Quick Streaming Algorithms for Maximization of Monotone Submodular Functions in Linear Time

Alan Kuhnle
Department of Computer Science, Florida State University
kuhnle@cs.fsu.edu

Abstract

We consider the problem of monotone, submodular maximization over a ground set of size n subject to cardinality constraint k. For this problem, we introduce the first deterministic algorithms with linear time complexity; these algorithms are streaming algorithms. Our single-pass algorithm obtains a constant ratio in $\lceil n/c \rceil + c$, for any $c \geq 1$. In addition, we propose a deterministic, multi-pass streaming algorithm with a constant number of passes that achieves nearly the optimal ratio with linear query and time complexities. We prove a lower bound that implies no constant-factor approximation exists using $o(n)$ queries, even if queries to infeasible sets are allowed. An empirical analysis demonstrates that our algorithms require fewer queries (often substantially less than n) yet still achieve better objective value than the current state-of-the-art algorithms, including single-pass, multi-pass, and non-streaming algorithms.

1 Introduction

k A nonnegative, set function $f : 2^{\mathcal{U}} \to \mathbb{R}^+$, where ground set $\mathcal{U}$ is of size n, is *submodular* if for all $S \subseteq T \subseteq \mathcal{U}$, $u \in \mathcal{U} \setminus T$, $f(T \cup \{u\}) - f(T) \leq f(S \cup \{u\}) - f(S)$ and *monotone* if $f(A) \leq f(B)$ if $A \subseteq B$. Intuitively, submodularity captures a natural diminishing returns property that arises in many machine learning applications, such as viral marketing (Kempe et al., 2003), network monitoring (Leskovec et al., 2007), sensor placement (Krause and Guestrin, 2007), video summarization (Mirzasoleiman et al., 2018), and MAP Inference for Determinantal Point Processes (Gillenwater et al., 2012).

A well-studied NP-hard optimization problem in this context is submodular maximization subject to a cardinality constraint (SMCC): $\arg\max_{|S| \leq k} f(S)$, where the cardinality constraint k is an input parameter and the function f is submodular and monotone. A simple greedy procedure (Nemhauser et al., 1978) achieves approximation ratio of $1 - 1/e \approx 0.632$ for SMCC in $O(kn)$ time; this ratio is optimal under the value query model (Nemhauser and Wolsey, 1978). In the value query model, the function f is provided to an algorithm as a value oracle, which when queried with set S returns $f(S)$ in a single operation that requires $O(1)$ time. In this work, the time complexity of an algorithm is measured in terms of the number of arithmetic operations and number of oracle queries.

For $k = \Omega(n)$, the standard greedy algorithm has $\Omega(n^2)$ time complexity, which is prohibitive on modern instance sizes. Further, loading the entire ground set into memory may be impossible. Therefore, much effort has gone into the design of algorithms with lower time complexity (Badanidiyuru and Vondrák, 2014; Mirzasoleiman et al., 2015; Buchbinder et al., 2015; Kuhnle, 2019; Crawford, 2020); and into streaming algorithms (Gomes and Krause, 2010; Badanidiyuru et al., 2014; Chakrabarti and Kale, 2015). In this context, a *streaming algorithm*[1] accesses elements by one or more sequential passes through the ground set and stores at most $O(k \log(n))$ elements in memory.

Several randomized approximation algorithms (Mirzasoleiman et al., 2015; Buchbinder et al., 2015; Fahrbach et al., 2019) have been designed that require $O(n)$ time, independent of k. However, the ratios of these algorithms hold only in expectation, which is undesirable for applications in which a good solution is required with high probability. Furthermore, these algorithms are

Proceedings of the 24th International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130.

[1] Formally, this is the semi-streaming model since k could be large relative to n. In this work, we will assume each element of the ground set requires $O(1)$ space.

Table 1: State-of-the-art algorithms for SMCC in terms of time complexity.

Reference	Passes	Ratio	Memory	Queries	Time
LTL (Mirzasoleiman et al., 2015)	k	$1-1/e-\varepsilon$	$O(n)$	$n\log(1/\varepsilon)$	$O(n)$
P-Pass (Norouzi-Fard et al., 2018)	$O(1/\varepsilon)$	$1-1/e-\varepsilon$	$O(k\log(k)/\varepsilon)$	$O(n\log(k)/\varepsilon^2)$	$O(n\log k)$
SieveStream++ (Kazemi et al., 2019)	1	$1/2-\varepsilon$	$O(k/\varepsilon)$	$O(n\log(k)/\varepsilon)$	$O(n\log k)$
C&K (Chakrabarti and Kale, 2015)	1	1/4	$O(k)$	$2n$	$O(n\log k)$
QuickStream$_c$, $c\geq 1$ (Theorem 1)	1	$1/(4c)-\varepsilon$	$O\left(ck\log(k)\log(1/\varepsilon)\right)$	$\lceil n/c\rceil+c$	$O(n)$
QS+BR (Theorem 2)	$O(1/\varepsilon)$	$1-1/e-\varepsilon$	$O(k\log(k))$	$O(n/\varepsilon)$	$O(n)$

not streaming algorithms and require the entire ground set to be loaded into memory. Indeed, every deterministic or streaming algorithm with constant ratio that has been described in the literature requires $\Omega(n\log k)$ time. This statement remains true if "deterministic" is replaced by "with high probability" (that is, probability that converges to 1 as $n\to\infty$). Moreover, every deterministic or streaming algorithm requires $\Omega(n\log k)$ queries to the value oracle, except for the single-pass streaming algorithm of Chakrabarti and Kale (2015), which obtains a ratio of $1/4$ with $2n$ oracle queries and $O(n\log k)$ arithmetic operations.

Contributions In this work, we propose the first deterministic, streaming algorithms for SMCC that have linear time complexity in the size n of the ground set. The first algorithm is a single-pass streaming algorithm that obtains a constant ratio, and the second is a multi-pass streaming algorithm that obtains nearly the optimal ratio. Specifically:

- We provide a linear-time, single-pass algorithm QuickStream (Section 2 and Appendix B), which achieves a constant ratio of while making at most $\lceil n/c\rceil+c$ queries to the value oracle for f, for any $c\geq 1$. This is the lowest query complexity[2] of any constant factor algorithm, which is important as the cost to evaluate the function f may be expensive. The following theorem summarizes the guarantees for QuickStream.

  Theorem 1. *Let $c\geq 1$ be an integer, and let $\varepsilon>0$. There exists a deterministic, single-pass streaming algorithm that makes at most $\lceil n/c\rceil+c$ queries, has memory complexity $O\left(ck\log(k)\log(1/\varepsilon)\right)$ has approximation ratio at least $1/(4c)-\varepsilon$ for* SMCC*, and the ratio converges to $(1-1/e)/(c+1)$ as $k\to\infty$. Further, the time complexity of the algorithm is $O(n)$.*

  We also show a lower bound of $\Omega(n/k)$ on the time complexity to obtain a constant ratio (Section 2.2).

- We propose a multi-pass algorithm QS+BR (Section 3), which achieves nearly the optimal ratio $1-1/e-\varepsilon$ in a constant number of passes and linear time complexity. In addition, this algorithm is the first deterministic algorithm for SMCC to obtain nearly the optimal ratio with a linear query complexity.

  Theorem 2. *There exists a deterministic, multi-pass streaming algorithm for* SMCC *that achieves approximation ratio $1-1/e-\varepsilon$, makes $O(n/\varepsilon)$ oracle queries, requires $O(1/\varepsilon)$ passes over the ground set, and requires $O(k\log k)$ memory. Further, the time complexity of the algorithm is $O(n)$.*

- An empirical evaluation (Section 4) of our single-pass algorithm QuickStream shows that if QuickStream is supplemented with a linear-time post-processing procedure (which does not compromise any of the theoretical guarantees of the algorithm), it empirically exceeds the objective value of the state-of-the-art single-pass streaming algorithm SieveStream++ (Kazemi et al., 2019) and the non-streaming LTL algorithm, while using fewer queries than either algorithm. Further, QS+BR obtains an even greater objective value while remaining query efficient.

Table 1 shows how our algorithms compare theoretically to the current state-of-the-art algorithms for SMCC.

1.1 Related Work

The literature studying SMCC is vast, so we only discuss algorithms for SMCC with monotone objective and cardinality constraint in this section. Streaming algorithms for more generalized constraints and submodular but not necessarily monotone functions include the works of Chekuri et al. (2015), Mirzasoleiman et al. (2016), Mirzasoleiman et al. (2018), and Feldman et al. (2018), among others.

[2]The query complexity of an algorithm is the total number of queries made to the value oracle for f and is upper-bounded by the time complexity.

Fast Approximation Algorithms The stochastic greedy algorithm LTL of Mirzasoleiman et al. (2015) obtains a ratio of $1-1/e-\varepsilon$ in $O(n)$ time, and thus has nearly optimal ratio and time complexity. However, its ratio holds only in expectation: LTL returns a poor solution with constant probability if $k = O(1)$. We refer the reader to Hassidim and Singer (2017) for discussion and further analysis of the ratio of LTL; also, in Section 4, we empirically explore the behavior of LTL for large values of ε. In addition to LTL, two other randomized approximation algorithms with linear query and time complexities have been developed. The algorithm of Buchbinder et al. (2015) achieves ratio $1/e-\varepsilon$ in $O(n\log(1/\varepsilon)/\varepsilon^2)$ time. Very recently, the randomized, parallelizable algorithm of Fahrbach et al. (2019) obtains ratio $1-1/e-\varepsilon$ in expectation with time complexity $O(n\log(1/\varepsilon)/\varepsilon^3)$. In contrast to our algorithms, none of these algorithms are streaming algorithms or are deterministic. For some applications of SMCC, an approximation ratio that holds only in expectation (rather than deterministically or with high probability) may be undesirable.

Single-Pass Streaming Algorithms Chakrabarti and Kale (2015) provided the first single-pass streaming algorithm for SMCC; they designed a (1/4)-approximation with one pass, $2n$ total queries, and $O(k)$ memory. However, this algorithm requires time complexity of $\Omega(n\log k)$. Badanidiyuru et al. (2014) improved the ratio for a single-pass algorithm to $1/2-\varepsilon$ in $O(k\log(k)/\varepsilon)$ memory, and $O(n\log(k)/\varepsilon)$ total queries and time. Kazemi et al. (2019) have provided the single pass $1/2-\varepsilon$ approximation SieveStream++, which improves the algorithm of Badanidiyuru et al. (2014) to have memory complexity of $O(k/\varepsilon)$ as indicated in Table 1. The current state-of-the-art, single-pass algorithm is SieveStream++, which is empirically compared to our algorithms in Section 4. Finally, Feldman et al. (2020) recently showed that any one-pass algorithm with approximation guarantee of $1/2+\varepsilon$ must essentially store all elements of the stream. In contrast to our single-pass algorithm, none of these algorithms have linear time complexity. Further, they require more oracle queries by at least a constant factor.

Multi-Pass Streaming Algorithms The first multi-pass streaming algorithm for SMCC has been given by Gomes and Krause (2010), which obtains value $\mathrm{OPT}/2-k\varepsilon$ using $O(k)$ memory and $O(B/\varepsilon)$ passes, where f is upper bounded by B. Norouzi-Fard et al. (2018) designed a multi-pass algorithm P-Pass that obtains ratio $1-1/e-\varepsilon$ in $O(1/\varepsilon)$ passes, $O(k\log(k)/\varepsilon)$ memory, $O(n\log(k)/\varepsilon^2)$ time. This is a generalization of the multi-pass algorithm of McGregor and Vu (2019) for the maximum coverage problem. The current state-of-the-art, multi-pass algorithm is P-Pass, which is empirically compared to our algorithms in Section 4. In contrast to our multi-pass algorithm, no multi-pass algorithm has linear time complexity; further, our algorithm makes fewer passes than P-Pass to achieve the same ratio of $1-1/e-\varepsilon$.

Algorithm 1 For each $c \geq 1$, a single-pass algorithm with approximation ratio $(1/(4c)-\varepsilon)$ if $k \geq 2$, query complexity $\lceil n/c \rceil + c$, and memory complexity $O\left(ck\log(k)\log(1/\varepsilon)\right)$.

procedure QuickStream$_c(f, k, \varepsilon)$
  Input: oracle f, cardinality constraint k, $\varepsilon > 0$
  $A \leftarrow \emptyset$, $A' \leftarrow \emptyset$, $C \leftarrow \emptyset$, $\ell \leftarrow \lceil \log_2(1/(4\varepsilon)) \rceil + 3$
  for element e received **do**
    $C \leftarrow C \cup \{e\}$
    if $|C| = c$ or stream has ended **then**
      if $f(A \cup C) - f(A) \geq f(A)/k$ **then**
        $A \leftarrow A \cup C$
      if $|A| > 2c\ell(k+1)\log_2(k)$ **then**
        $A \leftarrow \{c\ell(k+1)\log_2(k)$ elements most recently added to $A\}$
      $C \leftarrow \emptyset$
  $A' \leftarrow \{ck$ elements most recently added to $A\}$.
  Partition A' arbitrarily into at most c sets of size at most k. Return the set of the partition with highest f value.

2 The QuickStream$_c$ Algorithm

The algorithm QuickStream$_c$ is a single-pass, deterministic streaming algorithm. The parameter c is the number of elements buffered before the algorithm processes them together; this parameter determines the approximation ratio, query complexity, and memory complexity of the algorithm: respectively, $1/(4c)$, $\lceil n/c \rceil + c$, and $O\left(ck\log(k)\log(1/\varepsilon)\right)$. Notably, this algorithm is the first deterministic algorithm for SMCC to obtain linear time complexity. To handle the case that $k = 1$ and obtain better ratios if $k \geq 8c/e$, we provide two related algorithms in Appendix B.

The algorithm QuickStream$_c$ maintains a set A, initially empty. We refer to the sets of size at most c of elements processed together as *blocks* of size c. When a new block C is received, the algorithm makes one query of $f(A \cup C)$. If $f(A \cup C) - f(A) \geq f(A)/k$, the block C is added to A; otherwise, it is discarded. If the size $|A|$ exceeds $2\ell c(k+1)\log_2 k$, elements are deleted from A. At the end of the stream, the algorithm partitions the last ck elements added to A into c pieces of size at most k and return the one with highest f value. Pseudocode is given in Alg. 1.

At a high level, our algorithm resembles a swapping algorithm such as Chakrabarti and Kale (2015) or Buchbinder et al. (2014), which replaces previously added elements with better ones as they arrive. However, our algorithm uses simply the order in which elements were added to A to compare elements; which bypasses the need of a direct comparison of the value of an incoming element with the other elements of A. This indirect method of comparison allows us to obtain an algorithm with linear time complexity.

Below, we prove the following theorem.

Theorem 3. *Let $c \geq 1$, $\varepsilon \geq 0$, and let (f, k) be an instance of* SMCC *with $k \geq 2$. The solution S returned by* QuickStream$_c$ *satisifes $f(S) \geq (1/(4c) - \varepsilon)\,\mathrm{OPT}$, where* OPT *is the optimal solution value on this instance. Further,* QuickStream$_c$ *makes at most $\lceil n/c \rceil + c$ queries and has memory complexity $O\left(ck \log(k) \log(1/\varepsilon)\right)$.*

We remark that using the the value $f(A)$ of a potentially infeasible set A is an important feature of our algorithm; the use of infeasible sets is necessary to obtain a constant ratio with fewer than n queries to the oracle.

Proof of Theorem 3. The query complexity, time complexity, and memory complexity of QuickStream$_c$ are clear from the limit on the size of A, the choice of ℓ, and the fact that one query is required every c elements together with c queries at the termination of the stream. The rest of the proof establishes the approximation ratio of QuickStream$_c$.

First, we argue it is sufficient to prove the ratio in the case $c = 1$. Let $\mathcal{N} = \{C_1, \ldots, C_m\}$, where each C_i is the i-th block of at most c elements of $\mathcal{U}$ considered for addition to A on line 7. Define monotone, submodular function $g : 2^{\mathcal{N}} \to \mathbb{R}^+$ by $g(S) = f(\bigcup_{C \in S} C)$. Observe that if we omit lines 12 and 13, the behavior of QuickStream$_c$ on instance (f, k) is equivalent to QuickStream$_1$ run on instance (g, k) of SMCC; further, $\arg\max_{|S| \leq k} g(S) \geq \arg\max_{|S| \leq k} f(S)$. Let S be the solution returned by QuickStream$_1$ on instance (g, k). Then the value of A' at termination of QuickStream$_c$ is $A' = \bigcup_{C \in S} C$. Let $\{D_1, \ldots, D_c\}$ be the partition of A on line 13 of Alg. 1. Then by submodularity of f

$$g(S) = f(A') \leq \sum_{i=1}^{c} f(D_i) \leq c \max_{1 \leq i \leq c} f(D_i).$$

Since QuickStream$_c$ returns $\arg\max_{1 \leq i \leq c} f(D_i)$, it suffices to show that QuickStream$_1$ has approximation ratio $(1/4 - \varepsilon)$.

For the rest of the proof, we let $c = 1$. We require the following claim, which follows directly from the inequality $\log x \geq 1 - 1/x$ for $x > 0$.

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

Throughout the proof, let A_i denote the value of A at the beginning of the i-th iteration of **for** loop; let A_{n+1} be the value of A after the **for** loop completes. Also, let $A^* = \bigcup_{1 \leq i \leq n+1} A_i$, and let e_i denote the element received at the beginning of iteration i. We refer to line numbers of the pseudocode Alg. 1. First, we show the value of $f(A)$ does not decrease between iterations of the **for** loop, despite the possibility of deletions from A.

Lemma 1. *For any $1 \leq i \leq n$, it holds that $f(A_i) \leq f(A_{i+1})$.*

Proof. If no deletion is made during iteration i of the **for** loop, then any change in $f(A)$ is clearly nonnegative. So suppose deletion of set B from A occurs on line 10 of Alg. 1 during this iteration. Observe that $A_{i+1} = (A_i \setminus B) \cup \{e_i\}$, because the deletion is triggered by the addition of e_i to A_i. In addition, at some iteration $j < i$ of the **for** loop, it holds that $A_j = B$. From the beginning of iteration j to the beginning of iteration i, there have been $\ell(k+1)\log_2(k) - 1 \geq (\ell - 1)(k+1)\log_2(k)$ additions and no deletions to A, which add precisely the elements in $(A_i \setminus A_j)$.

It holds that

$$\begin{aligned} f\left(A_i \setminus A_j\right) &\overset{(a)}{\geq} f\left(A_i\right) - f\left(A_j\right) \\ &\overset{(b)}{\geq} \left(1 + \frac{1}{k}\right)^{(\ell-1)(k+1)\log k} \cdot f(A_j) - f(A_j) \\ &\overset{(c)}{\geq} (k^{\ell-1} - 1) f(A_j), \end{aligned}$$

where inequality (a) follows from submodularity and nonnegativity of f, inequality (b) follows from the fact that each addition from A_j to A_i increases the value of $f(A)$ by a factor of at least $(1 + 1/k)$, and inequality (c) follows from Claim 1. Therefore

$$\begin{aligned} f(A_i) &\leq f\left(A_i \setminus A_j\right) + f\left(A_j\right) \\ &\leq \left(1 + \frac{1}{k^{\ell-1} - 1}\right) f\left(A_i \setminus A_j\right). \end{aligned} \tag{1}$$

Next,

$$\begin{aligned} & f\left(\left(A_i \setminus A_j\right) \cup \{e_i\}\right) - f\left(A_i \setminus A_j\right) \\ &\overset{(d)}{\geq} f\left(A_i \cup \{e_i\}\right) - f\left(A_i\right) \\ &\overset{(e)}{\geq} f\left(A_i\right)/k \geq f\left(A_i \setminus A_j\right)/k, \end{aligned} \tag{2}$$

where inequality (d) follows from submodularity, and inequality (e) is by the condition to add e_i to A_i on line 7. Finally, using Inequalities (1) and (2) as indicated below, we have

$$\begin{aligned} f(A_{i+1}) &= f(A_i \setminus A_j \cup \{e_i\}) \\ &\overset{\text{By (2)}}{\geq} \left(1+\frac{1}{k}\right) f(A_i \setminus A_j) \\ &\overset{\text{By (1)}}{\geq} \frac{1+\frac{1}{k}}{1+\frac{1}{k^{\ell-1}-1}} \cdot f(A_i) \geq f(A_i), \end{aligned}$$

where the last inequality follows since $k \geq 2$ and $\ell \geq 3$. □

Next, we bound the total value of $f(A)$ lost from deletion throughout the run of the algorithm.

Lemma 2. $f(A^*) \leq \left(1+\frac{1}{k^\ell-1}\right) f(A_{n+1})$.

Proof. Observe that $A^* \setminus A_{n+1}$ may be written as the union of pairwise disjoint sets, each of which is size $\ell(k+1)\log_2(k)+1$ and was deleted on line 10 of Alg. 1. Suppose there were m sets deleted from A; write $A^* \setminus A_{n+1} = \{B^i : 1 \leq i \leq m\}$, where each B^i is deleted on line 10, ordered such that $i < j$ implies B^i was deleted after B^j (the reverse order in which they were deleted); finally, let $B^0 = A_{n+1}$.

Claim 2. Let $0 \leq i \leq m$. Then $f\left(B^i\right) \geq k^\ell f\left(B^{i+1}\right)$.

Proof. Let B^i, $B^{i+1} \in \mathcal{B}$. There are at least $\ell(k+1)\log k + 1$ elements added to A and exactly one deletion event during the period between starting when $A = B^{i+1}$ until $A = B^i$. Moreover, each addition except possibly one (corresponding to the deletion event) increases $f(A)$ by a factor of at least $1 + 1/k$. Hence, by Lemma 1 and Claim 1, $f\left(B^i\right) \geq k^\ell f\left(B^{i+1}\right)$. □

By Claim 2, for any $0 \leq i \leq m$, $f(A_{n+1}) \geq k^{\ell i} f\left(B^i\right)$. Thus,

$$\begin{aligned} f(A^*) &\leq f(A^* \setminus A_{n+1}) + f(A_{n+1}) \\ &\overset{(a)}{\leq} \sum_{i=0}^{m} f\left(B^i\right) \\ &\overset{(b)}{\leq} f(A_{n+1}) \sum_{i=0}^{\infty} k^{-\ell i} \\ &\overset{(c)}{=} f(A_{n+1}) \left(\frac{1}{1-k^{-\ell}}\right), \end{aligned}$$

where inequality (a) follows from submodularity and nonnegativity of f, inequality (b) follows Claim 2, and inequality (c) follows from the sum of a geometric series. □

Next, we bound the value of OPT in terms of $f(A_{n+1})$.

Lemma 3. $\left(2+\frac{1}{k^\ell-1}\right) f(A_{n+1}) \geq \text{OPT}$.

Proof. Let $O \subseteq \mathcal{U}$ be an optimal solution of size k to SMCC; for each $o \in O$, let $i(o)$ be the iteration in which o was processed. Then

$$\begin{aligned} f(O) - f(A^*) &\leq f(O \cup A^*) - f(A^*) \\ &\leq \sum_{o \in O \setminus A^*} f(A^* + o) - f(A^*) \\ &\leq \sum_{o \in O \setminus A^*} f\left(A_{i(o)}\right)/k \\ &\leq \sum_{o \in O \setminus A^*} f(A_{n+1})/k \leq f(A_{n+1}), \end{aligned}$$

by monotonicity and submodularity of f, the condition of Line 7, Lemma 1, and the size of O. From here, the result follows from Lemma 2. □

Recall that QuickStream$_1$ returns the set A', the last k elements added to A. Lemma 4 shows that $2f(A') \geq f(A_{n+1})$.

Lemma 4. $f(A_{n+1}) \leq 2f(A')$.

Proof. If $|A_{n+1}| \leq k$, $f(A') \geq f(A_{n+1})$ by monotonicity, and the lemma holds. Therefore, suppose $|A_{n+1}| > k$. Let $A' = \{a'_1, \ldots, a'_k\}$, in the order these elements were added to A. Let $A'_i = \{a'_1, \ldots, a'_i\}$, $A'_0 = \emptyset$. Then

$$\begin{aligned} f(A') &\geq f(A_{n+1}) - f(A_{n+1} \setminus A') \\ &= \sum_{i=1}^{k} \left[f\left((A_{n+1} \setminus A') \cup A'_{i-1} + a'_i\right)\right. \\ &\qquad \left. - f\left((A_{n+1} \setminus A') \cup A'_{i-1}\right)\right] \\ &\overset{(a)}{\geq} \sum_{i=1}^{k} \frac{f\left((A_{n+1} \setminus A') \cup A'_{i-1}\right)}{k} \\ &\overset{(b)}{\geq} \sum_{i=1}^{k} \frac{f(A_{n+1} \setminus A')}{k} = f(A_{n+1} \setminus A'), \end{aligned}$$

where inequality (a) is by the condition on Line 7, and inequality (b) is from monotonicity of f. Thus $f(A_{n+1}) \leq f(A_{n+1} \setminus A') + f(A') \leq 2f(A')$. □

Since $k \geq 2$, Lemmas 3 and 4 show that the set A' of QuickStream$_1$ satisifes $f(A') \geq \left(\frac{1}{4+2/(k^\ell-1)}\right) \text{OPT}$. By the choice of ℓ, $f(A') \geq (1/4 - \varepsilon)\,\text{OPT}$. □

2.1 Post-Processing: QuickStream$_c$++

In this section, we describe a simple post-processing procedure to improve the objective value obtained by QuickStream$_c$. At the termination of the stream, QuickStream$_c$ stores a set A of size $O(k \log k)$ from which the set A' and solution are extracted, on which the worst-case approximation ratio is proven in the previous section. However, the set A may be regarded as a filtered ground set of size $O(k \log k) \leq n$, upon which any algorithm may be run to extract a solution. As long as the post-processing algorithm has query and time complexity and runtime $O(n)$, Theorem 3 still holds for the resulting single-pass streaming algorithm with post-processing. This modification of QuickStream is termed QuickStream++.

We remark that the condition of Line 7 of QuickStream$_c$ may be changed to the following condition: $f(A \cup C) - f(A) \geq \delta f(A)/k$, for input parameter $\delta > 0$. In this case, it is not difficult to extend the analysis in the previous section to show that the algorithm achieves ratio $[c(1+\delta)(1+1/\delta)]^{-1}$, in memory $O(k \log k)$ and the same query complexity and runtime. This ratio is optimized for $\delta = 1$, but when using post-processing with QuickStream$_c$++, smaller values of δ result in larger sets A, although still bounded in $O(k \log k) \leq n$. We found in our empirical evaluation in Section 4 that setting $\delta = c/10$ for QuickStream$_c$++ yields good empirical results.

2.2 Lower Bound on Query and Time Complexity

While it is clear that at least n queries are required for any constant factor if the algorithm is only allowed to query feasible sets (consider $k = 1$), our algorithms bypass this restriction. Our next result is a lower bound on the number of queries (and hence also the time complexity) required to obtain a constant-factor approximation.

Theorem 4. *Let $c \geq 2$ be an integer, and let $\varepsilon > 0$. Any (randomized) approximation algorithm for* SMCC *with ratio $1/c + \varepsilon$ for* SMCC *with probability δ requires at least $\lceil \delta n/(ck-1) \rceil$ oracle queries and hence $\Omega(n/k)$ time.*

Theorem 4 implies no constant-factor approximation exists with $o(n)$ time in the value query model. Another consequence of Theorem 4 is that any algorithm with ratio $(1/2 + \varepsilon)$ with probability greater than $1 - 1/n$ requires at least n queries.

Proof. We prove the theorem for instances of SMCC with cardinaity constraint $k \geq 1$. Let $c \in \mathbb{N}$, $c \geq 2$, and let $0 < \varepsilon < 1$. Let $n \in \mathbb{N}$, and let $\mathcal{U}_n = \{0, 1, \ldots, n-1\}$.

Algorithm 2 A procedure to boost to from constant ratio α to ratio $1 - e^{-1+\varepsilon}$ in $O(1/\varepsilon)$ passes, 1 query per element per pass, and $O(k)$ memory.

```
1: procedure BoostRatio(f, k, α, Γ, ε)
2:     Input: evaluation oracle f : 2^N → R+, constraint k, constant α, value Γ such that Γ ≤ OPT ≤ Γ/α, and 0 < ε < 1.
3:     τ ← Γ/(αk), A ← ∅.
4:     while τ ≥ (1 − ε)Γ/(4k) do
5:         τ ← τ(1 − ε)
6:         for n ∈ N do
7:             if f(A + n) − f(A) ≥ τ then
8:                 A ← A + n
9:             if |A| = k then
10:                return A
11:    return A
```

Define $f : 2^{\mathcal{U}_n} \to \mathbb{R}^+$ by $f(A) = \min\{|A|, ck\}$, for $A \subseteq \mathcal{U}_n$. Next, we define a function g that is hard to distinguish from f: pick $a \in \mathcal{U}_n$ uniformly randomly. Let $g(A) = f(A)$ if $a \notin A$, and $g(A) = ck$ otherwise. Clearly, both f and g are monotone and submodular.

Now, consider queries to f and g of a set $A \subseteq \mathcal{U}_n$. These queries can only distinguish between f and g if $|A| \leq ck - 1$ and $a \in A$; in any other case, the values of $f(A)$ and $g(A)$ are equal. Consider a (possibly adaptive) sequence of queries of sets $A_1, A_2, \ldots, A_m$. Without loss of generality, we may assume $|A_i| \leq ck - 1$ for each i, since the query of any set of larger size yields no information about the element a. Then the algorithm can correctly distinguish f from g iff $a \in \bigcup A_i$, which happens with probability at most $m(ck-1)/n$, since $|\bigcup A_i| \leq m(ck-1)$. Therefore, to distinguish between f and g with probability at least δ requires at least $\lceil \delta n/(ck-1) \rceil$ queries.

Since any approximation algorithm with ratio at least $1/c + \varepsilon$ with probability δ would distinguish between f, g with probability δ, since the optimal solution with f has value k, while $g(a) = ck$, the theorem is proven. □

3 Multi-Pass Streaming Algorithm to Boost Constant Ratio to $1 - 1/e - \varepsilon$

In this section, we describe BoostRatio (Alg. 2), which given any α-approximation $\mathcal{A}$ for SMCC can boost the ratio to $1 - e^{-1+\varepsilon} \geq 1 - 1/e - \varepsilon$ using the output of $\mathcal{A}$. Theorem 5 is proven below.

Theorem 5. *Let $0 < \varepsilon < 1$. Suppose a deterministic α-approximation $\mathcal{A}$ exists for* SMCC*. Then algorithm* BoostRatio *is a multi-pass streaming algorithm that when applied to the solution of $\mathcal{A}$ yields a solution within factor $1 - e^{-1+\varepsilon} \geq 1 - 1/e - \varepsilon$ of optimal in at most $n(\log(4/\alpha)/\varepsilon + 1$ queries, $\log(4/\alpha)/\varepsilon + 1$ passes,*

and $O(k)$ memory.

If the algorithm $\mathcal{A}$ is the algorithm provided by Theorem 1, this establishes Theorem 2.

As input, the algorithm BoostRatio takes an instance (f, k) of SMCC, an approximate solution value Γ, and accuracy parameter $\varepsilon > 0$. On the instance (f, k), it must hold that $\Gamma \leq \mathrm{OPT} \leq \Gamma/\alpha$, where OPT is the value of an optimal solution. The algorithm works by making one pass (line 6) through the ground set for each threshold value τ, during which any element with marginal gain at least τ to A is added to A (lines 7 – 8). The maximum and minimum values of τ are determined by Γ, α, and k: initally $\tau = \Gamma/(\alpha k)$, and the algorithm terminates if $\tau < (1-\varepsilon)\Gamma/(4k)$; each iteration of the **while** loop, τ is decreased by a factor of $(1-\varepsilon)$. The set A is initially empty; if $|A| = k$, the algorithm terminates and returns A; otherwise, at most $O(\log(1/\alpha)/\varepsilon)$ passes are made until the minimum threshold value is reached.

Intuitively, the $1-1/e-\varepsilon$ ratio is achieved since the α-approximate solution Γ allows the algorithm to approximate the value for τ of OPT/k in a constant number of guesses. Once this threshold has been reached, only $\log(1/4)/\varepsilon$ more values of τ are needed to achieve the desired ratio. While BoostRatio may be used with any α-approximation, if it is used with QuickStream$_1$, the resulting algorithm is the first linear-time, deterministic, $(1-1/e-\varepsilon)$-approximation for SMCC, which is a multi-pass streaming algorithm.

Proof of Theorem 5. Suppose $0 < \varepsilon < 1$. Let (f, k) be an instance of SMCC. The algorithm is to first run $\mathcal{A}$, to obtain set A'. Next, BoostRatio is called with parameters $(f, k, \alpha, f(A'), \varepsilon)$. Observe that the inital value of the threshold τ in the **while** loop is at least $(1-\varepsilon)\mathrm{OPT}/k$, and the final value of τ is at most $\mathrm{OPT}/(4k)$.

Consider the case that at termination $|A| < k$. Then by the last iteration of the **while** loop, submodularity and monotonicity of f,

$$\begin{aligned} f(O) - f(A) &\leq f(O \cup A) - f(A) \\ &\leq \sum_{o \in O \setminus A} f(A \cup \{o\}) - f(A) \\ &\leq \sum_{o \in O \setminus A} \Gamma/(4k) \leq \mathrm{OPT}/4, \end{aligned}$$

from which $f(A) \geq 3\mathrm{OPT}/4 \geq (1 - e^{-1+\varepsilon})\mathrm{OPT}$.

Next, consider the case that at termination $|A| = k$. Let $A_i = \{a_1, a_2, \ldots, a_i\}$, ordered by the addition of elements to A, and let $A_0 = \emptyset$.

Claim 3. Let $i \in \{0, \ldots, k-1\}$. Then

$$f(A_{i+1}) - f(A_i) \geq \frac{(1-\varepsilon)}{k}(\mathrm{OPT} - f(A_i))$$

Proof. Let $i \in \{0, \ldots, k-1\}$. First, suppose a_{i+1} is added to A_i during an iteration with $\tau \geq (1-\varepsilon)\mathrm{OPT}/k$. In this case, $f(A_{i+1}) - f(A_i) \geq \tau \geq (1-\varepsilon)\mathrm{OPT}/k \geq \frac{(1-\varepsilon)}{k}(\mathrm{OPT} - f(A_i))$.

Next, suppose a_{i+1} is added to A_i during an iteration with $\tau < (1-\varepsilon)\mathrm{OPT}/k$. Consider the set $O \setminus A_i$; in the previous iteration of the **while** loop, no element of $O \setminus A_i$ is added to A; hence, by submodularity, for all $o \in O \setminus A_i$, $f(A_i + o) - f(A_i) < \tau/(1-\varepsilon)$. Therefore,

$$\begin{aligned} f(A_{i+1}) - f(A_i) &\geq \tau \\ &\geq \frac{(1-\varepsilon)}{k} \sum_{o \in O \setminus A_i} f(A_i \cup \{o\}) - f(A_i) \\ &\geq \frac{(1-\varepsilon)}{k}(f(O \cup A_i) - f(A_i)) \\ &\geq \frac{(1-\varepsilon)}{k}(\mathrm{OPT} - f(A_i)). \end{aligned}$$

□

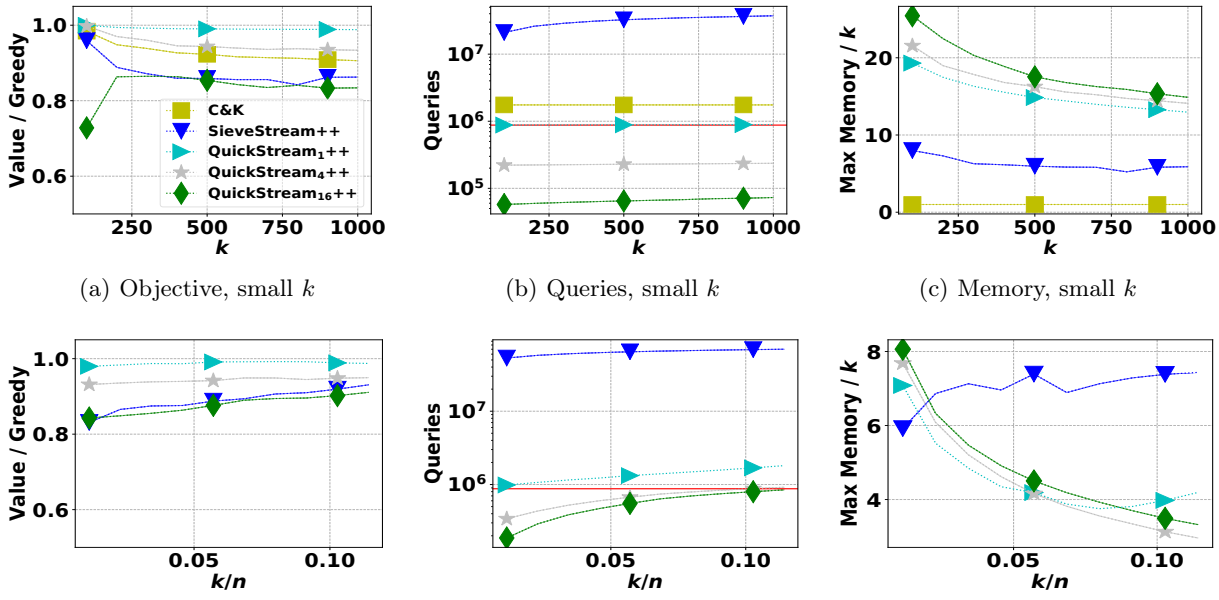
From Claim 3, standard arguments show the $f(A_k) \geq \mathrm{OPT}\left(1 - e^{-1+\varepsilon}\right) \geq \mathrm{OPT}(1 - 1/e - \varepsilon)$.

For the query complexity, observe that the **for** loop of BoostRatio makes at most n queries, and the **while** loop requires $\log(\alpha/4)/\log(1-\varepsilon) + 1 \leq \log(4/\alpha)/\varepsilon + 1$ iterations. □

4 Empirical Evaluation

In this section, we demonstrate that the objective value achieved empirically by our algorithm QuickStream$_c$++ beats that of the state-of-the-art algorithms LTL, SieveStream++, and C&K, while using the fewest queries and only a single pass. Our multi-pass algorithm QS+BR (QuickStream$_1$ followed by BoostRatio) achieved mean objective value better than 0.99 of the standard Greedy value across all instances tested.

Algorithms Our algorithms are compared to the following methods: Greedy, the standard greedy algorithm analyzed by Nemhauser et al. (1978), LTL (Mirzasoleiman et al., 2015), SieveStream++ (Kazemi et al., 2019), P-Pass (Norouzi-Fard et al., 2018), and C&K (Chakrabarti and Kale, 2015), as described in Section 1. Randomized algorithms were averaged over 10 independent runs and the shaded regions in plots correspond to one standard deviation. Any algorithm with an accuracy parameter ε is run with $\varepsilon = 0.1$ unless otherwise specified.



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

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

Figure 1: Evaluation of single-pass streaming algorithms on web-Google ($n = 875713$), in terms of objective value normalized by the standard greedy value, total number of queries, and the maximum memory used by each algorithm normalized by k. The legend shown in **(a)** applies to all subfigures.

We evaluate our algorithm QuickStream$_c$++ for various values of c. The post-processing procedure run on A is taken to be our linear time BoostRatio and we set parameter $\delta = c/10$ (see Section 2.1 for the definition of δ). We also evaluate our multi-pass algorithm QS+BR.

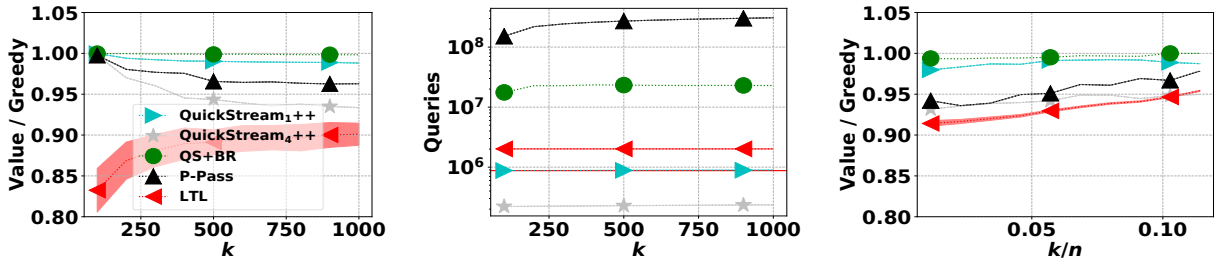
Applications We evaluate all of the algorithms on two applications of SMCC: the first is maximum coverage on a graph: for each set of vertices S, the value of $f(S)$ is the number of vertices adjacent to the set S. The second application is the revenue maximization problem on a social network (Hartline et al., 2008), a variant of influence maximization. For detailed specification of these applications, see Appendix C. We evaluate on a variety of network technologies from the Stanford Large Network Dataset Collection (Leskovec and Krevl, 2020), including ego-Facebook ($n = 4039$) and web-Google ($n = 875713$), among others listed in Appendix C. Values of k evaluated include small values ($k \leq 1000$) and large values ($k = \Omega(n)$).

Results: Single-Pass Algorithms In Fig. 1, representative results are shown for the single-pass algorithms. Results were qualitatively similar across applications and datasets; additional results are shown in Appendix C.

Objective Value For small k ($k \leq 1000$), the mean objective value (normalized by the standard Greedy value) obtained by each single-pass algorithm across all instances is as follows: QuickStream$_1$++ 0.99; QuickStream$_4$++ 0.95; C&K 0.93; SieveStream++ 0.87; QuickStream$_{16}$++ 0.84. On the instances with large k ($k \leq 0.1n$), the means are: QuickStream$_1$++ 0.99; QuickStream$_4$++ 0.94; SieveStream++ 0.89; QuickStream$_{16}$++ 0.88.

Queries In terms of queries, QuickStream$_c$++ required roughly n/c queries for small k; the the next smallest was C&K, which required $2n$ queries, followed by SieveStream++, which started at more than $10n$ queries and increased logarithmically with k. For large k, the queries of QuickStream$_c$++ increased due to the $O(n)$ post-processing step which depends on k, but always remained less than $2n$.

The algorithm C&K, while very efficient in terms of queries, was unable to run in a reasonable timeframe on our larger instances. Most of the algorithms we evaluate (including both of our algorithms) use a marginal gain query of sets that only increase in size, which yields an optimized implementation for the maximum cover application. However, C&K cannot be implemented with this optimization and requires the full $O(n)$ oracle query; thus, on some instances we were able to run the standard greedy algorithm but not C&K. This illustrates the fact that the oracle query complexity only constitutes partial information about the runtime of the algorithm.



(a) (b) (c)

Figure 2: Evaluation of our algorithms compared with the multi-pass P-Pass and non-streaming algorithm LTL. We compare the objective value (normalized by the standard Greedy objective value) and total queries on web-Google for the maximum cover application for both small and large k values. The large k values are given as a fraction of the number of nodes in the network. The legend shown in **(a)** applies to all subfigures.

Memory As shown in Figs. 1(c) and 1(f), the memory usage of the algorithms remained at most a constant times k; for QuickStream$_c$++, this constant decreased as k increased, and with large enough k, the algorithms used less memory than SieveStream++. In terms of memory, C&K is optimal both theoretically and in practice, as it stores only k elements.

Results: Multi-Pass and Non-Streaming Algorithms In Fig. 2, we show results of our algorithms QuickStream$_c$++ and QS+BR, in comparison with the multi-pass P-Pass algorithm and the non-streaming LTL algorithm on web-Google. Surprisingly, our single-pass algorithm QuickStream$_1$++ beats the objective values of both P-Pass and LTL, as it obtained 0.99 of the standard greedy value on average across all instances (both small and large k). The only algorithm with better objective value than QuickStream$_1$++ is our multipass QS+BR. The algorithm QuickStream$_4$++ exceeded the objective value of LTL despite using 1/8 of the queries.

5 Conclusions

In this work, we have provided the first constant-factor algorithms for SMCC that make a linear number of oracle queries and arithmetic operations. Supplemented with post-processing heuristics, our single-pass algorithm QuickStream achieves state-of-the-art empirical objective value while using fewer than n queries of the objective function. Our multi-pass algorithm QS+BR nearly achieves the optimal worst-case ratio of $1 - 1/e$ and is the first deterministic algorithm to do so with linear query complexity.

6 Acknowledgments

The work of A. Kuhnle was partially supported by Florida State University. We thank Victoria G. Crawford and the anonymous reviewers for helpful feedback on earlier versions of the manuscript.

References

Ashwinkumar Badanidiyuru and Jan Vondrák. Fast algorithms for maximizing submodular functions. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2014.

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)*, pages 671–680, 2014.

Niv Buchbinder, Moran Feldman, and Roy Schwartz. Online Submodular Maximization with Preemption. In *ACM-SIAM Symposium on Discrete Algorithms*, 2014.

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.

Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Mathematical Programming*, 154(1-2):225–247, 2015.

T. H.Hubert Chan, Zhiyi Huang, Shaofeng H.C. Jiang, Ning Kang, and Zhihao Gavin Tang. Online Submodular Maximization with Free Disposal: Randomization Beats 1/4 for Partition Matroids. *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1204–1223, 2017.

Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming Algorithms for Submodular Function Maximization. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2015.

Victoria G. Crawford. Faster Guarantees of Pareto

Optimization for Submodular Maximization. In *arxiv preprint arXiv:1908:01230*, 2020.

Matthew Fahrbach, Vahab Mirrokni, and Morteza Zadimoghaddam. Submodular Maximization with Nearly Optimal Approximation, Adaptivity, and Query Complexity. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 255–273, 2019.

Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, Get More: Streaming Submodular Maximization with Subsampling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The One-way Communication Complexity of Submodular Maximization with Applications to Streaming and Robustness. In *arXiv preprint arXiv:2003.13459*, 2020.

Jennifer Gillenwater, Alex Kulesza, and Ben Taskar. Near-Optimal MAP Inference for Determinantal Point Processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

Ryan Gomes and Andreas Krause. Budgeted Nonparametric Learning from Data Streams. In *International Conference on Machine Learning (ICML)*, 2010.

Jason Hartline, Vahab S. Mirrokni, and Mukund Sundararajan. Optimal marketing strategies over social networks. *International Conference on World Wide Web (WWW)*, pages 189–198, 2008.

Avinatan Hassidim and Yaron Singer. Robust Guarantees of Stochastic Greedy Algorithms. *International Conference on Machine Learning (ICML)*, 2017.

Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular Streaming in All its Glory: Tight Approximation, Minimum Memory and Low Adaptive Complexity. In *International Conference on Machine Learning (ICML)*, 2019.

David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2003.

Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. *AAAI Conference on Artificial Intelligence*, 2007.

Alan Kuhnle. Interlaced Greedy Algorithm for Maximization of Submodular Functions in Nearly Linear Time. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Jure Leskovec and Andrej Krevl. {SNAP Datasets}: {Stanford} Large Network Dataset Collection. \url{http://snap.stanford.edu/data}, jun 2020.

Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective Outbreak Detection in Networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2007.

Andrew McGregor and Hoa T. Vu. Better Streaming Algorithms for the Maximum Coverage Problem. *Theory of Computing Systems*, 63(7):1595–1619, 2019.

Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrak, and Andreas Krause. Lazier Than Lazy Greedy. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, and Amin Karbasi. Fast Constrained Submodular Maximization : Personalized Data Summarization. In *International Conference on Machine Learning (ICML)*, 2016.

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.

G L Nemhauser and L A Wolsey. Best Algorithms for Approximating the Maximum of a Submodular Set Function. *Mathematics of Operations Research*, 3(3): 177–188, 1978.

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

Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-Approximation for Submodular Maximization on Massive Data Streams. In *International Conference on Machine Learning (ICML)*, volume 9, 2018.