Beyond $1/2$-Approximation for Submodular Maximization on Massive Data Streams

Ashkan Norouzi-Fard [*1] **Jakub Tarnawski** [*1] **Slobodan Mitrović** [*1] **Amir Zandieh** [*1] **Aidasadat Mousavifar** [1] **Ola Svensson** [1]

Abstract

Many tasks in machine learning and data mining, such as data diversification, non-parametric learning, kernel machines, clustering etc., require extracting a small but representative summary from a massive dataset. Often, such problems can be posed as maximizing a *submodular* set function subject to a cardinality constraint. We consider this question in the *streaming setting*, where elements arrive over time at a fast pace and thus we need to design an efficient, low-memory algorithm. One such method, proposed by Badanidiyuru et al. (2014), always finds a 0.5-approximate solution. Can this approximation factor be improved? We answer this question affirmatively by designing a new algorithm SALSA for streaming submodular maximization. It is the first low-memory, single-pass algorithm that improves the factor 0.5, under the natural assumption that elements arrive in a random order. We also show that this assumption is necessary, i.e., that there is no such algorithm with better than 0.5-approximation when elements arrive in arbitrary order. Our experiments demonstrate that SALSA significantly outperforms the state of the art in applications related to exemplar-based clustering, social graph analysis, and recommender systems.

1. Introduction

We are experiencing an unprecedented growth in the sizes of modern datasets. Streams of data of massive volume are generated every second, coming from many different sources in industry and science such as: image and video streams, sensor data, social networks, stock markets, and many others. Sometimes, such data is produced so rapidly that most of it cannot even be stored in any way. In this context, a critical task is *data summarization*: one of extracting a representative subset of manageable size from rich, large-scale data streams. A central topic in machine learning and data mining today, its main challenge is to produce such a concise yet high-value summary while doing so efficiently and on-the-fly.

In many applications, this challenge can be viewed as optimizing a *submodular function* subject to a cardinality constraint. Indeed, submodularity – an intuitive notion of diminishing returns, which postulates that an element should contribute more to a smaller set than to a larger one – plays a similar role in this setting as convexity does in continuous optimization. Namely, it is general enough to model a multitude of practical scenarios, such as viral marketing (Kempe et al., 2003), recommender systems (El-Arini & Guestrin, 2011), search result diversification (Agrawal et al., 2009) or active learning (Golovin & Krause, 2011), while allowing for both theoretically and practically sound and efficient algorithms. In particular, a celebrated result by Nemhauser et al. (1978) shows that the GREEDY algorithm – one that iteratively picks the element with the largest marginal contribution to the current summary – is a $(1 - 1/e)$-approximation for maximizing a monotone submodular function subject to a cardinality constraint. That is, the objective value that it attains is at least a $(1 - 1/e)$-fraction of the optimum. This approximation factor is NP-hard to improve (Feige, 1998). Unfortunately, GREEDY requires repeated access to the complete dataset, which precludes it from use in large-scale applications in terms of both memory and running time.

The sheer bulk of large datasets and the infeasibility of GREEDY together imply a growing need for faster and memory-efficient algorithms, ideally ones that can work in the *streaming setting*: one where input arrives one element at a time, rather than being available all at once, and only a small portion of the data can be kept in memory at any point. The first such algorithm was given by Chakrabarti and Kale (2014), yielding a 0.25-approximation while requiring only a single pass over the data, in arbitrary order, and using

[*]Equal contribution [1]Theory of Computation Laboratory, EPFL, Lausanne, Vaud, Switzerland. Correspondence to: Ashkan Norouzi-Fard <ashkan.afn@gmail.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

$O(k)$ function evaluations per element and $O(k)$ memory.[1] A more accurate and efficient method SIEVE-STREAMING was proposed by Badanidiyuru et al. (2014). For any $\varepsilon > 0$, it provides a $(0.5 - \varepsilon)$-approximation and uses $O\left(\log k/\varepsilon\right)$ function evaluations per element and $O\left(k \log k/\varepsilon\right)$ memory. While well-suited for use in big data stream scenarios, its approximation guarantee is nevertheless still inferior to that of GREEDY. It is natural to wonder: can the ratio 0.5 be improved upon by a more accurate algorithm?

It turns out that in general, the answer is no (modulo the natural assumption that the submodular function is only evaluated on feasible sets). As one of our results, we show that:

Theorem 1.1. *Any algorithm for streaming submodular maximization that only queries the value of the submodular function on feasible sets (i.e., sets of cardinality at most k) and is an α-approximation for a constant $\alpha > 0.5$ must use $\Omega(n/k)$ memory, where n is the length of the stream.*

This hardness includes randomized algorithms, and applies even for *estimating* the optimum value to within this factor, without necessarily returning a solution (see the full version).[2] Note that usually $n/k \gg k$; such an algorithm therefore cannot run in a large-scale streaming setting.

However, this bound pertains to *arbitrary-order* streams. An immediate question, then, is whether inherent randomness present in the stream can be helpful in obtaining higher accuracy. Namely, in many real-world scenarios the data arrives in random order, or can be processed in random order. This can be seen as a sweet spot between assuming that the data is drawn randomly from an underlying prior distribution – which is usually unrealistic – and needing to allow for instances whose contents and order are both adversarially designed – which also do not appear in applications. Is it possible to obtain a better approximation ratio under this natural assumption?

Again, we begin with a negative result: the performance of the state-of-the-art SIEVE-STREAMING algorithm remains capped at 0.5 in this setting.

Theorem 1.2. *There exists a family of instances on which the approximation ratio of* SIEVE-STREAMING *is at most $0.5 + o(1)$ even if elements arrive in random order.*

[1] We make the usual assumption that one can store any element, or the value of any set, using $O(1)$ memory. The memory usage calculation in (Chakrabarti & Kale, 2014) is lower-level, which results in an extra $\log n$ factor. Furthermore, their algorithm can be implemented using a priority queue, which would result in a runtime of $O(\log k)$ per element.

[2] Moreover, note that Theorem 1.1 does *not* follow from the work of Buchbinder et al. (2015), who proved an approximation hardness of 0.5 for *online* algorithms whose memory state must always be a feasible solution (consisting of at most k elements).

We remark that Theorem 1.2 also extends to certain natural modifications of SIEVE-STREAMING (with a different value of a threshold parameter used in the algorithm, or multiple such values that are tried in parallel). Thus, new ideas are required to go beyond an approximation ratio of 0.5.

As the main result of this paper we present a new algorithm SALSA (Streaming ALgorithm for Submodular maximization with Adaptive thresholding), which does break the 0.5 barrier in the random-order case. SALSA, like SIEVE-STREAMING, works in the streaming model and takes only a single pass over the data, selecting those elements whose marginal contribution is above some current threshold. However, it employs an *adaptive* thresholding scheme, where the threshold is chosen dynamically based on the objective value obtained until a certain point in the data stream. This additional power allows us to prove the following guarantee:

Theorem 1.3. *[Main Theorem] There exists a constant $\alpha > 0.5$ such that, for any stream of elements that arrive in random order, the value of the solution returned by* SALSA *is at least $\alpha \cdot \mathrm{OPT}$ in expectation (where OPT is the value of the optimum solution).* SALSA *uses $O(k \log k)$ memory (independent of the length of the stream) and processes each element using $O(\log k)$ evaluations of the objective function.*

We remark that even if the stream is adversarial-order, SALSA still guarantees a $(0.5 - \varepsilon)$-approximation.

A different way to improve the accuracy of an algorithm is allowing it to make multiple passes over the stream. In this paper we also consider this setting and present a 2-pass algorithm TWO-PASS for streaming submodular maximization. We show that TWO-PASS achieves a $5/9$ approximation ratio using the same order of memory and function evaluations as SIEVE-STREAMING. Formally, for any $\varepsilon > 0$ we show that:

Theorem 1.4. TWO-PASS *is a $(5/9 - \varepsilon)$-approximation for streaming submodular maximization. It uses $O\left(k \log k/\varepsilon\right)$ memory and processes each element with $O\left(\log k/\varepsilon\right)$ evaluations of the objective function.*

Furthermore, we generalize our ideas to design a p-pass algorithm P-PASS for any constant $p \geq 2$. McGregor and Vu (2016) showed that, regardless of the running time, no p-pass algorithm can beat the $(1 - 1/e)$ approximation guarantee using memory $\mathrm{poly}(k)$. In this work we show that P-PASS quickly converges to a $(1 - 1/e)$-approximation as p grows. We show that:

Theorem 1.5. P-PASS *is a $\left(1 - \left(p/p+1\right)^p - \varepsilon\right)$-approximation for streaming submodular maximization. It uses $O\left(k \log k/\varepsilon\right)$ memory and processes each element with $O\left(p \log k/\varepsilon\right)$ evaluations of the objective function.*

Applications and experiments We assess the accuracy of our algorithms and show their versatility in several real-world scenarios. In particular, we study maximum coverage in graphs, exemplar-based clustering, and personalized movie recommendation. We find that SALSA significantly outperforms the state of the art, SIEVE-STREAMING, in all tested datasets. In fact, our experimental results show that SALSA reduces the gap between GREEDY, which is the benchmark algorithm even for the offline setting (a "tractable optimum"), and the best known streaming algorithm by a factor of two on average.

Note that we are able to obtain these practical improvements even though, in our experiments, the order of arrival of elements is *not* manually randomized. This suggests that the random-order assumption, which allows us to obtain our improved theoretical guarantees, does well in approximating the nature of real-world datasets, which are not stochastic but also not adversarial.

Related work The benchmark algorithm for monotone submodular maximization under a cardinality constraint is GREEDY. Unfortunately, it is not efficient and requires k passes over the entire dataset. There has thus been much interest in obtaining more efficient versions of GREEDY, such as LAZY-GREEDY (Minoux, 1978; Leskovec et al., 2007; Krause et al., 2008), the algorithm of Badanidiyuru and Vondrk (2014), or STOCHASTIC-GREEDY (Mirzasoleiman et al., 2015).

The first *multi-pass* algorithm for streaming submodular maximization has been given by Gomes and Krause (2010). If f is upper-bounded by B, then for any $\varepsilon > 0$ their algorithm attains the value $0.5 \cdot \mathrm{OPT} - k\varepsilon$ and uses $O(k)$ memory while making $O(B/\varepsilon)$ passes. Interestingly, it converges to the optimal solution for a restricted class of submodular functions.

Many different settings are considered under the streaming model. One important requirement often arising in practice is that the returned solution be robust against deletions (Mirzasoleiman et al., 2017; Mitrovi et al., 2017; Kazemi et al., 2017). Non-monotone submodular functions have also been considered (Chekuri et al., 2015; Mirzasoleiman et al., 2017).

McGregor and Vu (2016) consider the k-coverage problem in the multi-pass streaming setting. They give an algorithm achieving $(1 - 1/e - \varepsilon)$-approximation in $O(1/\varepsilon)$ passes. They also show that improving upon the ratio $(1 - 1/e)$ in a constant number of passes would require an almost linear memory. Their results generalize from k-coverage to submodular maximization.

In the *online* setting, the stream length n is unknown and the algorithm must always maintain a feasible solution. This model allows preemption, i.e., the removal of previous elements from the solution (otherwise no constant competitive ratio is possible). Chakrabarti and Kale (2014), Chekuri et al. (2015) and Buchbinder et al. (2015) have obtained 0.25-competitive algorithms for monotone submodular functions under a cardinality constraint. This competitive ratio was later improved to 0.29 by Chan et al. (2017). Buchbinder et al. (2015) also prove a hardness of 0.5.

A different large-scale scenario is the *distributed* one, where the elements are partitioned across m machines. The algorithm GREEDI (Mirzasoleiman et al., 2013) consists in running GREEDY on each machine and then combining the resulting summaries on a single machine using another run of GREEDY. This yields an $O\left(1/\min(\sqrt{k}, m)\right)$-approximation. Barbosa et al. (2015) showed that when the elements are partitioned *randomly*, one obtains a $(1 - 1/e)/2$-approximation. Mirrokni and Zadimoghaddam (2015) provide a different two-round strategy: they compute *coresets* of size $O(k)$ and then greedily merge them, yielding a 0.545-approximation. The algorithm of Kumar et al. (2015) consists of a logarithmic number of rounds in the MapReduce setting and approaches the GREEDY ratio. Barbosa et al. (2016) provide a general reduction that implies a $(1 - 1/e - \varepsilon)$-approximation in $O(1/\varepsilon)$ rounds.

Korula et al. (2015) study the Submodular Welfare Maximization problem – where a set of items needs to be partitioned among agents in order to maximize social welfare, i.e., the sum of the (monotone submodular) utility functions of the agents – in the online setting. The best possible competitive ratio in general is 0.5. However, they show that the greedy algorithm is 0.505-competitive if elements arrive in random order.

2. Preliminaries

We consider a (potentially large) collection V of n items, also called the *ground set*. We study the problem of maximizing a *non-negative monotone submodular function* $f : 2^V \to \mathbb{R}_+$. Given two sets $X, Y \subseteq V$, the *marginal gain* of X with respect to Y is defined as

$$f(X|Y) = f(X \cup Y) - f(Y),$$

which quantifies the increase in value when adding X to Y. We say that f is *monotone* if for any element $e \in V$ and any set $Y \subseteq V$ it holds that $f(e|Y) \geq 0$. The function f is *submodular* if for any two sets X and Y such that $X \subseteq Y \subseteq V$ and any element $e \in V \setminus Y$ we have

$$f(e|X) \geq f(e|Y).$$

Throughout the paper, we assume that f is given in terms of a value oracle that computes $f(S)$ for given $S \subseteq V$. We also assume that f is *normalized*, i.e. $f(\emptyset) = 0$.

SUBMODULARITY UNDER CARDINALITY CONSTRAINT

The problem of maximizing function f under *cardinality constraint* k is defined as selecting a set $S \subseteq V$ with $|S| \leq k$ so as to maximize $f(S)$. We will use $\mathcal{O}$ to refer to such a set S, OPT to denote $f(\mathcal{O})$, and the name SUBMAX to refer to this problem.

3. Overview of SALSA

In this section, we present an overview of our algorithm. We also explain the main ideas and the key ingredients of its analysis. In the full version of this paper, we combine these ideas into a proof of Theorem 1.3. Throughout this section, we assume that the value OPT of an optimal solution $\mathcal{O} = \{o_1, ..., o_k\}$ is known in advance. We show how to remove this assumption using standard techniques in the full version.

We start by defining the notion of *dense* optimum solutions. We say that $\mathcal{O}$ is dense if there exists a set $D \subseteq \mathcal{O}$ of size at most $k/100$ such that $f(D) \geq OPT/10$.[3] Our algorithm runs three procedures, and each procedure outputs a set of at most k elements. One of the procedures performs well in the case when $\mathcal{O}$ is dense. The other two approaches are designed to collect high utility when $\mathcal{O}$ is not dense. We run these procedures in parallel and, out of the three returned sets, we report the one attaining the highest utility. In what follows, we first describe our algorithm for the case when $\mathcal{O}$ is not dense.

Case: $\mathcal{O}$ is not dense. We present the intuition behind the algorithm under the simplifying assumption that $f(o) = \text{OPT}/k$ for every $o \in \mathcal{O}$. However, the algorithm that we state provides an approximation guarantee better than 0.5 in expectation for *any* instance that is not dense.

Over the first 0.1-fraction of the stream, both procedures for this case behave identically: they maintain a set of elements S; initially, $S = \emptyset$; each element e from the stream is added to S if its marginal gain is at least $T_1 = \frac{\text{OPT}}{k}(1/2 + \epsilon)$, i.e.,

$$f(e|S) \geq \frac{\text{OPT}}{k}(1/2 + \epsilon).$$

Consider the first element $o \in \mathcal{O}$ that the procedures encounter on the stream. Since the stream is randomly ordered, o is a random element of $\mathcal{O}$. Due to this, we claim that if $f(S)$ is small, then it is likely that the procedures add o to S. This follows from the fact that each element of $\mathcal{O}$ is worth OPT/k. Namely, if $f(S) < \text{OPT}(1/2 - \epsilon')$, for a small constant $\epsilon' > 0$, then the average marginal contribution of the elements of $\mathcal{O}$ with respect to S is more than T_1, hence it is likely that the procedures select o. By repeating the same argument we can conclude that after processing a 0.1-fraction of the stream, either: (1) $f(S)$ is large, i.e., $f(S) > \text{OPT}(1/2 - \epsilon')$; or (2) the procedures have selected $k/100$ elements from $\mathcal{O}$ (which are worth $\text{OPT}/100$).

Up to this point, both procedures for the non-dense case behaved identically. In the remaining 0.9-fraction of the stream, the procedure corresponding to case (1) above uses a threshold $\frac{\text{OPT}}{k}(1/2 - \delta)$, which is lower than T_1. Since there are still $0.9n$ elements left on the stream, and already after the first 0.1-fraction we have $f(S) > \text{OPT}(1/2 - \epsilon')$, it is very likely that by the end the procedure will have added enough further elements to S so that $f(S) \geq \text{OPT}(1/2 + \epsilon)$.

In case (2) above, the procedure has already selected a set S that contains at least $k/100$ elements from $\mathcal{O}$, i.e., $|S \cap \mathcal{O}| \geq k/100$. Now, the procedure corresponding to this case continues with the threshold $T_1 = \frac{\text{OPT}}{k}(1/2 + \epsilon)$. If by the end of the stream the procedure has selected k elements, then clearly $f(S) \geq \text{OPT}(1/2 + \epsilon)$, since each element has marginal gain at least T_1. Otherwise, the procedure has selected fewer than k elements. This means that the marginal gain of any element of the stream with respect to S is less than T_1. Now we claim that $f(S) > \text{OPT}/2$. First, there are at most $99k/100$ elements in $\mathcal{O} \setminus S$. Furthermore, adding each such element to the set S gives marginal gain less than T_1. Therefore, the total benefit that the elements of $\mathcal{O} \setminus S$ give to S is at most $\frac{\text{OPT}}{k}(1/2 + \epsilon) \cdot 99k/100$, which is less than $\text{OPT}(1/2 - 1/300)$ for small enough ϵ, therefore

$$\text{OPT} \leq f(S \cup \mathcal{O}) = f(S) + f(\mathcal{O}|S)$$

and thus

$$\begin{aligned} f(S) &> \text{OPT} - \text{OPT}(1/2 - 1/300) \\ &= \text{OPT}(1/2 + 1/300). \end{aligned}$$

Case: $\mathcal{O}$ is dense. We now give a brief overview of the procedure that is designed for the case when $\mathcal{O}$ is dense. Over the first 0.8-fraction of the stream, the procedure uses a (high) threshold $T_1' = \frac{\text{OPT}}{k} \cdot 2$. Let $D \subseteq \mathcal{O}$ be the dense part of $\mathcal{O}$. Note that the average value of the elements of D is at least $\frac{\text{OPT}}{k} \cdot 10$, which is significantly higher than the threshold T_1'.

Hence, even over the 0.8-fraction of the stream, the algorithm will in expectation collect some elements with large marginal gain. This, intuitively, means that the algorithm in expectation selects k' elements of total value significantly larger than $k'\text{OPT}/(2k)$. This enables us to select the remaining $k - k'$ elements with marginal gain below $\text{OPT}/(2k)$ and still collect a set of utility larger than $\text{OPT}/2$. We implement this observation by letting the algorithm use a threshold lower than $\text{OPT}/(2k)$ for the remaining 0.2-fraction of the stream. This increases the chance that the algorithm collects $k - k'$ more elements.

[3] In the full version, we slightly alter the constants in the definition of a dense optimal solution.

In what follows, we provide pseudo-codes of our three algorithms. For sake of brevity, we fix the values of constants and give the full analysis of the algorithms in the full version of this paper.

We begin with the dense case, presented in Algorithm 1. In the pseudo-code, C_1, C_2 are large absolute constants and β is the fraction of the stream that we process with a high threshold.

Algorithm 1 DENSE

$S := \emptyset$
for the i-th element e_i on the stream **do**
  if $i \leq \beta n$ and $f(e_i|S) \geq \frac{C_1}{k}\text{OPT}$ and $|S| < k$ **then**
    $S := S \cup \{e_i\}$
  else if $i > \beta n$ and $f(e_i|S) \geq \frac{1}{C_2 \cdot k}\text{OPT}$ and $|S| < k$ **then**
    $S := S \cup \{e_i\}$
return S

For the case when $\mathcal{O}$ is not dense, we use two algorithms as described above. The first algorithm (Algorithm 2) goes over the stream and selects any element whose marginal gain to the currently selected elements is at least $\frac{\text{OPT}}{k}(1/2 + \epsilon)$. The second algorithm (Algorithm 3) starts with the same threshold, but after passing over βn elements it decreases the threshold to $\frac{\text{OPT}}{k}(1/2 - \delta)$.

Algorithm 2 FIXED THRESHOLD

$S := \emptyset$
for the i-th element e_i on the stream **do**
  if $f(e_i|S) \geq \frac{\text{OPT}}{k}(1/2 + \epsilon)$ and $|S| < k$ **then**
    $S := S \cup \{e_i\}$
return S

Algorithm 3 HIGH-LOW THRESHOLD

$S := \emptyset$
for the i-th element e_i on the stream **do**
  if $i \leq \beta n$ and $f(e_i|S) \geq \frac{\text{OPT}}{k}(1/2 + \epsilon)$ and $|S| < k$ **then**
    $S := S \cup \{e_i\}$
  else if $i > \beta n$ and $f(e_i|S) \geq \frac{\text{OPT}}{k}(1/2 - \delta)$ and $|S| < k$ **then**
    $S := S \cup \{e_i\}$
return S

Since we do not know in advance whether the input is dense or not, we run these three algorithms in parallel and output the best solution at the end.

4. TWO-PASS Algorithm

In this section, we describe our TWO-PASS algorithm. Recall that we denote the optimum solution by $\mathcal{O} = \{o_1, \ldots, o_k\}$ and we let $\text{OPT} = f(\mathcal{O})$. Throughout this section, we assume that OPT is known. We show how to remove this assumption in the full version of this paper. Also, in the full version we present a (more general) p-pass algorithm.

Our TWO-PASS algorithm (Algorithm 4) is simple: in the first pass we pick any element whose marginal gain with respect to the currently picked elements is higher than the threshold $T_1 = \frac{2}{3} \cdot \frac{\text{OPT}}{k}$. In the second pass we do the same using the threshold $T_2 = \frac{4}{9} \cdot \frac{\text{OPT}}{k}$.

Algorithm 4 TWO-PASS Algorithm

$S := \emptyset$
for the i-th element e_i on the stream **do**
  if $f(e_i|S) \geq \frac{2\text{OPT}}{3k}$ and $|S| < k$ **then**
    $S := S \cup \{e_i\}$
for the i-th element e_i on the stream **do**
  if $f(e_i|S) \geq \frac{4\text{OPT}}{9k}$ and $|S| < k$ **then**
    $S := S \cup \{e_i\}$
return S

Theorem 4.1. TWO-PASS *is a* $5/9$*-approximation for* SUBMAX.

Proof. We prove this theorem in two cases depending on $|S|$. First we consider the case $|S| < k$. For any element $o \in \mathcal{O} \setminus S$ we have $f(o|S_i) \leq T_2$ since we have not picked it in the second pass. Therefore

$$f(\mathcal{O}|S) \leq \sum_{o \in \mathcal{O}} f(o|S) \leq k \cdot T_2 = 4/9 \cdot \text{OPT}.$$

Thus

$$\text{OPT} \leq f(S \cup \mathcal{O}) = f(S) + f(\mathcal{O}|S)$$

and so

$$f(S) \geq \text{OPT} \cdot (1 - 4/9) = 5/9 \cdot \text{OPT}.$$

Therefore in this case we get the desired approximation ratio.

Now we consider the second case, i.e., $|S| = k$. It is clear that if we have picked k elements in the first round, then we get a $2/3$-approximation guarantee. Therefore assume that we picked fewer than k elements in the first round, and let S_1 denote these elements. With a similar argument as in the previous case we get that $f(S_1) \geq \text{OPT}/3$. One can see that in the worst-case scenario, in the first pass we have picked $k/2$ elements with marginal gain exactly T_1 each and in the second pass we have picked $k/2$ elements with

marginal gain exactly T_2 each (we present a formal proof of this statement in the full version). Therefore we have:

$$\begin{aligned} f(S) &\geq k/2 \cdot T_1 + k/2 \cdot T_2 \\ &= k/2 \cdot \frac{2}{3} \cdot \frac{\text{OPT}}{k} + k/2 \cdot \frac{4}{9} \cdot \frac{\text{OPT}}{k} \\ &\geq \frac{\text{OPT}}{2} \cdot (2/3 + 4/9) \\ &\geq 5/9 \cdot \text{OPT}\,. \end{aligned}$$

□

5. Empirical Evaluation

In this section, we numerically validate our theoretical findings. Namely, we compare our algorithms, SALSA and TWO-PASS, with two baselines, GREEDY and SIEVE-STREAMING. For this purpose, we consider three applications: (i) dominating sets on graphs, (ii) exemplar-based clustering, and (iii) personalized movie recommendation. In each of the experiments we find that SALSA outperforms SIEVE-STREAMING.

It is natural to consider the utility obtained by GREEDY as a proxy for an optimum, as it is theoretically tight and difficult to beat in practice. The majority of our evaluations demonstrate that the gap between the solutions constructed by SALSA and GREEDY is more than two times smaller than the gap between the solutions constructed by SIEVE-STREAMING and GREEDY.

For each of the experiments we invoke our algorithms with the following values of the parameters: Algorithm 1 with $C_1 = 10$, $C_2 = 0.2$, $\beta = 0.8$; Algorithm 2 with $\varepsilon = 1/6$; Algorithm 3 with $\beta = 0.1$, $\varepsilon = 0.05$, $\delta = 0.025$.

5.1. Maximum coverage in big graphs

Maximum coverage is a classic graph theory problem with many practical applications, including influence maximization in social networks (Kempe et al., 2015) and community detection in graphs (Fortunato & Lancichinetti, 2009). The goal in this problem is to find a small subset of vertices of a graph that is connected to a large fraction of the vertices.

Maximum coverage can be cast as maximization of a submodular function subject to a cardinality constraint. More formally, we are given a graph $G = (V, E)$, where $n = |V|$ denotes the number of vertices and $m = |E|$ denotes the number of edges. The goal is to find a set $S \subseteq V$ of size k that maximizes the number of vertices in the neighborhood of S.[4] We consider three graphs for this problem from the SNAP data library (Leskovec & Krevl, 2014).

Pokec social network Pokec is the most popular online social network in Slovakia. This graph has $n = 1,632,803$ and $m = 30,622,564$.

LiveJournal social network LiveJournal (Backstrom et al., 2006) is a free online community that enables members to maintain journals and individual and/or group blogs. This graph has $n = 4,847,571$ and $m = 68,993,773$.

Orkut social network Similar to Pokec, Orkut (Yang & Leskovec, 2015) is also an online social network. This graph has $n = 3,072,441$ vertices and $m = 117,185,083$ edges.

We compare our algorithms, SALSA and TWO-PASS, with both baselines on these datasets for different values of k – from 100 to 10,000. The results show that SALSA always outperforms SIEVE-STREAMING by around 10%, and also reduces the gap between GREEDY and the best streaming algorithm by a factor of two. Furthermore, the performance of our TWO-PASS algorithm is very close to that of GREEDY. The results can be found in Figure 1, where (a) and (b) correspond to the Orkut dataset, (c) and (d) correspond to LiveJournal, and (e) to Pokec.

5.2. Exemplar-based clustering

Imagine that we are given a collection of emails labeled as spam or non-spam and asked to design a spam classifier. In addition, every email is equipped with an m-dimensional vector corresponding to the features of that email. One possible approach is to view these m-dimensional vectors as points in the Euclidean space, decompose them into k clusters and fix a representative point for each cluster. Then, whenever a new email arrives, it is assigned the same label as the cluster representative closest to it. Let V denote the set of all the labeled emails. To obtain the described set of cluster representatives, we maximize the following submodular function:

$$f(S) = L(\{e_0\}) - L(S \cup \{e_0\}),$$

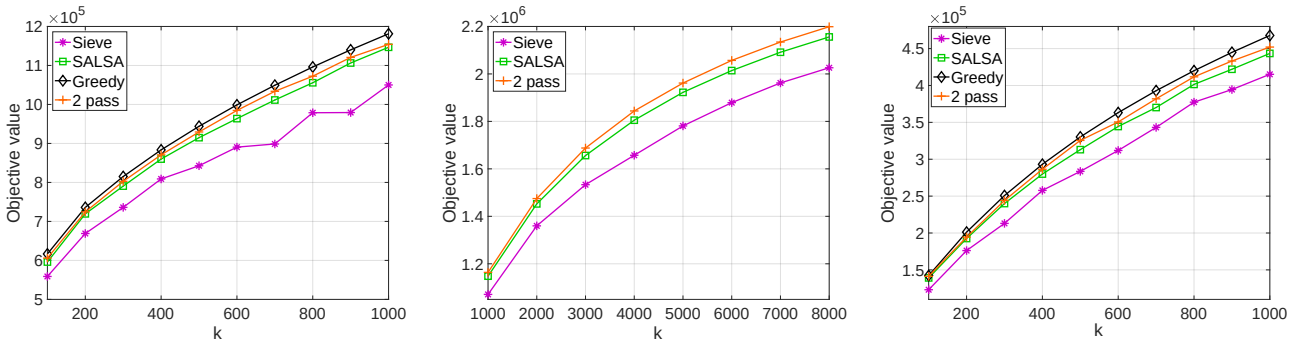
where e_0 is the all-zero vector, and $L(S)$ is defined as follows (Gomes & Krause, 2010):

$$L(S) = \frac{1}{|V|} \sum_{e \in V} \min_{v \in S} d(e, v).$$

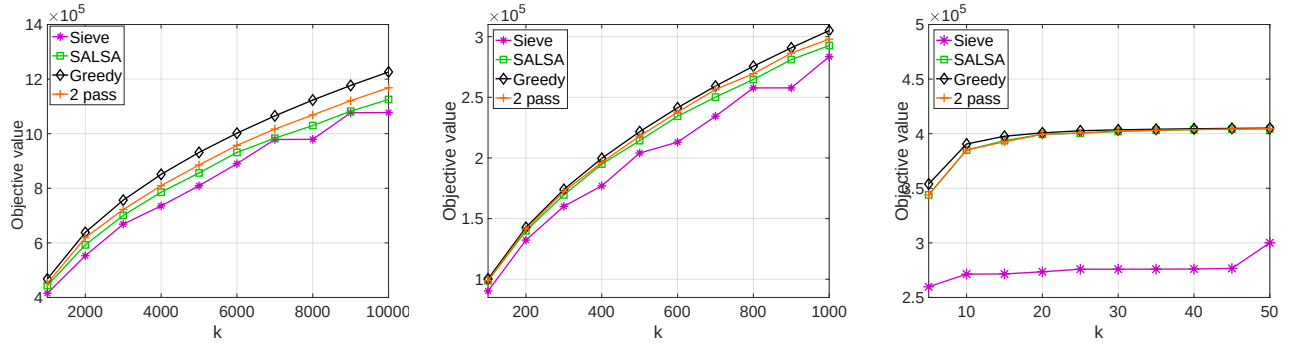
In the definition of the function $L(S)$, $d(x, y) = \|x - y\|^2$ denotes the squared Euclidean distance.[5]

[4] This problem has been also referred to as the dominating set problem in the literature.

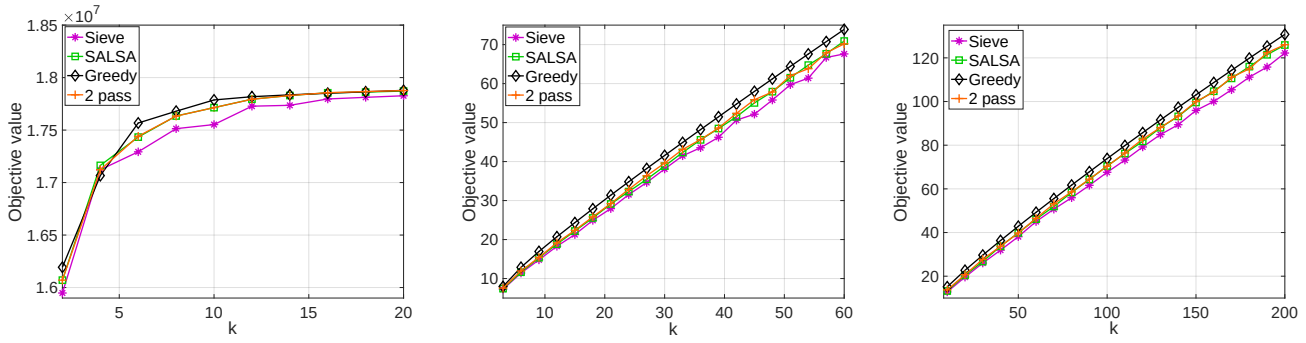
[5] Notice that we turn a minimization problem over $L(S)$ into a maximization problem over $f(S)$. The approximation guarantee for maximizing $f(S)$ does not transfer to an approximation guarantee for minimizing $L(S)$. Nevertheless, maximizing $f(S)$ gives very good practical performance, and hence we use it in place of $L(S)$.



(a) Orkut (b) Orkut (c) LiveJournal



(d) LiveJournal (e) Pokec (f) Spambase



(g) CIFAR-10 (h) Movies, $\alpha = 0.75$ (i) Movies, $\alpha = 0.85$

Figure 1: Numerical comparisons of our two algorithms (SALSA and TWO-PASS) and baselines (GREEDY and SIEVE-STREAMING). In plot (b) we could not run GREEDY on the underlying dataset due to its prohibitively slow running time on this instance. Each plot demonstrates the performance of the algorithms for varying values of the cardinality k. The datasets used for plots (a)-(e) are described in Section 5.1, for plots (f) and (g) in Section 5.2, and for plots (h) and (i) in Section 5.3.

Similarly to spam classification, and among many other applications, the exemplar submodular function can also be used for image clustering. In light of these applications, we perform experiments on two datasets:

Spambase This dataset consists of $4,601$ emails, each email described by 57 attributes (Lichman, 2013). We do not consider mail-label as one of the attributes.

CIFAR-10 This dataset consists of $50,000$ color images, each of size 32×32, divided into 10 classes. Each image is represented as a $3,072$-dimensional vector – three coordinates corresponding to the red, green and blue channels of each pixel (Krizhevsky et al., 2014).

Before running these experiments, we subtract the mean of the corresponding dataset from each data point.

The results for the Spambase dataset are shown in Figure 1(f). We can observe that both of our algorithms attain a significantly higher utility than SIEVE-STREAMING. Also, at their point of saturation, our algorithms equalize with GREEDY. We can also observe that SIEVE-STREAMING saturates at a much lower value than our algorithms, which suggests that the strategy we develop filters elements from the stream more carefully than SIEVE-STREAMING does.

Our results for the CIFAR-10 dataset, depicted in Figure 1(g), show that, before the point of saturation our algorithms select elements of around 5% higher utility than SIEVE-STREAMING. After the point of saturation our algorithms achieve the same utility as GREEDY, while SIEVE-STREAMING approaches that value slowly. Saturation happens around $k = 10$, which is expected since the images in CIFAR-10 are decomposed into 10 classes.

5.3. Personalized movie recommendation

We use the Movielens 1M dataset (Harper & Konstan, 2016) to build a recommender system for movies. The dataset contains over a million ratings for 3,900 movies by 6,040 users. For a given user u and a number k, the system should recommend a collection of k movies personalized for user u.

We use the scoring function proposed by Mitrovi et al. (2017). We first compute low-rank feature vectors $w_u \in \mathbb{R}^{20}$ for each user u and $v_m \in \mathbb{R}^{20}$ for each movie m. These are obtained via low-rank matrix completion (Troyanskaya et al., 2001) so as to make each inner product $\langle w_u, v_m \rangle$ approximate the rating of m by u, if known. Now we define the submodular function

$$f_{u,\alpha}(S) = \alpha \cdot \sum_{m' \in M} \max_{m \in S} \langle v_{m'}, v_m \rangle + (1-\alpha) \cdot \sum_{m \in S} \langle w_u, v_m \rangle .$$

The first term is a facility-location objective (Lindgren et al., 2016) that measures how well S covers the space M of all movies (thus promoting diversity). The second term aggregates the user-dependent scores of items in S. The parameter α can be adjusted depending on the user's preferences.

Our experiments consist in recommending collections of movies for $\alpha = 0.75$ and values of k up to 60 (see Figure 1(h)), as well as for $\alpha = 0.85$ and values of k up to 200 (see Figure 1(i)). We do this for 8 randomly selected users and report the averages. We find that the performance of both SALSA and TWO-PASS falls at around 40% of the gap between SIEVE-STREAMING and GREEDY. This quantity improves as k increases.

6. Conclusion

In this paper, we consider the monotone submodular maximization problem subject to a cardinality constraint. For the case of adversarial-order streams, we show that a $1/2$ approximation guarantee is tight. Motivated by real-world applications, we also study this problem in random-order streams. We show that the previously known techniques are not sufficient to improve upon $1/2$ even in this setting. We design a novel approach that exploits randomness of the stream and achieves a better-than-$1/2$ approximation guarantee. We also present a multi-pass algorithm that approaches $(1 - 1/e)$-approximation using only a constant number of passes, even in adversarial-order streams. We validate the performance of our algorithm on real-world data. Our evaluations demonstrate that we outperform the state of the art SIEVE-STREAMING algorithm by a considerable margin. In fact, our results are closer to GREEDY than to SIEVE-STREAMING. Although we make a substantial progress in the context of streaming submodular maximization, there is still a gap between our approximation guarantee and the currently best known lower bound. It would be very interesting to reduce (or close) this gap, and we hope that our techniques will provide insight in this direction.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback. Ola Svensson and Jakub Tarnawski were supported by ERC Starting Grant 335288-OptApprox.

References

Agrawal, R., Gollapudi, S., Halverson, A., and Ieong, S. Diversifying search results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, WSDM '09, pp. 5–14, New York, NY, USA, 2009. ACM.

Backstrom, L., Huttenlocher, D., Kleinberg, J., and Lan, X. Group formation in large social networks: Membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pp. 44–54, New York, NY, USA, 2006. ACM.

Badanidiyuru, A. and Vondrák, J. Fast algorithms for maximizing submodular functions. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pp. 1497–1514, Philadelphia, PA, USA, 2014. Society for Industrial and Applied Mathematics.

Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., and Krause, A. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the*

20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, pp. 671–680, New York, NY, USA, 2014. ACM.

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*, pp. 1236–1244, 2015.

Barbosa, R. D. P., Ene, A., Nguyen, H. L., and Ward, J. A new framework for distributed submodular maximization. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 645–654, Oct 2016. doi: 10.1109/FOCS.2016.74.

Buchbinder, N., Feldman, M., and Schwartz, R. Online submodular maximization with preemption. In *Proceedings of the Twenty-sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, pp. 1202–1216, Philadelphia, PA, USA, 2015. Society for Industrial and Applied Mathematics.

Chakrabarti, A. and Kale, S. Submodular maximization meets streaming: Matchings, matroids, and more. In Lee, J. and Vygen, J. (eds.), *Integer Programming and Combinatorial Optimization*, pp. 210–221, Cham, 2014. Springer International Publishing.

Chan, T.-H. H., Huang, Z., Jiang, S. H.-C., Kang, N., and Tang, Z. G. Online submodular maximization with free disposal: Randomization beats 1/4 for partition matroids. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pp. 1204–1223, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics.

Chekuri, C., Gupta, S., and Quanrud, K. Streaming algorithms for submodular function maximization. In Halldórsson, M. M., Iwama, K., Kobayashi, N., and Speckmann, B. (eds.), *Automata, Languages, and Programming*, pp. 318–330, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

El-Arini, K. and Guestrin, C. Beyond keyword search: Discovering relevant scientific literature. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pp. 439–447, New York, NY, USA, 2011. ACM.

Feige, U. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

Fortunato, S. and Lancichinetti, A. Community detection algorithms: a comparative analysis: invited presentation, extended abstract. In *4th International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS '09, Pisa, Italy, October 20-22, 2009*, pp. 27, 2009.

Golovin, D. and Krause, A. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *J. Artif. Int. Res.*, 42(1):427–486, September 2011. ISSN 1076-9757.

Gomes, R. and Krause, A. Budgeted nonparametric learning from data streams. In *In Proc. International Conference on Machine Learning (ICML*, 2010.

Harper, F. M. and Konstan, J. A. The MovieLens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19, 2016.

Kazemi, E., Zadimoghaddam, M., and Karbasi, A. Deletion-Robust Submodular Maximization at Scale. *ArXiv e-prints*, November 2017.

Kempe, D., Kleinberg, J., and Tardos, E. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pp. 137–146, New York, NY, USA, 2003. ACM.

Kempe, D., Kleinberg, J. M., and Tardos, É. Maximizing the spread of influence through a social network. *Theory of Computing*, 11:105–147, 2015.

Korula, N., Mirrokni, V., and Zadimoghaddam, M. Online submodular welfare maximization: Greedy beats 1/2 in random order. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pp. 889–898, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3536-2. doi: 10.1145/2746539.2746626. URL http://doi.acm.org/10.1145/2746539.2746626.

Krause, A., Singh, A., and Guestrin, C. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *J. Mach. Learn. Res.*, 9:235–284, June 2008.

Krizhevsky, A., Nair, V., and Hinton, G. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

Kumar, R., Moseley, B., Vassilvitskii, S., and Vattani, A. Fast greedy algorithms in mapreduce and streaming. *ACM Trans. Parallel Comput.*, 2(3):14:1–14:22, September 2015.

Leskovec, J. and Krevl, A. SNAP Datasets: Stanford large network dataset collection, June 2014.

Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., and Glance, N. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pp. 420–429, New York, NY, USA, 2007. ACM.

Lichman, M. UCI machine learning repository, 2013.

Lindgren, E., Wu, S., and Dimakis, A. G. Leveraging sparsity for efficient submodular data summarization. In *Advances in Neural Information Processing Systems*, pp. 3414–3422, 2016.

McGregor, A. and Vu, H. T. Better streaming algorithms for the maximum coverage problem. *arXiv preprint arXiv:1610.06199*, 2016.

Minoux, M. Accelerated greedy algorithms for maximizing submodular set functions. In Stoer, J. (ed.), *Optimization Techniques*, pp. 234–243, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg.

Mirrokni, V. and Zadimoghaddam, M. Randomized composable core-sets for distributed submodular maximization. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pp. 153–162, New York, NY, USA, 2015. ACM.

Mirzasoleiman, B., Karbasi, A., Sarkar, R., and Krause, A. Distributed submodular maximization: Identifying representative elements in massive data. In *Advances in Neural Information Processing Systems*, pp. 2049–2057, 2013.

Mirzasoleiman, B., Badanidiyuru, A., Karbasi, A., Vondrák, J., and Krause, A. Lazier than lazy greedy. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pp. 1812–1818. AAAI Press, 2015.

Mirzasoleiman, B., Jegelka, S., and Krause, A. Streaming Non-monotone Submodular Maximization: Personalized Video Summarization on the Fly. *ArXiv e-prints*, June 2017.

Mirzasoleiman, B., Karbasi, A., and Krause, A. Deletion-robust submodular maximization: Data summarization with "the right to be forgotten". In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2449–2458, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

Mitrovi, S., Bogunovi, I., Norouzi-Fard, A., Tarnawski, J., and Cevher, V. Streaming robust submodular maximization: A partitioned thresholding approach. In *Advances in Neural Information Processing Systems*, 2017.

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

Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., and Altman, R. B. Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17(6):520–525, 2001.

Yang, J. and Leskovec, J. Defining and evaluating network communities based on ground-truth. *Knowl. Inf. Syst.*, 42 (1):181–213, 2015.