http://pubsonline.informs.org/journal/moor

MATHEMATICS OF OPERATIONS RESEARCH
Articles in Advance, pp. 1–24
ISSN 0364-765X (print), ISSN 1526-5471 (online)

An Optimal Streaming Algorithm for Submodular Maximization with a Cardinality Constraint

Naor Alaluf,[a] Alina Ene,[b] Moran Feldman,[c] Huy L. Nguyen,[d] Andrew Suh[b]

[a] Department of Mathematics and Computer Science, Open University of Israel, Raanana 4353701, Israel; [b] Department of Computer Science, Boston University, Boston, Massachusetts 02215; [c] Department of Computer Science, University of Haifa, Haifa 3498838, Israel; [d] Khoury College of Computer Sciences, Northeastern University, Boston, Massachusetts 02115

Contact: naoralaluf@gmail.com (NA); aene@bu.edu, https://orcid.org/0000-0002-5818-1807 (AE); moranfe@cs.haifa.ac.il, https://orcid.org/0000-0002-1535-2979 (MF); hlnguyen@cs.princeton.edu (HLN); asuh9@bu.edu (AS)

Received: August 16, 2020
Revised: August 15, 2021
Accepted: October 5, 2021
Published Online in Articles in Advance: February 2, 2022

MSC2020 Subject Classifications: Primary: 90C27; secondary: 68W27

https://doi.org/10.1287/moor.2021.1224

Copyright: © 2022 INFORMS

Abstract. We study the problem of maximizing a *nonmonotone* submodular function subject to a cardinality constraint in the streaming model. Our main contribution is a single-pass (semi) streaming algorithm that uses roughly $O(k/\varepsilon^2)$ memory, where k is the size constraint. At the end of the stream, our algorithm postprocesses its data structure using any off-line algorithm for submodular maximization and obtains a solution whose approximation guarantee is $\alpha/(1+\alpha)-\varepsilon$, where α is the approximation of the off-line algorithm. If we use an exact (exponential time) postprocessing algorithm, this leads to $1/2-\varepsilon$ approximation (which is nearly optimal). If we postprocess with the state-of-the-art offline approximation algorithm, whose guarantee is $\alpha = 0.385$, we obtain a 0.2779-approximation in polynomial time, improving over the previously best polynomial-time approximation of 0.1715. It is also worth mentioning that our algorithm is combinatorial and deterministic, which is rare for an algorithm for nonmonotone submodular maximization, and enjoys a fast update time of $O(\varepsilon^{-2}(\log k + \log(1+\alpha)))$ per element.

Funding: The work of N. Alaluf and M. Feldman was supported in part by the Israel Science Foundation [Grant 1357/16]. The work of A. Ene and A. Suh was supported in part by the National Science Foundation (NSF) [CAREER Grant CCF-1750333 and Grants CCF-1718342 and III-1908510]. The work of H. L. Nguyen was supported in part by the NSF [CAREER Grant CCF-1750716 and Grant CCF-1909314].

Keywords: submodular maximization • cardinality constraint • semi-streaming algorithms

1. Introduction

In this paper, we study the problem of maximizing a *nonmonotone* 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, in which the goal is to select a small subset of the data that captures the salient features of the overall data set (Badanidiyuru et al. [4]). 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 because it makes it possible to analyze the data using algorithms that would be prohibitive to run on the entire data set.

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 because 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 in both the distributed and streaming settings, leading to many new results and insights (Badanidiyuru et al. [4], Barbosa et al. [5, 6], Chakrabarti and Kale [14], Chekuri et al. [17], Epasto et al. [21], Feldman et al. [25], Kumar et al. [33], Mirrokni and Zadimoghaddam [37], Mirzasoleiman et al. [38–40], Norouzi-Fard et al. [43]).

Despite this significant progress, several fundamental questions remain open in both the streaming and distributed settings. 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*—that is, 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 (Nemhauser et al. [42]), can be adapted to the streaming model (Badanidiyuru et al. [4], Kumar et al. [33]). 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 $f(\mathrm{OPT})/(2k)$, 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 $1/2$-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 (Badanidiyuru et al. [4]). The algorithm uses the maximum singleton value to make $O(\varepsilon^{-1}\log k)$ guesses for $f(\mathrm{OPT})$, and for each guess, it runs the single-threshold greedy algorithm, which leads to $(1/2-\varepsilon)$-approximation. Remarkably, this approximation guarantee is optimal in the streaming model even if we allow unbounded computational power: Feldman et al. [27] show that any algorithm for this problem that achieves an approximation better than $1/2$ requires $\Omega(n/k^3)$ 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 uses only $O(\varepsilon^{-1}k\log k)$ space.[1]

In contrast, the general problem with a nonmonotone objective has proven to be considerably more challenging. Even in the centralized setting, the greedy algorithm fails to achieve any constant approximation guarantee when the objective is nonmonotone. Thus, several approaches have been developed for handling nonmonotone objectives in this setting, including local search (Feige et al. [22], Lee et al. [35, 34]), continuous optimization (Buchbinder and Feldman [8], Ene and Nguyen [20], Feldman et al. [26]), and sampling (Buchbinder et al. [12], Feldman et al. [24]). The currently best approximation guarantee is 0.385 (Buchbinder and Feldman [8]), the strongest inapproximability is 0.491 (Gharan and Vondrák [29]), and it remains a long-standing open problem to settle the approximability of submodular maximization subject to a cardinality constraint.

Adapting these techniques to the streaming setting is challenging, and the approximation guarantees are weaker. The main approach for nonmonotone maximization in the streaming setting has been to extend the local search algorithm of Chakrabarti and Kale [14] from monotone to nonmonotone objectives. This approach was employed in a sequence of works (Chekuri et al. [17], Feldman et al. [25], Mirzasoleiman et al. [38]), leading to the currently best approximation of $(3+2\sqrt{2})^{-1} \approx 0.1715$.[2] This naturally leads to the following questions:

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

1.1. Our Contributions

In this work, we give an affirmative answer to all of these questions. Specifically, we give a streaming algorithm[3] that performs a single pass over the stream and outputs sets of size $O(k/\varepsilon)$ that can be postprocessed using any off-line algorithm for submodular maximization. The postprocessing is itself quite straightforward: we simply run the off-line algorithm on the output set to obtain a solution of size at most k. We show that, if the off-line algorithm achieves α-approximation, then we obtain $(\alpha/(1+\alpha)-\varepsilon)$-approximation. One can note that, if we postprocess using an exact (exponential time) algorithm, we obtain $(1/2-\varepsilon)$-approximation. This matches the inapproximability result proven by Feldman et al. [27] for the special case of a monotone objective function. Furthermore, we show that, in the nonmonotone case, any streaming algorithm guaranteeing $(1/2+\varepsilon)$-approximation for some positive constant ε must use in fact $\Omega(n)$ space.[4] 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 off-line setting is $\alpha = 0.385$ (Buchbinder and Feldman [8]). If we postprocess using this algorithm, we obtain 0.2779-approximation in polynomial time, improving over the previously best polynomial-time approximation of 0.1715 from Feldman et al. [25]. The offline algorithm of Buchbinder and Feldman [8] 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 Buchbinder et al. [12] that achieves $1/e$-approximation. Furthermore, any existing heuristic for the off-line problem can be used for postprocessing, exploiting their effectiveness beyond the worst case.

1.2. Variants of Our Algorithm

Essentially, every algorithm for nonmonotone submodular maximization includes a randomized component. Oftentimes this component is explicit, but in some cases, it takes more subtle forms such as maintaining multiple solutions that are viewed as the support of a distribution (Buchbinder and Feldman [7], Feldman et al. [24]) or using the multilinear extensions (which are defined via expectations) (Buchbinder and Feldman [8], Feldman et al. [26]). We present in this work three variants of our algorithm based on the preceding three methods of introducing a randomized component into the algorithm.

Perhaps the most straightforward way to introduce a randomized component into the single-threshold greedy algorithm is to use the multilinear extension as the objective function and include only fractions of elements in the solution (which corresponds to including the elements in the solution only with some bounded probability). This has the advantage of keeping the algorithm almost deterministic (in fact, completely deterministic when the multilinear extension can be evaluated deterministically), which allows for a relatively simple analysis of the algorithm and a low space complexity of $O(k\log\alpha^{-1}/\varepsilon^2)$. However, the time complexity of an algorithm obtained via this approach depends on the complexity of evaluating the multilinear extension, which, in general, can be quite high. In Appendix C, we describe and analyze a variant of our algorithm (named StreamProcessExtension), which is based on the multilinear extension.

To avoid the multilinear extension and its associated time-complexity penalty, one can use true randomization and pass every arriving element to single-threshold greedy only with a given probability. However, analyzing such a combination of single-threshold greedy with true 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 earlier in one of the results of Chekuri et al. [17]. Our approach for handling this issue is to consider two cases depending on the probability that the budget is filled up in a run (this is a good event because 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 in parallel 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. Another issue with true randomness is that some elements can be randomly discarded despite being highly desirable. Following ideas from distributed algorithms (Barbosa et al. [5, 6], Mirrokni and Zadimoghaddam [37]), this issue can also be solved by running multiple copies of the algorithm in parallel because such a run guarantees that, with high probability, every desirable element is processed by some copy. Using this approach, we get a variant of our algorithm named StreamProcessRandomized, which can be found in Appendix D. StreamProcessRandomized is combinatorial and fast (it has an update time of $\tilde{O}(\varepsilon^{-2})$ marginal value computations per element), but because of the heavy use of parallel runs, it has a slightly worse space complexity of $\tilde{O}(k/\varepsilon^3)$.

These two discussed variants of our algorithm appeared already in an earlier conference version of this paper (Alaluf et al. [1]). Our main result, however, is a new variant of our algorithm (named StreamProcess) based on the technique of maintaining multiple solutions and treating them as the support of a distribution. In retrospect, creating a variant based on this technique is natural because it combines the advantages of the two previous approaches. Specifically, we get an algorithm that is deterministic and combinatorial, has a relatively simple analysis, and enjoys a low space complexity of $O(k\varepsilon^{-2}\log\alpha^{-1})$ and a low update time of $O(\varepsilon^{-2}\log(k/\alpha))$ marginal value computations per element.

Although the last variant of our algorithm has the best time and space guarantees, we give also the two earlier variants for two reasons. The first reason is that they demonstrate the first use of techniques such as continuous extensions and random partitions in the context of streaming algorithms for submodular maximization and, thus, greatly expand the toolkit available in this context. These techniques have proven to be quite versatile in the sequential and distributed settings, and we hope that they will lead to further developments in the streaming setting as well. The second reason is that, although the asymptotic approximation guarantees of all three variants of our algorithm are identical given a black box off-line α-approximation algorithm, they might differ when the off-line algorithm has additional properties. For example, the approximation ratio of the off-line algorithm might depend on the ratio between k and the number of elements in its input (see Buchbinder et al. [12] for an example of such an algorithm). Given such an off-line algorithm, it might be beneficial to set the parameters controlling the number of elements passed to the off-line algorithm so that only a moderate number of elements is passed, which is a regime in which the three variants of our algorithm produce different approximation guarantees.

Hence, one of the first two variants of our algorithm might end up having a better approximation guarantee than that of the last variant if future research yields off-line algorithms with relevant properties.

1.3. Additional Related Work

The problem of maximizing a nonnegative monotone submodular function subject to a cardinality or a matroid constraint was studied (in the off-line model) already in the 1970s. In 1978, Nemhauser et al. [42] and Fisher et al. [28] showed that a natural greedy algorithm achieves an approximation ratio of $1-1/e \approx 0.632$ for this problem when the constraint is a cardinality constraint and an approximation ratio of $1/2$ for matroid constraints. The $1-1/e$-approximation ratio for cardinality constraints was shown to be optimal already in the same year by Nemhauser and Wolsey [41], but the best possible approximation ratio for matroid constraints was open for a long time. Only a decade ago, Călinescu et al. [13] managed to show that a more involved algorithm, known as "continuous greedy," can achieve $(1-1/e)$-approximation for this type of constraint, which is tight because matroid constraints generalize cardinality constraints.

Unlike the natural greedy algorithm, continuous greedy is a randomized algorithm, which raises an interesting question regarding the best possible approximation ratio for matroid constraints that can be achieved by a deterministic algorithm. Very recently, Buchbinder et al. [9] made a slight step toward answering this question. Specifically, they describe a deterministic algorithm for maximizing a monotone submodular function subject to a matroid constraint whose approximation ratio is 0.5008. This algorithm shows that the $1/2$-approximation of the greedy algorithm is not the right answer for the aforementioned question.

Many works also study the off-line problem of maximizing a nonnegative (not necessarily monotone) submodular function subject to a cardinality or matroid constraint (Buchbinder and Feldman [8], Buchbinder et al. [12], Chekuri et al. [19], Ene and Nguyen [20], Feldman [23], Feldman et al. [26]). The most recent of these works achieves an approximation ratio of 0.385 for both cardinality and matroid constraints (Buchbinder and Feldman [8]). In contrast, it is known that no polynomial-time algorithm can achieve an approximation ratio of 0.497 for cardinality constraints or 0.478 for matroid constraints, respectively (Gharan and Vondrák [29]).

The study of streaming algorithms for submodular maximization problems is related to the study of online algorithms for such problems. A partial list of works on algorithms of the last kind includes Azar et al. [3], Buchbinder et al. [10, 11], Chan et al. [15], Kapralov et al. [30], and Korula et al. [32].

2. Preliminaries

2.1. 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, for example, $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ to denote a vector in $\mathbb{R}^V$. For two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^V$, we let $\boldsymbol{x} \vee \boldsymbol{y}$ and $\boldsymbol{x} \wedge \boldsymbol{y}$ be the vectors such that $(\boldsymbol{x} \vee \boldsymbol{y})_e = \max\{\boldsymbol{x}_e, \boldsymbol{y}_e\}$ and $(\boldsymbol{x} \wedge \boldsymbol{y})_e = \min\{x_e, y_e\}$ for all $e \in V$. For a set $S \subseteq V$, we let $\mathbf{1}_S$ denote the indicator vector of S, that is, the vector that has one in every coordinate $e \in S$ and zero in every coordinate $e \in V \setminus S$. Given an element $e \in V$, we use $\mathbf{1}_e$ as 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 $\boldsymbol{p}$ such that $p_e = \Pr[e \in S]$ for all $e \in V$ (i.e., the expectation is applied coordinate-wise).

2.2. Submodular Functions

In this paper, we consider the problem of maximizing a nonnegative submodular function subject to a cardinality constraint. A set function $f : 2^V \to \mathbb{R}$ is submodular if $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ for all subsets $A, B \subseteq V$. Additionally, given a set $S \subseteq V$ and an element $e \in V$, we use $f(e \mid S)$ to denote the marginal contribution of e to S with respect to the set function f, that is, $f(e \mid S) = f(S \cup \{e\}) - f(S)$.

2.3. 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 $\mathtt{R}(\boldsymbol{x})$. For every vector $\boldsymbol{x} \in [0,1]^V$, $\mathtt{R}(\boldsymbol{x})$ 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 $\boldsymbol{x} \in [0,1]^V$ by

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

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

The second extension we need is known as the *Lovász extension*. Unlike the multilinear extension that explicitly appears in one of the variants of our algorithm, the Lovász extension is not part of any of these variants. However, it plays a central role in the analyses of two of them. The Lovász extension $\hat{f} : [0,1]^V \to \mathbb{R}$ is defined as follows. For every $\boldsymbol{x} \in [0,1]^V$, $\hat{f}(\boldsymbol{x}) = \mathbb{E}_{\theta\sim[0,1]}[f(\{e \in V : x_e \geq \theta\})]$, 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 nonnegative submodular function has the following properties: (1) convexity: $c\hat{f}(\boldsymbol{x}) + (1-c)\hat{f}(\boldsymbol{y}) \geq \hat{f}(c\boldsymbol{x} + (1-c)\boldsymbol{y})$ for all $\boldsymbol{x}, \boldsymbol{y} \in [0,1]^V$ and all $c \in [0, 1]$ (Lovász [36]); (2) restricted scale invariance: $\hat{f}(c\boldsymbol{x}) \geq c\hat{f}(\boldsymbol{x})$ for all $\boldsymbol{x} \in [0,1]^V$ and all $c \in [0,1]$; (3) it lower bounds the multilinear extension, that is, $F(\boldsymbol{x}) \geq \hat{f}(\boldsymbol{x})$ for every $\boldsymbol{x} \in [0,1]^V$ (Vondrák [44, lemma A.4]).

3. Simplified Algorithm

The properties of our main algorithm (StreamProcess—the third variant discussed earlier) are summarized by the following theorem.

Theorem 3.1. *Assume there exists an α-approximation off-line algorithm OfflineAlg for maximizing a nonnegative submodular function subject to a 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 $(\alpha/(1+\alpha) - \varepsilon)$-approximation semistreaming algorithm for maximizing a nonnegative submodular function subject to a cardinality constraint. The algorithm stores $O(k\varepsilon^{-2})$ elements and makes $O(\varepsilon^{-2}\log k)$ marginal value computations while processing each arriving element.*[5] *Furthermore, if OfflineAlg is deterministic, then so is the algorithm that we get.*

In this section, we introduce a simplified version of the algorithm used to prove Theorem 3.1. This simplified version (given as Algorithm 1) captures our main new ideas but makes the simplifying assumption that it has access to an estimate τ of $f(\mathrm{OPT})$ obeying $(1-\varepsilon/2)\cdot f(\mathrm{OPT}) \leq \tau \leq f(\mathrm{OPT})$. Such an estimate can be produced using well-known techniques at the cost of a slight increase in the space complexity and update time of the algorithm. More specifically, in Section 4, we formally show that one such technique from Kazemi et al. [31] can be used for that purpose and it increases the space complexity and update of the algorithm only by factors of $O(\varepsilon^{-1}\log\alpha^{-1})$ and $O(\varepsilon^{-1}\log(k/\alpha))$, respectively.

Algorithm 1 gets two parameters: the approximation ratio α of OfflineAlg and an integer $p \geq 1$. The algorithm maintains p solutions $S_1, S_2, \ldots, S_p$. All these solutions start empty, and the algorithm may add each arriving element to at most one of them. Specifically, when an element e arrives, the algorithm checks whether there exists a solution S_i such that (1) S_i does not already contain k elements, and (2) the marginal contribution of e with respect to S_i exceeds the threshold of $c\tau/k$ for $c = \alpha/(1+\alpha)$. If there is such a solution, the algorithm adds e to it (if there are multiple such solutions, the algorithm adds e to an arbitrary one of them); otherwise, the algorithm discards e. After viewing all of the elements, Algorithm C.1 generates one more solution S_o by executing OfflineAlg on the union of the p solutions $S_1, S_2, \ldots, S_p$. The output of the algorithm is then the best solution among the $p+1$ generated solutions.

Algorithm 1 (StreamProcess, Simplified (p, α))

1. Let $c \leftarrow \alpha/(1+\alpha)$.
2. **for** $i = 1$ **to** p **do** Let $S_i \leftarrow \emptyset$.
3. **for** each arriving element e **do**
4. ⎸ **if** there exists an integer $1 \leq i \leq p$ such that $|S_i| < k$ and $f(e \mid S_i) \geq \frac{c\tau}{k}$, **then**
5. ⎸⎸ Update $S_i \leftarrow S_i \cup \{e\}$ (if there are multiple options for i, pick an arbitrary one).
6. Find another feasible solution $S_o \subseteq \cup_{i=1}^{p} S_i$ by running OfflineAlg with $\cup_{i=1}^{p} S_i$ as the ground set.
7. **return** the solution maximizing f among S_o and the p solutions $S_1, S_2, \ldots, S_p$.

Because Algorithm 1 stores elements only in the $p+1$ solutions it maintains and all these solutions are feasible (and, thus, contain only k elements), we immediately get the following observation. Note that this observation implies (in particular) that Algorithm 1 is a semistreaming algorithm for a constant p when the space complexity of OfflineAlg is nearly linear.

Observation 3.1. Algorithm 1 stores $O(pk)$ elements and makes at most p marginal value calculations while processing each arriving element.

We now divert our attention to analyzing the approximation ratio of Algorithm 1. Let us denote by $\hat{S}_i$ the final set S_i (i.e., the content of this set when the stream ends) and consider two cases. The first (easy) case is when at

least one of the solutions $S_1, S_2, \ldots, S_p$ reaches a size of k. The next lemma analyzes the approximation guarantee of Algorithm 1 in this case.

Lemma 3.1. *If there is an integer $1 \le i \le p$ such that $|\hat{S}_i| = k$, then the output of Algorithm 1 has value of at least $\frac{\alpha\tau}{1+\alpha}$.*

Proof. Denote by $e_1, e_2, \ldots, e_k$ the elements of $\hat{S}_i$ in the order of their arrival. Using this notation, the value of $f(\hat{S}_i)$ can be written as follows:

$$f(\hat{S}_i) = f(\emptyset) + \sum_{j=1}^{k} f(e_j \mid \{e_1, e_2, \ldots, e_{j-1}\}) \ge \sum_{i=1}^{k} \frac{\alpha\tau}{(1+\alpha)k} = \frac{\alpha\tau}{1+\alpha},$$

where the inequality holds because the nonnegativity of f implies $f(\emptyset) \ge 0$ and Algorithm 1 adds an element e_j to S_i only when $f(e_j \mid \{e_1, e_2, \ldots, e_{j-1}\}) \ge \frac{c\tau}{k} = \frac{\alpha\tau}{(1+\alpha)k}$. The lemma now follows because the solution output by Algorithm 1 is at least as good as S_i. □

Consider now the case in which no set S_i reaches the size of k. In this case, our objective is to show that at least one of the solutions computed by Algorithm 1 has a large value. Lemmas 3.3 and 3.4 lower bound the value of the average solution among $S_1, S_2, \ldots, S_p$ and the solution S_o, respectively. The proof of Lemma 3.3 uses the following known lemma.

Lemma 3.2 (Buchbinder et al. [12, Lemma 2.2]). *Let $f : 2^V \to \mathbb{R}_{\ge 0}$ be a nonnegative submodular function. Denote by $A(p)$ a random subset of A, where each element appears with probability at most p (not necessarily independently). Then, $\mathbb{E}[f(A(p))] \ge (1-p) \cdot f(\emptyset)$.*

Let $O = \mathrm{OPT} \setminus \cup_{i=1}^{p} \hat{S}_i$, and let $b = |O|/k$.

Lemma 3.3. *If $|\hat{S}_i| < k$ for every integer $1 \le i \le p$, then for every fixed set $A \subseteq V$, $p^{-1} \cdot \sum_{i=1}^{p} f(\hat{S}_i \cup A) \ge (1-p^{-1}) \cdot f(O \cup A) - \alpha b\tau/(1+\alpha)$.*

Proof. The *elements* in O were rejected by Algorithm 1. Because no set S_i reaches a size of k, this means that the marginal contribution of the elements of O with respect to every set S_i at the time of their arrival was smaller than $c\tau/k$. Moreover, because Algorithm 1 only adds elements to its solutions during its execution, the submodularity of f guarantees that the marginals of the elements of O are below this threshold also with respect to $\hat{S}_1 \cup A, \hat{S}_2 \cup A, \ldots, \hat{S}_p \cup A$. More formally, we get

$$f(e \mid \hat{S}_i \cup A) < \frac{c\tau}{k} = \frac{\alpha\tau}{k(1+\alpha)} \quad \forall\, e \in O \text{ and integer } 1 \le i \le p.$$

Using the submodularity of f again, this implies that, for every integer $1 \le i \le p$,

$$f(\hat{S}_i \cup O \cup A) \le f(\hat{S}_i \cup A) + \sum_{e \in O} f(e \mid \hat{S}_i \cup A) \le f(\hat{S}_i \cup A) + |O| \cdot \frac{\alpha\tau}{k(1+\alpha)} = f(\hat{S}_i \cup A) + \frac{\alpha b\tau}{1+\alpha}.$$

Adding up these inequalities (and dividing by p), we get

$$p^{-1} \cdot \sum_{i=1}^{p} f(\hat{S}_i \cup A) \ge p^{-1} \cdot \sum_{i=1}^{p} f(\hat{S}_i \cup O \cup A) - \frac{\alpha b\tau}{1+\alpha}.$$

We now note that $p^{-1} \cdot \sum_{i=1}^{p} f(\hat{S}_i \cup O \cup A)$ can be viewed as the expected value of a nonnegative submodular function $g(S) = f(S \cup O \cup A)$ over a random set S that is equal to every one of the sets $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_p$ with probability $1/p$. Because the sets $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_p$ are disjoint, S contains every element with probability at most $1/p$, and thus, by Lemma 3.2,

$$p^{-1} \cdot \sum_{i=1}^{p} f(\hat{S}_i \cup O \cup A) = \mathbb{E}[g(S)] \ge (1-p^{-1}) \cdot g(\emptyset) = (1-p^{-1}) \cdot f(O \cup A).$$

The lemma now follows by combining the last two inequalities. □

As mentioned, our next step is to get a lower bound on the value of $f(S_o)$. One easy way to get such a lower bound is to observe that $\mathrm{OPT} \setminus O$ is a subset of $\cup_{i=1}^{p} \hat{S}_i$ of size at most k and, thus, is a feasible solution for the instance faced by the algorithm OfflineAlg used to find S_o, which implies $\mathbb{E}[f(S_o)] \ge \alpha \cdot f(\mathrm{OPT} \setminus O)$ because OfflineAlg is an α-approximation algorithm. The following lemma proves a more involved lower bound by

considering the vectors $(b\mathbf{1}_{\hat{S}_i}) \vee \mathbf{1}_{\mathrm{OPT}\setminus O}$ as feasible fractional solutions for the same instance (using rounding methods, such as pipage rounding or swap rounding (Călinescu et al. [13], Chekuri et al. [18]), such feasible factional solutions can be converted into integral feasible solutions of at least the same value).

Lemma 3.4. *If $|\hat{S}_i| < k$ for every integer $1 \le i \le p$, then $\mathbb{E}[f(S_o)] \ge \alpha b\tau(1 - p^{-1} - \alpha b/(1+\alpha)) + \alpha(1-b) \cdot f(\mathrm{OPT} \setminus O)$.*

Proof. Fix some integer $1 \le i \le p$ and consider the vector $(b\mathbf{1}_{\hat{S}_i}) \vee \mathbf{1}_{\mathrm{OPT}\setminus O}$. Clearly,

$$\|(b\mathbf{1}_{\hat{S}_i}) \vee \mathbf{1}_{\mathrm{OPT}\setminus O}\|_1 \le b \cdot |\hat{S}_i| + |\mathrm{OPT} \setminus O| \le |O| + |\mathrm{OPT} \setminus O| = |\mathrm{OPT}| \le k,$$

where the second inequality holds by the definition of b because $|\hat{S}_i| < k$. Given the last property of the vector $(b\mathbf{1}_{\hat{S}_i}) \vee \mathbf{1}_{\mathrm{OPT}\setminus O}$, standard rounding techniques such as pipage rounding (Călinescu et al. [13]) and swap rounding (Chekuri et al. [18]) can be used to produce from this vector a set $A_i \subseteq \hat{S}_i \cup (\mathrm{OPT} \setminus O) \subseteq \cup_{i=1}^{p} \hat{S}_i$ of size at most k such that $f(A_i) \ge F((b\mathbf{1}_{\hat{S}_i}) \vee \mathbf{1}_{\mathrm{OPT}\setminus O})$. Because S_o is produced by the algorithm OfflineAlg whose approximation ratio is α, and A_i is one possible output for this algorithm, this implies $\mathbb{E}[f(S_o)] \ge \alpha \cdot f(A_i) \ge \alpha \cdot F((b\mathbf{1}_{\hat{S}_i}) \vee \mathbf{1}_{\mathrm{OPT}\setminus O})$. Furthermore, by averaging this equation over all the possible values of i, we get

$$\begin{aligned}
\mathbb{E}[f(S_o)] &\ge \alpha p^{-1} \cdot \sum_{i=1}^{p} F((b\mathbf{1}_{\hat{S}_i}) \vee \mathbf{1}_{\mathrm{OPT}\setminus O}) \\
&\ge \alpha p^{-1} \cdot \left[(1-b) \cdot \sum_{i=1}^{p} f(\mathrm{OPT} \setminus O) + b \cdot \sum_{i=1}^{p} f(\hat{S}_i \cup (\mathrm{OPT} \setminus O))\right] \\
&\ge \alpha(1-b) \cdot f(\mathrm{OPT} \setminus O) + \alpha b[(1-p^{-1}) \cdot f(\mathrm{OPT}) - \alpha b\tau/(1+\alpha)],
\end{aligned}$$

where the second inequality holds because the multilinear extension F of a submodular function f is known to be concave along nonnegative directions (Călinescu et al. [13]), which implies that $F((t \cdot \mathbf{1}_{\hat{S}_i}) \vee \mathbf{1}_{\mathrm{OPT}\setminus O})$ is a concave function of t within the range $[0,1]$ (and b falls inside this range). The last inequality holds by Lemma 3.3 (for $A = \mathrm{OPT} \setminus O$), and the lemma now follows by recalling that $f(OPT) \ge \tau$. □

Combining the guarantees of the last two lemmas, we can now obtain a lower bound on the value of the solution of Algorithm 1 in the case that $|S_i| < k$ for every integer $1 \le i \le p$, which is independent of b.

Corollary 3.1. *If $|\hat{S}_i| < k$ for every integer $1 \le i \le p$, then the output of Algorithm 1 has value of at least $\left(\frac{\alpha}{\alpha+1} - 2p^{-1}\right)\tau$.*

Proof. The corollary follows immediately from the nonnegativity of f when $p = 1$. Thus, we may assume $p \ge 2$ in the rest of the proof.

Because Algorithm 1 outputs the best solution among the $p+1$ solutions it creates, we get by Lemmas 3.3 (for $A = \emptyset$) and 3.4 that the value of the solution it outputs is at least

$$\begin{aligned}
&\max\left\{p^{-1}\sum_{i=1}^{p} f(\hat{S}_i), f(S_o)\right\} \ge \max\{(1-p^{-1}) \cdot f(O) - \alpha b\tau/(1+\alpha), \\
&\quad \alpha b\tau(1 - p^{-1} - \alpha b/(1+\alpha)) + \alpha(1-b) \cdot f(\mathrm{OPT} \setminus O)\} \\
&\ge \frac{\alpha(1-b)}{\alpha(1-b) + 1 - p^{-1}} \cdot [(1-p^{-1}) \cdot f(O) - \alpha b\tau/(1+\alpha)] \\
&\quad + \frac{1-p^{-1}}{\alpha(1-b) + 1 - p^{-1}} \cdot [\alpha b\tau(1 - p^{-1} - \alpha b/(1+\alpha)) + \alpha(1-b) \cdot f(\mathrm{OPT} \setminus O)].
\end{aligned}$$

Note now that the submodularity and nonnegativity of f guarantee together $f(O) + f(\mathrm{OPT} \setminus O) \ge f(\mathrm{OPT}) \ge \tau$. Using this fact and the nonnegativity of f, the previous inequality yields

$$\begin{aligned}
&\frac{\max\left\{p^{-1}\sum_{i=1}^{p} f(\hat{S}_i), f(S_o)\right\}}{\alpha\tau} \\
&\ge \max\left\{0, \frac{(1-b)(1-p^{-1}) - \alpha b(1-b)/(1+\alpha) + b(1-p^{-1})(1 - p^{-1} - \alpha b/(1+\alpha))}{\alpha(1-b) + 1 - p^{-1}}\right\} \\
&\ge \max\left\{0, \frac{(1-b)(1-p^{-1}) - \alpha b(1-b)/(1+\alpha) + b(1-p^{-1})(1 - p^{-1} - \alpha b/(1+\alpha))}{\alpha(1-b) + 1}\right\} \\
&\ge \frac{(1-b) - \alpha b(1-b)/(1+\alpha) + b(1 - \alpha b/(1+\alpha))}{\alpha(1-b) + 1} - 2p^{-1},
\end{aligned}$$

where the last two inequalities hold because $\alpha \in (0,1]$, $b \in [0,1]$ and $p \geq 2$. Simplifying the last inequality, we get

$$\frac{\max\left\{p^{-1}\sum_{i=1}^{p} f(\hat{S}_i), f(S_o)\right\}}{\alpha\tau} \geq \frac{(1-b)(1+\alpha)-\alpha b(1-b)+b(1+\alpha-\alpha b)}{(1+\alpha)[\alpha(1-b)+1]} - 2p^{-1}$$
$$= \frac{1+\alpha(1-b)}{(1+\alpha)[\alpha(1-b)+1]} - 2p^{-1} = \frac{1}{1+\alpha} - 2p^{-1}.$$

The corollary now follows by rearranging this inequality (and recalling that $\alpha \in (0,1]$). □

Note that the guarantee of Corollary 3.1 (for the case it considers) is always weaker than the guarantee of Lemma 3.1. Thus, we can summarize the results we have proven so far using the following proposition.

Proposition 3.1. *Algorithm 1 stores $O(pk)$ elements and makes at most p marginal value calculations while processing each arriving element, and its output set has an expected value of at least $(\alpha/(1+\alpha) - 2p^{-1})\tau$.*

Using the last proposition, we can now prove the following theorem. As discussed at the beginning of the section, in Section 4, we explain how the assumption that τ is known can be dropped at the cost of a slight increase in the number of elements stored by the algorithm and its update time, which yields Theorem 3.1.

Theorem 3.2. *Assume there exists an α-approximation off-line algorithm OfflineAlg for maximizing a nonnegative submodular function subject to a 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 a semistreaming algorithm that assumes access to an estimate τ of $f(\mathrm{OPT})$ obeying $(1-\varepsilon/2)\cdot f(\mathrm{OPT}) \leq \tau \leq f(\mathrm{OPT})$ and provides a $(\alpha/(1+\alpha)-\varepsilon)$-approximation for the problem of maximizing a nonnegative submodular function subject to a cardinality constraint. This algorithm stores $O(k\varepsilon^{-1})$ elements and calculates $O(\varepsilon^{-1})$ marginal values while processing each arriving element.*

Proof. Consider the algorithm obtained from Algorithm 1 by setting $p = \lceil 4/\varepsilon \rceil$. By Proposition 3.1 and the nonnegativity of f, this algorithm stores only $O(pk) = O(k\varepsilon^{-1})$ elements and calculates only $p = O(\varepsilon^{-1})$ marginal values while processing each arriving element, and the expected value of its output set is at least

$$\max\left\{0, \left(\frac{\alpha}{\alpha+1} - 2p^{-1}\right)\tau\right\} \geq \left(\frac{\alpha}{\alpha+1} - \frac{\varepsilon}{2}\right)\cdot(1-\varepsilon/2)\cdot f(\mathrm{OPT}) \geq \left(\frac{\alpha}{\alpha+1} - \varepsilon\right)\cdot f(\mathrm{OPT}),$$

where the first inequality holds because $p \geq 4/\varepsilon$ and τ obeys, by assumption, $\tau \geq (1-\varepsilon/2)\cdot f(\mathrm{OPT})$. □

4. Complete Algorithm

In this section, we explain how one can drop the assumption from Section 3 that the algorithm has access to an estimate τ of $f(\mathrm{OPT})$. This completes the proof of Theorem 3.1.

Technically, we analyze in this section the variant of Algorithm 1 given as Algorithm 2. It gets the same two parameters α and p received by Algorithm 1 plus an additional parameter $\varepsilon' \in (0,1)$ controlling the guaranteed quality of the output. The algorithm is based on a technique originally from Kazemi et al. [31]. Throughout its execution, Algorithm 2 tracks in m a lower bound on the value of $f(\mathrm{OPT})$. The algorithm also maintains a set $T = \{(1+\varepsilon')^i | m/(1+\varepsilon') \leq (1+\varepsilon')^i \leq mk/c, i \in \mathbb{Z}\}$ of values that, given the current value of the lower bound m, are (1) possible estimates for $f(\mathrm{OPT})$ at the current point and (2) not so large that they dwarf this lower bound (of course, T includes only a subset of the possible estimates obeying these requirements). For every estimate τ in T, the algorithm maintains p sets $S_1^\tau, S_2^\tau, \ldots, S_p^\tau$. We note that the set of solutions maintained is updated every time that T is updated (which happens after every update of m). Specifically, whenever a new value τ is added to T, the algorithm instantiate p new sets $S_1^\tau, S_2^\tau, \ldots, S_p^\tau$, and whenever a value τ is dropped from T, the algorithm deletes $S_1^\tau, S_2^\tau, \ldots, S_p^\tau$.

Although a value τ remains in T, Algorithm 2 maintains the sets $S_1^\tau, S_2^\tau, \ldots, S_p^\tau$ in exactly the same way that Algorithm 1 maintains its sets $S_1, S_2, \ldots, S_p$ given this value τ as an estimate for $f(\mathrm{OPT})$. Moreover, we show that, if τ remains in T when the algorithm terminates, then the contents of $S_1^\tau, S_2^\tau, \ldots, S_p^\tau$ when the algorithm terminates are equal to the contents of the sets $S_1, S_2, \ldots, S_p$ when Algorithm 1 terminates after executing with this τ as the estimate for $f(\mathrm{OPT})$. Thus, one can view Algorithm 2 as parallel execution of Algorithm 1 for many estimates of $f(\mathrm{OPT})$ at the same time. After viewing the last element, Algorithm 2 calculates for every $\tau \in T$ an output set $\overline{S}_\tau$

based on the sets $S_1^\tau, S_2^\tau, \ldots, S_p^\tau$ in the same way Algorithm 1 does that and then outputs the best output set computed for any $\tau \in T$.

Algorithm 2 (StreamProcess $(p, \alpha, \varepsilon')$)

1 Let $c \leftarrow \alpha/(1+\alpha)$.
2 Let $m \leftarrow f(\emptyset)$ and $T \leftarrow \{(1+\varepsilon')^h \mid m/(1+\varepsilon') \le (1+\varepsilon')^h \le mk/c, h \in \mathbb{Z}\}$.
3 **for** *each arriving element* e **do**
4 Let $m' \leftarrow \max\{f(\{e\}), \max_{\tau \in T, 1 \le i \le p} f(S_i^\tau)\}$.
5 **if** $m < m'$ **then**
6 Update $m \leftarrow m'$, and then $T \leftarrow \{(1+\varepsilon')^h \mid m/(1+\varepsilon') \le (1+\varepsilon')^h \le mk/c, h \in \mathbb{Z}\}$.
7 Delete $S_1^\tau, S_2^\tau, \ldots, S_p^\tau$ for every value τ removed from T in line 6.
8 Initialize $S_i^\tau \leftarrow \emptyset$ for every value τ added to T in line 6 and integer $1 \le i \le p$.
9 **for** *every* $\tau \in T$ **do**
10 **if** *there exists an integer* $1 \le i \le p$ *such that* $|S_i^\tau| < k$ *and* $f(e \mid S_i^\tau) \ge \frac{c\tau}{k}$ **then**
11 Update $S_i^\tau \leftarrow S_i^\tau \cup \{e\}$ (if there are multiple options for i, pick an arbitrary one).
12 **for** *every* $\tau \in T$ **do**
13 Find another feasible solution $S_o^\tau \subseteq \cup_{i=1}^p S_i^\tau$ by running OfflineAlg with $\cup_{i=1}^p S_i^\tau$ as the ground set.
14 Let $\overline{S}_\tau$ be the better solution among S_o^τ and the p solutions $S_1^\tau, S_2^\tau, \ldots, S_p^\tau$.
15 **return** *the best solution among* $\{\overline{S}_\tau\}_{\tau \in T}$ *or the empty set if* $T = \emptyset$.

We begin the analysis of Algorithm 2 by providing a basic lower bound on the value of each set S_i^τ. This lower bound can be viewed as a generalized counterpart of Lemma 3.1.

Lemma 4.1. *At every point during the execution of the algorithm, for every* $\tau \in T$ *and* $1 \le i \le p$, *it holds that* $f(S_i^\tau) \ge \alpha\tau \cdot |S_i^\tau|/[(1+\alpha)k]$.

Proof. Denote by $e_1, e_2, \ldots, e_{|S_i^\tau|}$ the elements of S_i^τ in the order of their arrival. Using this notation, the value of $f(S_i^\tau)$ can be written as follows:

$$f(S_i^\tau) = f(\emptyset) + \sum_{j=1}^{|S_i^\tau|} f(e_j \mid \{e_1, e_2, \ldots, e_{j-1}\}) \ge \sum_{i=1}^{|S_i^\tau|} \frac{\alpha\tau}{(1+\alpha)k} = \frac{\alpha\tau \cdot |S_i^\tau|}{(1+\alpha)k},$$

where the inequality holds because the nonnegativity of f implies $f(\emptyset) \ge 0$ and Algorithm 2 adds an element e_j to S_i^τ only when $f(e_j \mid \{e_1, e_2, \ldots, e_{j-1}\}) \ge c\tau/k = \alpha\tau/[(1+\alpha)k]$. □

Using the last lemma, we can now prove the following observation, which upper bounds the size of each set S_i^τ maintained by Algorithm 2 and, thus, serves as a first step toward bounding the space complexity of this algorithm.

Observation 4.1. At the end of every iteration of Algorithm 2, the size of the set S_i^τ is at most $mk/(c\tau) + 1$ for every $\tau \in T$ and integer $1 \le i \le p$.

Proof. Let $\overline{S}_i^\tau$ denote the content of the set S_i^τ at the beginning of the iteration. Because at most a single element is added to S_i^τ during the iteration, we get via Lemma 4.1 that the value of the set $\overline{S}_i^\tau$ is at least

$$\frac{\alpha\tau \cdot |\overline{S}_i^\tau|}{(1+\alpha)k} \ge \frac{\alpha\tau \cdot (|S_i^\tau| - 1)}{(1+\alpha)k} = \frac{c\tau \cdot (|S_i^\tau| - 1)}{k}.$$

The way in which Algorithm 2 updates m guarantees that, immediately after the update of m at the beginning of the iteration, the value of m was at least as large as the value of $\overline{S}_i^\tau$, and thus, we get

$$m \ge \frac{c\tau \cdot (|S_i^\tau| - 1)}{k},$$

which implies the observation. □

We are now ready to bound the space complexity and update time of Algorithm 2.

Lemma 4.2. *Algorithm 2 can be implemented so that it stores* $O(pk(\varepsilon')^{-1}\log\alpha^{-1})$ *elements and makes only* $O(p(\varepsilon')^{-1}\log(k/\alpha))$ *marginal value calculations when processing each arriving element.*

Proof. We begin the proof with two technical calculations. First, note that the number of estimates in T is upper bounded at all times by

$$1+\log_{1+\varepsilon'}\left(\frac{km/c}{m/(1+\varepsilon')}\right)=2+\frac{\ln k-\ln c}{\ln(1+\varepsilon')}\le 2+\frac{\ln k-\ln c}{2\varepsilon'/3}=O((\varepsilon')^{-1}(\log k-\log c)).$$

Second, note that

$$\begin{aligned}
&\sum_{\tau\in T}\sum_{i=1}^{p}(|S_i^\tau|-1)\le\sum_{\tau\in T}\min\left\{pk,\frac{pmk}{c\tau}\right\}\le\sum_{\substack{h\in\mathbb{Z}\\(1+\varepsilon')^{h+1}\ge m}}\min\left\{pk,\frac{pmk}{c(1+\varepsilon')^h}\right\}\\
&\le pk\cdot\left|\left\{h\in\mathbb{Z}\mid\frac{m}{1+\varepsilon'}\le(1+\varepsilon')^h<m/c\right\}\right|+\sum_{h=0}^{\infty}\frac{pk}{(1+\varepsilon')^h}\\
&\le pk\cdot\left(\log_{1+\varepsilon'}\left(\frac{m/c}{m}\right)+2\right)+\frac{pk}{1-1/(1+\varepsilon')}\le\frac{pk\cdot(\ln c^{-1}+2)}{\ln(1+\varepsilon')}+\frac{pk(1+\varepsilon')}{\varepsilon'}\\
&=O(pk(\varepsilon')^{-1}\log c^{-1})+O(pk(\varepsilon')^{-1})=O(pk(\varepsilon')^{-1}\log c^{-1}),
\end{aligned}$$

where the first inequality follows from Observation 4.1 and the fact that all the sets maintained by Algorithm 2 are of size at most k, the second inequality follows from the definition of T, and the third inequality holds because the first term on its right-hand side upper bounds the part of the sum that appears on its left-hand side corresponding to h values obeying $m\le(1+\varepsilon')^{h+1}\le m(1+\varepsilon')/c$, and the second term on the right-hand side of the inequality upper bounds the remaining part of the aforementioned sum.

We now observe that, when processing each arriving element, Algorithm 2 makes p marginal value calculations in association with every value $\tau\in T$ and another set of $O(p)$ such calculations that are not associated with any value of T (in line 4 of the algorithm). Thus, the total number of marginal value calculations made by the algorithm during the processing of an arriving element is

$$p\cdot|T|+O(p)=p\cdot O((\varepsilon')^{-1}(\log k-\log c))+O(p)=O(p(\varepsilon')^{-1}(\log k-\log c))=O(p(\varepsilon')^{-1}\log(k/c)).$$

We also note that Algorithm 2 has to store only the elements belonging to S_i^τ for some $\tau\in T$ and integer $1\le i\le p$. Thus, in all these sets together, the algorithm stores $O(pk(\varepsilon')^{-1}\log c^{-1})$ elements because

$$\begin{aligned}
\sum_{\tau\in T}\sum_{i=1}^{p}|S_i^\tau|&=\sum_{\tau\in T}\sum_{i=1}^{p}(|S_i^\tau|-1)+p|T|\\
&=O(pk(\varepsilon')^{-1}\log c^{-1})+p\cdot O((\varepsilon')^{-1}(\log k-\log c))=O(pk(\varepsilon')^{-1}\log c^{-1}).
\end{aligned}$$

To complete the proof of the lemma, it remains to note that $c=\alpha/(1+\alpha)\ge\alpha/2$. □

At this point, we divert our attention to analyzing the approximation guarantee of Algorithm 2. Let us denote by $\hat{m}$ and $\hat{T}$ the final values of m and T, respectively.

Observation 4.2. We always have $c\in(0,1/2]$, and thus, $\hat{T}$ is not empty unless $\hat{m}=0$.

Proof. Because $\alpha\in(0,1]$ and $\alpha/(\alpha+1)$ is an increasing function of α,

$$c=\frac{\alpha}{\alpha+1}\in\left(\frac{0}{0+1},\frac{1}{1+1}\right]=(0,1/2].$$ □

The last observation immediately implies that, when $\hat{T}$ is empty, all the singletons have zero values, and the same must be true for every other nonempty set by the submodularity and nonnegativity of f. Thus, $\mathrm{OPT}=\emptyset$, which makes the output of Algorithm 2 optimal in the rare case in which $\hat{T}$ is empty. Hence, we can assume from now on that $\hat{T}\neq\emptyset$. The following lemma shows that $\hat{T}$ contains a good estimate for $f(\mathrm{OPT})$ in this case.

Lemma 4.3. *The set* $\hat{T}$ *contains a value* $\hat{\tau}$ *such that* $(1-\varepsilon')\cdot f(\mathrm{OPT})\le\hat{\tau}\le f(\mathrm{OPT})$.

Proof. Observe that $\hat{m} \geq \max\{f(\emptyset), \max_{e \in V}\{f(\{e\})\}\}$. Thus, by the submodularity of f,

$$f(\mathrm{OPT}) \leq f(\emptyset) + \sum_{e \in \mathrm{OPT}} [f(\{e\}) - f(\emptyset)] \leq \max\left\{f(\emptyset), \sum_{e \in \mathrm{OPT}} f(\{e\})\right\} \leq k\hat{m} \leq \frac{k\hat{m}}{c}.$$

In contrast, one can note that m is equal to the value of f for some feasible solution, and thus, $f(\mathrm{OPT}) \geq \hat{m}$. Because $\hat{T}$ contains all the values of the form $(1+\varepsilon')^i$ in the range $[\hat{m}/(1+\varepsilon'), k\hat{m}/c]$, the preceding inequalities imply that it contains, in particular, the largest value of this form that is still not larger than $f(\mathrm{OPT})$. Let us denote this value by $\hat{\tau}$. By definition, $\hat{\tau} \leq f(\mathrm{OPT})$. Additionally,

$$\hat{\tau} \cdot (1+\varepsilon') \geq f(\mathrm{OPT}) \Rightarrow \hat{\tau} \geq \frac{f(\mathrm{OPT})}{1+\varepsilon'} \geq (1-\varepsilon') \cdot f(\mathrm{OPT}). \quad \square$$

Let us now concentrate on the value $\hat{\tau}$ whose existence is guaranteed by Lemma 4.3 and let $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_p$ denote the sets maintained by Algorithm 1 when it gets $\hat{\tau}$ as the estimate for $f(\mathrm{OPT})$. Additionally, let us denote by $e_1, e_2, \ldots, e_n$ the elements of V in the order of their arrival, and let e_j be the element whose arrival made Algorithm 2 add $\hat{\tau}$ to T (i.e., $\hat{\tau}$ was added to T by Algorithm 2 while processing e_j). If $\hat{\tau}$ belonged to T from the very beginning of the execution of Algorithm 2, then we define $j = 1$.

Lemma 4.4. *All the sets $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_p$ maintained by Algorithm 1 are empty immediately prior to the arrival of e_j.*

Proof. If $j = 1$, then the lemma is trivial. Thus, we assume $j > 1$ in the rest of this proof. Prior to the arrival of e_j, $\hat{\tau}$ was not part of T. Nevertheless, because $\hat{\tau} \in \hat{T}$, we must have at all times

$$\hat{\tau} \geq \frac{\hat{m}}{1+\varepsilon'} \geq \frac{m}{1+\varepsilon'},$$

where the second inequality holds because the value m only increases over time. Therefore, the reason that $\hat{\tau}$ did not belong to T prior to the arrival of e_j must have been that m was smaller than $c\hat{\tau}/k$. Because m is at least as large as the value of any singleton containing an element already viewed by the algorithm we get, for every two integers $1 \leq t \leq j-1$ and $1 \leq i \leq p$,

$$\frac{c\hat{\tau}}{k} > f(\{e_t\}) \geq f(\{e_t\} \mid \emptyset) \geq f(e_t | \hat{S}_i \wedge \mathbf{1}_{\{e_1, e_2, \ldots, e_{t-1}\}}),$$

where the second inequality follows from the nonnegativity of f and the last from its submodularity. Thus, e_t is not added by Algorithm 1 to any one of the sets $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_p$, which implies that all these sets are empty at the moment e_j arrives. $\square$

According to the preceding discussion, from the moment $\hat{\tau}$ gets into T, Algorithm 2 updates the sets $S_1^{\hat{\tau}}, S_2^{\hat{\tau}}, \ldots, S_p^{\hat{\tau}}$ in the same way that Algorithm 1 updates $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_p$ (note that, once $\hat{\tau}$ gets into T, it remains there for good because $\hat{\tau} \in \hat{T}$). Together with the previous lemma, which shows that $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_p$ are empty just like $S_1^{\hat{\tau}}, S_2^{\hat{\tau}}, \ldots, S_{p\hat{\tau}}$ at the moment e_j arrives, this implies that the final contents of $S_1^{\hat{\tau}}, S_2^{\hat{\tau}}, \ldots, S_p^{\hat{\tau}}$ are equal to the final contents of $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_p$, respectively. Because the set $\overline{S}_{\hat{\tau}}$ is computed based on the final contents of $S_1^{\hat{\tau}}, S_2^{\hat{\tau}}, \ldots, S_p^{\hat{\tau}}$ in the same way that the output of Algorithm 1 is computed based on the final contents of $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_p$, we get the following corollary.

Corollary 4.1. *If it is guaranteed that the approximation ratio of Algorithm 1 is at least β when $(1-\varepsilon') \cdot f(\mathrm{OPT}) \leq \tau \leq f(\mathrm{OPT})$ for some choice of the parameters α and p, then the approximation ratio of Algorithm 2 is at least β as well for this choice of α and p.*

We are now ready to prove Theorem 3.1.

Theorem 3.1 (Repeated for Clarity). *Assume there exists an α-approximation off-line algorithm OfflineAlg for maximizing a nonnegative submodular function subject to a 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 $(\alpha/(1+\alpha)-\varepsilon)$-approximation semistreaming algorithm for maximizing a nonnegative submodular function subject to a cardinality constraint. The algorithm stores $O(k\varepsilon^{-2}\log\alpha^{-1})$ elements and makes $O(\varepsilon^{-2}\log(k/\alpha))$ marginal value computations while processing each arriving element. Furthermore, if OfflineAlg is deterministic, then so is the algorithm that we get.*

Proof of Theorem 3.1. The proof of Theorem 3.2 shows that Algorithm 1 achieves an approximation guarantee of $\alpha/(1+\alpha)-\varepsilon$ when it has access to a value τ obeying $(1-\varepsilon/2)\cdot f(\mathrm{OPT}) \le \tau \le f(\mathrm{OPT})$ and its parameter p is set to $\lceil 4/\varepsilon \rceil$. According to Corollary 4.1, this implies that, by setting the parameter p of Algorithm 2 in the same way and setting ε' to $\varepsilon/2$, we get an algorithm whose approximation ratio is at least $\alpha/(1+\alpha)-\varepsilon$ and does not assume access to an estimate τ of $f(\mathrm{OPT})$.

It remains to bound the space requirement and update the time of the algorithm obtained in this way. Plugging the equalities $p = \lceil 4/\varepsilon \rceil$ and $\varepsilon' = \varepsilon/2$ into the guarantee of Lemma 4.2, we get that the algorithm we have obtained stores $O(k\varepsilon^{-2}\log\alpha^{-1})$ elements and makes $O(\varepsilon^{-2}\log(k/\alpha))$ marginal value calculations while processing each arriving element. □

Acknowledgments

An earlier version of this paper appeared in the 47th International Colloquium on Automata, Languages and Programming (ICALP 2020) (Alaluf et al. [1]).

Appendix A. Details About the Error in a Previous Work

As mentioned, Chekuri et al. [17] describes a semistreaming algorithm for the problem of maximizing a nonnegative (not necessarily monotone) submodular function subject to a cardinality constraint and claims an approximation ratio of roughly 0.212 for this algorithm. However, an error was later found in the proof of this result (Chekuri [16]) (the error does not affect the other results of Chekuri et al. [17]). For completeness, we briefly describe in this appendix the error found.

In the proof presented by Chekuri et al. [17], the output set of their algorithm is denoted by $\tilde{S}$. As is standard in the analysis of algorithms based on single-threshold greedy, the analysis distinguishes between two cases: one case in which $|\tilde{S}| = k$ and a second case in which $|\tilde{S}| < k$. To argue about the second case, the analysis then implicitly uses the inequality

$$\mathbb{E}[f(\tilde{S} \cup \mathrm{OPT}) \mid |S| < k] \ge \left(1 - \max_{e \in V} \Pr[e \in \tilde{S}]\right) \cdot f(\mathrm{OPT}). \tag{A.1}$$

It is claimed by Chekuri et al. [17] that this inequality follows from Lemma 3.2 (originally from Buchbinder et al. [12]). However, this lemma can yield only the inequalities

$$\mathbb{E}[f(\tilde{S} \cup \mathrm{OPT})] \ge \left(1 - \max_{e \in V} \Pr[e \in \tilde{S}]\right) \cdot f(\mathrm{OPT})$$

and

$$\mathbb{E}[f(\tilde{S} \cup \mathrm{OPT}) \mid |S| < k] \ge \left(1 - \max_{e \in V} \Pr[e \in \tilde{S} \mid |S| < k]\right) \cdot f(\mathrm{OPT}),$$

which are similar to (A.1), but do not imply it.

Appendix B. Inapproximability

In this appendix, we prove an inapproximability result for the problem of maximizing a nonnegative submodular function subject to a cardinality constraint in the data stream model. This result is given by the next theorem. The proof of the theorem is an adaptation of a proof given by Buchbinder et al. [10] for a similar result applying to an online variant of the same problem.

Theorem B.1. *For every constant $\varepsilon > 0$, no data stream algorithm for maximizing a nonnegative submodular function subject to cardinality constraint is $(1/2+\varepsilon)$-competitive unless it uses $\Omega(|V|)$ memory.*

Proof. Let $k \ge 1$ and $h \ge 1$ be two integers to be chosen later, and consider the nonnegative submodular function $f : 2^V \to \mathbb{R}^+$, where $V = \{u_i\}_{i=1}^{k-1} \cup \{v_i\}_{i=1}^{h} \cup \{w\}$, defined as follows:

$$f(S) = \begin{cases} |S| & \text{if } w \notin S, \\ k + \left|S \cap \{u_i\}_{i=1}^{k-1}\right| & \text{if } w \in S. \end{cases}$$

It is clear that f is nonnegative. One can also verify that the marginal value of each element in V is nonincreasing, and hence, f is submodular.

Let *ALG* be an arbitrary data stream algorithm for the problem of maximizing a nonnegative submodular function subject to a cardinality constraint, and let us consider what happens when we give this algorithm the preceding function f as input, the last element of V to arrive is the element w, and we ask the algorithm to pick a set of size at most k. One can

observe that, before the arrival of w, *ALG* has no way to distinguish between the other elements of V. Thus, if we denote by M the set of elements stored by *ALG* immediately before the arrival of w and assume that the elements of $V \setminus \{w\}$ arrive in a random order, then every element of $V \setminus \{w\}$ belongs to M with the same probability of $\mathbb{E}[|M|]/|V \setminus \{w\}|$. Hence, there must exist some arrival order for the elements of $V \setminus \{w\}$ guaranteeing that

$$\mathbb{E}\left[\left|M \cap \{u_i\}_{i=1}^{k-1}\right|\right] = \sum_{i=1}^{k-1} \Pr[u_i \in M] \leq \frac{k \cdot \mathbb{E}[|M|]}{|V \setminus \{w\}|}.$$

Note now that the preceding implies that the expected value of the output set produced by *ALG* given the preceding arrival order is at most

$$k + \frac{k \cdot \mathbb{E}[|M|]}{|V \setminus \{w\}|}.$$

In contrast, the optimal solution is the set $\{u_i\}_{i=1}^{k-1} \cup \{w\}$, whose value is $2k-1$. Therefore, the competitive ratio of *ALG* is at most

$$\frac{k + k \cdot \mathbb{E}[|M|]/|V \setminus \{w\}|}{2k-1} = \frac{1 + \mathbb{E}[|M|]/|V \setminus \{w\}|}{2 - 1/k} \leq \frac{1}{2} + \frac{1}{k} + \frac{\mathbb{E}[|M|]}{|V \setminus \{w\}|}.$$

To prove the theorem, we need to show that, when the memory used by *ALG* is $o(|V|)$, we can choose large enough values for k and h that guarantee that the rightmost side of the last inequality is at most $1/2 + \varepsilon$. We do so by showing that the two terms $1/k$ and $\mathbb{E}[|M|]/|V \setminus \{w\}|$ can both be upper bounded by $\varepsilon/2$ when the integers k and h are large enough, respectively. For the term $1/k$ this is clearly the case when k is larger than $2/\varepsilon$. For the term $\mathbb{E}[|M|]/|V \setminus \{w\}|$ this is true because increasing h can make V as large as want and, thus, can make the ratio $\mathbb{E}[|M|]/|V \setminus \{w\}|$ as small as necessary because of our assumption that the memory used by *ALG* (which includes M) is $o(|V|)$. □

Appendix C. Multilinear Extension-Based Algorithm

The properties of the multilinear extension-based variant of our algorithm are summarized by the following theorem.

Theorem C.1. *Assume there exists an α-approximation off-line algorithm* OfflineAlg *for maximizing a nonnegative submodular function subject to a 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 $(\alpha/(1+\alpha) - \varepsilon)$-approximation semistreaming algorithm for maximizing a nonnegative submodular function subject to a cardinality constraint. The algorithm stores at most $O(k\varepsilon^{-2}\log\alpha^{-1})$ elements.*

For ease of reading, we present only a simplified version of the multilinear extension-based variant of our algorithm. This simplified version (given as Algorithm C.1) captures our main new ideas but makes two simplifying assumptions that can be avoided using standard techniques.

- The first assumption is that Algorithm C.1 has access to an estimate τ of $f(\mathrm{OPT})$ obeying $(1-\varepsilon/8)\cdot f(\mathrm{OPT}) \leq \tau \leq f(\mathrm{OPT})$. Such an estimate can be produced using well-known techniques, such as a technique of Kazemi et al. [31] used in Section 4, at the cost of increasing the space complexity of the algorithm only by a factor of $O(\varepsilon^{-1}\log\alpha^{-1})$.
- The second assumption is that Algorithm C.1 has value oracle access to the multilinear extension F. If the time complexity of Algorithm C.1 is not important, then this assumption is of no consequence because 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, in which case, this assumption can be dropped using standard sampling techniques (such as the one used in Călinescu et al. [13]).[6] Interestingly, the rounding step of the algorithm and the sampling technique are the only parts of the extension-based algorithm that employ randomness. Because the rounding can be made deterministic given either exponential time or value oracle access to F, we get the following observation.

Observation C.1. If OfflineAlg is deterministic, then our multilinear extension-based algorithm is also deterministic when it is allowed either exponential computation time or value oracle access to F.

A more detailed discussion of the techniques for removing the preceding assumptions can be found in an earlier version of this paper (available in Alaluf et al. [2]) that had this variant of our algorithm as one of its main results.

Algorithm C.1 has two constant parameters $p \in (0,1)$ and $c > 0$ and maintains a fractional solution $\boldsymbol{x} \in [0,1]^V$. 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 $\boldsymbol{x}$. If this marginal contribution exceeds the threshold of $c\tau/k$, then the algorithm tries to add to $\boldsymbol{x}$ a p-fraction of e but might end up adding a smaller fraction of e if adding a full p-fraction of e to $\boldsymbol{x}$ makes $\boldsymbol{x}$ an infeasible solution, that is, makes $\|\boldsymbol{x}\|_1 > k$ (note that $\|\boldsymbol{x}\|_1$ is the sum of the coordinates of $\boldsymbol{x}$).

After viewing all of the elements, Algorithm C.1 uses the fractional solution $\boldsymbol{x}$ 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 $\boldsymbol{x}$. As mentioned in Section 3, two rounding procedures, pipage and swap rounding, are suggested for this task in the literature (Călinescu et al. [13], Chekuri et al. [18]). 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 $F(\boldsymbol{x})$ of the fractional solution $\boldsymbol{x}$. The set S_2 is generated by applying OfflineAlg to the support of the vector $\boldsymbol{x}$, 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 C.1 simply returns the better one of them.

Algorithm C.1 (StreamProcessExtension (simplified) (p, c))

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

Let us denote by $\hat{\boldsymbol{x}}$ the final value of the fractional solution $\boldsymbol{x}$ (i.e., its value when the stream ends). We begin the analysis of Algorithm C.1 with the following useful observation. In the statement of observation and in the rest of the section, we denote by $\text{supp}(\boldsymbol{x})$ the support of vector $\boldsymbol{x}$, that is, the set $\{e \in V \mid x_e > 0\}$.

Observation C.2. If $\|\hat{\boldsymbol{x}}\|_1 < k$, then $\hat{\boldsymbol{x}}_e = p$ for every $e \in \text{supp}(\hat{\boldsymbol{x}})$. Otherwise, $\|\hat{\boldsymbol{x}}\|_1 = k$, and the first part of the observation is still true for every element $e \in \text{supp}(\hat{\boldsymbol{x}})$ except for maybe a single element.

Proof. For every element e added to the support of $\boldsymbol{x}$ by Algorithm C.1, the algorithm sets x_e to p unless this makes $\|\boldsymbol{x}\|_1$ exceed k, in which case the algorithm sets x_e to be the value that makes $\|\boldsymbol{x}\|_1$ equal to k. Thus, after a single coordinate of $\boldsymbol{x}$ is set to a value other than p (or the initial zero), $\|\boldsymbol{x}\|_1$ becomes k, and Algorithm C.1 stops changing $\boldsymbol{x}$. □

Using the last observation, we can now bound the space complexity of Algorithm C.1 and show (in particular) that it is a semistreaming algorithm for a constant p when the space complexity of OfflineAlg is nearly linear.

Observation C.3. Algorithm C.1 can be implemented so that it stores $O(k/p)$ elements.

Proof. To calculate the sets S_1 and S_2, Algorithm C.1 needs access only to the elements of V that appear in the support of $\boldsymbol{x}$. Thus, the number of elements it needs to store is $O(|\text{supp}(\hat{\boldsymbol{x}})|) = O(k/p)$, where the equality follows from Observation C.2. □

We now divert our attention to analyzing the approximation ratio of Algorithm C.1. The first step in this analysis is lower bounding the value of $F(\hat{\boldsymbol{x}})$, which we do by considering two cases: one when $\|\hat{\boldsymbol{x}}\|_1 = k$ and the other when $\|\hat{\boldsymbol{x}}\|_1 < k$. The following lemma bounds the value of $F(\hat{\boldsymbol{x}})$ in the first of these cases. Intuitively, this lemma holds because $\text{supp}(\hat{\boldsymbol{x}})$ contains many elements, and each one of these elements must have increased the value of $F(\boldsymbol{x})$ significantly when added (otherwise, Algorithm C.1 would not have added this element to the support of x).

Lemma C.1. *If* $\|\hat{\boldsymbol{x}}\|_1 = k$, *then* $F(\hat{\boldsymbol{x}}) \geq c\tau$.

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

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

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

Consider now the case in which $\|\hat{\boldsymbol{x}}\|_1 < k$. Recall that our objective is to lower bound $F(\hat{\boldsymbol{x}})$ in this case as well. To do that, we need a tool for upper bounding the possible increase in the value of $F(\boldsymbol{x})$ when some of the indices of $\boldsymbol{x}$ are zeroed. The next lemma provides such an upper bound.

Lemma C.2. *Let $f : 2^V \to \mathbb{R}_{\geq 0}$ be a nonnegative submodular function, let p be a number in the range $[0, 1]$, and let $\boldsymbol{x}$, $\boldsymbol{y}$ be two vectors in $[0,1]^V$ such that*

- $\text{supp}(\boldsymbol{x}) \cap \text{supp}(\boldsymbol{y}) = \emptyset$.
- $y_e \leq p$ *for every* $e \in V$.

Then, the multilinear extension F of f obeys $F(\boldsymbol{x} + \boldsymbol{y}) \geq (1-p) \cdot F(\boldsymbol{x})$.

Proof. Let us define the function $G_{\boldsymbol{x}}(S) = \mathbb{E}[f(\mathtt{R}(\boldsymbol{x}) \cup S)]$. It is not difficult to verify that $G_{\boldsymbol{x}}$ is nonnegative and submodular and $G_{\boldsymbol{x}}(\emptyset) = F(\boldsymbol{x})$. Additionally, because $\text{supp}(\boldsymbol{x}) \cap \text{supp}(\boldsymbol{y}) = \emptyset$, $\mathtt{R}(\boldsymbol{x}+\boldsymbol{y})$ has the same distribution as $\mathtt{R}(\boldsymbol{x}) \cup \mathtt{R}(\boldsymbol{y})$, and therefore,

$$\begin{aligned} F(\boldsymbol{x}+\boldsymbol{y}) &= \mathbb{E}[f(\mathtt{R}(\boldsymbol{x}+\boldsymbol{y}))] = \mathbb{E}[f(\mathtt{R}(\boldsymbol{x}) \cup \mathtt{R}(\boldsymbol{y}))] \\ &= \mathbb{E}[G_{\boldsymbol{x}}(\mathtt{R}(\boldsymbol{y}))] \geq (1-p) \cdot G_{\boldsymbol{x}}(\emptyset) = (1-p) \cdot F(\boldsymbol{x}), \end{aligned}$$

where the inequality follows from Lemma 3.2. □

Using the last lemma, we now prove two lemmas proving upper and lower bounds on the expression $F(\hat{\boldsymbol{x}} + \mathbf{1}_{\text{OPT} \setminus \text{supp}(\hat{\boldsymbol{x}})})$.

Lemma C.3. *If $\|\hat{\boldsymbol{x}}\|_1 < k$, then $F(\hat{\boldsymbol{x}} + \mathbf{1}_{\text{OPT} \setminus \text{supp}(\hat{\boldsymbol{x}})}) \geq (1-p) \cdot [p \cdot f(\text{OPT}) + (1-p) \cdot f(\text{OPT} \setminus \text{supp}(\hat{\boldsymbol{x}}))]$.*

Proof. Because $\|\hat{\boldsymbol{x}}\|_1 < k$, Observation C.2 guarantees that $\hat{\boldsymbol{x}}_e = p$ for every $e \in \text{supp}(\hat{\boldsymbol{x}})$. Thus, $\hat{\boldsymbol{x}} = p \cdot \mathbf{1}_{\text{OPT} \cap \text{supp}(\hat{\boldsymbol{x}})} + p \cdot \mathbf{1}_{\text{supp}(\hat{\boldsymbol{x}}) \setminus \text{OPT}}$, and therefore,

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

where the first inequality follows from Lemma C.2, the second inequality holds because 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 $b = k^{-1} \cdot |\text{OPT} \setminus \text{supp}(\hat{\boldsymbol{x}})|$. Intuitively, the lemma holds because the fact that the elements of $\text{OPT} \setminus \text{supp}(\hat{\boldsymbol{x}})$ were not added to the support of $\boldsymbol{x}$ by Algorithm C.1 implies that their marginal contribution is small.

Lemma C.4. *If $\|\hat{\boldsymbol{x}}\|_1 < k$, then $F(\hat{\boldsymbol{x}} + \mathbf{1}_{\text{OPT} \setminus \text{supp}(\hat{\boldsymbol{x}})}) \leq F(\hat{\boldsymbol{x}}) + bc\tau$.*

Proof. The elements in $\text{OPT} \setminus \text{supp}(\hat{\boldsymbol{x}})$ were rejected by Algorithm C.1, which means that their marginal contribution with respect to the fractional solution $\boldsymbol{x}$ at the time of their arrival was smaller than $c\tau/k$. Because the fractional solution x only increases during the execution of the algorithm, the submodularity of f guarantees that the same is true also with respect to $\hat{\boldsymbol{x}}$. More formally, we get

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

Using the submodularity of f again, this implies

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

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

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

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 $\text{OPT} \cap \text{supp}(\hat{\boldsymbol{x}})$ is a subset of the support of $\hat{\boldsymbol{x}}$ of size at most k and, thus, is a feasible solution for OfflineAlg to return, which implies $\mathbb{E}[f(S_2)] \geq \alpha \cdot f(\text{OPT} \cap \text{supp}(\hat{\boldsymbol{x}}))$ because 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 $(b\hat{\boldsymbol{x}}) \vee \mathbf{1}_{\text{OPT} \cap \text{supp}(\hat{\boldsymbol{x}})}$ as a fractional feasible solution (using the rounding methods discussed before it, it can be converted into an integral feasible solution of at

least the same value). The proof of the lemma lower bounds the value of the vector $(b\hat{\boldsymbol{x}}) \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{\boldsymbol{x}})}$ using the concavity of the function $F((t\cdot\hat{\boldsymbol{x}}) \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{\boldsymbol{x}})})$ as well as ideas used in the proofs of the previous claims.

Lemma C.5. *If $\|\hat{\boldsymbol{x}}\|_1 < k$, then $\mathbb{E}[f(S_2)] \geq \alpha b(1-p-cb)\tau + \alpha(1-b)\cdot f(\mathrm{OPT}\cap\mathrm{supp}(\hat{\boldsymbol{x}}))$.*

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

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

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

The first step toward proving the last inequality is getting a lower bound on $F(\hat{\boldsymbol{x}} \vee \mathbf{1}_{\mathrm{OPT}\cap\mathrm{supp}(\hat{\boldsymbol{x}})})$. Recall that we already show in the proof of Lemma C.4 that

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

Thus, the submodularity of f implies

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

Rearranging this inequality yields

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

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

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

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

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 C.1 in the case of $\|\hat{\boldsymbol{x}}\|_1 < k$, which is a function of α, τ, and p alone. We note that both the guarantees of Corollary C.1 and Lemma C.5 are lower bounds on the expected value of the output of the algorithm in this case because $\mathbb{E}[f(S_1)] \geq F(\hat{\boldsymbol{x}})$. 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 C.2. *If $\|\hat{\boldsymbol{x}}\|_1 < k$ and c is set to $\alpha(1-p)/(\alpha+1)$, then $\mathbb{E}[\max\{f(S_1), f(S_2)\}] \geq (1-p)\alpha\tau/(\alpha+1)$.*

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

By the definition of S_1, $\mathbb{E}[f(S_1)] \geq F(\hat{\boldsymbol{x}})$. Thus, by Corollary C.1 and Lemma C.5,

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

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

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

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

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

which is always nonnegative because both q and α are numbers between zero and one. Thus, we get that the minimal value of Expression (C.1) is obtained for $d = 0$ for any choice of q and α. Plugging this value into d yields

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

Note that Lemma C.1 and Corollary C.2 both 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 C.2 (because $\mathbb{E}[\max\{f(S_1), f(S_2)\}] \geq \mathbb{E}[f(S_1)] \geq F(\hat{\mathbf{x}})$). Thus, we can summarize the results we have proved so far using the following proposition.

Proposition C.1. *Algorithm C.1 is a semistreaming algorithm storing $O(k/p)$ elements. Moreover, for the value of the parameter c given in Corollary C.2, the output set produced by this algorithm has an expected value of at least $\alpha\tau(1-p)/(\alpha+1)$.*

Using the last proposition, we can now prove the following theorem. As discussed at the beginning of the section, the assumption that τ is known can be dropped at the cost of a slight increase in the number of elements stored by the algorithm, which yields Theorem C.1.

Theorem C.2. *For every constant $\varepsilon \in (0,1]$, there exists a semistreaming 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 $(\alpha/(1+\alpha)-\varepsilon)$-approximation for the problem of maximizing a nonnegative submodular function subject to a cardinality constraint. This algorithm stores $O(k\varepsilon^{-1})$ elements.*

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

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

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

Appendix D. Algorithm with True Randomization

The variant of our algorithm that involves true randomization is shown in Algorithm D.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 Badanidiyuru et al. [4]. 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 (recall that α is the approximation guarantee of the off-line algorithm OfflineAlg that we use in the postprocessing 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 midstream 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 Badanidiyuru et al. [4] for the full details.

Theorem D.1. *There exists a streaming algorithm* StreamProcessRandomized *for nonnegative, nonmonotone 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 $O(\varepsilon^{-3}/k \log(k/\alpha)\log(1/\varepsilon))$ space.*
- *The update time per item is $O(\varepsilon^{-2}/\log(k/\alpha)\log(1/\varepsilon))$ marginal gain computations.*

At the end of the stream, we postprocess the output of StreamProcessRandomized using any off-line algorithm OfflineAlg for submodular maximization. The resulting solution is a $\alpha/(1+\alpha)-\varepsilon$ approximation, where α is the approximation of OfflineAlg.

Algorithm D.1 (Streaming Algorithm for $\max_{|S|\le k} f(S)$). PostProcess uses any off-line algorithm OfflineAlg with approximation α. The algorithm does **not** store the sets $V_{i,j}$; they are defined for analysis purposes only.

```
1  //f : 2^V → ℝ_{≥0}: submodular and nonnegative
2  //k: cardinality constraint
3  //ε ∈ (0,1]: accuracy parameter
4  //κ: threshold
5  StreamProcessRandomized(f,k,ε,κ)
6  r ← Θ(ln(1/ε)/ε)
7  m ← 1/ε
8  S_{i,j} ← ∅ for all i ∈ [r], j ∈ [m]
9  V_{i,j} ← ∅ for all i ∈ [r], j ∈ [m] //not stored, defined for analysis purposes only
10 for each arriving element e do
11     for i = 1 to r do
12         choose an index j ∈ [m] uniformly and independently at random
13         V_{i,j} ← V_{i,j} ∪ {e} //not stored, defined for analysis purposes only
14         if f(S_{i,j} ∪ {e}) − f(S_{i,j}) ≥ κ and |S_{i,j}| < k then
15             S_{i,j} ← S_{i,j} ∪ {e}
16 return {S_{i,j} : i ∈ [r], j ∈ [m]}
17 PostProcess(f,k,ε)
18 //Assumes an estimate for f(OPT); see text on how to remove this assumption
19 //Uses any off-line algorithm OFFLINEALG with approximation α
20 κ ← α/[(1+α)k] · f(OPT) //threshold
21 {S_{i,j}} ← StreamProcessRandomized (f,k,ε,κ)
22 if |S_{i,j}| = k for some i and j then
23     return S_{i,j}
24 else
25     U ← ∪_{i,j} S_{i,j}
26     T ← OfflineAlg (f,k,U)
27     return arg max{f(S_{1,1}), f(T)}
```

Algorithm D.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 StreamProcessRandomized.

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

In the remainder of this section, we analyze Algorithm D.1 and show that it achieves a $\alpha/(1+\alpha)-\varepsilon$ approximation, where α is the approximation guarantee of the off-line 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 StreamProcessRandomized has size k. For every $i \in [r]$, let $\mathcal{F}_i$ be the event that $|S_{i,1}| = k$. Because each of the r repetitions (iterations of the for loop of StreamProcessRandomized) use independent randomness to partition V, the events $\mathcal{F}_1, \ldots, \mathcal{F}_r$ are independent. Additionally, the events $\mathcal{F}_1, \ldots, \mathcal{F}_r$ have the same probability. We divide the analysis into two cases, depending on whether $\Pr[\mathcal{F}_1] \ge \varepsilon$ or $\Pr[\mathcal{F}_1] < \varepsilon$. In the first case, because 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 because

$f(S_{i,j}) \geq \kappa\,|S_{i,j}| = \kappa k = \alpha f(\mathrm{OPT})/(1+\alpha)$. In the second case, we have $\Pr[\overline{\mathcal{F}_1}] \geq 1-\varepsilon$, and we argue that $\cup_{i,j} S_{i,j}$ contains a good solution. We now give the formal argument for each of the cases.

D.1. The Case $Pr[\mathcal{F}_1] \geq \varepsilon$

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

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

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

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

Lemma D.1. *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

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

D.2. The Case $Pr[\mathcal{F}_1] < \varepsilon$

In this case, we show that the solution $\arg\max\{f(T), f(S_{1,1})\}$, which is 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. [6]: the probabilities p_e defined as follows 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 Barbosa et al. [6, section 7.3]; Lemma D.3 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. [6]. We emphasize that Barbosa et al. [6] use these concepts in a different context (specifically, monotone maximization in the distributed setting). When applied to our context—nonmonotone maximization in the streaming setting—the framework of Barbosa et al. [6] 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.

D.2.1. Notation and Definitions. For analysis purposes only, we make use of the Lovász extension $\hat{f}$. We fix an optimal solution $\mathrm{OPT} \in \arg\max\{f(A) : A \subseteq V, |A| \leq k\}$. 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 StreamProcessRandomized). Additionally, for each $i \in [r]$, $V_{i,1}, \ldots, V_{i,m}$ is a partition of V into $1/m$ samples.

For a subset $N \subseteq V$, we let STGreedy(N) be the output of the single threshold greedy algorithm when run as follows (see also Algorithm D.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 StreamProcessRandomized; 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 $S_{i,j} = \text{STGreedy}\,(V_{i,j})$ for all $i \in [r], j \in [m]$. For analysis purposes only, we also consider STGreedy(N) for sets N that do not correspond to any set $V_{i,j}$.

For each $e \in V$, we define

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

We partition OPT into two sets:

$$O_1 = \{e \in \mathrm{OPT} : p_e \geq \varepsilon\}$$
$$O_2 = \mathrm{OPT} \setminus O_1.$$

We also define the following subset of O_2:

$$O_2' = \{e \in O_2 : e \notin \text{STGreedy}(V_{1,1} \cup \{e\})\}.$$

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' 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 $U = \cup_{i,j} S_{i,j}$ with good probability: each iteration $i \in [r]$ ensures that an element $e \in O_1$ is in $S_{i,1} \cup \cdots \cup S_{i,m}$ with probability $p_e \geq \varepsilon$, and because we repeat $r = \Theta(\ln(1/\varepsilon)/\varepsilon)$ times, we ensure that $\mathbb{E}[\mathbf{1}_{O_1 \cap U}] \geq (1-\varepsilon)\mathbf{1}_{O_1}$. We also have that $\mathbb{E}[\mathbf{1}_{O_2'}] \geq (1-\varepsilon)\mathbf{1}_{O_2}$: 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 (because $V_{1,1} \sim \mathcal{V}(1/m)$) and $p_e \leq \varepsilon$ (because $e \in O_2$). Thus, $\mathbb{E}[\mathbf{1}_{(O_1 \cap U) \cup O_2'}] \geq (1-\varepsilon)\mathbf{1}_{\mathrm{OPT}}$, 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 postprocessing 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 Lovász extension and its properties.

Lemma D.2. *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, $\mathbb{E}[f(S \cup O)] \ge (v-u)f(\mathrm{OPT})$.*

Proof. Let $\hat{f}$ be the Lovász extension of f. Using the fact that $\hat{f}$ is an extension and it is convex, we obtain

$$\mathbb{E}[f(S \cup O)] = \mathbb{E}\left[\hat{f}(\mathbf{1}_{S\cup O})\right] \ge \hat{f}(\mathbb{E}[\mathbf{1}_{S\cup O}]) = \hat{f}(\mathbb{E}[\mathbf{1}_S] + \mathbb{E}[\mathbf{1}_O]).$$

Let $\mathbf{x} := \mathbb{E}[\mathbf{1}_S] + \mathbb{E}[\mathbf{1}_O]$. Note that

$$x_e = \begin{cases} \Pr[e \in S] \le u & \text{if } e \in V \setminus \mathrm{OPT} \\ \Pr[e \in O] \ge v & \text{if } e \in \mathrm{OPT}. \end{cases}$$

Thus, we have

$$\hat{f}(\mathbf{x}) = \int_0^1 f(\{e \in V : x_e \ge \theta\})d\theta \ge \int_u^v f(\{e \in V : x_e \ge \theta\})d\theta = (v-u)f(\mathrm{OPT}).$$

The first equality is the definition of $\hat{f}$. The inequality is by the nonnegativity of f. The second equality is because, for $u < \theta \le v$, we have $\{e \in V : x_e \ge \theta\} = \mathrm{OPT}$. □

The following lemma establishes a consistency property for the STGreedy algorithm, analogous to the consistency property shown and used by Barbosa et al. [6] for algorithms such as the standard greedy algorithm. The proof is also very similar to the proof shown by Barbosa et al. [6].

Lemma D.3. *Conditioned on the event $|S_{1,1}| < k$,*

$$\textsc{STGreedy}(V_{1,1} \cup O_2') = \textsc{STGreedy}(V_{1,1}) = S_{1,1}.$$

Proof. To simplify notation, we let $V_1 = V_{1,1}$ and $S_1 = S_{1,1}$. Let $X = \textsc{STGreedy}(V_1 \cup O_2')$. Suppose for contradiction that $S_1 \ne X$. Let $e_1, e_2, \ldots, e_{|V_1 \cup O_2'|}$ be the elements of $V_1 \cup O_2'$ in the order in which they arrived in the stream. Let i be the smallest index such that $\textsc{STGreedy}(\{e_1, \ldots, e_i\}) \ne \textsc{STGreedy}(\{e_1, \ldots, e_i\} \cap V_1)$. By the choice of i, we have

$$\textsc{STGreedy}(\{e_1, \ldots, e_{i-1}\}) = \textsc{STGreedy}(\{e_1, \ldots, e_{i-1}\} \cap V_1) := A.$$

Note that $|A| < k$ because $A \subseteq S_1$ and $|S_1| < k$ by assumption. Because $\textsc{STGreedy}(\{e_1, \ldots, e_i\}) \ne \textsc{STGreedy}(\{e_1, \ldots, e_i\} \cap V_1)$, we must have $e_i \notin V_1$ (and, thus, $e_i \in O_2' \setminus V_1$) and $f(A \cup \{e_i\}) - f(A) \ge \kappa$. The latter implies that $e_i \in \textsc{STGreedy}(V_1 \cup \{e_i\})$: after processing all of the elements of V_1 that arrived before e_i, the partial greedy solution is A; when e_i arrives, it is added to the solution because $|A| < k$ and $f(A \cup \{e_i\}) - f(A) \ge \kappa$. But then $e_i \notin O_2'$, which is a contradiction. □

We now proceed with the main analysis. Recall that PostProcess runs 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 because we could have $|S_{1,1}| > |O_2|$. Therefore, it is natural that an equation relating the value of this set to the values of feasible sets (such as $S_{1,1}$ and T) include a factor to correct for the "unfair" advantage that $S_{1,1} \cup (O_1 \cap U)$ might have as an infeasible set. This correction factor takes the form of $1 - |O_2|/k$ in the next lemma.

Lemma D.4. *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 $\mathbb{E}[\mathbf{1}_X] = b\mathbf{1}_{S_1}/k$. 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 $p := |S_1|b/(sk)$.

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

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

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

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

Next, we lower bound $\max\{f(S_1), f(T)\}$ using a convex combination $(1-\theta)f(S_1) + \theta f(T)$ with coefficient $\theta = 1/(1+\alpha(1-b/k))$. Note that $1-\theta = \theta\alpha(1-b/k)$. 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

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

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 $f(A \mid B)$ to denote the marginal gain of set A on top of set B, that is, $f(A \mid B) = f(A \cup B) - f(B)$.

Lemma D.5. *We have* $\mathbb{E}[f(O_2' \mid S_{1,1})] \leq \kappa b + \varepsilon f(\mathrm{OPT})$.

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:

$$\begin{aligned} \mathbb{E}[f(O_2' \mid S_1)] &= \mathbb{E}[f(O_2' \mid S_1) \mid |S_1| < k] \cdot \underbrace{\Pr[|S_1| < k]}_{\leq 1} + \underbrace{\mathbb{E}[f\left(O_2' \mid S_1\right) \mid |S_1| = k]}_{\leq f(\mathrm{OPT})} \cdot \underbrace{\Pr[|S_1| = k]}_{\leq \varepsilon} \\ &\leq \mathbb{E}[f(O_2' \mid S_1) \mid |S_1| < k] + \varepsilon f(\mathrm{OPT}). \end{aligned}$$

We have used that $f(O_2' \mid S_1) \leq f(O_2') \leq f(\mathrm{OPT})$, for which 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 $\mathbb{E}[f(O_2' \mid S_1) \mid |S_1| < k] \leq \kappa b$.

We condition on the event $|S_1| < k$ for the remainder of the proof. By Lemma D.3, we have $\mathrm{STGreedy}(V_1 \cup O_2') = S_1$. Because $|S_1| < k$, 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

$$f(O_2' \mid S_1) \leq \kappa |O_2'| \leq \kappa b. \quad \square$$

Lemma D.6. *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 D.2 to the following sets:

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

We show that $\mathbb{E}[\mathbf{1}_S] \leq \varepsilon \mathbf{1}_{V \setminus \mathrm{OPT}}$ 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 D.2, which gives the desired result.

Because $S \subseteq S_{1,1} \subseteq V_{1,1}$ 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 $\Pr[e \in O] \geq 1-\varepsilon$. Because $(O_1 \cap U) \cup O_2' \subseteq O$, 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 StreamProcessRandomized ensures that e is in $S_{i,1} \cup \cdots \cup S_{i,m}$ with probability $p_e \geq \varepsilon$. Because 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

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

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 D.5 and D.6 immediately imply the following:

Lemma D.7. *We have* $\mathbb{E}[f(S_{1,1} \cup (O_1 \cap U))] \geq (1-3\varepsilon) f(\mathrm{OPT}) - \kappa b$.

Proof. Recall that we use the notation $f(A \mid B) = f(A \cup B) - f(B)$. We have

$$\begin{aligned} f(S_{1,1} \cup (O_1 \cap U)) &= f(S_{1,1} \cup (O_1 \cap U) \cup O_2') - f(O_2' \mid S_{1,1} \cup (O_1 \cap U)) \\ &\geq f(S_{1,1} \cup (O_1 \cap U) \cup O_2') - f(O_2' \mid S_{1,1}), \end{aligned}$$

where the inequality is by submodularity.

By taking expectation and using Lemmas D.5 and D.6, we obtain the desired result. □

Finally, Lemmas D.4 and D.7 give the approximation guarantee.

Lemma D.8. *We have* $\mathbb{E}[\max\{f(S_{1,1}), f(T)\}] \geq (\alpha/(1+\alpha) - 3\varepsilon) f(\mathrm{OPT})$.

Proof. By Lemmas D.4 and D.7, we have

$$\begin{aligned} \mathbb{E}[\max\{f(S_{1,1}), f(T)\}] &\geq \frac{\alpha}{1+\alpha\left(1-\frac{b}{k}\right)} \mathbb{E}[f(S_{1,1} \cup (O_1 \cap U))] \\ &\geq \frac{\alpha}{1+\alpha\left(1-\frac{b}{k}\right)} ((1-3\varepsilon) f(\mathrm{OPT}) - \kappa b) \\ &= \frac{\alpha}{1+\alpha\left(1-\frac{b}{k}\right)} \left((1-3\varepsilon) f(\mathrm{OPT}) - \frac{\alpha}{1+\alpha}\frac{b}{k} f(\mathrm{OPT})\right) \\ &\geq \left(\frac{\alpha}{1+\alpha} - 3\varepsilon\right) f(\mathrm{OPT}). \quad \square \end{aligned}$$

Endnotes

[1] A variant of the algorithm from Kazemi et al. [31] has an even better space complexity of $O(k/\varepsilon)$.

[2] Chekuri et al. [17] claimed an improved approximation ratio of $(2+e)^{-1} - \varepsilon$ for a cardinality constraint, but an error was later found in the proof of this improved ratio Chekuri [16]. See Appendix A for more details.

[3] Formally, all the algorithms we present are semistreaming algorithms, that is, their space complexity is nearly linear in k. Because 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 semistreaming algorithms in this paper and use the two terms interchangeably.

[4] This result is a simple adaptation of a result from Buchbinder et al. [10]. For completeness, we include the proof in Appendix B.

[5] Formally, the number of elements stored by the algorithm and the number of marginal value computations also depend on $\log \alpha^{-1}$. Because 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.

[6] We note, however, that, although the sampling technique keeps the algorithm polynomial, it does increase the time complexity of the algorithm significantly because $\tilde{\Theta}(k^4 \varepsilon^{-4} \alpha^{-2})$ samples are necessary in order to replace each value oracle query to F.

References

[1] Alaluf N, Ene A, Feldman M, Nguyen HL, Suh A (2020) Optimal streaming algorithms for submodular maximization with cardinality constraints. Czumaj A, Dawar A, Merelli E, eds. *Proc. 47th Internat. Colloquium on Automata, Languages, and Programming ICALP* (Schloss Dagstuhl - Leibniz-Zentrum für Informatik), 6:1–6:19.
[2] Alaluf N, Ene A, Feldman M, Nguyen HL, Suh A (2020) Optimal streaming algorithms for submodular maximization with cardinality constraints. Preprint, submitted February 20, https://arxiv.org/abs/1911.12959v2.
[3] Azar Y, Gamzu I, Roth R (2011) Submodular max-sat. Demetrescu C, Halldórsson M, eds. *Proc. 19th Annual European Sympos. ESA* (Springer), 323–334.
[4] Badanidiyuru A, Mirzasoleiman B, Karbasi A, Krause A (2014) Streaming submodular maximization: Massive data summarization on the fly. Macskassy SA, Perlich C, Leskovec J, Wang W, Ghani R, eds. *Proc. 20th ACM SIGKDD Internat. Conf. Knowledge Discovery and Data Mining* (ACM), 671–680.
[5] Barbosa R, Ene A, Nguyen HL, Ward J (2015) The power of randomization: Distributed submodular maximization on massive datasets. Bach FR, Blei DM, eds. *Proc. 32nd Internat. Conf. Machine Learn. ICML* (JMLR.org), 1236–1244.
[6] Barbosa R, Ene A, Nguyen HL, Ward J (2016) A new framework for distributed submodular maximization. Dinur I, ed. *Proc. 57th Annual Sympos. Foundations of Comput. Sci. (FOCS)* (IEEE Computer Society), 645–654.
[7] Buchbinder N, Feldman M (2018) Deterministic algorithms for submodular maximization problems. *ACM Trans. Algorithms* 14(3): 32:1–32:20.
[8] Buchbinder N, Feldman M (2019) Constrained submodular maximization via a non-symmetric technique. *Math. Oper. Res.* 44(3): 988–1005.
[9] Buchbinder N, Feldman M, Garg M (2019) *Deterministic* $(1/2+\varepsilon)$-approximation for submodular maximization over a matroid. Chan TM, ed. *Proc. 13th Annual Sympos. Discrete Algorithms (SODA)* (SIAM), 241–254.
[10] Buchbinder N, Feldman M, Schwartz R (2019) Online submodular maximization with preemption. *ACM Trans. Algorithms* 15(3): 30:1–30:31.
[11] Buchbinder N, Feldman M, Filmus Y, Garg M (2020) Online submodular maximization: Beating 1/2 made simple. *Math. Programming* 183(1):149–169.
[12] Buchbinder N, Feldman M, Naor J, Schwartz R (2014) Submodular maximization with cardinality constraints. Chekuri C, ed. *Proc. 25th Annual Sympos. Discrete Algorithms (SODA)* (SIAM), 1433–1452.
[13] Călinescu G, Chekuri C, Pál M, Vondrák J (2011) Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.* 40(6):1740–1766.
[14] Chakrabarti A, Kale S (2015) Submodular maximization meets streaming: Matchings, matroids, and more. *Math. Programming* 154(1–2): 225–247.
[15] Chan TH, Huang Z, Jiang SH, Kang N, Tang ZG (2018) Online submodular maximization with free disposal. *ACM Trans. Algorithms* 14(4):56:1–56:29.
[16] Chekuri C (2018) Personal communication. The communication took place over e-mail on February 8, 2018.
[17] Chekuri C, Gupta S, Quanrud K (2015) Streaming algorithms for submodular function maximization. Halldórsson MM, Iwama K, Kobayashi N, Speckmann B, eds. *Proc. 42nd Internat. Colloquium Automata Languages Programming* (Springer), 318–330.
[18] Chekuri C, Vondrák J, Zenklusen R (2010) Dependent randomized rounding via exchange properties of combinatorial structures. *Proc. 51st Annual Sympos. Foundations of Comput. Sci. (FOCS)* (IEEE Computer Society), 575–584.
[19] Chekuri C, Vondrák J, Zenklusen R (2014) Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM J. Comput.* 43(6):1831–1879.
[20] Ene A, Nguyen HL (2016) Constrained submodular maximization: Beyond 1/e. Dinur I, ed. *Proc. 57th Annual Sympos. on Foundations of Computer Sci. (FOCS)* (IEEE Computer Society), 248–257.
[21] Epasto A, Mirrokni V, Zadimoghaddam M (2017) Bicriteria distributed submodular maximization in a few rounds. Scheideler C, Taghi Hajiaghayi M, eds. *Proc. 29th Sympos. Parallelism in Algorithms and Architectures (SPAA)* (ACM), 25–33.
[22] Feige U, Mirrokni VS, Vondrák J (2011) Maximizing non-monotone submodular functions. *SIAM J. Comput.* 40(4):1133–1153.
[23] Feldman M (2017) Maximizing symmetric submodular functions. *ACM Trans. Algorithms* 13(3):39:1–39:36.
[24] Feldman M, Harshaw C, Karbasi A (2017) Greed is good: Near-optimal submodular maximization via greedy optimization. Kale S, Shamir O, eds. *Proc. Machine Learn. Research 65 (COLT 2017)* (PMLR), 758–784.
[25] Feldman M, Karbasi A, Kazemi E (2018) Do less, get more: Streaming submodular maximization with subsampling. Bengio S, Wallach HM, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, eds. *Proc. Annual Conf. Neural Inform. Processing Systems*, 732–742.
[26] Feldman M, Naor JS, Schwartz R (2011) A unified continuous greedy algorithm for submodular maximization. Ostrovsky R, ed. *Proc. IEEE 52nd Annual Sympos. Foundations of Comput. Sci. (FOCS)* (IEEE Computer Society), 570–579.
[27] Feldman M, Norouzi-Fard A, Svensson O, Zenklusen R (2020) The one-way communication complexity of submodular maximization with applications to streaming and robustness. Makarychev K, Makarychev Y, Tulsiani M, Kamath G, Chuzhoy J, eds. *Proc. 52nd Annual (SIGACT) Sympos. Theory of Comput. (STOC)* (ACM), 1363–1374.
[28] Fisher ML, Nemhauser GL, Wolsey LA (1978) An analysis of approximations for maximizing submodular set functions—II. *Math. Programming Stud.* 8:73–87.
[29] Gharan SO, Vondrák J (2011) Submodular maximization by simulated annealing. Randall D, ed. *Proc. 22nd Annual Sympos. Discrete Algorithms (SODA)* (SIAM).
[30] Kapralov M, Post I, Vondrák J (2013) Online submodular welfare maximization: Greedy is optimal. Khanna S, ed. *Proc. 24th Annual Sympos. Discrete Algorithms (SODA)* (SIAM), 1216–1225.
[31] Kazemi E, Mitrovic M, Zadimoghaddam M, Lattanzi S, Karbasi A (2019) Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. Chaudhuri K, Salakhutdinov R, eds. *Proc. 36th Internat. Conf. Machine Learn. (ICML)* (PMLR), 3311–3320.
[32] Korula N, Mirrokni VS, Zadimoghaddam M (2018) Online submodular welfare maximization: Greedy beats 1/2 in random order. *SIAM J. Comput.* 47(3):1056–1086.

[33] Kumar R, Moseley B, Vassilvitskii S, Vattani A (2015) Fast greedy algorithms in mapreduce and streaming. *ACM Trans. Parallel Comput.* 2(3):14:1–14:22.
[34] Lee J, Sviridenko M, Vondrák J (2010) Submodular maximization over multiple matroids via generalized exchange properties. *Math. Oper. Res.* 35(4):795–806.
[35] Lee J, Mirrokni VS, Nagarajan V, Sviridenko M (2009) Non-monotone submodular maximization under matroid and knapsack constraints. Mitzenmacher M, ed. *Proc. 41st Annual Sympos. Theory Comput. (STOC)* (ACM), 323–332.
[36] Lovász L (1982) Submodular functions and convexity. *Math. Programming State Art, XIth Internat. Sympos. Math. Programming*, 235–257.
[37] Mirrokni V, Zadimoghaddam M (2015) Randomized composable core-sets for distributed submodular maximization. Servedio RA, Rubinfeld R, eds. *Proc. 47th Annual Sympos. Theory Comput. (STOC)* (ACM).
[38] Mirzasoleiman B, Jegelka S, Krause A (2018) Streaming non-monotone submodular maximization: Personalized video summarization on the fly. McIlraith SA, Weinberger KQ, eds. *Proc. 32nd Conf. Artificial Intelligence (AAAI)* (AAAI Press), 1379–1386.
[39] Mirzasoleiman B, Karbasi A, Badanidiyuru A, Krause A (2015) Distributed submodular cover: Succinctly summarizing massive data. Cortes C, Lawrence ND, Lee DD, Sugiyama M, Garnett R, eds. *Proc. Annual Conf. Neural Informat. Processing Systems (NIPS)* 28:2881–2889.
[40] Mirzasoleiman B, Karbasi A, Sarkar R, Krause A (2013) Distributed submodular maximization: Identifying representative elements in massive data. Burges CJC, Bottou L, Ghahramani Z, Weinberger KQ, eds. *Proc. 27th Annual Conf. Neural Informat. Processing Systems (NIPS)*, 2049–2057.
[41] Nemhauser GL, Wolsey LA (1978) Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Res.* 3(3): 177–188.
[42] Nemhauser GL, Wolsey LA, Fisher ML (1978) An analysis of approximations for maximizing submodular set functions—I. *Math. Programming* 14(1):265–294.
[43] Norouzi-Fard A, Tarnawski J, Mitrovic S, Zandieh A, Mousavifar A, Svensson O (2018) Beyond 1/2-approximation for submodular maximization on massive data streams. Dy JG, Krause A, eds. *Proc. 35th Internat. Conf. Machine Learn. (ICML)* (PMLR), 3826–3835.
[44] Vondrák J (2013) Symmetry and approximability of submodular maximization problems. *SIAM J. Comput.* 42(1):265–304.