Linear-Time and Deterministic Algorithms for Cardinality-Constrained Non-Monotone Submodular Maximization

Alan Kuhnle
Department of Computer Science & Engineering
Texas A & M University
College Station, Texas
alan@alankuhnle.com
May 26, 2022

Abstract

For the problem of maximizing a nonnegative, (not necessarily monotone) submodular function with respect to a cardinality constraint, we propose the first deterministic, approximation algorithms with linear time complexity. Our main contribution is a single-pass streaming algorithm that obtains ratio $23.314 + \varepsilon$ and makes only a constant number of oracle queries per received element. Along the way, we provide a simpler (non-streaming) deterministic, linear-time algorithm with ratio at most 11.657; and a deterministic, linear-time algorithm for the unconstrained maximization problem with competitive ratio 4. Finally, we present a streaming algorithm that, with a constant number of passes, improves the approximation ratio to $4 + \varepsilon$ in linear time. Empirically, the algorithms are validated to use fewer queries than state-of-the-art algorithms.

1 Introduction

Within discrete optimization, the submodularity property has been shown to be a fundamental and useful property. Intuitively, submodularity captures the idea of diminishing returns, where the marginal gain in utility decreases as the set becomes larger. Submodular objective functions arise in many learning objectives, e.g. interpreting neural networks [EDFK17], nonlinear sparse regression [EKDN18], among many others (see Iyer et al. [IKBA20] and references therein). In this work, we study submodular maximization subject to a size constraint, defined formally as follows.

Submodularity and Problem Definition. A nonnegative, set function $f : 2^U \rightarrow \mathbb{R}^+$, where ground set $U$ is of size $n$, is submodular if for all $S \subseteq T \subseteq U$, $u \in U \setminus T$, $f(T \cup \{u\}) - f(T) \leq f(S \cup \{u\}) - f(S)$. In this work, we study
the cardinality-constrained submodular maximization problem (SMCC): given submodular \( f \) and integer \( k \), determine
\[
\arg\max_{|S| \leq k} f(S).
\]
The function \( f \) is not required to be monotone\(^1\). We consider the value query model, in which the function \( f \) is available to an algorithm as an oracle that returns, in a single operation, the value \( f(S) \) of any queried set \( S \); we measure the time complexity of an algorithm by the number of oracle queries, since the time required for oracle evaluation typically dominates other parts of the computation. Observe that problem SMCC is NP-hard, since the classical maximum coverage problem is a special case. Therefore, we seek approximation algorithms that obtain a performance ratio with respect to an optimal solution.

**Challenges from Big Data.** Because of ongoing exponential growth in data size [Mis+08; LBS17] over the past decades, much effort has gone into the design of algorithms for submodular optimization with low time complexity, e.g. [BV14; Mir+15; BFS15; FMZ19; Kuh19]. Moreover, much effort has also gone into the design of memory efficient algorithms that do not need to store all of the data. In this context, researchers have studied streaming algorithms for submodular optimization [BMKK14; CK15; CGQ15; FKK18; MJK18; Ala+20; HKFK20; LRVZ21]. In the context of submodular optimization, a streaming algorithm takes a constant number of passes through the ground set (preferably a single pass) while staying within a small memory footprint of \( O(k \log n) \) elements of the ground set, where \( k \) is the maximum size of a solution and \( n \) is the size of the ground set. In this work, we consider an arbitrary order of element arrival in the stream.

**Randomized vs. Deterministic Algorithms.** An approximation ratio that holds only in expectation may result in a poor solution with constant probability. To obtain the ratio with high probability, \( O(\log n) \) independent repetitions of the algorithm are typically required, which may be infeasible or undesirable in practice, especially in a streaming context. Moreover, the derandomization of algorithms for submodular optimization has proven difficult, although a method to derandomize some algorithms at the cost of a polynomial increase in time complexity was given by Buchbinder and Feldman [BF18]. Therefore, in this work we seek to design linear-time, streaming approximation algorithms that in addition are deterministic.

**Motivating Questions.** The fastest deterministic algorithm in prior literature with constant ratio is the \((4 + \varepsilon)\)-approximation of Kuhnle [Kuh19] which takes time \( O_\varepsilon(n \log(k)) \). The fastest streaming algorithm is the \( O_\varepsilon(n \log(k)) \)-time algorithm of Alaluf et al. [Ala+20]. Therefore, this work seeks to answer the questions:

\[
Q1: \text{Does there exist a deterministic, linear-time algorithm for SMCC with constant approximation factor?} \quad Q2: \text{Does there exist a linear-time, single-pass streaming algorithm for SMCC?}
\]

\(^1\)A function \( f \) is monotone if \( f(S) \leq f(T) \) whenever \( S \subseteq T \).
Table 1: The symbol * indicates that the ratio holds only in expectation; while † indicates the ratio does not hold under adversarial stream order. Two entries for Alaluf et al. [Ala+20] are shown with different choice of post-processing algorithm. **Top section:** state-of-the-art streaming algorithms; **Middle section:** deterministic and randomized algorithms with lowest time complexity; **Bottom section:** our algorithms.

| Reference       | Ratio        | Time                                      | Passes | Memory               |
|-----------------|--------------|-------------------------------------------|--------|----------------------|
| [Ala+20]+[BF16] | 3.597+\(\varepsilon\) * | \(O\left(\frac{1}{\varepsilon} \log \left(\frac{1}{\varepsilon}\right) (\frac{n}{\varepsilon} + \text{poly}\left(\frac{1}{\varepsilon}\right))\right)\) | 1      | \(O(k/\varepsilon^2)\) |
| [Ala+20]+[Kuh19]| 5+\(\varepsilon\) * | \(O\left(\frac{1}{\varepsilon} \log\left(\frac{1}{\varepsilon}\right)\left(\frac{n}{\varepsilon} + \frac{1}{\varepsilon} + \log\left(\frac{1}{\varepsilon}\right)\right)\right)\) | 1      | \(O(k/\varepsilon^2)\) |
| [LRVZ21]         | \(e + \varepsilon + o(1)^{\dagger}\) | \(O\left(\frac{1}{\varepsilon} k^2.5\right)\) | 1      | \(O(k/\varepsilon)\)  |
| [BF15]           | \(e + \varepsilon^*\) | \(O\left(\frac{\varepsilon}{k} \log\left(\frac{1}{k}\right)\right)\) | \(k\) | \(O(n)\)             |
| [Kuh19]          | 4+\(\varepsilon\) | \(O\left(\frac{1}{\varepsilon} k^2\log\left(\frac{\varepsilon}{k}\right)\right)\) | \(k\) | \(O(k)\)             |

In the case the function \(f\) is monotone, Kuhnle [Kuh21] obtained a deterministic, single-pass streaming algorithm with ratio 4 in linear time; and a deterministic, multi-pass streaming algorithm with ratio 1.582+\(\varepsilon\) in linear time, which is nearly the optimal ratio in the value query model for monotone SMCC [NW78]. However, the general (i.e. non-monotone) case is more difficult; many algorithms for the general case depend on a critical lemma (Lemma 2.2) of Feige, Mirrokni, and Vondrák [FMV11] that establishes a partial monotonicity in the presence of randomness.

1.1 Contributions

Our first contribution answers Q1 affirmatively. We provide a linear-time, deterministic approximation algorithm (Alg. 2) with ratio at most 11.657 in Section 3. Although Alg. 2 takes one pass through the ground set, it is not a streaming algorithm since it may store \(O(n)\) elements. Next, we modify Alg. 2 to stay within \(O(k \log n)\) memory and hence answer Q2 affirmatively (Section 4). The result is a single-pass, deterministic streaming algorithm \textsc{LinearStream} that runs in linear time, with ratio 23.314+\(\varepsilon\).

The algorithm \textsc{LinearStream} requires as a subroutine a deterministic, linear-time algorithm for the unconstrained maximization problem that, upon receival of a new element, can update its solution in constant time and maintain a competitive ratio to the offline optimal. No such algorithm exists in prior literature, so we provide a 4-competitive algorithm Alg. 1. Since this algorithm uses related ideas to our other algorithms and is the simplest, we present it first in Section 2.

Next, we improve the best approximation ratio in deterministic linear time.
to $4 + \varepsilon$ by allowing multiple passes through the ground set. In Section 5, we provide MultiPassLinear, a multi-pass streaming algorithm that builds upon LinearStream and runs in $O\left(\frac{n}{\varepsilon} \log \left(\frac{1}{\varepsilon}\right)\right)$ time.

Finally, an empirical evaluation in Section 6 shows improvement in query complexity and solution value of our single-pass algorithm over the current state-of-the-art streaming algorithms on two applications of SMCC. Table 1 shows how our algorithms compare theoretically to state-of-the-art algorithms for SMCC.

Preliminaries. An alternative characterization of submodularity is the following: $f$ is submodular iff. $\forall A, B \subseteq U, f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. We use the following notation of the marginal gain of adding $T \subseteq U$ to set $S \subseteq U$: $\delta_T(S) = f(S \cup T) - f(S)$. For element $x \in U$, $\delta_x(S) = \delta_{\{x\}}(S)$.

1.2 Additional Related Work

In the rest of this section, we focus on the most closely related works to ours; specifically, state-of-the-art fast and streaming algorithms for general SMCC.

Single-Pass Streaming Algorithms. Alaluf et al. [Ala+20] introduced a single-pass streaming algorithm that obtains ratio $1 + \alpha + \varepsilon$, where $\alpha$ is the ratio of an offline post-processing algorithm $A$ for SMCC with time complexity $T(A, m)$ on an input of size $m$. The time complexity of their algorithm is $O\left(\frac{\log(k/\varepsilon)}{\varepsilon} \cdot \left(\frac{n}{\varepsilon} + T(A, k/\varepsilon)\right)\right)$. The currently best offline ratio that may be used for $A$ is the $2.597$ algorithm of Buchbinder and Feldman [BF16], which yields ratio $3.597 + \varepsilon$ in expectation for Alaluf et al. [Ala+20] in polynomial time. This is the state-of-the-art ratio for single-pass streaming under no assumptions on the stream order. If the $4 + \varepsilon$ algorithm of Kuhnle [Kuh19] is used for post-processing, the resulting algorithm is a deterministic, single-pass algorithm with time complexity $O\left(\frac{n}{\varepsilon} \log \left(\frac{k}{\varepsilon}\right)\right)$ and ratio $5 + \varepsilon$; this is the state-of-the-art time complexity for a single-pass streaming algorithm. While we do not improve on the state-of-the-art ratio in this paper, we improve the state-of-the-art time complexity to $O(n)$ with LinearStream.

To the best of our knowledge the only algorithm for the general SMCC under random stream order is that of Liu et al. [LRVZ21]. Their algorithm achieves ratio $e + \varepsilon + o(1)$ in expectation with time complexity $O\left(nk^{2.5}/\varepsilon^3\right)$. We compare with this algorithm empirically in Section 6 and find that due to the large numbers of queries involved, this algorithm only completes on very small instances.

Fastest Algorithms. Kuhnle [Kuh19] presented a deterministic algorithm that achieves ratio $4 + \varepsilon$ in $O\left(\frac{n}{\varepsilon} \log \left(\frac{1}{\varepsilon}\right)\right)$ time; this is the fastest deterministic algorithm in previous literature. Our multipass algorithm MultiPassLinear obtains ratio $4 + \varepsilon$ in $O\left(\frac{n}{\varepsilon} \log \left(\frac{1}{\varepsilon}\right)\right)$ time, thereby improving the state-of-the-art time complexity to linear time while keeping the same ratio. In the case of randomized algorithms, Buchbinder, Feldman, and Schwartz [BFS15] obtained ratio $e + \varepsilon$ in expectation in $O\left(\frac{n}{\varepsilon^2} \log \left(\frac{1}{\varepsilon}\right)\right)$ time.

Unconstrained Maximization. Given a submodular function $f$, the unconstrained maximization problem (UNCMAX) is to determine $\arg \max_{S \subseteq N} f(S)$. 4
This problem is also NP-hard; and a \((2 - \varepsilon)\)-approximation requires exponentially many oracle queries [FMV11]. Buchbinder et al. [BFNS12] gave a 2-approximation algorithm for UncMax in linear time. In our work, we need a deterministic, linear-time algorithm for UncMax that can update its solution on receipt of a new element in constant time, as described in Section 4. Buchbinder, Feldman, and Schwartz [BFS14] give two online algorithms for UncMax with competitive ratios 4 and \(e\), but these algorithms do not meet our requirements. Therefore, we present Alg. 1 in Section 2.

2 Linear-Time Algorithm for UncMax with Competitive Ratio

Algorithm 1 The 4-competitive linear-time algorithm for UncMax.

1: procedure \((f)\)
2: \textbf{Input:} oracle \(f\)
3: \(X \leftarrow \emptyset, Y \leftarrow \emptyset\)
4: for element \(e\) received do
5: \(S \leftarrow \arg \max\{\delta_e(X), \delta_e(Y)\}\)
6: if \(\delta_e(S) > 0\) then
7: \(S \leftarrow S \cup \{e\}\)
8: \(S' \leftarrow \arg \max\{f(X), f(Y)\}\)
9: return \(S'\)

In this section, we present a 4-competitive linear-time algorithm for UncMax, which is required by our single-pass algorithm as a subroutine in Section 4. In addition, this algorithm can be regarded as the starting point of our algorithms for cardinality constraint. The latter algorithms can be regarded as increasingly sophisticated adaptations of the basic approach to handle first the cardinality constraint and subsequently the memory requirements for streaming.

Algorithm Overview. The algorithm (Alg. 1) maintains two candidate solutions \(X\) and \(Y\), which are initially empty. As each element \(e\) is received, it is added to the set to which it gives the largest marginal gain, as long as such marginal gain is non-negative. Let \(X_l, Y_l\) denote the value of \(X, Y\), respectively, after receipt of \(\{e_1, \ldots, e_l\}\). Below, we show a competitive ratio of 4 to the maximum on the set of elements received thus far; i.e.

\[
4 \max\{f(X_l), f(Y_l)\} \geq \max_{S \subseteq \{e_1, \ldots, e_l\}} f(S).
\]

While this is the standard definition of competitive ratio, Alg. 1 is not an online algorithm as defined in Buchbinder, Feldman, and Schwartz [BFS14] (see discussion in Appendix A). Further, none of the existing online algorithms satisfy the properties needed by LinearStream, which are described in Section 4.
**Theorem 1.** Alg. 1 is a deterministic, linear-time algorithm for UncMax with competitive ratio 4, which runs in linear time in the number of received elements and updates its solution in constant time upon receipt of a new element.

**Proof.** Suppose Alg. 1 has received elements \( \{e_1, \ldots, e_l\} \); and let \( X, Y \) have their values after processing these elements; let \( S = \arg \max \{f(X), f(Y)\} \). Let \( O \subseteq \{e_1, \ldots, e_l\} \) satisfy \( f(O) = \text{UncMax}(\{e_1, \ldots, e_l\}) \). We will show that \( 4f(S) \geq f(O) \). For each \( o \in O \cap Y \), let \( Y_i(o) \) denote the value of \( Y \) at the beginning of the iteration in which \( o \) was added to \( Y \). We have

\[
\begin{align*}
    f(O \cup X) - f(X) & \leq \sum_{o \in O \setminus (X \cup Y)} \delta_o(X) + \sum_{o \in O \cap Y} \delta_o(X) \\
                  & \leq 0 + \sum_{o \in O \cap Y} \delta_o(Y_i(o)) \leq f(Y),
\end{align*}
\]

where the second inequality follows from submodularity and the comparison on Line 5. Analogously, \( f(O \cup Y) - f(Y) \leq f(X) \). Hence

\[
f(O) \leq f(O \cup X) + f(O \cup Y) \leq 2(f(X) + f(Y)) \leq 4f(S),
\]

where the first inequality follows by submodularity, nonnegativity of \( f \) and the fact that \( X \cap Y = \emptyset \). \( \square \)

### 3 Linear-Time Algorithm for SMCC

In this section, we present a 11.657-approximation, linear-time algorithm for SMCC. The algorithm is an adaptation of Alg. 1 to handle the size constraint.

**Algorithm 2** The \((2b + 4)(1 + 1/b)\)-competitive algorithm for SMCC.

1: procedure \((f, b, k)\)
2: Input: oracle \( f \), \( b > 0, k \in \mathbb{N} \)
3: \( X \leftarrow \emptyset, Y \leftarrow \emptyset \)
4: for element \( e \) received do
5: \( S \leftarrow \arg \max \{\delta_e(X), \delta_e(Y)\} \)
6: if \( \delta_e(S) > bf(S)/k \) then
7: \( S \leftarrow S \cup \{e\} \)
8: \( X' \leftarrow \{k \text{ elements most recently added to } X\} \)
9: \( Y' \leftarrow \{k \text{ elements most recently added to } Y\} \)
10: return \( S' \leftarrow \arg \max \{f(X'), f(Y')\} \)

**Algorithm 2 Overview.** The algorithm has a strategy similar to Alg. 1, with two differences. First, an element is only added to \( S \in \{X, Y\} \) if its marginal gain is at least a threshold of \( bf(S)/k \), where \( b \) is a parameter and \( k \) is the cardinality constraint; in the unconstrained version, this threshold
was 0. Second, instead of returning max\{f(X), f(Y)\}, the algorithm instead considers the last k elements added to X or Y: X' and Y', respectively. Part of the analysis below shows that because each addition results in \(Ω(f(S)/k)\) gain in value, there is a concentration of value in the last k elements of each set. Rather than an approximation ratio, we again prove a competitive ratio after each element is received, but the ratio has worsened from 4 to \(\leq 11.657\) which is achieved with \(b = \sqrt{2}\).

**Theorem 2.** Alg. 2 is a deterministic, linear-time algorithm for SMCC with competitive ratio \((2b + 4)(1 + 1/b)\), which runs in linear time in the number of received elements.

**Proof.** Let X, Y have their values after receiving a set \(\mathcal{N}\) of elements; let \(O \subseteq \mathcal{N}\) be an optimal solution to SMCC \((f, \mathcal{N}', k)\). First, we will bound \(f(O)\) in terms of \(M_k = \max\{f(X), f(Y)\}\); subsequently, we will bound \(M_1\) in terms of \(M_2 = \max\{f(X'), f(Y')\}\). Observe that \(f(X), f(Y)\) do not decrease during the execution of the algorithm; so that, at any point during the execution, we have \(\max\{f(X), f(Y)\} \leq M_1\).

For each \(o \in O \cap Y\), let \(Y_{i(o)}\) denote the value of \(Y\) at the beginning of the iteration in which \(o\) was added to \(Y\). We have

\[
f(O \cup X) - f(X) \leq \sum_{o \in O \setminus (X \cup Y)} \delta_o(X) + \sum_{o \in O \cap Y} \delta_o(X)
\leq bM_1 + \sum_{o \in O \cap Y} \delta_o(Y_{i(o)}) \leq bM_1 + f(Y),
\]

where the second inequality follows from submodularity and the comparisons on Lines 5 and 6. Analogously, \(f(O \cup Y) - f(Y) \leq bM_1 + f(X)\). Hence

\[
f(O) \leq f(O \cup X) + f(O \cup Y) \leq 2(f(X) + f(Y)) + 2bM_1 \leq (4 + 2b)M_1,
\]

where the first inequality follows by submodularity, nonnegativity of \(f\) and the fact that \(X \cap Y = \emptyset\).

Next, we turn to the bound of \(M_1\) in terms of \(M_2\). Consider \(X' \subseteq X\); we will show that \(f(X) \leq (1 + 1/b)f(X')\). First, if \(|X'| < k\), then \(X' = X\). So assume that \(|X'| = \{x_1, \ldots, x_k\}\), where the order is by when these elements were added to \(X\). Let \(X'_i = \{x_1, \ldots, x_i\}\). For \(i \in \{1, \ldots, k\}\), we have

\[
f(X'_i) - f(X'_{i-1}) \geq f((X \setminus X') \cup X'_i) - f((X \setminus X') \cup X'_{i-1})
\geq \frac{b}{k} f((X \setminus X') \cup X'_{i-1})
\geq \frac{b}{k} f(X \setminus X'),
\]

where Inequality 2 follows from submodularity of \(f\); Inequality 3 follows from the addition of \(x_i\) on Line 6; and Inequality 4 follows from the fact that \(f(X)\) does not decrease during the execution of the algorithm. The summation of
Algorithm 3 A single-pass algorithm for SMCC.

1: procedure LinearStream($f, k, \varepsilon, b$)
2:     Input: oracle $f$, cardinality constraint $k$, $\varepsilon > 0$, $b > 0$
3:     $\alpha \leftarrow 1 + \beta/b$
4:     $\ell \leftarrow \lceil \log((6\alpha)/\varepsilon + 1) \rceil + 3$
5:     $A \leftarrow \emptyset$, $B \leftarrow \emptyset$, $\tau \leftarrow \text{um}_A \leftarrow \text{um}_B \leftarrow f(\emptyset)$
6:     for element $e$ received do
7:         $S \leftarrow \text{arg max}\{\delta_e(A), \delta_e(B)\}$
8:         if $\delta_e(S) \geq b\tau/k$ then
9:             $S \leftarrow S \cup \{e\}$
10:            $\text{um}_S = \text{UncMaxUpdate}(e)$
11:            if max\{$\text{um}_S, f(S)$\} $> \tau$ then
12:                $\tau \leftarrow \text{max}\{\text{um}_S, f(S)\}$
13:        if $|S| > 2\ell(k/b + 1) \log_2(k)$ then
14:            $S \leftarrow \{\ell(k/b + 1) \log_2(k) \text{ elements most recently added to } S\}$
15:            $\text{um}_S \leftarrow \text{UncMaxCompetitiveAlg}(S)$
16:            $\tau \leftarrow \text{max}\{f(A), f(B), \text{um}_A, \text{um}_B\}$.
17:        $A' \leftarrow \{k \text{ elements most recently added to } A\}$.
18:        $B' \leftarrow \{k \text{ elements most recently added to } B\}$.
19:     return $S' \leftarrow \text{arg max}\{f(A'), f(B')\}$

these inequalities yields $f(X') - f(\emptyset) \geq bf(X \setminus X')$. By submodularity and nonnegativity of $f$, $f(X) \leq f(X \setminus X') + f(X') \leq (1 + 1/b)f(X')$. Symmetrically, $f(Y) \leq (1 + 1/b)f(Y')$, so we have $M_1 \leq (1 + 1/b)M_2$. Together with Inequality 1, we have $f(O) \leq (4 + 2b)M_1 \leq (4 + 2b)(1 + 1/b)M_2$. □

4 Single-Pass Streaming Algorithm for SMCC

In this section, a linear-time, constant-factor algorithm is described. This algorithm (LinearStream, Alg. 3) is a deterministic streaming algorithm that makes one pass through the ground set.

LinearStream Overview. The starting point of the algorithm is Alg. 2 of Section 3; several modifications are needed to ensure the algorithm stays with $O_\epsilon(k \log n)$ space. First, we add a deletion procedure on Line 14. The intuition is that if the size of $A$ (resp. $B$) is large, then because the threshold required to add elements on Line 8 depends on $f(A)$, the value of the initial elements is small. Therefore, deleting these elements can cause only a small loss in the value of $f(A)$. However, because $f$ may be non-monotone, such deletion may actually cause an increase in the value of $f(A)$, which interferes with the concentration of value of $A$ into its last $k$ elements. Therefore, to ensure enough value accumulates in the last $k$ elements, we need to ensure that each addition
adds $\Omega(um(A)/k)$ value, where $um(A)$ is the solution to the UncMAX problem restricted to $A$.

To ensure each addition adds $\Omega(um(A)/k)$ value, we require a $\beta$-competitive algorithm for UncMAX; and to ensure our algorithm stays linear-time, we need to be able to update the estimate for $um(A)$ to $um(A \cup \{e\})$ in constant time. Therefore, we need algorithms UncMaxCompetitiveAlg and UncMaxUpdate such that 1) both algorithms are deterministic; 2) UncMaxCompetitiveAlg is linear-time; 3) UncMaxUpdate is constant time; 4) a $\beta$-competitive estimate of $um(A)$ is maintained. Observe that using Alg. 1 (Alg. 2, Section 2) for UncMaxCompetitiveAlg; and Lines 5–8 of Alg. 1 for UncMaxUpdate, all of the above requirements are met with $\beta = 4$.

**Theoretical Guarantees.** Next, we prove the following theorem concerning the performance of LinearStream (Alg. 3). With $\beta = 4$, the ratio is optimized to $12 + 8\sqrt{2} \leq 23.314 + \varepsilon$ at $b = 2\sqrt{2}$.

**Theorem 3.** Let $\varepsilon, b \geq 0$, and let $(f, k)$ be an instance of SMCC; and suppose UncMaxCompetitiveAlg and UncMaxUpdate satisfy the requirements discussed in Section 4. Then the solution $S'$ returned by LinearStream$(f, k, \varepsilon, b)$ satisfies

$$OPT \leq ((2b + 4)(1 + \beta/b + \varepsilon) f(S')).$$

Further, LinearStream has time complexity $O(n)$, memory complexity $O(k \log(k) \log(1/\varepsilon))$, and makes one pass over the ground set.

**Proof of Theorem 3.** The time and memory complexities of LinearStream are immediate, so we focus on the approximation ratio. The first lemma (Lemma 1) establishes basic facts about the growth of the value in the sets $A$ and $B$ as elements are received. Lemma 1 considers a general sequence of elements that satisfy the same conditions on addition and deletion as elements of $A$ or $B$, respectively. The proof is deferred to Appendix B.1 and depends on a condition to add elements and uses submodularity of $f$ to bound the loss in value due to periodic deletions.

**Lemma 1.** Let $(c_0, \ldots, c_{m-1})$ be a sequence of elements, and $(C_0, \ldots, C_m)$ a sequence of sets, such that $C_0 = \emptyset$, and $C_i^+ = C_i \cup \{c_i\}$ satisfies $f(C_i^+)^+ \geq (1 + b/k) f(C_i)$, and $C_{i+1}^+ = C_{i+1}^+$, unless $|C_i^+| > 2f(k/b + 1) \log_2(k)$, in which case $C_{i+1}^+ = C_{i+1}^+ \setminus C_j$, where $j = i - \ell(k/b + 1) \log_2(k)$. Then 1) $f(C_{i+1}^+) \geq f(C_i)$, for any $i \in \{0, \ldots, m - 1\}$; and 2) Let $C^* = \{c_0, \ldots, c_{m-1}\}$. Then $f(C^*) \leq \left(1 + \frac{1}{k^i-1}\right) f(C_m)$.

**Notation.** Next, we define notation used throughout the proof. Let $A_i, B_i$ denote the respective values of variables $A, B$ at the beginning of the $i$-th iteration of for loop; let $A_{n+1}, B_{n+1}$ denote their respective final values. Also, let $A^* = \bigcup_{1 \leq i \leq n+1} A_i$; analogously, define $B^*$. Let $e_i$ denote the element received at the beginning of iteration $i$. We refer to line numbers of the pseudocode Alg. 3. Notice that after deletion of duplicate entries, the sequences $(A_n), (B_n)$ satisfy the hypotheses of Lemma 1, with the sequence of elements in $A^*, B^*$,
respectively. Since many of the following lemmata are symmetric with respect to \( A \) and \( B \), we state them generically, with variables \( C, D \) standing in for one of \( A, B \), respectively. The notations \( C_i, C^*, D_i, D^* \) are defined analogously to \( A_i, A^* \) defined above. Finally, if \( D \in \{ A, B \} \), define \( \Delta D_i = f(D_{i+1}) - f(D_i) \).

Observe that \( \sum_{i=0}^{n-1} \Delta D_i = f(D_{n+1}) - f(\emptyset) \).

The analyses of both Algs. 1 and 2 above use the fact that the marginal gain of an element to one set can be bounded by the increase in value of the other because of the competition between the sets (i.e. the comparison on Line 7).

If a deletion occurs after the comparison on Line 7, this bound may no longer hold. The next lemma shows that an approximate form of the bound holds.

**Lemma 2.** Let \( C, D \in \{ A, B \} \), such that \( C \neq D \). Let \( o \in O \cap D^* \). Let \( i(o) \) denote the iteration in which \( o \) was processed. Then

\[
\delta_o(C_{i(o)}) \leq \frac{\Delta D_{i(o)} + \gamma f(D_{n+1})}{1 + \gamma}.
\]

The next lemma uses Lemma 2 to bound the gain of adding the entire set \( O \) into \( C^* \).

**Lemma 3.** Let \( C, D \in \{ A, B \} \), such that \( C \neq D \). Then

\[
f(C^* \cup O) - f(C^*) \leq bf(C_{n+1}) + (1 + k\gamma)f(D_{n+1})
\]

As in the analysis of Alg. 2, we need to show a concentration of value in the last \( k \) elements added to our sets. The next lemma accomplishes this by using that each element gives a gain of \( \Omega(\text{um}(C)) \) by using the \( \beta \)-competitive procedure for UncMax.

**Lemma 4.** Let \( C \in \{ A, B \} \), and let \( C' \subseteq C_{n+1} \) be the set of \( \min\{|C_{n+1}|, k\} \) elements most recently added to \( C_{n+1} \). Then \( f(C_{n+1}) \leq (1 + \beta/b) f(C') \).

**Proof.** For simplicity of notation, let \( C = C_{n+1} \). If \( |C| \leq k \), the result follows since \( C' = C \). So suppose \( |C| > k \), and let \( C' = \{c_1, \ldots, c_k\} \) be ordered by the iteration in which each element was added to \( C \). Also, let \( C'_i = \{c_1, c_2, \ldots, c_i\} \), for \( 1 \leq i \leq k \), and let \( C'_0 = \emptyset \). Let \( C_i \) denote the value of \( C \) at the beginning of the iteration in which \( c_i \) is added. For any set \( X \), let \( \text{um}(X) \) abbreviate UncMax\((X)\).

Observe that \( C_i \setminus C' \geq C \setminus C' \) for all \( 1 \leq i \leq k \), regardless of whether a deletion occurs at any point during the addition of elements of \( C' \). From this observation, submodularity, and the condition to add an element to \( C \) on Line 8 and the fact that UncMaxCompetitiveAlg is a \( \beta \)-competitive algorithm, we have that

\[
f(C'_i) - f(C'_{i-1}) \geq f((C_i \setminus C) \cup C'_i) - f((C_i \setminus C) \cup C'_{i-1})
\]

\[
\geq \frac{b \cdot \text{um}(C_i \setminus C') \cup C'_{i-1})}{\beta k} \geq \frac{b}{\beta k} \text{um}(C \setminus C').
\]
Therefore, \( f(C') \geq (b/\beta)u_m(C \setminus C') \). Hence, by submodularity, nonnegativity of \( f \), we have
\[
f(C) \leq f(C \setminus C') + f(C') \leq u_m(C \setminus C') + f(C') \leq (1 + \beta/b)f(C'). \quad \square
\]

By application of Lemma 3 with \( A = C \) and then again with \( B = C \), we obtain
\[
\delta_O(A^*) \leq bf(A_{n+1}) + (1 + k\gamma)f(B_{n+1}), \quad \text{and} \quad \delta_O(B^*) \leq bf(B_{n+1}) + (1 + k\gamma)f(A_{n+1}).
\]

Next, we have that
\[
f(O) \leq f(A^* \cup O) + f(B^* \cup O)
\leq f(A^*) + f(B^*) + \((b + 1 + k\gamma)(f(A_{n+1}) + f(B_{n+1}))\),
\]
where Inequality 7 follows from the fact that \( A^* \cap B^* = \emptyset \) and submodularity and nonnegativity of \( f \). Inequality 8 follows from the summation of Inequalities 5 and 6. By application of Property 2 of Lemma 1, we have from Inequality 8
\[
f(O) \leq (b + 2 + (k + 2)\gamma)(f(A_{n+1}) + f(B_{n+1}))
\leq (2b + 4 + 2(k + 2)\gamma)f(C_{n+1}),
\]
where \( C_{n+1} = \arg \max\{f(A_{n+1}), f(B_{n+1})\} \). Observe that the choice of \( \ell \) on Line 4 ensures that \( 2(k + 2)\gamma < \varepsilon(1 + \beta/b)^{-1} \), by Lemma 6. Therefore, by application of Lemma 4, we have from Inequality 9
\[
f(O) \leq ((2b + 4)(1 + \beta/b + \varepsilon))f(C'). \quad \square
\]

### 4.1 Post-Processing: LinearStream+

In this section, we briefly describe a modification LinearStream+ that improves the empirical performance of LinearStream. Instead of choosing, on Line 19, the best of \( A' \) and \( B' \) as the solution; introduce a third candidate solution as follows: use an offline algorithm for SMCC in a post-processing procedure on the restricted universe \( A \cup B \) to select a set of size at most \( k \) to return. This method can only improve the objective value of the returned solution and therefore does not compromise the theoretical analysis of the preceding section. The empirical solution value can be further improved by lowering the parameter \( b \) as this increases the size of \( A \cup B \), potentially improving the quality of the solution found by the selected post-processing algorithm.

### 5 Multi-Pass Streaming Algorithm for SMCC

In this section, we describe a multi-pass streaming algorithm for SMCC that obtains ratio \( 4 + O(\varepsilon) \) in linear time.
Algorithm 4 A multi-pass algorithm for SMCC.

1: procedure MultiPassLinear($f, k, \varepsilon, \Gamma, \alpha$) 
2: Input: oracle $f$, cardinality constraint $k$, $\varepsilon > 0$, parameters $\Gamma, \alpha$, such that $\Gamma \leq \text{OPT} \leq \Gamma/\alpha$. 
3: $\tau \leftarrow \Gamma/(4k\alpha)$ & Choice satisfies $\tau \geq \text{OPT}/(4k)$. 
4: while $\tau \geq \varepsilon \Gamma/(16k)$ do 
5: for $u \in \mathcal{N}$ do 
6: $S \leftarrow \arg \max \left\{ \delta_u(X) : X \in \{A, B\} \text{ and } |X| < k \right\}$ & If arg max is empty, break from loop. 
7: if $\delta_u(S) \geq \tau$ then 
8: $S \leftarrow S \cup \{u\}$ 
9: $\tau \leftarrow \tau(1 - \varepsilon)$ 
10: return $S \leftarrow \arg \max \{f(A), f(B)\}$

Algorithm Overview. The algorithm MultiPassLinear (Alg. 4) starts with $\Gamma$, an initial estimate of OPT, which is used to compute an upper bound for $\text{OPT}/(4k)$. Then, a fast greedy approach with descending thresholds is used, in which two disjoint sets $A$ and $B$ compete for elements with gain above a threshold. This scheme is related to the approach of FastInterlaceGreedy algorithm of Kuhnle [Kuh19], as discussed further in Appendix A. We improve by a $\log(k)$ factor to achieve linear time since the interval containing OPT is reduced to constant size by LinearStream.

We prove the following theorem concerning the performance of MultiPassLinear (Alg. 4).

**Theorem 4.** Let $0 \leq \varepsilon \leq 1/2$, and let $(f, k)$ be an instance of SMCC, with optimal solution value OPT. Suppose $\Gamma, \alpha \in \mathbb{R}$ satisfy $\Gamma \leq \text{OPT} \leq \Gamma/\alpha$. The solution $S$ returned by MultiPassLinear($f, k, \varepsilon, \Gamma, \alpha$) satisfies $\text{OPT} \leq (4 + 6\varepsilon)f(S)$. Further, MultiPassLinear has time and query complexity $O\left(\frac{n}{\varepsilon} \log \left(\frac{1}{\alpha \varepsilon}\right)\right)$, memory complexity $O(k)$, and makes $O(\log(1/(\alpha \varepsilon))/\varepsilon)$ passes over the ground set.

6 Empirical Evaluation

In this section, we evaluate our single-pass algorithms LinearStream (LS) and LinearStream+ (LS+) in the context of the following single-pass streaming algorithms:

- Algorithm 3 (LRVZ) of Liu, Rubinstein, Vondrak, and Zhao [LRVZ21], which achieves ratio $e + \varepsilon + o(1)$ in expectation in time $O\left(\frac{n}{\varepsilon^2} k^{2.5}\right)$, if the stream is in random order.
- Algorithm 2 (FKK) of Feldman, Karbasi, and Kazemi [FKK18]; this algorithm achieves ratio 5.828 in expectation and has $O(kn)$ time complexity.
Algorithm 1 (AEFNS) of Alaluf, Enc, Feldman, Nguyen, and Suh [Ala+20]; the implementation of this algorithm requires choice of a post-processing algorithm. For a fair comparison, AEFNS and LS+ used the same post-processing algorithm MultiPassLinear as discussed in Appendix D.2, which for AEFNS yields ratio $5 + \varepsilon$ and time complexity $O_\varepsilon(n \log k)$.

Randomized algorithms were repeated 40 times; plots show sample mean (symbol) and standard deviation (shaded region) of each metric. A timeout of four hours was used for each repetition. LRVZ received the stream in uniformly random order for each repetition; all other algorithms used the stream order determined by the data representation.

**Applications and Datasets.** The algorithms were evaluated on two applications: cardinality constrained maximum cut (maxcut) and revenue maximization on social networks (revmax). A variety of network topologies from the Stanford Large Network Dataset Collection [LK20] were used, as well as synthetic random graphs. For more details on the applications and datasets, see Appendix D.2.

**Results.** Results for the objective value (normalized by the standard greedy value) and total queries (normalized by the number of vertices $n$ in the graph) for each application are shown in Fig. 1 as the cardinality constraint $k$ varies. Results on other datasets exhibited qualitatively similar trends and are given in Appendix D.2.

**Discussion.** LS+ returned nearly the greedy value ($\geq 90\%$) on all instances while using $\approx 2n$ queries. The objective value of LS+ exceeded all competitors
with the exception of LRVZ, which required more than 1000n queries (despite being an idealized implementation, see Appendix D.2) on the instances in which it completed within the timeout of 4 hours.
References

[Ala+20] Naor Alaluf, Alina Ene, Moran Feldman, Huy L. Nguyen, and Andrew Suh. “Optimal streaming algorithms for submodular maximization with cardinality constraints”. In: 47th International Colloquium on Automata, Languages, and Programming (ICALP). 2020.

[BF16] Niv Buchbinder and Moran Feldman. “Constrained Submodular Maximization via a Non-symmetric Technique”. In: Mathematics of Operations Research 44.3 (2016).

[BF18] Niv Buchbinder and Moran Feldman. “Deterministic Algorithms for Submodular Maximization”. In: ACM Transactions on Algorithms 14.3 (2018).

[BFNS12] Niv Buchbinder, Moran Feldman, Joseph Seffi Naor, and Roy Schwartz. “A Tight Linear Time (1 / 2)-Approximation for Unconstrained Submodular Maximization”. In: Symposium on Foundations of Computer Science (FOCS). 2012.

[BFS14] Niv Buchbinder, Moran Feldman, and Roy Schwartz. “Online Submodular Maximization with Preemption”. In: ACM-SIAM Symposium on Discrete Algorithms. 2014.

[BFS15] Niv Buchbinder, Moran Feldman, and Roy Schwartz. “Comparing Apples and Oranges: Query Tradeoff in Submodular Maximization”. In: ACM-SIAM Symposium on Discrete Algorithms (SODA). 2015.

[BMKK14] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. “Streaming Submodular Maximization: Massive Data Summarization on the Fly”. In: ACM SIGKDD Knowledge Discovery and Data Mining (KDD). 2014, pp. 671–680.

[BV14] Ashwinkumar Badanidiyuru and Jan Vondrák. “Fast algorithms for maximizing submodular functions”. In: ACM-SIAM Symposium on Discrete Algorithms (SODA). 2014.

[CGQ15] Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. “Streaming Algorithms for Submodular Function Maximization”. In: International Colloquium on Automata, Languages, and Programming (ICALP). 2015.

[CK15] Amit Chakrabarti and Sagar Kale. “Submodular maximization meets streaming: matchings, matroids, and more”. In: Mathematical Programming 154.1-2 (2015), pp. 225–247.

[EDFK17] Ethan R. Elenberg, Alexandros G. Dimakis, Moran Feldman, and Amin Karbasi. “Streaming Weak Submodularity: Interpreting Neural Networks on the Fly”. In: Advances in Neural Information Processing Systems (NeurIPS). 2017.
[Mir+15] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrak, and Andreas Krause. “Lazier Than Lazy Greedy”. In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2015.

[Mis+08] Alan Mislove, Hema Swetha Koppula, Krishna P Gummadi, Peter Druschel, and Bobby Bhattacharjee. “Growth of the Flickr Social Network”. In: *First Workshop on Online Social Networks*. 2008.

[MJK18] Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. “Streaming Non-Monotone Submodular Maximization: Personalized Video Summarization on the Fly”. In: *AAAI Conference on Artificial Intelligence*. 2018.

[NW78] GL Nemhauser and L A Wolsey. “Best Algorithms for Approximating the Maximum of a Submodular Set Function”. In: *Mathematics of Operations Research* 3.3 (1978), pp. 177–188.
A Additional Discussion of Related Work

Online Algorithms. Buchbinder, Feldman, and Schwartz [BFS14] defined the following model for an online algorithm for the UncMax problem: the algorithm must maintain a solution $X$ with competitive ratio to the offline optimal; and when a new element $e$ is received, the algorithm must choose a subset $X' \subseteq X \cup \{e\}$ that maintains the competitive ratio. Since our Algorithm 1 keeps both $X,Y$ and may alternate which set is the solution, it does not satisfy this definition, although it does maintain the competitive ratio of $4$ as proven in Section 2.

Relationship of MultiPassLinear to FastInterlaceGreedy (FIG) of Kuhnle [Kuh19]. The overall strategy of FastInterlaceGreedy is very similar to MultiPassLinear. There are two main differences: 1) the initial threshold value of FIG is chosen using the $k$ times the maximum singleton value as an upper bound of OPT. In MultiPassLinear, a different upper bound on OPT is computed using the initial run of LinearStream. This difference is responsible for the improvement in asymptotic runtime. 2) The second difference is that FIG requires two runs of the descending threshold procedure. We require only a single run for MultiPassLinear due to a slight change in how elements are allotted to sets and a tightening of the analysis.

B Proofs for Section 4

B.1 Proof of Lemma 1

Claim 1. For any $y \geq 1$, $b > 0$, if $i \geq (k/b + 1) \log y$, then $(1 + b/k)^i \geq y$.

Proof. Follows directly from the inequality $\log x \geq 1 - 1/x$ for $x > 0$. □

Proof of Property 1 of Lemma 1. If no deletion is made at element $i$ of the sequence, then the result follows directly from $f(C_i^+) \geq (1+b/k) f(C_i)$. So suppose deletion of set $C_j$ from $C_i$ occurs. Observe that $C_{i+1} = (C_i \setminus C_j) \cup \{c_i\}$, because the deletion is triggered by the addition of $c_i$ to $C_i$.

Claim 2. From index $j$ to index $i$, there have been $\ell(k/b + 1) \log_2(k) - 1 \geq (\ell - 1)(k/b + 1) \log_2(k)$ additions and no deletions in the sequence.

Proof. The criterion for deletion at index $l$ is $|C_{l-1}^+| > 2\ell(k/b + 1) \log_2(k)$. Since initially $C_0 = \emptyset$, a deletion occurs only at indices $l$ for which $|C_{l-1}^+| = 2\ell(k/b + 1) \log_2(k) + 1$; so $|C_l| = \ell(k/b + 1) \log_2(k) + 1$. Therefore, there are at least $\ell(k/b + 1) \log_2(k) - 1$ indices between successive deletions. □

If $f(C_i \setminus C_j) \geq f(C_i)$, the lemma follows from submodularity and the condition $f(C_i^+) \geq (1 + b/k) f(C_i)$. Therefore, for the rest of the proof, suppose $f(C_i \setminus C_j) < f(C_i)$.
It holds that
\[ f(C_i \setminus C_j) \geq (a) \quad f(C_i) - f(C_j) \geq \left( 1 + \frac{b}{k} \right)^{(t-1)(k/b+1)\log k} \cdot f(C_j) - f(C_i) \geq (b) \quad \frac{1}{k^{t-1}} \cdot f(C_j), \]

where Inequality a follows from submodularity and nonnegativity of \( f \), Inequality b follows from the fact that each addition from \( C_j \) to \( C_i \) increases the value of \( f(C) \) by a factor of at least \( 1 + b/k \), and Inequality c follows from Claim 1. Therefore
\[ f(C_i) \leq f(C_i \setminus C_j) + f(C_j) \leq \left( 1 + \frac{1}{k^{t-1}-1} \right) f(C_i \setminus C_j). \tag{10} \]

Next,
\[ f((C_i \setminus C_j) \cup \{e_i\}) - f(C_i \setminus C_j) \geq f(C_i \setminus \{e_i\}) - f(C_i) \geq bf(C_i)/k \geq (d) \quad f(C_i \setminus C_j)/k, \tag{11} \]

where Inequality d follows from submodularity; Inequality e is by the condition \( f(C_i^+) \geq (1 + b/k) f(C_i) \); and Inequality f holds since \( b \geq 1 \) and \( f(C_i) > f(C_i \setminus C_j) \). Finally, using Inequalities (10) and (11) as indicated below, we have
\[ f(C_{i+1}) = f(C_i \setminus C_j \cup \{e_i\}) \geq (e) \quad f(C_i \setminus C_j) \geq f(C_i) \geq f(C_i), \]

where the last inequality follows since \( k \geq 2 \) and \( t \geq 3 \).

**Proof of Property 2 of Lemma 1.**

**Lemma 5.** \( f(C^*) \leq \left( 1 + \frac{1}{k^{t-1}} \right) f(C_{n+1}) \).

**Proof.** Observe that \( C^* \setminus C_{n+1} \) may be written as the union of pairwise disjoint sets, each of which is size \( \ell(k/b+1) \log_2(k) \). Suppose there were \( m \) sets deleted during the sequence; write \( C^* \setminus C_{n+1} = \{D^i : 1 \leq i \leq m\} \), ordered such that \( i < j \) implies \( D^i \) was deleted after \( D^j \) (the reverse order in which they were deleted); finally, let \( D^0 = C_{n+1} \).

**Claim 3.** Let \( 0 \leq i \leq m \). Then \( f(D^i) \geq k^i f(D^{i+1}) \).

**Proof.** There are at least \( \ell(k/b+1) \log k + 1 \) elements added to \( C \) and exactly one deletion event during the period between starting when \( C = D^{i+1} \) until \( C = D^i \). Moreover, each addition except possibly one (corresponding to the deletion event) increases \( f(C) \) by a factor of at least \( 1 + b/k \). Hence, by Lemma 1 and Claim 1, \( f(D^i) \geq k^i f(D^{i+1}) \).
By Claim 3, for any $0 \leq i \leq m$, $f(C_{n+1} = D^0) \geq k^{\ell_i} f(D^i)$. Thus,

$$f(C^*) \leq f(C^* \setminus C_{n+1}) + f(C_{n+1}) \leq \sum_{i=0}^{m} f(D^i)$$

(Submodularity, Nonnegativity of $f$)

$$\leq f(C_{n+1}) \sum_{i=0}^{\infty} k^{-\ell_i} \quad \text{(Claim 3)}$$

$$= f(C_{n+1}) \left( \frac{1}{1 - k^{-\ell}} \right) \quad \text{(Sum of geometric series)}$$

$\square$

### B.2 Proof of Lemma 2

**Proof.** Since $o \in O \cap D^*$, we know that $o$ is added to the set $D$ during iteration $i(o)$; therefore, by the comparison on Line 7 of Alg. 3, it holds that

$$\delta_o (C_{i(o)}(o)) \leq \delta_o (D_{i(o)}(o)). \quad (12)$$

If no deletion from $D$ occurs during iteration $i(o)$, the lemma follows from the fact that $\Delta D_{i(o)} = \delta_o (D_{i(o)}(o))$.

For the rest of the proof, suppose that a deletion from $D$ does occur during iteration $i(o)$. For convenience, denote by $D^-$ the value of $D$ after the deletion from $D_{i(o)}$. By Inequality 10 in the proof of Lemma 1, it holds that

$$f(D_{i(o)}) \leq (1 + \gamma) f(D^-) \quad (13)$$

Hence,

$$\Delta D_{i(o)} + \gamma f(D_{n+1}) = f(D^- + o) - f(D_{i(o)}) + \gamma f(D_{n+1})$$

$$\geq (1 + \gamma) f(D^- + o) - f(D_{i(o)}) \quad (14)$$

$$\geq (1 + \gamma) f(D^-) - (1 + \gamma) f(D^-) \quad (15)$$

$$= (1 + \gamma) \delta_o (D^-) \quad \text{(Sum of geometric series)}$$

$$\geq (1 + \gamma) \delta_o (D_{i(o)}) \quad (16)$$

$$\geq (1 + \gamma) \delta_o (C_{i(o)}). \quad (17)$$

where Inequality 14 follows from Lemma 1, Inequality 15 follows from Inequality 13, Inequality 16 follows from submodularity of $f$, and Inequality 17 follows from Inequality 12. $\square$
#### B.3 Proof of Lemma 3

Proof.

\[
\begin{align*}
& f(C^* \cup O) - f(C^*) \leq \sum_{o \in O \setminus C^*} \delta_o(C^*) \\
& \leq \sum_{o \in O \setminus C^*} \delta_o(C_{i(o)}) \\
& \leq \sum_{o \in O \setminus C^*} \frac{bf(C_{i(o)})}{k} + \frac{\Delta D_{i(o)} + \gamma f(D_{n+1})}{1 + \gamma} \\
& \leq bf(C_{n+1}) + \frac{1 + k \gamma}{1 + \gamma} f(D_{n+1}) \\
& \leq bf(C_{n+1}) + (1 + k \gamma) f(D_{n+1}),
\end{align*}
\]

where Inequalities 18, 19 follow from submodularity of \(f\). Inequality 20 holds by the following argument: let \(o \in O \setminus C^*\). If \(o \notin D^*\), then it holds that \(\delta_o(C_{i(o)}) < bf(C_{i(o)})/k\) by Line 7. Otherwise, if \(o \in D^*\), Lemma 2 yields \(\delta_o(C_{i(o)}) \leq \frac{\Delta D_{i(o)} + \gamma f(D_{n+1})}{1 + \gamma}\). Inequality 21 follows from the fact that \(|O \setminus C^*| \leq k\), Lemma 1, and the fact that \(f(D_{n+1}) - f(\emptyset) = \sum_{i=0}^{n} \Delta D_{i}\), where each \(\Delta D_{i} \geq 0\). \(\square\)

#### B.4 Justification of choice of \(\ell\)

**Lemma 6.** Let \(\varepsilon > 0\), and let \(\alpha = 1 + \beta/b\). Choose \(\ell \geq 1 + \log((6\alpha)/\varepsilon + 1)\), and let \(\gamma = 1/(k^\ell - 1)\). Then

\[2(k + 2)\gamma < \varepsilon \alpha^{-1}.\]

Proof. First, one may verify that \(\ell > \frac{\log((2k + 4)\alpha) + 1}{\log k} \implies 2(k + 2)\gamma < \varepsilon \alpha^{-1}\). Next, since \(k \geq 1\),

\[
\frac{1}{\log k} \left( \log \left( \frac{(2k + 4)\alpha}{\varepsilon} + 1 \right) \right) \leq \frac{1}{\log k} \left( \log \left( \frac{(2 + 4)\alpha}{\varepsilon} + 1 \right) \right) \\
= \frac{1}{\log k} \left( \log \left( \frac{6\alpha}{\varepsilon} + 1 \right) + \log(k) \right) \\
\leq 1 + \log \left( \frac{6\alpha}{\varepsilon} + 1 \right).
\]

Hence it suffices to take \(\ell\) greater than the last expression. \(\square\)

#### C Proofs for Section 5

Proof. To establish the approximation ratio, consider first the case in which \(C \in \{A, B\}\) satisfies \(|C| = k\) after the first iteration of the while loop. Let
\( C = \{c_1, \ldots, c_k\} \) be ordered by the order in which elements were added to \( C \) on Line 7, let \( C_i = \{c_1, \ldots, c_i\}, C_0 = \emptyset \), and let \( \Delta C_i = f(C_i) - f(C_{i-1}) \). Then \( f(C) = \sum_{i=1}^k \Delta C_i \geq \Gamma/(4\alpha) \geq \text{OPT}/4 \), and the ratio is proven.

Therefore, for the rest of the proof, suppose \( |A| < k \) and \( |B| < k \) immediately after the execution of the first iteration of the \textbf{while} loop. First, let \( C, D \in \{A, B\} \), such that \( C \neq D \) have their values at the termination of the algorithm. For the definition of \( D' \) and the proofs of the next two lemmata, see Appendix C. These lemmata together establish an upper bound on \( \delta_O(C) \) in terms of the gains of elements added to \( C \) and \( D \).

**Lemma 7.**

\[
\sum_{o \in O \setminus (C \cup D')} \delta_o(C) \leq (1 + 2\varepsilon) \sum_{i : c_i \notin O} \Delta C_i + \varepsilon \text{OPT}/16.
\]

**Lemma 8.**

\[
\delta_O(C) \leq \sum_{i : d_i \notin O} \Delta D_i + (1 + 2\varepsilon) \sum_{i : c_i \notin O} \Delta C_i + \varepsilon \text{OPT}/16.
\]

Applying Lemma 8 with \( C = A \) and separately with \( C = B \) and summing the resulting inequalities yields

\[
\delta_O(A) + \delta_O(B) \leq (1 + 2\varepsilon) \left[ \sum_{i=1}^k \Delta B_i + \sum_{i=1}^k \Delta A_i \right] + \frac{\varepsilon \text{OPT}}{8}
\]

\[
= (1 + 2\varepsilon) \left[ f(A) + f(B) \right] + \varepsilon \text{OPT}/8.
\]

Thus,

\[
f(O) \leq f(O \cup A) + f(O \cup B)
\]

\[
\leq (2 + 2\varepsilon)(f(A) + f(B)) + \varepsilon \text{OPT}/8,
\]

from which the result follows.

Let \( C, D \in \{A, B\} \), such that \( C \neq D \) have their values at the termination of the algorithm. If \( |C| = k \), let \( D' \) have the value of its corresponding variable when the \( k \)th element is added to \( C \); otherwise, if \( |C| < k \) let \( D' = D \).

**Proof of Lemma 7.** Suppose \( |C| = k \). Let \( \tau' \) be the value of \( \tau \) during the iteration of the \textbf{while} loop in which the last element was added to \( C \). Let \( o \in O \setminus (C \cup D') \). Then, since \( o \) was not added to \( C \) or \( D' \) during the previous iteration of the \textbf{while} loop, \( \delta_o(C) < \tau'/(1 - \varepsilon) \). Further, \( \Delta C_i \geq \tau' \) for all \( i \). Hence,

\[
\sum_{o \in O \setminus (C \cup D')} \delta_o(C) \leq \frac{1}{1 - \varepsilon} \sum_{i : c_i \notin O} \Delta C_i
\]

\[
\leq (1 + 2\varepsilon) \sum_{i : c_i \notin O} \Delta C_i.
\]
Next, suppose that $|C| < k$. In this case, the last threshold $\tau$ of the while loop ensures that $\sum_{o \in O \setminus (C \cup D')} \delta_o(C) < \varepsilon \Gamma/16 \leq \varepsilon \text{OPT}/16$. □

Proof of Lemma 8. Observe that
\[
\delta_O(C) \leq \sum_{o \in O \cap D'} \delta_o(C) + \sum_{o \notin (C \cup D')} \delta_o(C),
\]
where Inequality 22 follows from submodularity and Inequality 23 follows from submodularity and the comparison on Line 6 for each element $o \in O \cap D'$. From Inequality 23, the lemma follows from application of Lemma 7. □

D Empirical Evaluation

The source code and scripts to reproduce all plots are given at https://gitlab.com/kuhnle/linear-nm.

D.1 Environment

All experiments were run on a linux server running Ubuntu 20.04, with 2 × Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz and 504 GB RAM.

D.2 Implementation and Parameter Settings

All algorithms were implemented in C++ and used the same code for evaluation of the application oracle. An optimized marginal gain computation was available to the algorithms that could benefit from such optimization and when the application permitted such optimization.

All algorithms used lazy evaluations whenever possible as follows. Suppose $\delta_x(S)$ has already been computed, and the algorithm needs to check if $\delta_x(T) \geq \tau$, for some $\tau \in \mathbb{R}$ and $T \supseteq S$. Then if $\delta_x(S) < \tau$, this evaluation may be safely skipped due to the submodularity of $f$. The single-pass streaming algorithms evaluated do not benefit from lazy evaluations, except for those algorithms (LS+ and AEFNS) that use post-processing.

The accuracy parameter $\varepsilon$ of each algorithm is set to 0.1. The parameter $b$ of LS is set to 1; while for LS+, $b$ is set to 0.1. These choices for $b$ worked well empirically, although they depart from the theoretical analysis; for a justification, see Section 4.1.

As mentioned in Section 6, both LS+ and AEFNS used MultiPassLinear for post-processing. Recall that MultiPassLinear requires an input of $\Gamma$ and $\alpha$. LS+ used its solution value (before post-processing) and its approximation ratio for $\Gamma$ and $\alpha$, respectively (which means that MultiPassLinear will run in linear-time for its post-processing). However, AEFNS does not have an approximation ratio before post-processing, so the maximum singleton value
Figure 2: Solution value vs. $k$ for single-pass algorithms for the maxcut application on each dataset.

and $k$ were used for $\Gamma$ and $\alpha$, respectively. Both algorithms used their respective value for accuracy parameter $\varepsilon$ for the same parameter in MULTIPASSLINEAR.

### D.3 Applications and Datasets

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

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

In general, this is a non-monotone, submodular function.

The revenue maximization objective is defined as follows. Let graph $G = (V, E)$ represent a social network, with nonnegative edge weight $w_{ij}$ on each edge $(i, j) \in E$. We use the concave graph model introduced by Hartline, Mirrokni, and Sundararajan [HMS08]. In this model, each user $i \in V$ is associated with a non-negative, concave function $f_i : \mathbb{R}^+ \rightarrow \mathbb{R}^+$. The value $v_i(S) = f_i(\sum_{j \in S} w_{ij})$ encodes how likely the user $i$ is to buy a product if the set $S$ has adopted it. Then the total revenue for seeding a set $S$ is

$$f(S) = \sum_{i \in V \setminus S} f_i \left( \sum_{j \in S} w_{ij} \right).$$
Figure 3: Oracle queries vs. $k$ for single-pass algorithms for the maxcut application on each dataset.

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

We evaluate the algorithms on synthetic random graphs as well as real social network datasets from the Stanford Network Analysis Project [LK20]. The specific datasets used were as follows:

- **er**, an Erdős-Rényi random graph with number of nodes $n = 5000$ and edge probability $p = 0.01$.

- **ba**, a random graph in the Barabási-Albert preferential attachment model with parameter $n = 5000$ and initially $m_0 = 3$ nodes, and 3 nodes added each iteration.

- **fb**, the ego-Facebook from Leskovec and Krevl [LK20] with $n = 4039$, $m = 88,234$.

- **slashdot**, the soc-Slashdot-0811 social network from Leskovec and Krevl [LK20] with $n = 77,360$, $m = 905,468$.

- **pokec**, the social network from Leskovec and Krevl [LK20] with $n = 1,632,803$, and $m = 30,622,564$.
Figure 4: Solution value vs. $k$ for single-pass algorithms for the revmax application on each dataset.

D.4 Additional Results

Figs. 2 and 3 show the solution value and number of oracle queries for the maxcut application; and Figs. 4 and 5 show the same for the revmax application.

Observe that while occasionally LS+ (gold star) obtains a lower solution value than the other algorithms, it more consistently returns high solution values ($\geq 90\%$ of the greedy algorithm) across the five datasets and two applications than the other algorithms. Moreover, it uses fewer queries, frequently by more than an order of magnitude over the next most efficient algorithm.
Figure 5: Oracle queries vs. $k$ for single-pass algorithms for the revmax application on each dataset.