
        if $f(S_{i,j} \cup \{e\}) - f(S_{i,j}) \ge \kappa$ *and* $|S_{i,j}| < k$ **then**
            $S_{i,j} \leftarrow S_{i,j} \cup \{e\}$
return $\{S_{i,j} : i \in [r], j \in [m]\}$

PostProcess(f, k, ε)
$\kappa \leftarrow \frac{\alpha}{1+\alpha} \cdot \frac{1}{k} \cdot f(\mathrm{OPT})$ // threshold
$\{S_{i,j}\} \leftarrow$ StreamProcess$(f, k, \varepsilon, \kappa)$
if $|S_{i,j}| = k$ *for some i and j* **then**
    return $S_{i,j}$
else
    $U \leftarrow \bigcup_{i,j} S_{i,j}$
    $T \leftarrow$ OfflineAlg(f, k, U)
    return $\arg\max\{f(S_{1,1}), f(T)\}$

Algorithm 2 Single threshold Greedy algorithm. The algorithm processes the elements in the order in which they arrive in the stream, and it uses the same threshold κ as StreamProcess.

```
STGreedy(f, N, k, κ):
S ← ∅
for each e ∈ N in the stream order do
    if f(S ∪ {e}) − f(S) ≥ κ and |S| < k then
        S ← S ∪ {e}
return S
```

In the remainder of this section, we analyze Algorithm 1 and show that it achieves a $\frac{\alpha}{1+\alpha} - \varepsilon$ approximation, where α is the approximation guarantee of the offline algorithm OfflineAlg.

We divide the analysis into two cases, depending on the probability of the event that a set $S_{i,1}$ (for some $i \in [r]$) constructed by StreamProcess has size k. For every $i \in [r]$, let $\mathcal{F}_i$ be the event that $|S_{i,1}| = k$. Since each of the r repetitions (iterations of the for loop of StreamProcess) use independent randomness to partition V, the events $\mathcal{F}_1, \dots, \mathcal{F}_r$ are independent. Additionally, the events $\mathcal{F}_1, \dots, \mathcal{F}_r$ have the same probability. We divide the analysis into two cases, depending on whether $\Pr[\mathcal{F}_1] \geq \varepsilon$ or $\Pr[\mathcal{F}_1] < \varepsilon$. In the first case, since we are repeating $r = \Theta(\ln(1/\varepsilon)/\varepsilon)$ times, the probability that there is a set $S_{i,j}$ of size k is at least $1 - \varepsilon$, and we obtain the desired approximation since $f(S_{i,j}) \geq \kappa\,|S_{i,j}| = \kappa k = \frac{\alpha}{1+\alpha} f(\mathrm{OPT})$. In the second case, we have $\Pr\left[\,\overline{\mathcal{F}_1}\,\right] \geq 1 - \varepsilon$ and we argue that $\bigcup_{i,j} S_{i,j}$ contains a good solution. We now give the formal argument for each of the cases.

The case $\Pr[\mathcal{F}_1] \geq \varepsilon$

As noted earlier, the events $\mathcal{F}_1, \dots, \mathcal{F}_r$ are independent and have the same probability. Thus,

$$\Pr\left[\,\overline{\mathcal{F}_1 \cup \dots \cup \mathcal{F}_r}\,\right] \leq (1-\varepsilon)^r \leq \exp(-\varepsilon r) \leq \varepsilon$$

since $r = \Theta(\ln(1/\varepsilon)/\varepsilon)$. Thus $\Pr[\mathcal{F}_1 \cup \dots \cup \mathcal{F}_r] \geq 1 - \varepsilon$.

Conditioned on the event $\mathcal{F}_1 \cup \dots \cup \mathcal{F}_r$, we obtain the desired approximation due to the following lemma. The lemma follows from the fact that the marginal gain of each selected element is at least κ.

▶ **Lemma 4.** *We have $f(S_{i,j}) \geq \kappa\,|S_{i,j}|$ for all $i \in [r]$, $j \in [m]$.*

We can combine the two facts and obtain the desired approximation as follows. Let $\mathcal{S}$ be the random variable equal to the solution returned by PostProcess. We have

$$\mathbb{E}[f(\mathcal{S})] \geq \mathbb{E}[f(\mathcal{S}) | \mathcal{F}_1 \cup \dots \cup \mathcal{F}_r] \Pr[\mathcal{F}_1 \cup \dots \cup \mathcal{F}_r] \geq (1-\varepsilon)\kappa k = (1-\varepsilon)\frac{\alpha}{1+\alpha} f(\mathrm{OPT})$$

The case $\Pr[\mathcal{F}_1] < \varepsilon$

In this case, we show that the solution $\arg\max\{f(T), f(S_{1,1})\}$ returned on the last line of PostProcess has good value in expectation. Our analysis borrows ideas and techniques from the work of Barbosa et al. [3]: the probabilities p_e defined below are analogous to the probabilities used in that work; the division of OPT into two sets based on these probabilities is analogous to the division employed in Section 7.3 in that work; Lemma 6 shows a consistency property for the single threshold greedy algorithm that is analogous to the consistency property shown for the standard greedy algorithm and other algorithms by Barbosa *et al.* Barbosa *et al.* use these concepts in a different context (specifically, *monotone* maximization in the *distributed* setting). When applied to our context – *non-monotone* maximization in the *streaming* setting – the framework of Barbosa *et al.* requires $\Omega(\sqrt{nk})$ memory if used with a single pass (alternatively, they use $\Omega(\min\{k, 1/\varepsilon\})$ passes) and achieves worse approximation guarantees.

Notation and definitions. For analysis purposes only, we make use of the Lovasz extension $\hat{f}$. We fix an optimal solution $\mathrm{OPT} \in \arg\max\{f(A) \colon A \subseteq V, |A| \leq k\}$. Let $\mathcal{V}(1/m)$ be the distribution of $1/m$-samples of V, where a $1/m$-sample of V includes each element of V independently at random with probability $1/m$. Note that $V_{i,j} \sim \mathcal{V}(1/m)$ for every $i \in [r]$, $j \in [m]$ (see StreamProcess). Additionally, for each $i \in [r]$, $V_{i,1}, \dots, V_{i,m}$ is a partition of V into $1/m$-samples.

For a subset $N \subseteq V$, we let $\mathrm{STG}\textsc{reedy}(N)$ be the output of the single threshold greedy algorithm when run as follows (see also Algorithm 2 for a formal description of the algorithm): the algorithm processes the elements of N *in the order in which they arrive in the stream* and it uses the same threshold κ as StreamProcess; starting with the empty solution and continuing until the size constraint of k is reached, the algorithm adds an element to the current solution if its marginal gain is above the threshold. Note that $S_{i,j} = \mathrm{STG}\textsc{reedy}(V_{i,j})$ for all $i \in [r], j \in [m]$. For analysis purposes only, we also consider $\mathrm{STG}\textsc{reedy}(N)$ for sets N that do not correspond to any set $V_{i,j}$.

For each $e \in V$, we define

$$p_e = \begin{cases} \Pr_{X \sim \mathcal{V}(1/m)}\left[e \in \mathrm{STG}\textsc{reedy}(X \cup \{e\})\right] & \text{if } e \in \mathrm{OPT} \\ 0 & \text{otherwise} \end{cases}$$

We partition OPT into two sets:

$$O_1 = \{e \in \mathrm{OPT} \colon p_e \geq \varepsilon\} \quad O_2 = \mathrm{OPT} \setminus O_1$$

We also define the following subset of O_2:

$$O_2' = \{e \in O_2 \colon e \notin \mathrm{STG}\textsc{reedy}\left(V_{1,1} \cup \{e\}\right)\}.$$

Note that (O_1, O_2) is a deterministic partition of OPT, whereas O_2' is a random subset of O_2. The role of the sets O_1, O_2, O_2' will become clearer in the analysis. The intuition is that, using the repetition, we can ensure that each element of O_1 ends up in the collected set $U = \bigcup_{i,j} S_{i,j}$ with good probability: each iteration $i \in [r]$ ensures that an element $e \in O_1$ is in $S_{i,1} \cup \dots \cup S_{i,m}$ with probability $p_e \geq \varepsilon$ and, since we repeat $r = \Theta(\ln(1/\varepsilon)/\varepsilon)$ times, we will ensure that $\mathbb{E}[\mathbf{1}_{O_1 \cap U}] \geq (1-\varepsilon)\mathbf{1}_{O_1}$. We also have that $\mathbb{E}\left[\mathbf{1}_{O_2'}\right] \geq (1-\varepsilon)\mathbf{1}_{O_2}$: an element $e \in O_2 \setminus O_2'$ ends up being picked by STGreedy when run on input $V_{1,1} \cup \{e\}$, which is a low probability event for the elements in O_2; more precisely, the probability of this event is equal to p_e (since $V_{1,1} \sim \mathcal{V}(1/m)$) and $p_e \leq \varepsilon$ (since $e \in O_2$). Thus $\mathbb{E}\left[\mathbf{1}_{(O_1 \cap U) \cup O_2'}\right] \geq (1-\varepsilon)\mathbf{1}_{\mathrm{OPT}}$, which implies that the expected value of $(O_1 \cap U) \cup O_2'$ is at least $(1-\varepsilon)f(\mathrm{OPT})$. However, whereas $O_1 \cap U$ is available in the post-processing phase, elements of O_2' may not be available and they may account for most of the value of O_2. The key insight is to show that $S_{1,1}$ makes up for the lost value from these elements.

We start the analysis with two helper lemmas, which follow from standard arguments that have been used in previous works. The first of these lemmas follows from an argument based on the Lovasz extension and its properties.

▶ **Lemma 5.** *Let* $0 \le u \le v \le 1$*. Let* $S \subseteq V \setminus \mathrm{OPT}$ *and* $O \subseteq \mathrm{OPT}$ *be random sets such that* $\mathbb{E}[\mathbf{1}_S] \le u\mathbf{1}_{V\setminus \mathrm{OPT}}$ *and* $\mathbb{E}[\mathbf{1}_O] \ge v\mathbf{1}_{\mathrm{OPT}}$*. Then* $\mathbb{E}[f(S \cup O)] \ge (v-u)f(\mathrm{OPT})$*.*

The following lemma establishes a consistency property for the STGreedy algorithm, analogous to the consistency property shown and used by Barbosa et al. for algorithms such as the standard Greedy algorithm. The proof is also very similar to the proof shown by Barbosa et al.

▶ **Lemma 6.** *Conditioned on the event* $|S_{1,1}| < k$*, we have* STGreedy$(V_{1,1} \cup O_2') =$ STGreedy$(V_{1,1}) = S_{1,1}$*.*

We now proceed with the main analysis. Recall that PostProcess runs the algorithm OfflineAlg on U to obtain a solution T, and returns the better of the two solutions $S_{1,1}$ and T. In the following lemma, we show that the value of this solution is proportional to $f(S_{1,1} \cup (O_1 \cap U))$. Note that $S_{1,1} \cup (O_1 \cap U)$ may not be feasible, since we could have $|S_{1,1}| > |O_2|$, and hence the scaling based on $\frac{|O_2|}{k}$.

▶ **Lemma 7.** *We have* $\max\{f(S_{1,1}), f(T)\} \ge \frac{\alpha}{1+\alpha\left(1-\frac{|O_2|}{k}\right)} f(S_{1,1} \cup (O_1 \cap U))$*.*

Proof. To simplify notation, we let $S_1 = S_{1,1}$. Let $b = |O_2|$. First, we analyze $f(T)$. Let $X \subseteq S_1$ be a random subset of S_1 such that $|X| \le b$ and $\mathbb{E}[\mathbf{1}_X] = \frac{b}{k}\mathbf{1}_{S_1}$. We can select such a subset as follows: we first choose a permutation of S_1 uniformly at random, and let $\tilde{X}$ be the first $s := \min\{b, |S_1|\}$ elements in the permutation. For each element of $\tilde{X}$, we add it to X with probability $p := |S_1|b/(sk)$. For each $e \in S_1$, we have

$$\Pr[e \in X] = \Pr\left[e \in X | e \in \tilde{X}\right] \Pr\left[e \in \tilde{X}\right] = p\frac{s}{|S_1|} = \frac{b}{k}$$

For each $e \notin S_1$, we have $\Pr[e \in X] = 0$. Thus $\mathbb{E}[\mathbf{1}_X] = \frac{b}{k}\mathbf{1}_{S_1}$.

Since $X \cup ((O_1 \cap U) \setminus S_1)$ is a feasible solution contained in U and OfflineAlg achieves an α-approximation, we have

$$f(T) \ge \alpha f(X \cup ((O_1 \cap U) \setminus S_1))$$

By taking expectation over X only (more precisely, the random sampling that we used to select X) and using that $\hat{f}$ is a convex extension, we obtain:

$$\begin{aligned} f(T) &\ge \alpha \mathbb{E}_X\left[f(X \cup ((O_1 \cap U) \setminus S_1))\right] = \alpha \mathbb{E}_X\left[\hat{f}\left(\mathbf{1}_{X \cup ((O_1 \cap U)\setminus S_1))}\right)\right] \\ &\ge \alpha \hat{f}\left(\mathbb{E}_X\left[\mathbf{1}_{X \cup ((O_1 \cap U)\setminus S_1)}\right]\right) = \alpha \hat{f}\left(\frac{b}{k}\mathbf{1}_{S_1} + \mathbf{1}_{(O_1 \cap U)\setminus S_1}\right) \end{aligned}$$

Next, we lower bound $\max\{f(S_1), f(T)\}$ using a convex combination $(1-\theta)f(S_1) + \theta f(T)$ with coefficient $\theta = 1/(1 + \alpha\left(1 - \frac{b}{k}\right))$. Note that $1 - \theta = \theta\alpha\left(1 - \frac{b}{k}\right)$. By taking this convex combination, using the previous inequality lower bounding $f(T)$, and the convexity and restricted scale invariance of $\hat{f}$, we obtain:

$$\max\{f(S_1), f(T)\} \geq (1-\theta)f(S_1) + \theta f(T) = \theta\alpha\left(1-\frac{b}{k}\right)f(S_1) + \theta f(T)$$
$$\geq \theta\alpha\left(1-\frac{b}{k}\right)\hat{f}(\mathbf{1}_{S_1}) + \theta\alpha\hat{f}\left(\frac{b}{k}\mathbf{1}_{S_1} + \mathbf{1}_{(O_1\cap U)\setminus S_1}\right)$$
$$= \theta\alpha\left(2-\frac{b}{k}\right)\left(\frac{1-\frac{b}{k}}{2-\frac{b}{k}}\hat{f}(\mathbf{1}_{S_1}) + \frac{1}{2-\frac{b}{k}}\hat{f}\left(\frac{b}{k}\mathbf{1}_{S_1} + \mathbf{1}_{(O_1\cap U)\setminus S_1}\right)\right)$$
$$\geq \theta\alpha\left(2-\frac{b}{k}\right)\hat{f}\left(\frac{1-\frac{b}{k}}{2-\frac{b}{k}}\mathbf{1}_{S_1} + \frac{1}{2-\frac{b}{k}}\left(\frac{b}{k}\mathbf{1}_{S_1} + \mathbf{1}_{(O_1\cap U)\setminus S_1}\right)\right)$$
$$= \theta\alpha\left(2-\frac{b}{k}\right)\hat{f}\left(\frac{1}{2-\frac{b}{k}}\mathbf{1}_{S_1\cup(O_1\cap U)}\right) \geq \frac{\alpha}{1+\alpha\left(1-\frac{b}{k}\right)}f(S_1\cup(O_1\cap U)). \qquad \blacktriangleleft$$

(We note that we chose θ to make the coefficients of $\mathbf{1}_{S_1}$ and $\mathbf{1}_{(O_1\cap U)\setminus S_1}$ equal, and this allowed us to relate the value of the final solution to $f(S_1\cup(O_1\cap U))$.)

Next, we analyze the expected value of $f(S_{1,1}\cup(O_1\cap U))$. We do so in two steps: first we analyze the marginal gain of O_2' on top of $S_{1,1}$ and show that it is suitably small, and then we analyze $f(S_{1,1}\cup(O_1\cap U)\cup O_2')$ and show that its expected value is proportional to $f(\mathrm{OPT})$. We use the notation $f(A|B)$ to denote the marginal gain of A on top of B, i.e., $f(A|B) = f(A\cup B) - f(B)$.

▶ **Lemma 8.** *We have* $\mathbb{E}[f(O_2'|S_{1,1})] \leq \kappa b + \varepsilon f(\mathrm{OPT})$.

Proof. As before, to simplify notation, we let $S_1 = S_{1,1}$ and $V_1 = V_{1,1}$. We break down the expectation using the law of total expectation as follows:

$$\mathbb{E}[f(O_2'|S_1)] = \mathbb{E}[f(O_2'|S_1)\,|\,|S_1|<k]\cdot\underbrace{\Pr[|S_1|<k]}_{\leq 1} + \underbrace{\mathbb{E}[f(O_2'|S_1)\,|\,|S_1|=k]}_{\leq f(\mathrm{OPT})}\cdot\underbrace{\Pr[|S_1|=k]}_{\leq\varepsilon}$$
$$\leq \mathbb{E}[f(O_2'|S_1)\,|\,|S_1|<k] + \varepsilon f(\mathrm{OPT})$$

Above, we have used that $f(O_2'|S_1) \leq f(O_2') \leq f(\mathrm{OPT})$, where the first inequality follows by submodularity. We have also used that $\Pr[|S_1|=k] = \Pr[\mathcal{F}_1] \leq \varepsilon$. Thus it only remains to show that $\mathbb{E}[f(O_2'|S_1)\,|\,|S_1|<k] \leq \kappa b$.

We condition on the event $|S_1|<k$ for the remainder of the proof. By Lemma 6, we have $\mathrm{STGreedy}(V_1\cup O_2') = S_1$. Since $|S_1|<k$, each element of $O_2'\setminus S_1$ was rejected because its marginal gain was below the threshold when it arrived in the stream. This, together with submodularity, implies that $f(O_2'|S_1) \leq \kappa|O_2'| \leq \kappa b$. ◀

▶ **Lemma 9.** *We have* $\mathbb{E}[f(S_{1,1}\cup(O_1\cap U)\cup O_2')] \geq (1-2\varepsilon)f(\mathrm{OPT})$.

Proof. We apply Lemma 5 to the following sets:

$$S = S_{1,1}\setminus\mathrm{OPT}$$
$$O = (S_{1,1}\cap\mathrm{OPT})\cup(O_1\cap U)\cup O_2'$$

We show below that $\mathbb{E}[\mathbf{1}_O] \leq \varepsilon\mathbf{1}_{V\setminus\mathrm{OPT}}$ and $\mathbb{E}[\mathbf{1}_O] \geq (1-\varepsilon)\mathbf{1}_{\mathrm{OPT}}$. Assuming these bounds, we can take $u=\varepsilon$ and $v=1-\varepsilon$ in Lemma 5, which gives the desired result.

Since $S\subseteq S_{1,1}\subseteq V_{1,1}$ and $V_{1,1}$ is a $(1/m)$-sample of V, we have $\mathbb{E}[\mathbf{1}_S] \leq \frac{1}{m}\mathbf{1}_{V\setminus\mathrm{OPT}} = \varepsilon\mathbf{1}_{V\setminus\mathrm{OPT}}$. Thus it only remains to show that, for each $e\in\mathrm{OPT}$, we have $\Pr[e\in O]\geq 1-\varepsilon$. Since $(O_1\cap U)\cup O_2'\subseteq O$, it suffices to show that $\Pr[e\in(O_1\cap U)\cup O_2'] \geq 1-\varepsilon$, or equivalently that $\Pr[e\in(O_1\setminus U)\cup(O_2\setminus O_2')] \leq \varepsilon$.

Recall that (O_1, O_2) is a deterministic partition of OPT. Thus e belongs to exactly one of O_1 and O_2 and we consider each of these cases in turn.

Suppose that $e \in O_1$. A single iteration of the for loop of StreamProcess ensures that e is in $S_{i,1} \cup \dots \cup S_{i,m}$ with probability $p_e \geq \varepsilon$. Since we perform $r = \Theta(\ln(1/\varepsilon)/\varepsilon)$ independent iterations, we have $\Pr[e \notin U] \leq (1-\varepsilon)^r \leq \exp(-\varepsilon r) \leq \varepsilon$.

Suppose that $e \in O_2$. We have

$$\Pr[e \in O_2 \setminus O_2'] = \Pr[e \in \text{STGreedy}\,(V_{1,1} \cup \{e\})] = p_e \leq \varepsilon$$

where the first equality follows from the definition of O_2', the second equality follows from the definition of p_e and the fact that $V_{1,1} \sim \mathcal{V}(1/m)$, and the inequality follows from the definition of O_2. ◀

Lemmas 8 and 9 immediately imply the following:

▶ **Lemma 10.** *We have* $\mathbb{E}[f\,(S_{1,1} \cup (O_1 \cap U))] \geq (1-3\varepsilon)f(\text{OPT}) - \kappa b$.

Finally, Lemmas 7 and 10 give the approximation guarantee:

▶ **Lemma 11.** *We have* $\mathbb{E}[\max\{f(S_{1,1}), f(T)\}] \geq \left(\frac{\alpha}{1+\alpha} - 3\varepsilon\right) f(\text{OPT})$.

4 Extension based algorithm

Using our extension based algorithm, we prove the following theorem.

▶ **Theorem 12.** *Assume there exists an α-approximation offline algorithm* OfflineAlg *for maximizing a non-negative submodular function subject to cardinality constraint whose space complexity is nearly linear in the size of the ground set. Then, for every constant $\varepsilon \in (0,1]$, there exists an $(\frac{\alpha}{1+\alpha} - \varepsilon)$-approximation semi-streaming algorithm for maximizing a non-negative submodular function subject to a cardinality constraint. The algorithm stores at most $O(k\varepsilon^{-2})$ elements.*[4]

In this section, we introduce a simplified version of the algorithm used to prove Theorem 12. This simplified version (given as Algorithm 3) captures our main new ideas, but makes two simplifying assumptions that can be avoided using standard techniques.

- The first assumption is that Algorithm 3 has access to an estimate τ of $f(\text{OPT})$ obeying $(1-\varepsilon/8) \cdot f(\text{OPT}) \leq \tau \leq f(\text{OPT})$. Such an estimate can be produced using well-known techniques, at the cost of a slight increase in the space complexity of the algorithm. In the full version of this paper we formally show that one such technique due to [21] can be used for that purpose, and that it increases the space complexity of the algorithm only by a factor of $O(\varepsilon^{-1} \log \alpha^{-1})$.
- The second assumption is that Algorithm 3 has value oracle access to the multilinear extension F. If the time complexity of Algorithm 3 is not important, then this assumption is of no consequence since a value oracle query to F can be emulated using an exponential number of value oracle queries to f. However, the assumption becomes problematic when we would like to keep the time complexity of the algorithm polynomial and we only have value oracle access to f. Thus, we explain in the full version of this paper how to drop this

[4] Formally, the number of elements stored by the algorithm also depends on $\log \alpha^{-1}$. Since α is typically a positive constant, or at least lower bounded by a positive constant, we omit this dependence from the statement of the theorem.

assumption via sampling. Interestingly, the rounding step and this sampling technique are the only parts of the extension based algorithm that employ randomness. Since the rounding can be made deterministic given either exponential time or value oracle access to F, we get the following observation.

▶ **Observation 13.** *If* OfflineAlg *is deterministic, then the algorithm whose existence is guaranteed by Theorem 12 is also* deterministic *when it is allowed either exponential computation time or value oracle access to F.*

Algorithm 3 has two constant parameters $p \in (0, 1)$ and $c > 0$ and maintains a fractional solution $x \in [0, 1]^V$. This fractional solution starts empty, and the algorithm adds to it fractions of elements as they arrive. Specifically, when an element e arrives, the algorithm considers its marginal contribution with respect to the current fractional solution x. If this marginal contribution exceeds the threshold of $c\tau/k$, then the algorithm tries to add to x a p-fraction of e, but might end up adding a smaller fraction of e if adding a full p-fraction of e to x will make x an infeasible solution, i.e., make $\|x\|_1 > k$ (note that $\|x\|_1$ is the sum of the coordinates of x).

After viewing all of the elements, Algorithm 3 uses the fractional solution x to generate two sets S_1 and S_2 that are feasible (integral) solutions. The set S_1 is generated by rounding the fractional solution x. Two rounding procedures, named Pipage Rounding and Swap Rounding, were suggested for this task in the literature [8, 12]. Both procedures run in polynomial time and guarantee that the output set S_1 of the rounding is always feasible, and that its expected value with respect to f is at least the value $F(x)$ of the fractional solution x. The set S_2 is generated by applying OfflineAlg to the support of the vector x, which produces a feasible solution that (approximately) maximizes f among all subsets of the support whose size is at most k. After computing the two feasible solutions S_1 and S_2, Algorithm 3 simply returns the better one of them.

Algorithm 3 StreamProcessExtension (simplified) (p, c).

Let $x \leftarrow \mathbf{1}_{\varnothing}$.
for *each arriving element e* **do**
  if $\partial_e F(x) \geq \frac{c\tau}{k}$ **then** $x \leftarrow x + \min\{p, k - \|x\|_1\} \cdot \mathbf{1}_e$.
Round the vector x to yield a feasible solution S_1 such that $\mathbb{E}[f(S_1)] \geq F(x)$.
Find another feasible solution $S_2 \subseteq \operatorname{supp}(x)$ by running OfflineAlg with $\operatorname{supp}(x)$ as the ground set.
return the better solution among S_1 and S_2.

Let us denote by $\hat{x}$ the final value of the fractional solution x (i.e., its value when the stream ends). We begin the analysis of Algorithm 3 with the following useful observation.

▶ **Observation 14.** *If $\|\hat{x}\|_1 < k$, then $\hat{x}_e = p$ for every $e \in \operatorname{supp}(\hat{x})$. Otherwise (when $\|\hat{x}\|_1 = k$), this is still true for every element $e \in \operatorname{supp}(\hat{x})$ except for maybe a single element.*

Proof. For every element e added to the support of x by Algorithm 3, the algorithm sets x_e to p unless this will make $\|x\|_1$ exceed k, in which case the algorithm set x_e to be the value that will make $\|x\|_1$ equal to k. Thus, after a single coordinate of x is set to a value other than p (or the initial 0), $\|x\|_1$ becomes k and Algorithm 3 stops changing x. ◀

Using the last observation we can now bound the space complexity of Algorithm 3, and show (in particular) that it is a semi-streaming algorithm for a constant p when the space complexity of OfflineAlg is nearly linear.

▶ **Observation 15.** *Algorithm 3 can be implemented so that it stores at most $O(k/p)$ elements.*

Proof. To calculate the sets S_1 and S_2, Algorithm 3 needs access only to the elements of V that appear in the support of x. Thus, the number of elements it needs to store is $O(|\operatorname{supp}(\hat{x})|) = O(k/p)$, where the equality follows from Observation 14. ◀

We now divert our attention to analyzing the approximation ratio of Algorithm 3. The first step in this analysis is lower bounding the value of $F(\hat{x})$, which we do by considering two cases, one when $\|\hat{x}\|_1 = k$, and the other when $\|\hat{x}\|_1 < k$. The following lemma bounds the value of $F(\hat{x})$ in the first of these cases. Intuitively, this lemma holds since $\operatorname{supp}(\hat{x})$ contains many elements, and each one of these elements must have increased the value of $F(x)$ significantly when added (otherwise, Algorithm 3 would not have added this element to the support of x).

▶ **Lemma 16.** *If $\|\hat{x}\|_1 = k$, then $F(\hat{x}) \geq c\tau$.*

Proof. Denote by $e_1, e_2, \dots, e_\ell$ the elements in the support of $\hat{x}$, in the order of their arrival. Using this notation, the value of $F(\hat{x})$ can be written as follows.

$$\begin{aligned}
F(\hat{x}) &= F(\mathbf{1}_\varnothing) + \sum_{i=1}^{\ell} \left(F\big(\hat{x} \wedge \mathbf{1}_{\{e_1,e_2,\dots,e_i\}}\big) - F\big(\hat{x} \wedge \mathbf{1}_{\{e_1,e_2,\dots,e_{i-1}\}}\big)\right)\\
&= F(\mathbf{1}_\varnothing) + \sum_{i=1}^{\ell} \left(\hat{x}_{e_i} \cdot \partial_{e_i} F\big(\hat{x} \wedge \mathbf{1}_{\{e_1,e_2,\dots,e_{i-1}\}}\big)\right)\\
&\geq F(\mathbf{1}_\varnothing) + \frac{c\tau}{k} \cdot \sum_{i=1}^{\ell} \hat{x}_{e_i} = F(\mathbf{1}_\varnothing) + \frac{c\tau}{k} \cdot \|\hat{x}\|_1 \geq c\tau \enspace ,
\end{aligned}$$

where the second equality follows from the multilinearity of F, and the first inequality holds since Algorithm 3 selects an element e_i only when $\partial_{e_i} F\big(\hat{x} \wedge \mathbf{1}_{\{e_1,e_2,\dots,e_{i-1}\}}\big) \geq \frac{c\tau}{k}$. The last inequality holds since f (and thus, also F) is non-negative and $\|\hat{x}\|_1 = k$ by the assumption of the lemma. ◀

Consider now the case in which $\|\hat{x}\|_1 < k$. Recall that our objective is to lower bound $F(\hat{x})$ in this case as well. Towards this goal, we bound the expression $F(\hat{x} + \mathbf{1}_{\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})})$ from below and above in the following two lemmata.

▶ **Lemma 17.** *If $\|\hat{x}\|_1 < k$, then $F\big(\hat{x} + \mathbf{1}_{\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})}\big) \geq (1-p) \cdot \big[p \cdot f(\mathrm{OPT}) + (1-p) \cdot f\big(\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})\big)\big]$.*

Proof. Since $\|\hat{x}\|_1 < k$, Observation 14 guarantees that $\hat{x}_e = p$ for every $e \in \operatorname{supp}(\hat{x})$. Thus $\hat{x} = p \cdot \mathbf{1}_{\mathrm{OPT} \cap \operatorname{supp}(\hat{x})} + p \cdot \mathbf{1}_{\operatorname{supp}(\hat{x}) \setminus \mathrm{OPT}}$, and therefore,

$$\begin{aligned}
F\big(\hat{x} + \mathbf{1}_{\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})}\big) &= F\big(p \cdot \mathbf{1}_{\mathrm{OPT} \cap \operatorname{supp}(\hat{x})} + p \cdot \mathbf{1}_{\operatorname{supp}(\hat{x}) \setminus \mathrm{OPT}} + \mathbf{1}_{\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})}\big)\\
&\geq (1-p) \cdot F\big(p \cdot \mathbf{1}_{\mathrm{OPT} \cap \operatorname{supp}(\hat{x})} + \mathbf{1}_{\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})}\big)\\
&\geq (1-p) \cdot \hat{f}\big(p \cdot \mathbf{1}_{\mathrm{OPT} \cap \operatorname{supp}(\hat{x})} + \mathbf{1}_{\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})}\big)\\
&= (1-p) \cdot \Big[p \cdot f(\mathrm{OPT}) + (1-p) \cdot f\big(\mathrm{OPT} \setminus \operatorname{supp}(\hat{x})\big)\Big] \enspace ,
\end{aligned}$$

where the first inequality follows from Corollary 2, the second inequality holds since the Lovász extension lower bounds the multilinear extension, and the last equality follows from the definition of the Lovász extension. ◀

In the following lemma, and the rest of the section, we use the notation $b = k^{-1} \cdot |\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})|$. Intuitively, the lemma holds since the fact that the elements of $\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})$ where not added to the support of x implies that their marginal contribution is small.

▶ **Lemma 18.** *If $\|\hat{x}\|_1 < k$, then $F\big(\hat{x} + \mathbf{1}_{\mathrm{OPT}\setminus\mathrm{supp}(\hat{x})}\big) \leq F(\hat{x}) + bc\tau$.*

Proof. The elements in $\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})$ were rejected by Algorithm 3, which means that their marginal contribution with respect to the fractional solution x at the time of their arrival was smaller than $c\tau/k$. Since the fractional solution x only increases during the execution of the algorithm, the submodularity of f guarantees that this is true also with respect to $\hat{x}$. More formally, we get

$$\partial_e F(\hat{x}) < \frac{c\tau}{k} \quad \forall\, e \in \mathrm{OPT} \setminus \mathrm{supp}(\hat{x}) \ .$$

Using the submodularity of f again, this implies

$$F\big(\hat{x} + \mathbf{1}_{\mathrm{OPT}\setminus\mathrm{supp}(\hat{x})}\big) \leq F(\hat{x}) + \sum_{e\in\mathrm{OPT}\setminus\mathrm{supp}(\hat{x})} \partial_e F(\hat{x}) \leq F(\hat{x}) + |\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})| \cdot \frac{c\tau}{k} = F(\hat{x}) + bc\tau. \blacktriangleleft$$

Combining the last two lemmata immediately yields the promised lower bound on $F(\hat{x})$. To understand the second inequality in the following corollary, recall that $\tau \leq f(\mathrm{OPT})$.

▶ **Corollary 19.** *If $\|\hat{x}\|_1 < k$, then $F(\hat{x}) \geq (1-p) \cdot \Big[p \cdot f(\mathrm{OPT}) + (1-p) \cdot f\big(\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})\big)\Big] - bc\tau \geq [p(1-p) - bc]\tau + (1-p)^2 \cdot f\big(\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})\big)$.*

Our next step is to get a lower bound on the expected value of $f(S_2)$. One easy way to get such a lower bound is to observe that $\mathrm{OPT} \cap \mathrm{supp}(\hat{x})$ is a subset of the support of $\hat{x}$ of size at most k, and thus, is a candidate to be OPT; which implies $\mathbb{E}[f(S_2)] \geq \alpha \cdot f(\mathrm{OPT} \cap \mathrm{supp}(\hat{x}))$ since the algorithm OfflineAlg used to find S_2 is an α-approximation algorithm. The following lemma proves a more involved lower bound by considering the vector $(b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})}$ as a fractional candidate to be OPT (using the rounding methods discussed above it, it can be converted into an integral candidate of at least the same value). The proof of the lemma lower bounds the value of the vector $(b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})}$ using the concavity of the function $F((t \cdot \hat{x}) \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})})$ as well as ideas used in the proofs of the previous claims.

▶ **Lemma 20.** *If $\|\hat{x}\|_1 < k$, then $\mathbb{E}[f(S_2)] \geq \alpha b(1 - p - cb)\tau + \alpha(1 - b) \cdot f(\mathrm{OPT} \cap \mathrm{supp}(\hat{x}))$.*

Proof. Consider the vector $(b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})}$. Clearly,

$$\begin{aligned}\big\|(b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})}\big\|_1 &\leq b \cdot \|\hat{x}\|_1 + \big\|\mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})}\big\|_1 \\ &\leq |\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})| + |\mathrm{OPT} \cap \mathrm{supp}(\hat{x})| = |\mathrm{OPT}| \leq k \ ,\end{aligned}$$

where the second inequality holds by the definition of b since $\|\hat{x}\|_1 < k$. Thus, due to the existence of the rounding methods discussed in Section 4, there must exist a set S of size at most k obeying $f(S) \geq F((b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})})$. Since S_2 is produced by OfflineAlg, whose approximation ratio is α, this implies $\mathbb{E}[f(S_2)] \geq \alpha \cdot F((b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})})$. Thus, to prove the lemma it suffices to show that $F((b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})})$ is always at least $b(1 - p - cb)\tau + (1 - b) \cdot f(\mathrm{OPT} \cap \mathrm{supp}(\hat{x}))$.

The first step towards proving the last inequality is getting a lower bound on $F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})})$. Recall that we already showed in the proof of Lemma 18 that

$$\partial_e F(\hat{x}) < \frac{c\tau}{k} \quad \forall\, e \in \mathrm{OPT} \setminus \mathrm{supp}(\hat{x}) \ .$$

Thus, the submodularity of f implies

$$\begin{aligned} F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT}}) &\leq F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})}) + \sum_{e\in\mathrm{OPT}\setminus\mathrm{supp}(\hat{x})} \partial_e F(\hat{x}) \\ &\leq F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})}) + \frac{c\tau \cdot |\mathrm{OPT} \setminus \mathrm{supp}(\hat{x})|}{k} = F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})}) + cb\tau \ . \end{aligned}$$

Rearranging this inequality yields

$$F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})}) \geq F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT}}) - cb\tau \geq (1-p)\cdot f(\mathrm{OPT}) - cb\tau \geq (1-p-cb)\tau \ ,$$

where the second inequality holds by Corollary 2 since Observation 14 guarantees that every coordinate of $\hat{x}$ is either 0 or p. This gives us the promised lower bound on $F(\hat{x}\vee\mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})})$.

We now note that the submodularity of f implies that $F((t\cdot\hat{x}) \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})})$ is a concave function of t within the range $[0, 1]$. Since b is inside this range,

$$\begin{aligned} F((b\hat{x}) \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})}) &\geq b\cdot F(\hat{x} \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{x})}) + (1-b)\cdot f(\mathrm{OPT}\cap\mathrm{supp}(\hat{x})) \\ &\geq b(1-p-cb)\tau + (1-b)\cdot f(\mathrm{OPT}\cap\mathrm{supp}(\hat{x})) \ , \end{aligned}$$

which completes the proof of the lemma. ◀

Using the last two claims we can now obtain a lower bound on the value of the solution of Algorithm 3 in the case of $\|\hat{x}\|_1 < k$ which is a function of α, τ and p alone. We note that both the guarantees of Corollary 19 and Lemma 20 are lower bounds on the expected value of the output of the algorithm in this case since $\mathbb{E}[f(S_1)] \geq F(\hat{x})$. Thus, any convex combination of these guarantees is also such a lower bound, and the proof of the following corollary basically proves a lower bound for one such convex combination – for the specific value of c stated in the corollary.

▶ **Corollary 21.** *If $\|\hat{x}\|_1 < k$ and c is set to $\frac{\alpha(1-p)}{\alpha+1}$, then $\mathbb{E}[\max\{f(S_1), f(S_2)\}] \geq \frac{(1-p)\alpha\tau}{\alpha+1}$.*

Proof. The corollary follows immediately from the non-negativity of f when $p = 1$. Thus, we may assume $p < 1$ in the rest of the proof.

By the definition of S_1, $\mathbb{E}[f(S_1)] \geq F(\hat{x})$. Thus, by Corollary 19 and Lemma 20,

$$\begin{aligned} &\mathbb{E}[\max\{f(S_1), f(S_2)\}] \geq \max\{\mathbb{E}[f(S_1)], \mathbb{E}[f(S_2)]\} \\ &\qquad \geq \max\{[p(1-p) - bc]\tau + (1-p)^2 \cdot f\big(\mathrm{OPT}\setminus\mathrm{supp}(\hat{x})\big), \\ &\qquad\quad \alpha b(1-p-cb)\tau + \alpha(1-b)\cdot f(\mathrm{OPT}\cap\mathrm{supp}(\hat{x}))\} \\ &\qquad \geq \frac{\alpha(1-b)}{\alpha(1-b)+(1-p)^2}\cdot\big[[p(1-p)-bc]\tau + (1-p)^2\cdot f\big(\mathrm{OPT}\setminus\mathrm{supp}(\hat{x})\big)\big] \\ &\qquad\quad + \frac{(1-p)^2}{\alpha(1-b)+(1-p)^2}\cdot[\alpha b(1-p-cb)\tau + \alpha(1-b)\cdot f(\mathrm{OPT}\cap\mathrm{supp}(\hat{x}))] \ . \end{aligned}$$

To keep the following calculations short, it will be useful to define $q = 1-p$ and $d = 1-b$. Using this notation and the fact that the submodularity and non-negativity of f guarantee together $f\big(\mathrm{OPT}\setminus\mathrm{supp}(\hat{x})\big) + f\big(\mathrm{OPT}\cap\mathrm{supp}(\hat{x})\big) \geq f(\mathrm{OPT}) \geq \tau$, the previous inequality implies

$$\begin{aligned}
\frac{\mathbb{E}[\max\{f(S_1), f(S_2)\}]}{\alpha\tau} &\geq \frac{(1-b)[p(1-p)-bc]+b(1-p)^2(1-p-bc)+(1-b)(1-p)^2}{\alpha(1-b)+(1-p)^2}\\
&= \frac{d[q(1-q)-(1-d)c]+q^2(1-d)[q-(1-d)c]+dq^2}{\alpha d+q^2}\\
&= \frac{d[q-(1-d)c]+q^2(1-d)[q-(1-d)c]}{\alpha d+q^2} = \frac{[d+q^2(1-d)][q-(1-d)c]}{\alpha d+q^2}\\
&= \frac{q[d+q^2(1-d)](d\alpha+1)}{(\alpha+1)(d\alpha+q^2)} = \frac{d^2\alpha+d\alpha q^2-d^2\alpha q^2+d+q^2-dq^2}{d\alpha+q^2}\cdot\frac{q}{\alpha+1} \enspace , \qquad (1)
\end{aligned}$$

where the fourth equality holds by plugging in the value we assume for c.

The second fraction in the last expression is independent of the value of d, and the derivative of the first fraction in this expression as a function of d is

$$\begin{aligned}
&\frac{(2d\alpha+\alpha q^2-2d\alpha q^2+1-q^2)[d\alpha+q^2]-\alpha(d^2\alpha+d\alpha q^2-d^2\alpha q^2+d+q^2-dq^2)}{[d\alpha+q^2]^2}\\
&\qquad = \frac{1-q^2}{[d\alpha+q^2]^2}\cdot[q^2(1-\alpha)+d\alpha(d\alpha+2q^2)] \enspace ,
\end{aligned}$$

which is always non-negative since both q and α are numbers between 0 and 1. Thus, we get that the minimal value of the expression (1) is obtained for $d = 0$ for any choice of q and α. Plugging this value into d yields

$$\mathbb{E}[\max\{f(S_1), f(S_2)\}] \geq \frac{q\alpha\tau}{\alpha+1} = \frac{(1-p)\alpha\tau}{\alpha+1} \enspace . \qquad \blacktriangleleft$$

Note that Lemma 16 and Corollary 21 prove the same lower bound on the expectation $\mathbb{E}[\max\{f(S_1), f(S_2)\}]$ when c is set to the value it is set to in Corollary 21 (because $\mathbb{E}[\max\{f(S_1), f(S_2)\}] \geq \mathbb{E}[f(S_1)] \geq F(\hat{x})$). Thus, we can summarize the results we have proved so far using the following proposition.

▶ **Proposition 22.** *Algorithm 3 is a semi-streaming algorithm storing $O(k/p)$ elements. Moreover, for the value of the parameter c given in Corollary 21, the output set produced by this algorithm has an expected value of at least $\frac{\alpha\tau(1-p)}{\alpha+1}$.*

Using the last proposition, we can now prove the following theorem. As discussed at the beginning of the section, in the full version of this paper we explain how the assumption that τ is known can be dropped at the cost of increasing of a slight increase in the number of of elements stored by the algorithm, which yields Theorem 12.

▶ **Theorem 23.** *For every constant $\varepsilon \in (0, 1]$, there exists a semi-streaming algorithm that assumes access to an estimate τ of $f(\mathrm{OPT})$ obeying $(1-\varepsilon/8)\cdot f(\mathrm{OPT}) \leq \tau \leq f(\mathrm{OPT})$ and provides $(\frac{\alpha}{1+\alpha}-\varepsilon)$-approximation for the problem of maximizing a non-negative submodular function subject to cardinality constraint. This algorithm stores at most $O(k\varepsilon^{-1})$ elements.*

Proof. Consider the algorithm obtained from Algorithm 3 by setting $p = \varepsilon/2$ and c as is set in Corollary 21. By Proposition 22, this algorithm stores only $O(k/p) = O(k\varepsilon^{-1})$ elements, and the expected value of its output set is at least

$$\frac{\alpha\tau(1-p)}{\alpha+1} \geq \frac{\alpha(1-\varepsilon/8)(1-\varepsilon/2)}{\alpha+1}\cdot f(\mathrm{OPT}) \geq \frac{\alpha(1-\varepsilon)}{\alpha+1}\cdot f(\mathrm{OPT}) \geq \left(\frac{\alpha}{\alpha+1}-\varepsilon\right)\cdot f(\mathrm{OPT}) \enspace ,$$

where the first inequality holds since τ obeys, by assumption, $\tau \geq (1-\varepsilon/8)\cdot f(\mathrm{OPT})$. ◀

Further discussion. Before concluding, let us discuss in more detail the value that should be assigned to the parameter p of Algorithm 3. In the proof of Theorem 23, we chose p to be very small. This makes sense whenever α is independent of p since the formula given by Proposition 22 for the guaranteed value of the output is non-increasing in p. However, for some choices of OfflineAlg the value of α might depend on p, and thus, it might be beneficial to choose a value for p which is not very small. To see why α might depend on p, note that the input passed to OfflineAlg by Algorithm 3 is of size at most $\lceil k/p \rceil$ due to Observation 14; and therefore, the ratio between the size of the ground set and k in this input is roughly $1/p$. Hence, α depends on p if the approximation ratio of OfflineAlg depends on the above ratio; which is the case, e.g., for one of the algorithms described in [6].

As a corollary of the above discussion, we get that for some offline algorithms a smart choice of p can yield a better approximation guarantee than the one stated in Theorem 23. At the current point this corollary is not very useful since the state-of-the-art offline approximation algorithm has an approximation ratio which is independent of p; however, this might change in the future.

References

1. Naor Alaluf and Moran Feldman. Making a sieve random: Improved semi-streaming algorithm for submodular maximization under a cardinality constraint. *CoRR*, abs/1906.11237, 2019. `arXiv:1906.11237`.
2. Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 671–680. ACM, 2014.
3. Rafael da Ponte Barbosa, Alina Ene, Huy L Nguyen, and Justin Ward. A new framework for distributed submodular maximization. In *IEEE Foundations of Computer Science (FOCS)*, pages 645–654, 2016.
4. Rafael D.P. Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. The power of randomization: Distributed submodular maximization on massive datasets. In *International Conference on Machine Learning (ICML)*, 2015.
5. Niv Buchbinder and Moran Feldman. Constrained submodular maximization via a non-symmetric technique. *Mathematics of Operations Research*, 44(3):988–1005, 2019.
6. Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In *SODA*, pages 1433–1452, 2014. `doi:10.1137/1.9781611973402.106`.
7. Niv Buchbinder, Moran Feldman, and Roy Schwartz. Online submodular maximization with preemption. *ACM Trans. Algorithms*, 15(3):30:1–30:31, 2019. `doi:10.1145/3309764`.
8. Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. `doi:10.1137/080733991`.
9. Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: Matchings, matroids, and more. *Mathematical Programming*, 154(1-2):225–247, 2015.
10. Chandra Chekuri. Personal communication, 2018.
11. Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 318–330. Springer, 2015.
12. Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *FOCS*, pages 575–584, 2010. `doi:10.1109/FOCS.2010.60`.

13 Alina Ene and Huy L Nguyen. Constrained submodular maximization: Beyond 1/e. In *IEEE Foundations of Computer Science (FOCS)*, pages 248–257. IEEE, 2016.

14 Alessandro Epasto, Vahab Mirrokni, and Morteza Zadimoghaddam. Bicriteria distributed submodular maximization in a few rounds. In *PACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 25–33, 2017.

15 Uriel Feige, Vahab S Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011.

16 Moran Feldman, Christopher Harshaw, and Amin Karbasi. Greed is good: Near-optimal submodular maximization via greedy optimization. In *Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*, pages 758–784, 2017. URL: `http://proceedings.mlr.press/v65/feldman17b.html`.

17 Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 732–742, 2018.

18 Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In *IEEE Foundations of Computer Science (FOCS)*, 2011. `doi:10.1109/FOCS.2011.46`.

19 Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness, 2020. To appear in STOC 2020.

20 Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2011.

21 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 *ICML*, pages 3311–3320, 2019. URL: `http://proceedings.mlr.press/v97/kazemi19a.html`.

22 Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in mapreduce and streaming. In *PACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 1–10, 2013.

23 Jon Lee, Vahab S Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In *ACM Symposium on Theory of Computing (STOC)*, pages 323–332, 2009.

24 Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Mathematics of Operations Research*, 35(4):795–806, 2010.

25 László Lovász. Submodular functions and convexity. In *Mathematical Programming The State of the Art, XIth International Symposium on Mathematical Programming, Bonn, Germany, August 23-27, 1982*, pages 235–257, 1982. `doi:10.1007/978-3-642-68874-4_10`.

26 Vahab Mirrokni and Morteza Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *ACM Symposium on Theory of Computing (STOC)*, 2015.

27 Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

28 Baharan Mirzasoleiman, Amin Karbasi, Ashwinkumar Badanidiyuru, and Andreas Krause. Distributed submodular cover: Succinctly summarizing massive data. In *Advances in Neural Information Processing Systems*, pages 2881–2889, 2015.

29 Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2049–2057, 2013.

30 G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions – I. *Mathematical Programming*, 14(1):265–294, 1978.

31 Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In *International Conference on Machine Learning*, pages 3826–3835, 2018.

32 Jan Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM J. Comput.*, 42(1):265–304, 2013. doi:10.1137/110832318.