Submodular Streaming in All Its Glory: Tight Approximation, Minimum Memory and Low Adaptive Complexity

Ehsan Kazemi [1] **Marko Mitrovic** [1] **Morteza Zadimoghaddam** [2] **Silvio Lattanzi** [2] **Amin Karbasi** [1]

Abstract

Streaming algorithms are generally judged by the quality of their solution, memory footprint, and computational complexity. In this paper, we study the problem of maximizing a monotone submodular function in the streaming setting with a cardinality constraint k. We first propose SIEVE-STREAMING++, which requires just one pass over the data, keeps only $O(k)$ elements and achieves the tight $1/2$-approximation guarantee. The best previously known streaming algorithms either achieve a suboptimal $1/4$-approximation with $\Theta(k)$ memory or the optimal $1/2$-approximation with $O(k \log k)$ memory. Next, we show that by buffering a small fraction of the stream and applying a careful filtering procedure, one can heavily reduce the number of adaptive computational rounds, thus substantially lowering the computational complexity of SIEVE-STREAMING++. We then generalize our results to the more challenging multi-source streaming setting. We show how one can achieve the tight $1/2$-approximation guarantee with $O(k)$ shared memory while minimizing not only the required rounds of computations but also the total number of communicated bits. Finally, we demonstrate the efficiency of our algorithms on real-world data summarization tasks for multi-source streams of tweets and of YouTube videos.

1. Introduction

Many important problems in machine learning, including data summarization, network inference, active set selection, facility location, and sparse regression can be cast as instances of constrained submodular maximization (Krause & Golovin, 2012). Submodularity captures an intuitive diminishing returns property where the gain of adding an element to a set decreases as the set gets larger. More formally, a non-negative set function $f : 2^V \to \mathbb{R}_{\geq 0}$ is **submodular** if for all sets $A \subseteq B \subset V$ and every element $e \in V \setminus B$, we have $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$. The submodular function f is **monotone** if for all $A \subseteq B$ we have $f(A) \leq f(B)$.

In this paper, we consider the following canonical optimization problem: given a non-negative monotone submodular function f, find the set S^* of size at most k that maximizes the function f:

$$S^* = \underset{S \subseteq V, |S| \leq k}{\arg\max} \ f(S). \tag{1}$$

We define $\texttt{OPT} = f(S^*)$. When the data is relatively small and it does not change over time, the greedy algorithm and other fast centralized algorithms provide near-optimal solutions. Indeed, it is well known that for problem (1) the greedy algorithm (which iteratively adds elements with the largest marginal gain) achieves a $1 - 1/e$ approximation guarantee (Nemhauser et al., 1978).

In many real-world applications, we are dealing with massive streams of images, videos, texts, sensor logs, tweets, and high-dimensional genomics data which are produced from different data sources. These data streams have an unprecedented volume and are produced so rapidly that they cannot be stored in memory, which means we cannot apply classical submodular maximization algorithms. In this paper, our goal is to design efficient algorithms for streaming submodular maximization in order to simultaneously provide the best approximation factor, memory complexity, running time, and communication cost.

For problem (1), Norouzi-Fard et al. (2018) proved that any streaming algorithm[1] with a memory $o(n/k)$ cannot provide an approximation guarantee better than $1/2$. SIEVE-STREAMING is the first streaming algorithm with a constant approximation factor (Badanidiyuru et al., 2014). This algorithm guarantees an approximation factor of $1/2-\varepsilon$ and memory complexity of $O(k \log(k)/\varepsilon)$. While the approximation guarantee of their SIEVE-STREAMING is optimal, the memory complexity is a factor of $\log(k)$ away from the desired lower bound $\Theta(k)$. In contrast, Buchbinder et al. (2015)

[1]Yale University [2]Google Research. Correspondence to: Ehsan Kazemi <ehsan.kazemi@yale.edu>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

[1]They assume the submodular function is evaluated only on feasible sets of cardinality at most k. In this paper, we make the same natural assumption regarding the feasible queries.

designed a streaming algorithm with a $1/4$-approximation factor and optimal memory $\Theta(k)$. The first contribution of this paper is to answer the following question: Is there a streaming algorithm with an approximation factor arbitrarily close to $1/2$ whose memory complexity is $O(k)$?

Our new algorithm, SIEVE-STREAMING++, closes the gap between the optimal approximation factor and memory complexity, but it still has some drawbacks. In fact, in many applications of submodular maximization, the function evaluations (or equivalently Oracle queries)[2] are computationally expensive and can take a long time to process.

In this context, the fundamental concept of adaptivity quantifies the number of sequential rounds required to maximize a submodular function, where in each round, we can make polynomially many independent Oracle queries in parallel. More formally, given an Oracle f, an algorithm is ℓ-adaptive if every query $\mathcal{Q}$ to the Oracle f at a round $1 \leq i \leq \ell$ is independent of the answers $f(\mathcal{Q}')$ to all other queries $\mathcal{Q}'$ at rounds j, $i \leq j \leq \ell$ (Balkanski & Singer, 2018). The adaptivity of an algorithm has important practical consequences as low adaptive complexity results in substantial speedups in parallel computing time.

All the existing streaming algorithms require at least one Oracle query for each incoming element. This results in an adaptive complexity of $\Omega(n)$ where n is the total number of elements in the stream. Furthermore, in many real-world applications, data streams arrive at such a fast pace that it is not possible to perform multiple Oracle queries in real time. This could result in missing potentially important elements or causing a huge delay.

Our idea to tackle the problem of adaptivity is to introduce a hybrid model where we allow a machine to buffer a certain amount of data, which allows us to perform many queries in parallel. We design a sampling algorithm that, in only a few adaptive rounds, picks items with good marginal gain and discards the rest. The main benefit of this method is that we can quickly empty the buffer and continue the optimization process. In this way, we obtain an algorithm with optimal approximation, query footprint, and near-optimal adaptivity.

Next, we focus on an additional challenge posed by real-world data where often multiple streams co-exist at the same time. In fact, while submodular maximization over only one stream of data is challenging, in practice there are many massive data streams generated simultaneously from a variety of sources. For example, these multi-source streams are generated by tweets from news agencies, videos and images from sporting events, or automated security systems and sensor logs. These data streams have an enormous volume and are produced so rapidly that they cannot be even transferred to a central machine. Therefore, in the multi-source streaming setting, other than approximation factor, memory and adaptivity, it is essential to keep communication cost low. To solve this problem, we show that a carefully-designed generalization of our proposed algorithm for single-source streams also has an optimal communication cost.

[2]The Oracle for a submodular function f receives a set S and returns its value $f(S)$.

2. Related Work

Badanidiyuru et al. (2014) were the first to consider a one-pass streaming algorithm for maximizing a monotone submodular function under a cardinality constraint. Buchbinder et al. (2015) improved the memory complexity of (Badanidiyuru et al., 2014) to $\Theta(k)$ by designing a $1/4$ approximation algorithm. Norouzi-Fard et al. (2018) introduced an algorithm for random order streams that beats the $1/2$ bound. Recently, Agrawal et al. (2019) substantially improved the result for random order streams to an almost tight $1 - 1/e - \varepsilon - O(k^{-1})$ approximation factor. Norouzi-Fard et al. (2018) also studied the multi-pass streaming submodular maximization problem.

Chakrabarti & Kale (2015) studied streaming submodular maximization problem subject to the intersection of p matroid constraints. These results were further extended to more general constraints such as p-matchoids (Chekuri et al., 2015; Feldman et al., 2018). Also, there have been some very recent works to generalize these results to non-monotone submodular functions (Chakrabarti & Kale, 2015; Chekuri et al., 2015; Chan et al., 2017; Mirzasoleiman et al., 2018; Feldman et al., 2018). Elenberg et al. (2017) provide a streaming algorithm with a constant factor approximation for a generalized notion of submodular objective functions, called weak submodularity. In addition, a few other works study the streaming submodular maximization over sliding windows (Chen et al., 2016; Epasto et al., 2017).

To scale to very large datasets, several solutions to the problem of submodular maximization have been proposed in recent years (Mirzasoleiman et al., 2015; 2016a; Feldman et al., 2017; Badanidiyuru & Vondrák, 2014; Mitrovic et al., 2017a). Mirzasoleiman et al. (2015) proposed the first linear-time algorithm for maximizing a monotone submodular function subject to a cardinality constraint that achieves a $(1 - 1/e - \varepsilon)$-approximation. Buchbinder et al. (2017) extended these results to non-monotone submodular functions.

Another line of work investigates the problem of scalable submodular maximization in the MapReduce setting where the data is split amongst several machines (Kumar et al., 2015; Mirzasoleiman et al., 2016b; Barbosa et al., 2015; Mirrokni & Zadimoghaddam, 2015; Mirzasoleiman et al., 2016c; Barbosa et al., 2016; Liu & Vondrák, 2018). Each machine runs a centralized algorithm on its data and passes the result to a central machine. Then, the central machine outputs the final answer. Since each machine runs a variant of the greedy algorithm, the adaptivity of all these approaches is linear in k, i.e., it is $\Omega(n)$ in the worst-case.

Practical concerns of scalability have motivated studying the adaptivity of submodular maximization algorithms. Balkanski & Singer (2018) showed that no algorithm can obtain a constant factor approximation in $o(\log n)$ adaptive rounds for monotone submodular maximization subject to a cardinality constraint. They introduced the first constant factor approximation algorithm for submodular maximization with logarithmic adaptive rounds. Their algorithm, in $O(\log n)$ adaptive rounds, finds a solution with an approximation arbitrarily close to $1/3$. These bounds were recently improved by $(1 - 1/e - \varepsilon)$-approximation algorithm with $O(\log(n)/\mathrm{poly}(\varepsilon))$ adaptivity (Fahrbach et al., 2019; Balkanski et al., 2019a; Ene & Nguyen, 2019). More recently and independently, Balkanski et al. (2019b) and Chekuri & Quanrud (2019) studied the additivity of submodular maximization under a matroid constraint. In addition, Balkanski et al. (2018) proposed an algorithm for maximizing a non-monotone submodular function with cardinality constraint k whose approximation factor is arbitrarily close to $1/(2e)$ in $O(\log^2 n)$ adaptive rounds. Fahrbach et al. (2018) improved the adaptive complexity of this problem to $O(\log(n))$. Chen et al. (2018) considered the unconstrained submodular maximization problem and proposed the first algorithm that achieves the optimal approximation guarantee in a constant number of adaptive rounds.

Contributions The main contributions of our paper are:

- We introduce SIEVE-STREAMING++ which is the first streaming algorithm with optimal approximation factor and memory complexity. Note that our optimality result for the approximation factor is under the natural assumption that the Oracle is allowed to make queries only over the feasible sets of cardinality at most k.
- We design an algorithm for a hybrid model of submodular maximization, where it enjoys a near-optimal adaptive complexity and it still guarantees both optimal approximation factor and memory complexity. We also prove that our algorithm has a very low communication cost in a multi-source streaming setting.
- We use multi-source streams of data from Twitter and YouTube to compare our algorithms against state-of-the-art streaming approaches.
- We significantly improve the memory complexity for several important problems in the submodular maximization literature by applying the main idea of SIEVE-STREAMING++ (see the Supplementary material).

3. Streaming Submodular Maximization

In this section, we propose an algorithm called SIEVE-STREAMING++ that has the optimal $1/2$-approximation factor and memory complexity $O(k)$. Our algorithm is designed based on the SIEVE-STREAMING algorithm (Badanidiyuru et al., 2014).

The general idea behind SIEVE-STREAMING is that choosing elements with marginal gain at least $\tau^* = \frac{\mathtt{OPT}}{2k}$ from a data stream returns a set S with an objective value of at least $f(S) \geq \frac{\mathtt{OPT}}{2}$. The main problem with this primary idea is that the value of $\mathtt{OPT}$ is not known. Badanidiyuru et al. (2014) pointed out that, from the submodularity of f, we can trivially deduce $\Delta_0 \leq \mathtt{OPT} \leq k\Delta_0$ where Δ_0 is the largest value in the set $\{f(\{e\}) \mid e \in V\}$. It is also possible to find an accurate guess for $\mathtt{OPT}$ by dividing the range $[\Delta_0, k\Delta_0]$ into small intervals of $[\tau_i, \tau_{i+1})$. For this reason, it suffices to try $\log k$ different thresholds τ to obtain a close enough estimate of $\mathtt{OPT}$. Furthermore, in a streaming setting, where we do not know the maximum value of singletons a priori, Badanidiyuru et al. (2014) showed it suffices to only consider the range $\Delta \leq \mathtt{OPT} \leq 2k\Delta$, where Δ is the maximum value of singleton elements observed so far. The memory complexity of SIEVE-STREAMING is $O(k \log k/\varepsilon)$ because there are $O(\log k/\varepsilon)$ different thresholds and, for each one, we could keep at most k elements.

3.1. The SIEVE-STREAMING++ Algorithm

In the rest of this section, we show that with a novel modification to SIEVE-STREAMING it is possible to significantly reduce the memory complexity of the streaming algorithm.

Our main observation is that in the process of guessing $\mathtt{OPT}$, the previous algorithm uses Δ as a lower bound for $\mathtt{OPT}$; but as new elements are added to sets S_τ, it is possible to get better and better estimates of a lower bound on $\mathtt{OPT}$. More specifically, we have $\mathtt{OPT} \geq \mathtt{LB} \triangleq \max_\tau f(S_\tau)$ and as a result, there is no need to keep thresholds smaller than $\frac{\mathtt{LB}}{2k}$. Also, for a threshold τ we can conclude that there is at most $\frac{\mathtt{LB}}{\tau}$ elements in set S_τ. These two important observations allow us to get a geometrically decreasing upper bound on the number of items stored for each guess τ, which gives the asymptotically optimal memory complexity of $O(k)$. The details of our algorithm (SIEVE-STREAMING++) are described in Algorithm 1. Note that we represent the marginal gain of a set A to the set B with $f(A \mid B) = f(A \cup B) - f(B)$. Theorem 1 guarantees the performance of SIEVE-STREAMING++. Table 1 compares the state-of-the-art streaming algorithms based on approximation ratio, memory complexity and queries per element.

Theorem 1. *For a non-negative monotone submodular function f subject to a cardinality constraint k,* SIEVE-STREAMING++ *returns a solution S such that (i) $f(S) \geq (1/2 - \varepsilon) \cdot \max_{A \subseteq V, |A| \leq k} f(A)$, (ii) memory complexity is $O(k/\varepsilon)$, and (iii) number of queries is $O(\log(k)/\varepsilon)$ per each element.*

3.2. The BATCH-SIEVE-STREAMING++ Algorithm

The SIEVE-STREAMING++ algorithm, for each incoming element of the stream, requires at least one query to the Oracle which increases its adaptive complexity to $\Omega(n)$. Since the adaptivity of an algorithm has a significant impact

Table 1. Streaming algorithms for non-negative and monotone submodular maximization subject to a cardinality constraint k. The SIEVE-STREAMING++ provides the best approximation ratio, memory complexity (up to a constant factor), and query complexity.

Algorithm	**Approx. Ratio**	**Memory**	**Queries per Element**	**Reference**
PREEMPTION-STREAMING	$1/4$	$O(k)$	$O(k)$	Buchbinder et al. (2015)
SIEVE-STREAMING	$1/2 - \varepsilon$	$O(k \log(k)/\varepsilon)$	$O(\log(k)/\varepsilon)$	Badanidiyuru et al. (2014)
SIEVE-STREAMING++	$1/2 - \varepsilon$	$O(k/\varepsilon)$	$O(\log(k)/\varepsilon)$	Ours

Algorithm 1 SIEVE-STREAMING++

Input: Submodular function f, data stream V, cardinality constraint k and error term ε

```
1: $\tau_{\min} \leftarrow 0$, $\Delta \leftarrow 0$ and $\texttt{LB} \leftarrow 0$
2: while there is an incoming item $e$ from $V$ do
3:     $\Delta \leftarrow \max\{\Delta, f(\{e\})\}$
4:     $\tau_{\min} = \frac{\max(\texttt{LB},\Delta)}{2k}$
5:     Discard all sets $S_\tau$ with $\tau < \tau_{\min}$
6:     for $\tau \in \{(1+\varepsilon)^i | \tau_{\min}/(1+\varepsilon) \leq (1+\varepsilon)^i \leq \Delta\}$ do
7:         if $\tau$ is a new threshold then $S_\tau \leftarrow \varnothing$
8:         if $|S_\tau| < k$ and $f(\{e\} \mid S_\tau) \geq \tau$ then
9:             $S_\tau \leftarrow S_\tau \cup \{e\}$ and $\texttt{LB} \leftarrow \max\{\texttt{LB}, f(S_\tau)\}$
10: return $\arg\max_{S_\tau} f(S_\tau)$
```

on its ability to be executed in parallel, there is a dire need to implement streaming algorithms with low adaptivity. To address this concern, our proposal is to first buffer a fraction of the data stream and then, through a parallel threshold filtering procedure, reduce the adaptive complexity, thus substantially lower the running time. Our results show that a small buffer memory can significantly parallelize streaming submodular maximization.

One natural idea for parallelization is to iteratively perform the following two steps: (i) for a threshold τ, in one adaptive round, compute the marginal gain of elements to set S_τ and discard those with a gain less than τ, and (ii) pick one of the remaining items with a good marginal gain and add it to S_τ. This process is repeated at most k times. We refer to this algorithm as SAMPLE-ONE-STREAMING and we will use it as a baseline in Section 5.3.

Although this method gives a $1/2 - \varepsilon$ approximation factor, the adaptive complexity of this algorithm is $\Omega(k)$ which is still prohibitive in the worst case. For this reason, we introduce a hybrid algorithm called BATCH-SIEVE-STREAMING++. This algorithm enjoys two important properties: (i) the number of adaptive rounds is near-optimal, and (ii) it has an optimal memory complexity (by adopting an idea similar to SIEVE-STREAMING++). Next, we explain BATCH-SIEVE-STREAMING++ (Algorithm 2) in detail.

First, we assume that the machine has a buffer $\mathcal{B}$ that can store at most $\texttt{B}$ elements. For a data stream V, whenever `Threshold` fraction of the buffer is full, the optimization process begins. The purpose of `Threshold` is to empty the buffer before it gets completely full and to avoid losing arriving elements. Similar to the other sieve streaming methods, BATCH-SIEVE-STREAMING++ requires us to guess the value of $\tau^* = \frac{\texttt{OPT}}{2k}$. For each guess τ, BATCH-SIEVE-STREAMING++ uses THRESHOLD-SAMPLING (Algorithm 3) as a subroutine. THRESHOLD-SAMPLING iteratively picks random batches of elements T. If their average marginal gain to the set of picked elements S_τ is at least $(1-\varepsilon)\tau$ it adds that batch to S_τ. Otherwise, all elements with marginal gain less than τ to the set S_τ are filtered out. THRESHOLD-SAMPLING repeats this process until the buffer is empty or $|S_\tau| = k$.

Note that in Algorithm 2, we define the function f_S as $f_S(A) = f(A \mid S)$, which calculates the marginal gain of adding a set A to S. It is straightforward to show that if f is a non-negative and monotone submodular function, then f_S is also non-negative and monotone submodular.

The adaptive complexity of BATCH-SIEVE-STREAMING++ is the number of times its buffer gets full (which is at most $N/\texttt{B}$) multiplied by the adaptive complexity of THRESHOLD-SAMPLING. The reason for the low adaptive complexity of THRESHOLD-SAMPLING is quite subtle. In Line 3 of Algorithm 3, with a non-negligible probability, a constant fraction of items is discarded from the buffer. Thus, the while loop continues for at most $O(\log \texttt{B})$ steps. Since we increase the batch size by a constant factor of $(1+\varepsilon)$ each time, the for loops within each while loop will run at most $O(\log(k)/\varepsilon)$ times. Therefore, the total adaptive complexity of BATCH-SIEVE-STREAMING++ is $O(\frac{N \log(\texttt{B}) \log(k)}{\texttt{B}\varepsilon})$ Note that when $|S| < 1/\varepsilon$, multiplying the size by $(1+\varepsilon)$ would increase it less than one, so we increase the batch size one by one for the first loop in Lines 4–10 of Algorithm 3. Theorem 2 guarantees the performance of BATCH-SIEVE-STREAMING++.

Theorem 2. *For a non-negative monotone submodular function f subject to a cardinality constraint k, define N to be the total number of elements in the stream, $\texttt{B}$ to be the buffer size and $\varepsilon < 1/3$ to be a constant. For* BATCH-SIEVE-STREAMING++ *we have: (i) the approximation factor is $1/2 - 3\varepsilon/2$, (ii) the memory complexity is $O(\texttt{B} + k/\varepsilon)$, and (iii) the expected adaptive complexity is $O(\frac{N \log(\texttt{B}) \log(k)}{\texttt{B}\varepsilon})$.*

Algorithm 2 BATCH-SIEVE-STREAMING++

Input: Stream of data V, submodular set function f, cardinality constraint k, buffer $\mathcal{B}$ with a memory $\texttt{B}$, $\texttt{Threshold}$, and error term ε.

```
1: $\Delta \leftarrow 0, \tau_{\min} \leftarrow 0, \texttt{LB} \leftarrow 0$ and $\mathcal{B} \leftarrow \varnothing$
2: while there is an incoming element $e$ from $V$ do
3:   Add $e$ to $\mathcal{B}$
4:   if the buffer $\mathcal{B}$ is $\texttt{Threshold}$ percent full then
5:     $\Delta \leftarrow \max\{\Delta, \max_{e \in \mathcal{B}} f(e)\}$, $\tau_{\min} = \frac{\max(\texttt{LB},\Delta)}{2k(1+\varepsilon)}$
       and discard all sets $S_\tau$ with $\tau < \tau_{\min}$
6:     for $\tau \in \{(1+\varepsilon)^i | \tau_{\min} \leq (1+\varepsilon)^i \leq \Delta\}$ do
7:       If $\tau$ is a new threshold then assign a new set $S_\tau$
         to it, i.e., $S_\tau \leftarrow \varnothing$
8:       if $|S_\tau| < k$ then
9:         $T \leftarrow$ THRESHOLD-SAMPLING$(f_{S_\tau}, \mathcal{B}, k - |S_\tau|, \tau, \varepsilon)$
10:        $S_\tau \leftarrow S_\tau \cup T$
11:    $\texttt{LB} = \max_{S_\tau} f(S_\tau)$ and $\mathcal{B} \leftarrow \varnothing$
12: return $\arg\max_{S_\tau} f(S_\tau)$
```

Algorithm 3 THRESHOLD-SAMPLING

Input: Submodular set function f, set of buffered items $\mathcal{B}$, cardinality constraint k, threshold τ, and error term ε

```
1: $S \leftarrow \varnothing$
2: while $|\mathcal{B}| > 0$ and $|S| < k$ do
3:   update $\mathcal{B} \leftarrow \{x \in \mathcal{B} : f(\{x\} \mid S) \geq \tau\}$ and filter
     out the rest
4:   for $i = 1$ to $\lceil \frac{1}{\varepsilon} \rceil$ do
5:     Sample $x$ uniformly at random from $\mathcal{B} \setminus S$
6:     if $f(\{x\}|S) \leq (1-\varepsilon)\tau$ then
7:       break and go to Line 2
8:     else
9:       $S \leftarrow S \cup \{x\}$
10:      if $|S| = k$ then return $S$
11:  for $i = \lfloor \log_{1+\varepsilon}(1/\varepsilon) \rfloor$ to $\lceil \log_{1+\varepsilon} k \rceil - 1$ do
12:    $t \leftarrow \min\{\lfloor (1+\varepsilon)^{i+1} - (1+\varepsilon)^i \rfloor, |\mathcal{B} \setminus S|, k - |S|\}$
13:    Sample a random set $T$ of size $t$ from $\mathcal{B} \setminus S$
14:    if $|S \cup T| = k$ then return $S \cup T$
15:    if $\frac{f(T \mid S)}{|T|} \leq (1-\varepsilon)\tau$ then
16:      $S \leftarrow S \cup T$ and break
17:    else
18:      $S \leftarrow S \cup T$
19: return $S$
```

Remark It is important to note that THRESHOLD-SAMPLING is inspired by recent progress for maximizing submodular functions with low adaptivity (Fahrbach et al., 2019; Balkanski et al., 2019a; Ene & Nguyen, 2019) but it uses a few new ideas to adapt the result to our setting. Indeed, if we had used the sampling routines from these previous works, it was even possible to slightly improve the adaptivity of the hybrid model. The main issue with these methods is that their adaptivity heavily depends on evaluating many random subsets of the ground set in each round. As it is discussed in the next section, we are interested in algorithms that are efficient in the multi-source setting. In that scenario, the data is distributed among several machines, so existing sampling methods dramatically increases the communication cost of our hybrid algorithm.

4. Multi-Source Data Streams

In general, the important aspects to consider for a single source streaming algorithm are approximation factor, memory complexity, and adaptivity. In the multi-source setting, the communication cost of an algorithm also plays an important role. While the main ideas of SIEVE-STREAMING++ give us an optimal approximation factor and memory complexity, there is always a trade-off between adaptive complexity and communication cost in any threshold sampling procedure.

As we discussed before, existing submodular maximization algorithms with low adaptivity need to evaluate the utility of random subsets several times to guarantee the marginal gain of sampled items. Consequently, this incurs high communication cost. In this section, we explain how BATCH-SIEVE-STREAMING++ can be generalized to the multi-source scenario with both low adaptivity and low communication cost.

We assume elements arrive from m different data streams and for each stream the elements are placed in a separate machine with a buffer $\mathcal{B}_i$. When the buffer memory of at least one of these m machines is $\texttt{Threshold}\%$ full, the process of batch insertion and filtering begins. The only necessary change to BATCH-SIEVE-STREAMING++ is to use a parallelized version of THRESHOLD-SAMPLING with inputs from $\{\mathcal{B}_i\}$. In this generalization, Lines 5 and 13 of Algorithm 3 are executed in a distributed way where the goal is to perform the random sampling procedure from the buffer memory of all machines. Indeed, in order to pick a batch of t random items, the central coordinator asks each machine to send a pre-decided number of items. Note that the set of picked elements S_τ for each threshold τ is shared among all machines. And therefore the filtering step at Line 3 of Algorithm 3 can be done independently for each stream in only one adaptive round. Our algorithm is shown pictorially in Figure 1.

Theorem 3 guarantees the communication cost of BATCH-SIEVE-STREAMING++ in the multi-source setting. Notably, the communication cost of our algorithm is independent of the buffer size $\texttt{B}$ and the total number of elements N.

Theorem 3. *For a non-negative and monotone submodular function f in a multi-source streaming setting subject to a cardinality constraint k, define Δ_0 as the largest singleton value when for the first time a buffer gets full, and $C = \frac{\texttt{OPT}}{\Delta_0}$. The total communication cost of* BATCH-SIEVE-STREAMING++ *is* $O(\frac{k \log C}{\varepsilon^2})$.

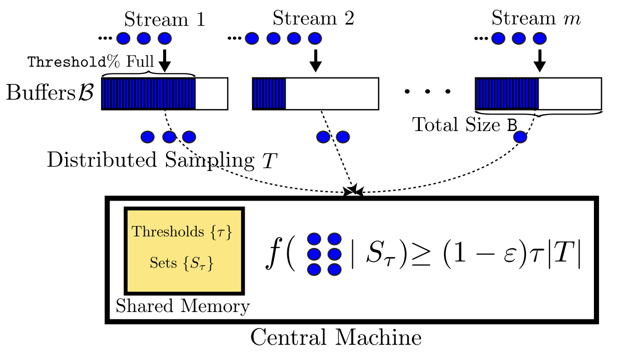


Figure 1. The schematic representation of our proposed hybrid algorithm: there are m simultaneous streams where data from each stream is buffered separately. When a buffer is `Threshold%` full, a central machine starts the sampling process. The thresholds $\{\tau\}$ and sets $\{S_\tau\}$ are stored in a shared memory. First, for each threshold τ, all elements with marginal gain less than τ are discarded from the buffers. Then the central machine randomly samples t items T (with geometrically increasing values of t) from the buffers of all streams and adds them to set S_τ if their average marginal gain is at least $(1-\varepsilon)\tau$. The sampling procedure stops when the average value of randomly picked items is not good enough. These iterative steps are performed until k items are picked or the buffer memories of all machines are emptied.

5. Experiments

In these experiments, we have three main goals:

1. For the single-source streaming scenario, we want to demonstrate the memory efficiency of SIEVE-STREAMING++ relative to SIEVE-STREAMING.
2. For the multi-source setting, we want to showcase how BATCH-SIEVE-STREAMING++ requires the fewest adaptive rounds amongst algorithms with optimal communication costs.
3. Lastly, we want to illustrate how a simple variation of BATCH-SIEVE-STREAMING++ can trade off communication cost for adaptivity, thus allowing the user to find the best balance for their particular problem.

5.1. Datasets

These experiments will be run on a Twitter stream summarization task and a YouTube Video summarization task, as described next.

Twitter Stream Summarization In this application, we want to produce real-time summaries for Twitter feeds. As of January 2019, six of the top fifty Twitter accounts (also known as "handles") are dedicated primarily to news reporting. Each of these handles has over thirty million followers, and there are many other news handles with tens of millions of followers as well. Naturally, such accounts commonly share the same stories. Whether we want to provide a periodic synopsis of major events or simply to reduce the clutter in a user's feed, it would be very valuable if we could produce a succinct summary that still relays all the important information.

To collect the data, we scraped recent tweets from 30 different popular news accounts, giving us a total of 42,104 unique tweets. In the multi-source experiments, we assume that each machine is scraping one page of tweets, so we have 30 different streams to consider.

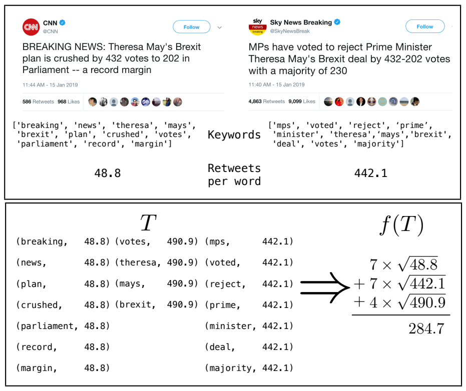
We want to define a submodular function that covers the important stories of the day without redundancy. To this end, we extract the keywords from each tweet and weight them proportionally to the number of retweets the post received. In order to encourage diversity in a selected set of tweets, we take the square root of the value assigned to each keyword. More formally, consider a function f defined over a ground set V of tweets. Each tweet $e \in V$ consists of a positive value val_e denoting its number of retweets and a set of ℓ_e keywords $W_e = \{w_{e,1}, \cdots, w_{e,\ell_e}\}$ from a general set of keywords $\mathcal{W}$. The score of a word $w \in W_e$ for a tweet e is defined by $\text{score}(w, e) = \text{val}_e$. If $w \notin W_e$, we define $\text{score}(w, e) = 0$. For a set $S \subseteq V$ of tweets, the function f is defined as follows:

$$f(S) = \sum_{w \in \mathcal{W}} \sqrt{\sum_{e \in S} \text{score}(w, e)}.$$

Figure 2 gives an example and Appendix E gives a proof of submodularity for this function.

YouTube Video Summarization In this second task, we want to select representative frames from multiple simultaneous and related video feeds. In particular, we consider YouTube videos of New Year's Eve celebrations from ten different cities around the world. Although the cities are not all in the same time zone, in our multi-source experiments we assume that we have one machine processing each video simultaneously.

Using the first 30 seconds of each video, we train an autoencoder that compresses each frame into a 4-dimensional representative vector. Given a ground set V of such vectors, we define a matrix M such that $M_{ij} = e^{-\text{dist}(v_i, v_j)}$, where $\text{dist}(v_i, v_j)$ is the euclidean distance between vectors $v_i, v_j \in V$. Intuitively, M_{ij} encodes the similarity between the frames represented by v_i and v_j.



CNN @CNN Follow

BREAKING NEWS: Theresa May's Brexit plan is crushed by 432 votes to 202 in Parliament -- a record margin

11:44 AM - 15 Jan 2019

586 Retweets 968 Likes

Sky News Breaking @SkyNewsBreak Follow

MPs have voted to reject Prime Minister Theresa May's Brexit deal by 432-202 votes with a majority of 230

11:40 AM - 15 Jan 2019

4,863 Retweets 9,099 Likes

```
['breaking', 'news', 'theresa', 'mays',             ['mps', 'voted', 'reject', 'prime',
 'brexit', 'plan', 'crushed', 'votes',   Keywords    'minister', 'theresa','mays','brexit',
 'parliament', 'record', 'margin']                   'deal', 'votes', 'majority']

            48.8                         Retweets                442.1
                                         per word
```

T

```
(breaking,   48.8) (votes,   490.9) (mps,      442.1)
(news,       48.8) (theresa, 490.9) (voted,    442.1)
(plan,       48.8) (mays,    490.9) (reject,   442.1)
(crushed,    48.8) (brexit,  490.9) (prime,    442.1)
(parliament, 48.8)                  (minister, 442.1)
(record,     48.8)                  (deal,     442.1)
(margin,     48.8)                  (majority, 442.1)
```

$f(T)$

$$\Longrightarrow \begin{array}{r} 7 \times \sqrt{48.8} \\ + 7 \times \sqrt{442.1} \\ + 4 \times \sqrt{490.9} \\ \hline 284.7 \end{array}$$

Figure 2. At the top, we show two tweets on the same subject from different accounts. We first extract the list of keywords, as well as the number of retweets per word. We combine these into a single list T of (keyword, score) pairs and then pass this list through our submodular function f.

The utility of a set $S \subseteq V$ is defined as a non-negative monotone submodular objective $f(S) = \log\det(I + \alpha M_S)$, where I is the identity matrix, $\alpha > 0$ and M_S is the principal sub-matrix of M indexed by S (Herbrich et al., 2003). Informally, this function is meant to measure the diversity of the vectors in S. Figure 3 shows the representative images selected by our BATCH-SIEVE-STREAMING++ algorithm for $k = 8$.

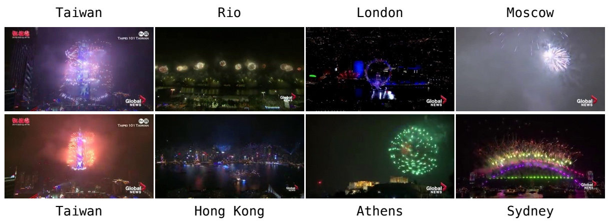


Figure 3. Eight representative frames chosen by BATCH-SIEVE-STREAMING++ from ten different simultaneous feeds of New Year's Eve fireworks from around the world.

5.2. Single-Source Experiments

In this section, we want to emphasize the power of SIEVE-STREAMING++ in the single-source streaming scenario. As discussed earlier, the two existing standard approaches for monotone k-cardinality submodular streaming are SIEVE-STREAMING and PREEMPTION-STREAMING.

As mentioned in Section 3, SIEVE-STREAMING++ theoretically has the best properties of both of these existing baselines, with optimal memory complexity and the optimal approximation guarantee. Figures 4a through 4d show the performance of these three algorithms on the YouTube task and confirm that this holds in practice as well.

For the purposes of this test, we simply combined the different video feeds into one single stream. We see that the memory required by SIEVE-STREAMING++ is much smaller than the memory required by SIEVE-STREAMING, but it still achieves the exact same utility. Furthermore, the memory requirement of SIEVE-STREAMING++ is within a constant factor of PREEMPTION-STREAMING, while its utility is much better. The Twitter experiment gives similar results so those graphs are deferred to Appendix F.

5.3. Multi-Source Experiments

Once we move into the multi-source setting, the communication cost of algorithms becomes a key concern also. In this section, we compare the performance of algorithms in terms of utility and adaptivity where their communication cost is optimal.

Our first baseline is a trivial extension of SIEVE-STREAMING++. The multi-source extension for this algorithm essentially functions by locally computing the marginal gain of each incoming element, and only communicating it to the central machine if the marginal gain is above the desired threshold. However, as mentioned at the beginning Section 3.1, this algorithm requires $\Omega(n)$ adaptive rounds. Our second baseline is SAMPLE-ONE-STREAMING, which was described in Section 3.2.

Figures 4e and 4f show the effect of the buffer size $\mathtt{B}$ on the performance of these algorithms for the Twitter task. The main observation is that BATCH-SIEVE-STREAMING++ can achieve roughly the same utility as the two baselines with many fewer adaptive rounds. Note that the number of adaptive rounds is shown in log scale.

Figures 4g and 4h show how these numbers vary with ε. Again, the utilities of the three baselines are similar. We also see that increasing ε results in a large drop in the number of adaptive rounds for BATCH-SIEVE-STREAMING++, but not for SAMPLE-ONE-STREAMING. Appendix F gives some additional graphs, as well as the results for the YouTube dataset.

5.4. Trade-off Between Communication and Adaptivity

In the multi-source setting, there is a natural exchange between communication cost and adaptivity. Intuitively, the idea is that if we sample items more aggressively (which translates into higher communication cost), a set S of k items is generally picked faster, thus it reduces the adaptivity. In the real world, the preference for one or the other can depend on a wide variety of factors ranging from resource constraints to the requirements of the particular problem.

In THRESHOLD-SAMPLING, we ensure the optimal communication performance by sampling $t_i = \lceil (1+\varepsilon)^{i+1} - (1+\varepsilon)^i \rceil$ items in each step of the for loop. Instead, to reduce the adaptivity by a factor of $\log(k)$, we could sample all the required k items in a single step. Thus, in one

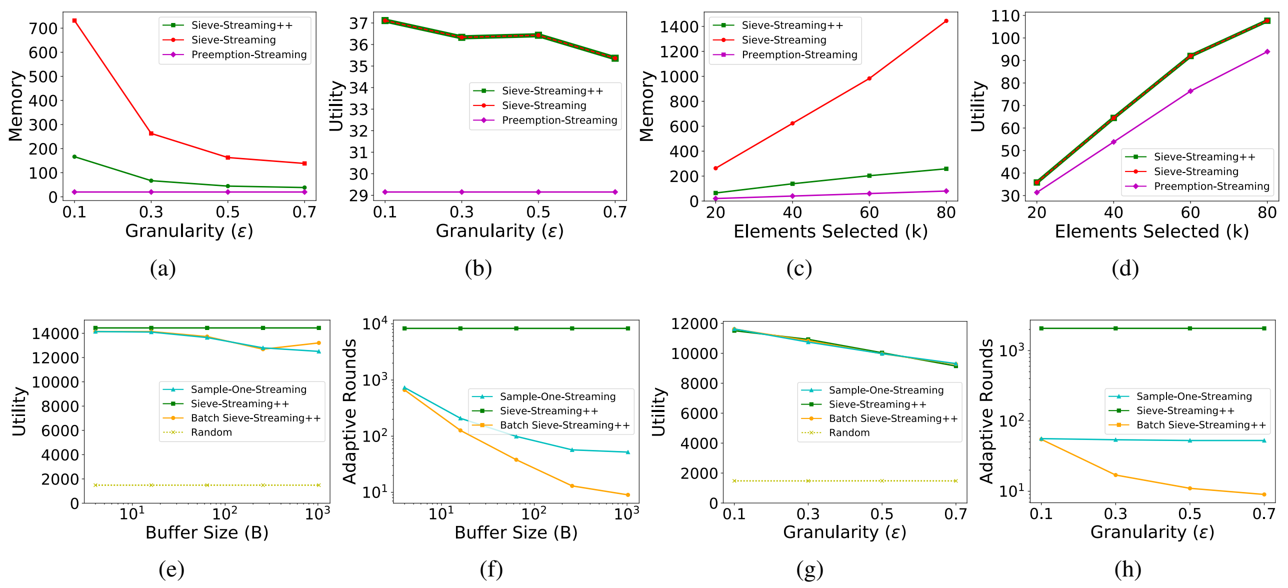


Figure 4. Graphs (a) to (d) show how the memory and utility of various single-source streaming algorithms vary with the cardinality k and granularity ε. Note that the utility of SIEVE-STREAMING++ and SIEVE-STREAMING exactly overlap. In (a) and (b) we use k = 20, while in (c) and (d) we use $\varepsilon = 0.3$. Graphs (e) to (h) show how the utility and adaptivity of various multi-source streaming algorithms vary with the buffer size $\mathtt{B}$ and the granularity ε. Unless they are being varied on the x-axis, we set $\varepsilon = 0.7$, $\mathtt{B} = 100$, and $k = 50$.

adaptive round we mimic the two for loops of THRESHOLD-SAMPLING. Doing this in each call to Algorithm 3 would reduce the expected adaptive complexity of THRESHOLD-SAMPLING to the optimal $\log(\mathtt{B})$, but dramatically increase the communication cost to $O(k \log \mathtt{B})$.

In order to trade off between communication and adaptivity, we can instead sample $t_i^R = \lceil (1+\varepsilon)^{i+R} - (1+\varepsilon)^i \rceil$ elements to perform R consecutive adaptive rounds in only one round. However, to maintain the same chance of a successful sampling, we still need to check the marginal gain. Finally, we pick a batch of the largest size t_i^j such that the average marginal gain of the first t_i^{j-1} items is above the desired threshold. Then we just add just this subset to S_τ, meaning we have wasted $\lceil (1+\varepsilon)^{i+R} - (1+\varepsilon)^{i+j} \rceil$ communication.

Scatter plots of Figure 5 shows how the number of adaptive rounds varies with the communication cost. Each individual dot represents a single run of the algorithm on a different subset of the data. The different colors cluster the dots into groups based on the value of R that we used in that run. Note that the parameter R controls the communication cost.

The plot on the left comes from the Twitter experiment, while the plot on the right comes from the YouTube experiment. Although the shapes of the clusters are different in the two experiments, we see that increasing R increases the communication cost, but also decreases the number of adaptive rounds, as expected.

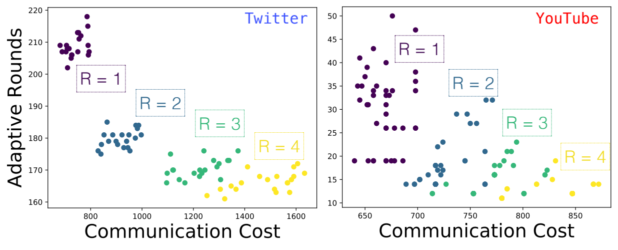


Figure 5. Scatter plots showing how we can lower the number of adaptive rounds by increasing communication. Each dot is the result of a single run of the algorithm and the colored clusters represent a particular setting for R.

6. Conclusion

In this paper, we studied the problem of maximizing a non-negative submodular function over a multi-source stream of data subject to a cardinality constraint k. We first proposed SIEVE-STREAMING++ with the optimum approximation factor and memory complexity for a single stream of data. Build upon this idea, we designed an algorithm for multi-source streaming setting with a $1/2$ approximation factor, $O(k)$ memory complexity, a very low communication cost, and near-optimal adaptivity. We evaluated the performance of our algorithms on two real-world data sets of multi-source tweet streams and video streams. Furthermore, by using the main idea of SIEVE-STREAMING++, we significantly improved the memory complexity of several important submodular maximization problems.

Acknowledgements

The work of Amin Karbasi is supported by AFOSR Young Investigator Award (FA9550-18-1-0160) and Grainger Award (PO 2000008083 2016012). We would like to thank Ola Svensson for his comment on the first version of this manuscript.

References

Agrawal, S., Shadravan, M., and Stein, C. Submodular Secretary Problem with Shortlists. In *Innovations in Theoretical Computer Science Conference, ITCS*, pp. 1:1–1:19, 2019.

Badanidiyuru, A. and Vondrák, J. Fast algorithms for maximizing submodular functions. In *Symposium on Discrete Algorithms, SODA*, pp. 1497–1514, 2014.

Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., and Krause, A. Streaming Submodular Maximization:Massive Data Summarization on the Fly. In *International Conference on Knowledge Discovery and Data Mining, KDD*, pp. 671–680, 2014.

Balkanski, E. and Singer, Y. The adaptive complexity of maximizing a submodular function. In *Symposium on Theory of Computing, STOC*, pp. 1138–1151, 2018.

Balkanski, E., Mirzasoleiman, B., Krause, A., and Singer, Y. Learning sparse combinatorial representations via two-stage submodular maximization. In *International Conference on Machine Learning (ICML)*, 2016.

Balkanski, E., Breuer, A., and Singer, Y. Non-monotone Submodular Maximization in Exponentially Fewer Iterations. In *Advances in Neural Information Processing Systems*, pp. 2359–2370, 2018.

Balkanski, E., Rubinstein, A., and Singer, Y. An Exponential Speedup in Parallel Running Time for Submodular Maximization without Loss in Approximation. In *Symposium on Discrete Algorithms (SODA)*, pp. 283–302, 2019a.

Balkanski, E., Rubinstein, A., and Singer, Y. An optimal approximation for submodular maximization under a matroid constraint in the adaptive complexity model. In *Symposium on Theory of Computing, STOC*, 2019b.

Bansal, N. and Sviridenko, M. The santa claus problem. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, pp. 31–40. ACM, 2006.

Barbosa, R., Ene, A., Nguyen, H., and Ward, J. The power of randomization: Distributed submodular maximization on massive datasets. In *International Conference on Machine Learning (ICML)*, pp. 1236–1244, 2015.

Barbosa, R., Ene, A., Nguyen, H. L., and Ward, J. A New Framework for Distributed Submodular Maximization. In *Annual Symposium on Foundations of Computer Science, FOCS*, pp. 645–654, 2016.

Buchbinder, N., Feldman, M., and Schwartz, R. Online submodular maximization with preemption. In *ACM-SIAM Symposium on Discrete Algorithms, SODA*, pp. 1202–1216, 2015.

Buchbinder, N., Feldman, M., and Schwartz, R. Comparing Apples and Oranges: Query Trade-off in Submodular Maximization. *Math. Oper. Res.*, 42(2):308–329, 2017.

Chakrabarti, A. and Kale, S. Submodular maximization meets streaming: matchings, matroids, and more. *Math. Program.*, 154(1-2):225–247, 2015.

Chan, T. H., Huang, Z., Jiang, S. H., Kang, N., and Tang, Z. G. Online Submodular Maximization with Free Disposal: Randomization Beats ¼ for Partition Matroids. In *Symposium on Discrete Algorithms, SODA*, pp. 1204–1223, 2017.

Chekuri, C. and Quanrud, K. Parallelizing greedy for submodular set function maximization in matroids and beyond. In *Symposium on Theory of Computing, STOC*, 2019.

Chekuri, C., Gupta, S., and Quanrud, K. Streaming algorithms for submodular function maximization. In *International Colloquium on Automata, Languages, and Programming*, pp. 318–330. Springer, 2015.

Chen, J., Nguyen, H. L., and Zhang, Q. Submodular Maximization over Sliding Windows. *CoRR*, abs/1611.00129, 2016.

Chen, L., Feldman, M., and Karbasi, A. Unconstrained submodular maximization with constant adaptive complexity. *CoRR*, abs/1811.06603, 2018.

Das, A. and Kempe, D. Submodular meets Spectral: Greedy Algorithms for Subset Selection, Sparse Approximation and Dictionary Selection. In *International Conference on Machine Learning (ICML)*, pp. 1057–1064, 2011.

Elenberg, E. R., Dimakis, A. G., Feldman, M., and Karbasi, A. Streaming Weak Submodularity: Interpreting Neural Networks on the Fly. In *Advances in Neural Information Processing Systems*, pp. 4047–4057, 2017.

Ene, A. and Nguyen, H. L. Submodular Maximization with Nearly-optimal Approximation and Adaptivity in Nearly-linear Time. In *Symposium on Discrete Algorithms (SODA)*, pp. 274–282, 2019.

Epasto, A., Lattanzi, S., Vassilvitskii, S., and Zadimoghaddam, M. Submodular Optimization Over Sliding Windows. In *WWW*, pp. 421–430, 2017.

Fahrbach, M., Mirrokni, V. S., and Zadimoghaddam, M. Non-monotone Submodular Maximization with Nearly Optimal Adaptivity Complexity. *CoRR*, abs/1808.06932, 2018.

Fahrbach, M., Mirrokni, V. S., and Zadimoghaddam, M. Submodular Maximization with Nearly Optimal Approximation, Adaptivity and Query Complexity. In *Symposium on Discrete Algorithms (SODA)*, pp. 255–273, 2019.

Feldman, M., Harshaw, C., and Karbasi, A. Greed Is Good: Near-Optimal Submodular Maximization via Greedy Optimization. In *Conference on Learning Theory*, 2017.

Feldman, M., Karbasi, A., and Kazemi, E. Do Less, Get More: Streaming Submodular Maximization with Subsampling. In *Advances in Neural Information Processing Systems*, pp. 730–740, 2018.

Herbrich, R., Lawrence, N. D., and Seeger, M. Fast sparse gaussian process methods: The informative vector machine. In *Advances in Neural Information Processing Systems*, pp. 625–632, 2003.

Kazemi, E., Zadimoghaddam, M., and Karbasi, A. Scalable Deletion-Robust Submodular Maximization: Data Summarization with Privacy and Fairness Constraints. In *International Conference on Machine Learning (ICML)*, pp. 2549–2558, 2018.

Krause, A. and Golovin, D. Submodular Function Maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, 2012.

Kumar, R., Moseley, B., Vassilvitskii, S., and Vattani, A. Fast Greedy Algorithms in MapReduce and Streaming. *TOPC*, 2(3):14:1–14:22, 2015.

Liu, P. and Vondrák, J. Submodular Optimization in the MapReduce Model. *CoRR*, abs/1810.01489, 2018.

Mirrokni, V. and Zadimoghaddam, M. Randomized composable core-sets for distributed submodular maximization. In *Symposium on Theory of Computing, , STOC*, pp. 153–162. ACM, 2015.

Mirzasoleiman, B., Badanidiyuru, A., Karbasi, A., Vondrak, J., and Krause, A. Lazier than Lazy Greedy. In *AAAI Conference on Artificial Intelligence*, pp. 1812–1818, 2015.

Mirzasoleiman, B., Badanidiyuru, A., and Karbasi, A. Fast constrained submodular maximization: Personalized data summarization. In *International Conference on Machine Learning (ICML)*, pp. 1358–1367, 2016a.

Mirzasoleiman, B., Karbasi, A., Sarkar, R., and Krause, A. Distributed Submodular Maximization. *Journal of Machine Learning Research (JMLR)*, 17:1–44, 2016b.

Mirzasoleiman, B., Zadimoghaddam, M., and Karbasi, A. Fast Distributed Submodular Cover: Public-Private Data Summarization. In *Advances in Neural Information Processing Systems*, 2016c.

Mirzasoleiman, B., Karbasi, A., and Krause, A. Deletion-Robust Submodular Maximization: Data Summarization with "the Right to be Forgotten". In *International Conference on Machine Learning (ICML)*, pp. 2449–2458, 2017.

Mirzasoleiman, B., Jegelka, S., and Krause, A. Streaming Non-Monotone Submodular Maximization: Personalized Video Summarization on the Fly. In *AAAI Conference on Artificial Intelligence*, 2018.

Mitrovic, M., Bun, M., Krause, A., and Karbasi, A. Differentially Private Submodular Maximization: Data Summarization in Disguise. In *International Conference on Machine Learning (ICML)*, pp. 2478–2487, 2017a.

Mitrovic, M., Kazemi, E., Zadimoghaddam, M., and Karbasi, A. Data Summarization at Scale: A Two-Stage Submodular Approach. In *International Conference on Machine Learning (ICML)*, pp. 3593–3602, 2018.

Mitrovic, S., Bogunovic, I., Norouzi-Fard, A., Tarnawski, J. M., and Cevher, V. Streaming Robust Submodular Maximization: A Partitioned Thresholding Approach. In *Advances in Neural Information Processing Systems*, pp. 4560–4569, 2017b.

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

Norouzi-Fard, A., Tarnawski, J., Mitrovic, S., Zandieh, A., Mousavifar, A., and Svensson, O. Beyond 1/2-Approximation for Submodular Maximization on Massive Data Streams. In *International Conference on Machine Learning (ICML)*, pp. 3826–3835, 2018.

Stan, S., Zadimoghaddam, M., Krause, A., and Karbasi, A. Probabilistic Submodular Maximization in Sub-Linear Time. In *International Conference on Machine Learning (ICML)*, 2017.