Adversarially Robust Submodular Maximization under Knapsack Constraints*

Dmitrii Avdiukhin
davdyukh@iu.edu
Indiana University

Slobodan Mitrović†
slobo@mit.edu
MIT

Grigory Yaroslavtsev
gyarosla@iu.edu
Indiana University & The Alan Turing Institute

Samson Zhou
samsonzhou@gmail.com
Indiana University

ABSTRACT

We propose the first adversarially robust algorithm for monotone submodular maximization under single and multiple knapsack constraints with scalable implementations in distributed and streaming settings. For a single knapsack constraint, our algorithm outputs a robust summary of almost optimal (up to polylogarithmic factors) size, from which a constant-factor approximation to the optimal solution can be constructed. For multiple knapsack constraints, our approximation is within a constant-factor of the best known non-robust solution.

We evaluate the performance of our algorithms by comparison to natural robustifications of existing non-robust algorithms under two objectives: 1) dominating set for large social network graphs from Facebook and Twitter collected by the Stanford Network Analysis Project (SNAP), 2) movie recommendations on a dataset from MovieLens. Experimental results show that our algorithms give the best objective for a majority of the inputs and show strong performance even compared to offline algorithms that are given the set of removals in advance.

CCS CONCEPTS

• **Theory of computation → Streaming, sublinear and near linear time algorithms**; *Distributed algorithms.*

KEYWORDS

submodular maximization, streaming algorithms, distributed algorithms

*Full version is available at [3]

†Supported by the Swiss NSF grant P2ELP2_181772 and MIT-IBM Watson AI Lab.

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330911

ACM Reference Format:
Dmitrii Avdiukhin, Slobodan Mitrović, Grigory Yaroslavtsev, and Samson Zhou. 2019. Adversarially Robust Submodular Maximization under Knapsack Constraints. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and DataMining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3292500.3330911

1 INTRODUCTION

Submodular maximization has a wide range of applications in data science, machine learning and optimization, including data summarization, personalized recommendation, feature selection and clustering under various constraints, e.g. budget, diversity, fairness and privacy among others. *Constrained* submodular optimization has been studied since the seminal work of [33]. It has recently attracted a lot of interest in various large-scale computation settings, including distributed [12, 29], streaming [1, 4, 34], and adaptive [5, 6, 11, 14–16] due to its applications in recommendation systems [13, 25], exemplar based clustering [17], and document summarization [27, 36, 39]. For monotone functions, constrained submodular optimization has been studied extensively under numerous constraints such as cardinality [4, 7], knapsack [19], matchings [10], and matroids [9].

With the increase in volume of data, the task of designing low-memory streaming and low-communication distributed algorithms for monotone submodular maximization has received significant attention. A series of results [1, 4, 34] culminated in a single pass algorithm over random-order stream that achieves close to $(1 - 1/e)$-approximation of monotone submodular maximization under cardinality constraint. This approximation almost matches the guarantee of a celebrated result [33]. Also in the context of streaming, [19, 24, 40] studied monotone submodular maximization under d knapsack constraints, resulting in a single pass algorithm that provides $(1/(1 + 2d))$-approximation. Another line of work [12, 24, 29, 31] focused on submodular maximization in distributed setting. In particular, [29] developed 2-round and $(1/\epsilon)$-round MapReduce algorithms that provide $1/2$ and $1 - 1/e - \epsilon$ approximation, respectively, for monotone submodular maximization under cardinality constraint.

In this paper, we focus on the robust version of this classic problem [21, 22, 32, 35]. Consider a situation where a set of recommendations (or advertisements) is constructed for a new user. It is standard to model this as a monotone submodular maximization problem under knapsack constraints, which allow incorporation of various

restrictions on available budget, screen space, user preferences, privacy and fairness, etc. However, new users are likely to find some of the recommended items familiar, annoying or otherwise undesirable. Hence, it is advisable to build recommendations in such a way that even if the user later decides to dismiss some of the recommended items, one can quickly compute a new high-quality set of recommended items without solving the entire problem from scratch. We refer to this property as "adversarial robustness" since the removals are allowed to be completely arbitrary (e.g. might depend on the algorithm's suggestions).

1.1 Adversarially Robust Monotone Submodular Maximization

Let V be a finite domain consisting of elements $e_1, \ldots, e_{|V|}$. For a set function $f\colon 2^V \to \mathbb{R}^{\geq 0}$, we use $f(e \mid S)$ to denote the *marginal gain* of an element e given a set $S \subseteq V$, i.e., $f(e \mid S) = f(S \cup \{e\}) - f(S)$. A set function f is *submodular* if for every $S \subseteq T \subseteq V$ and every $e \in V$ it holds that $f(e \mid T) \leq f(e \mid S)$. A set function f is *monotone* if for every $S \subseteq T \subseteq V$ it holds that $f(T) \geq f(S)$. Intuitively, elements in the universe contribute non-negative utility, but have diminishing gains as the cost of the set increases.

For a set $S \subseteq V$ we use notation $\mathbf{x}_S$ to denote the 0-1 indicator vector of S. We use $\mathbf{C} \in \mathbb{R}^{d \times |V|}$ to denote a matrix with positive entries and $\mathbf{b} \in \mathbb{R}^d$ to denote a vector with positive entries. Here, $\mathbf{C}$ and b should be interpreted as knapsack constraints, where set S satisfies these constraints if and only if $\mathbf{C}\mathbf{x}_S \leq b$.

Problem 1.1 (MSM under knapsack constraints). *In the monotone submodular maximization (MSM) problem subject to d knapsack constraints, we are given a monotone submodular set function $f\colon 2^V \to \mathbb{R}^{\geq 0}$ and are required to output:*

$$\mathcal{OPT}(V) = \operatorname*{argmax}_{S \subseteq V\colon \boldsymbol{C}\boldsymbol{x}_S \leq \boldsymbol{b}} f(S).$$

Since the constraints are scaling-invariant, one can rescale each row $\mathbf{C}_i$ by multiplying it (and the corresponding entry in $\mathbf{b}$) by b_1/b_i so that all entries in $\mathbf{b}$ are the same and equal to b_1. One can further rescale $\mathbf{C}$ and $\mathbf{b}$ by the smallest entry in $\mathbf{C}$ (or some lower bound on it), so that $\min_{i,j} C_{i,j} \geq 1$. We assume such rescaling below and let $K = b_i$ for all i. In the case of one constraint ($d = 1$), we further simplify the notation and set $c(e_i) = C_{1,i}$ and $K = b_1$ and refer to $c(e_i)$ simply as the cost of the i-th item.

An important role in our algorithms is played by the *marginal density* of an item. Formally, for a set $S \subseteq V$, an element e and a cost function $c\colon V \to \mathbb{R}^{\geq 0}$ we define the marginal density of e with respect to S under the cost function c as: $\rho(e|S) = \frac{f(e \mid S)}{c(e)}$. For multiple dimensions, we will specifically define the cost function $c(\cdot)$.

Motivated by applications to personalized recommendation systems, we consider the *adversarially robust* version of the above problem. In the *adversarially robust monotone submodular maximization (ARMSM) problem* the goal is to produce a small "adversarially robust" summary $\mathcal{S} \subseteq V$. Here "adversarial robustness" means that for any set E of cardinality at most m, which might be later removed, one should be able to compute a good approximation for the residual monotone submodular maximization problem over $V \setminus E$ based only on $\mathcal{S}$. In this paper, we propose a study of ARMSM under knapsack constraints:

Problem 1.2 (ARMSM under knapsack constraints). *An algorithm $\mathcal{A}$ solves the adversarially robust monotone submodular maximization problem ARMSM(m, K) subject to d knapsack constraints if it produces a summary $\mathcal{S} \subseteq V$ such that:*

$$\mathcal{OPT}(V \setminus E) = \operatorname*{argmax}_{S \subseteq \mathcal{S} \setminus E\colon \boldsymbol{C}\boldsymbol{x}_S \leq \boldsymbol{b}} f(S)$$

for any set of removals E of cardinality at most m. $\mathcal{A}$ gives an α-approximation if there exists a set $Z \subseteq \mathcal{S}$ with $\boldsymbol{C}\boldsymbol{x}_Z \leq \boldsymbol{b}$ such that $f(Z) \geq \alpha f(\mathcal{OPT}(V \setminus E))$.

The main goal of an adversarially robust algorithm is to minimize the size of the resulting summary. We remark that the above robustness model is very strong. In particular, the set of removals E does not have to be fixed in advance and might depend on the summary $\mathcal{S}$ produced by the algorithm. Hence, we choose to refer to it as *adversarial robustness* in order to avoid confusion with other notions of robustness known in the literature [2, 37].

1.2 Our Theoretical Results

Streaming algorithms. We first consider the ARMSM problem in the streaming setting. A streaming algorithm is given the vector $\mathbf{b}$ of knapsack budget bounds upfront. Then, the elements of the ground set $e_1, \ldots, e_{|V|}$ arrive in an arbitrary order. When an element e_i arrives, the algorithm sees the corresponding column $\mathbf{C}_{*,i}$, which lists the d costs associated with this item. The algorithm only sees each element once and is required to use only a small amount of space throughout the stream. In the end of the stream, an adversarially chosen set of removals E is revealed and the goal is to solve ARMSM over $V \setminus E$. The key objective of the streaming algorithm is to minimize the amount of space used while providing a good approximation for ARMSM for any E.

Our first set of results gives adversarially robust algorithms for the ARMSM problem under one knapsack constraint:

Theorem 1.3 (ARMSM under one knapsack constraint). *For the ARMSM(m, K) problem under one knapsack constraint, there exists an algorithm that gives a constant-factor approximation with a summary consisting of $\tilde{O}(K + m)$ elements of the ground set (Theorem 3.1).*

We also show that if the total cost of removed items is at most M then there is an algorithm with summary size $\tilde{O}(K + M)$ and improved approximation guarantee. For ARMSM under a single knapsack constraint, our bounds are tight up to polylogarithmic factors, since an optimal solution may contain K items of unit cost, and an adversary can remove up to m items of any set. Hence, storing $\Omega(K + m)$ elements is necessary to obtain a constant factor approximation.

For the ARMSM problem under d knapsack constraints, we give an algorithm with the following guarantee:

Theorem 1.4 (ARMSM under d knapsack constraints). *For the ARMSM(m, K) problem under d knapsack constraints, there exists an algorithm that gives an $\Omega(\frac{1}{d})$-approximation with a summary of size $\tilde{O}(K + m)$ (Theorem 3.2).*

Distributed algorithms. We also consider the ARMSM problem in the distributed setting. Here, our aim is to collect a robust set $\mathcal{S}$ of elements while distributing the work to a number of machines, minimizing the memory requirement per machine and the number of rounds in which the machines need to communicate with each other. As in the case of streaming setting, a set of removals E is revealed only after $\mathcal{S}$ is constructed. We obtain a 2-round algorithm that matches our result for streaming, in terms of approximation guarantees.

Theorem 1.5 (Distributed ARMSM). *For the ARMSM(m, K) problem on a dataset of size n under d knapsack constraints, there exists an algorithm that gives an $\Omega(\frac{1}{d})$-approximation with a summary of size $\tilde{O}(K + m)$. If oracle access to f is given, this algorithm can be implemented in two distributed rounds using $\tilde{O}((m + K)\sqrt{n})$ words of space per machine (Theorem 4.1).*

1.3 Empirical Evaluations

We evaluate the performance of our algorithms on both single knapsack and multiple knapsack constraints by comparison to natural generalizations of existing algorithms. We implement the algorithms for the objective of dominating set for large social network graphs from Facebook and Twitter collected by the Stanford Network Analysis Project (SNAP), and for the objective of coverage on a large dataset from MovieLens. We compare the objectives on the sets output as well as the total number of elements collected by each algorithm.

Our results show that our algorithms provide the best objective for a majority of the inputs. In fact, our streaming algorithms perform just as well as the standard *offline* algorithms, even when the offline algorithms *know in advance* which elements will be removed. Our results also indicate that the number of elements collected by our algorithms does not appear to correlate with the total number of elements, which is an attractive property for streaming algorithms. In fact, most of the baseline algorithms collect relatively the same number of elements for the robust summary, ensuring fair comparison. For more details, see Section 5.

1.4 Previous Work

The special case of ARMSM(m, K) with one constraint and equal costs for all elements is referred to as robust submodular maximization under the cardinality constraint. If at most k elements can be selected, we refer to this problem as ARMSM(m, k). The study of this problem was initiated by Krause *et al.* [22]. The first (non-streaming) constant-factor approximation for this problem was given by Orlin *et al.* [35] for $m = o(\sqrt{k})$. This was further extended by [8] who give algorithms for $m = o(k)$. In these works, the size of the summary is restricted to contain at most k elements and hence by design only $m < k$ removals can be handled.

Recently the focus has shifted to handling larger numbers of removals and so there has been increased interest in studying ARMSM(m, k) with summary of sizes *greater* than k. [30] solve this problem with summary size $O(k \cdot m)$, which was improved by [32] to $\tilde{O}(m + k)$. Moreover, their algorithms are applicable to arbitrary ordered streams. A different setup was considered by [21], who assume that E is chosen *independently* of the choice of a robust summary and give algorithms with summary size $\tilde{O}(m + k)$, but obtain better approximation guarantees than [32].

To the best of our knowledge, there is little known about the general ARMSM(m, K) problem considered here which asks for robustness under single or multiple knapsack constraints.

2 TECHNIQUES

Our general approach is to find a set $\mathcal{S}$ at the end of the stream, so that when a set E of items is removed, we show that running an offline algorithm, Offline, on the set $Z := \mathcal{S} \setminus E$ produces a good approximation to the value of the optimal solution of the entire stream. Since Offline on input Z is known to produce a good approximation to the optimal solution of constrained submodular maximization on input Z (see Theorem 2.1), then it suffices to show that $f(\mathcal{OPT}(Z))$ is a good approximation to $f(\mathcal{OPT})$, where we use $\mathcal{OPT}$ to denote $\mathcal{OPT}(V \setminus E)$.

Theorem 2.1. *[23, 38] There exists an algorithm Offline that gives a $(1 - 1/e)$-approximation for the monotone submodular maximization problem subject to d knapsack constraints in polynomial time.*

We assume that we have a good guess for $f(\mathcal{OPT})$ by making a number of exponentially increasing guesses τ. Our algorithms start with the partitions-and-buckets approach from [8, 32] for robust submodular maximization under cardinality constraints. Specifically, our algorithms create a number of partitions and also create a number of buckets for each partition, where the number of buckets is chosen to be "robust" to the removal of items at the end of the stream. An element in the stream is added to the first possible bucket in which its marginal density exceeds a certain threshold, which is governed by the partition. The thresholds are exponentially decreasing across the partitions, so that the number of partitions is logarithmic in K.

At a high level, our algorithms overcome several potential pitfalls. The first challenge we face is the issue of buckets being populated with items of small cost whose marginal density surpasses the threshold. These small items prevent large items (such as cost K) whose marginal density also surpasses the threshold from being added to any bucket. If the optimal solution consists of a single large item, then the approximation guarantee could potentially be as bad as $\frac{1}{K}$. Thus, we allow each bucket *double* the capacity and create an additional partition level with a smaller threshold to compensate.

The second challenge we face is relating the items in various partitions. Although we would like to argue that an item e in a bucket in a certain partition i does not have overwhelmingly large marginal gain, the most natural way to prove this would be to claim that e would have been placed in a previous partition less than i because the ratio is overwhelmingly large. However, this is no longer true because items in partition i can have up to cost 2^i and any non-empty bucket in previous partitions does not have enough capacity. Surprisingly, for the purposes of analysis, it suffices to prohibit any item in a bucket from using more than a certain fraction of the capacity. That is, any item added to a bucket $B_{i,j}$, which has capacity 2^{i+1}, must have cost at most 2^{i-1}.

2.1 Robustness to the Removal of m Items

We now describe AlgNum, which outputs a solution of cost K on a single knapsack constraint and is robust against the removal of up to m items. We would like to use an averaging arguments to show that some "saturated" bucket $B_{i^*,j}$ in a partition cannot have too much intersection with the elements E that are removed at the end of the stream. However, the removal of up to m items at the end of the stream may cause the removal of cost up to mK. But then the averaging argument fails unless the number of buckets in each partition also increases by a factor of K, which unfortunately gives an additional multiple of K in the space of the algorithm.

Instead, the key idea is to *dynamically* allocate a number of new buckets, depending on the total cost of the current items in the buckets of a partition. The goal is to maintain enough buckets to guarantee that a certain number of elements can be added to a partition, regardless of their cost. Therefore, the number of total buckets is not large unless the stored items have large cost, in which case the number of items is relatively low anyway. To do this, we maintain counters s_i that allocate a new bucket to partition i each time they exceed $\min\{2^i, K\}$. Each time an item e is added to partition i, the counter s_i is increased proportional to the cost of the item, $c(e)$. The creation of new buckets is allowed until a certain number of items have been collected by the partition. Intuitively, algorithms robust to the removal of m items, such as AlgNum, should strive to output at the end of the stream a set with a certain number of items, whereas algorithms robust to the removal of items with a certain cost M should strive to output a set with a certain cost.

At the end, we run a procedure Prune to further bound the number of elements output by the algorithm. Prune simply reorders the elements stored by AlgNum by cost of the elements, and again runs AlgNum on the sorted set of elements as an input stream. Since the items with smaller cost arrive first, this ensures that we cannot have too many items of large cost.

3 STREAMING ALGORITHMS

We now warm-up by providing the first streaming algorithm for the ARMSM(m, K) problem under a single knapsack constraint. We later show how to build on these ideas to obtain robustness subject to multiple knapsack constraints.

3.1 Single Knapsack Constraint

We describe our algorithm AlgNum, which is used to produce a summary consisting of $\tilde{O}(K+m)$ items. Recall that we use $\mathcal{OPT}$ to denote $\mathcal{OPT}(V \setminus E)$. In order to simplify presentation, we assume[1] that we have a good estimate τ^* for $f(\mathcal{OPT})$, such that $\tau^* \leq f(\mathcal{OPT}) \leq (1+\epsilon)\tau^*$. To simplify presentation, we further assume that K is a power of two and hence let $K = 2^\ell$ (see Algorithm 1 for how rounding is handled).

AlgNum creates ℓ partitions $B_1, \ldots, B_\ell$ where the i-th partition initially consists of $n_i = O(\ell(\frac{m}{2^i} + 1))$ buckets of capacity 2^{i+1} each. We refer to the j-th bucket in the i-th partition as $B_{i,j}$. When processing the stream, each element e is added to the first possible bucket $B_{i,j}$ in the first possible partition i such that the bucket has enough capacity remaining and the marginal density $\rho(e|B_{i,j})$ exceeds a certain threshold $\tau/2^i$ for this partition. Note that the thresholds exponentially decrease across the partitions while capacities of the buckets exponentially increase.

Our goal is to maintain enough buckets to guarantee that a certain number of elements can be added to a partition, regardless of their cost. To dynamically allocate a number of new buckets, AlgNum keeps counters s_i that create a new bucket while they exceed 2^i, after which the value of the counter is lowered. The counter s_i is increased proportional to the cost of an item $c(e)$ each time an item e is added to partition i. This process continues until a certain number of items are in the partition. Finally, we run the procedure Prune to further bound the number of elements output by the algorithm. See Figure 1 for an illustration of the data structure.

Algorithm 1 AlgNum: Picking elements with large marginal density.

Input: Parameters m, K, estimate τ^* of $f(\mathcal{OPT})$.

$\ell \leftarrow \lceil \log K \rceil, w \leftarrow \left\lceil \frac{4\ell m}{K} \right\rceil, \tau \leftarrow \frac{2\tau^*}{32(1-\frac{1}{2^\ell})+3}$
for $i \leftarrow 0$ to ℓ **do** ▷ Initialize parameters
  $n_i \leftarrow w\lceil K/2^i \rceil + 8\ell$
  $s_i \leftarrow 0$
  for $j \leftarrow 1$ to n_i **do**
    $B_{i,j} \leftarrow \emptyset$
for each element e in the stream **do**
  for $i \leftarrow 0$ to ℓ **do**
    if $c(e) > 2^{i-1}$ **then continue**
    for $j \leftarrow 1$ to n_i **do**
      if $\rho(e|B_{i,j}) < \frac{\tau}{2^i}$ **then continue**
      if $c(B_{i,j} \cup e) \leq 2^{i+1}$ **then**
        $B_{i,j} \leftarrow B_{i,j} \cup \{e\}$
        $s_i \leftarrow s_i + 8\ell c(e)$
        if $\sum_{j=1}^{n_i} |B_{i,j}| < 10w \cdot 2^i$ **then**
          while $s_i \geq 2^i$ **do**
            $B_{i,n_i+1} \leftarrow \emptyset$
            $n_i \leftarrow n_i + 1$
            $s_i \leftarrow s_i - 2^i$
          break: process next element e
return $S_\tau = \{B_{i,j}\}_{i,j}$

Algorithm 2 Prune: Decreasing the size of the output set.

Input: Output set S from AlgNum.

Sort S by size so that $c(s_1) \leq c(s_2) \leq \ldots$.
$T \leftarrow$ AlgNum with input stream $s_1, s_2, \ldots$.
return T ▷ AlgNum can be replaced with AlgMult.

By using Prune on the output of AlgNum, we have the following result.

Theorem 3.1. *There exists an algorithm that outputs a set $\mathcal{S}$ with $\tilde{O}(K+m)$ elements such that, for any set E of at most m removed*

[1] This assumption can be removed using standard techniques (see e.g. Appendix E of [32]) by maintaining $\frac{1}{\epsilon}\log K$ guesses to find such a τ^*.

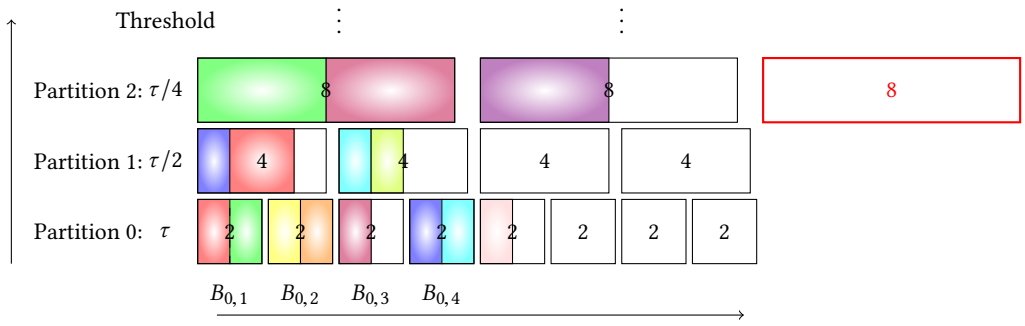

Figure 1: Partitions and buckets: $B_{1,1}$ and $B_{2,2}$ are partially occupied. $B_{2,3}$ (in red) has been dynamically created, since the elements of $B_{2,1}$ and $B_{2,2}$ are large.

Algorithm 3 Robust maximum submodular with knapsack constraint.

Input: Output set T from PRUNE, set E removed by adversary.

1: **return** OFFLINE($T \setminus E$)

items, one can compute from $\mathcal{S}$ a set $Z \subseteq V \setminus E$ with cost at most K and $f(Z)$ is a constant factor approximation to $f(\mathcal{OPT})$.

In fact, if the m items that are removed has total cost at most M, we can provide a better guarantee in terms of both approximation and number of elements stored (see the full version of the paper at [3]).

3.2 d Knapsack Constraints

We now consider the ARMSM(m, K) problem under d knapsack constraints. Recall that ALGNUM relies on guessing the correct threshold and then using a streaming framework that adds elements whose marginal gain surpasses the threshold. In the case where there are d knapsack constraints, a natural approach would be to have parallel instances that guess thresholds for each constraint, and then pick the instance with the best set. This would certainly work, but since there would be $O(\log K)$ guesses for each constraint, the total number of parallel instances would be $O\left(\log^d K\right)$, which is unacceptable for large values of d and K. On the other hand, it seems reasonable to believe that the space usage can be improved, at the expense of the approximation guarantee, by maintaining a smaller number of parallel instances. In that case, marginal gain to cost ratio is not well-defined, since there is a separate cost for each knapsack, so what would be the right quantity to consider?

Recall the standard normalization for multiple knapsack constraints discussed in Section 1.1. We define the largest cost of an item to be the maximum cost of the item across all knapsacks, after the normalization. It has been previously shown that the correct quantity to consider for the streaming model is the marginal gain of an item divided by its largest cost [40]. Namely, if the ratio of the marginal gain to the largest cost of an item exceeds the corresponding threshold, and the item fits into a bucket without violating any of the knapsack constraints, then we choose to add the item to the first such bucket. Since the threshold now compares the marginal gain to the largest cost, a natural question to be asked is what quantity should be used for the dynamic allocation of the buckets. Recall that the previous goal of ALGNUM was to maintain a specific number of items, so that it would be robust against the removal of m items. Thus, we would like to allocate a new bucket for a partition whenever the capacity of the bucket with respect to some knapsack becomes saturated. Hence, ALGMULT maintains a series of counters $_{i,a}$ for partition i and knapsack a, where $1 \le a \le d$. Whenever one of these counters exceeds K, we create a new bucket entirely in partition i, and lower $_{i,a}$ accordingly.

Algorithm 4 ALGMULT: Picking elements with large marginal gain to cost ratio.

Input: Parameters d, m, K, estimate τ^* of $f(\mathcal{OPT})$.

$\ell \leftarrow \lceil \log K \rceil$, $w \leftarrow \left\lceil \frac{4\ell m}{K} \right\rceil$, $\tau \leftarrow \frac{\tau^*}{4}$
for $i \leftarrow 0$ to ℓ **do** ▷ Initialize parameters
  $n_i \leftarrow w\lceil K/2^i \rceil + 8\ell$
  $s_{i,a} \leftarrow 0$
  for $j \leftarrow 1$ to n_i **do**
    $B_{i,j} \leftarrow \emptyset$
for each element e in the stream **do**
  $c(e) \leftarrow \max_{1 \le a \le d} c_a(e)$
  for $i \leftarrow 0$ to ℓ **do**
    if $c(e) > 2^{i-1}$ **then continue**
    for $j \leftarrow 1$ to n_i **do**
      if $\rho(e|B_{i,j}) < \frac{\tau}{2^i(1+2d)}$ **then continue**
      if $c_a(B_{i,j} \cup e) < 2^{i+1}$ for all $1 \le a \le d$ **then**
        $B_{i,j} \leftarrow B_{i,j} \cup \{e\}$
        $s_{i,a} \leftarrow s_{i,a} + 8\ell c_a(e)$ for all $1 \le a \le d$
        while $s_{i,a} \ge 2^i$ for some $1 \le a \le d$ and $\sum_{j=1}^{n_i} |B_{i,j}| < 10w \cdot 2^i$ **do**
          $B_{i,n_i+1} \leftarrow \emptyset$
          $n_i \leftarrow n_i + 1$
          $s_{i,a} \leftarrow \max\{0, s_{i,a} - 2^i\}$ for all $1 \le a \le d$
        break: process next element e
return $S_\tau = \{B_{i,j}\}_{i,j}$

By using PRUNE on the output of ALGNUM, we have the following result. As in Section 3.1, we do not attempt to optimize parameters

here, but observe that the number of elements stored is independent of d.

Theorem 3.2. *For the ARMSM(m, K) problem under d knapsack constraints, there exists an algorithm that outputs a set $\mathcal{S}$ of size $\tilde{O}(K+m)$, from which one can compute a set $Z \subseteq V \setminus E$ with cost at most K and $f(Z)$ is a $\Omega\left(\frac{1}{d}\right)$-approximation to $f(\mathcal{OPT})$.*

4 DISTRIBUTED ALGORITHM

In this section, we give a distributed algorithm for the ARMSM(m, K) problem under d knapsack constraints (see Definition 1.2). We use a variant of the MapReduce model of [20], in which we consider an input set V of size $n = |V|$ that is distributed across $\tilde{O}((K+m)\sqrt{n})$ machines, one of which is designated the *central machine* that will ultimately output a set of elements. For some parameters m and K that are known across all machines, we permit each machine to have $\tilde{O}((K+m)\sqrt{n})$ memory. The machines communicate to each other in a number of synchronous rounds to perform computation. In each round, each machine receives some input of size $\tilde{O}((K+m)\sqrt{n})$, on which the machine performs some local computation. The machine then communicates some output to other machines at the start of the next round. We require that the total input and output message size is $\tilde{O}((K+m)\sqrt{n})$ per machine. We assume that each machine has access to an oracle that computes f. Then our main result in the distributed model is the following.

Theorem 4.1. *For the ARMSM(m, K) problem under d knapsack constraints, there exists a two-round distributed algorithm that outputs a set $\mathcal{S}$, from which one can compute a set $Z \subseteq V \setminus E$ with cost at most K and $f(Z)$ is a $\Omega\left(\frac{1}{d}\right)$-factor approximation to $f(\mathcal{OPT})$. Moreover, each machine uses space $\tilde{O}\left(\sqrt{n \cdot \frac{1}{\epsilon}(K^2+mK)}\right)$.*

The analysis of our distributed algorithm is based on the analysis for our streaming algorithms, along with a recent work by [29]. We generalize their result to obtain a distributed algorithm that constructs a robust summary equivalent to that constructed by AlgMult.

In our algorithm and proofs, we use L to denote an upper bound on the number of elements collected by AlgMult. Let $\mathcal{B}$ be the data structure of sets $B_{i,j}$ maintained by AlgMult. We use AlgMult$_{\mathcal{B},W}$ to refer to the invocation of AlgMult with the following changes:

- The buckets $B_{i,j}$ are initialized by $\mathcal{B}$ and the loop on line 5 of AlgMult is ignored.
- In place of V, the ground set W is used.

Our distributed algorithm is explicitly given in Algorithm 6 and uses subroutine PartitionAndSample, which is given in Algorithm 5.

Algorithm 5 PartitionAndSample

Input: Set of elements V.

1: $F \leftarrow$ sample each $e \in V$ with probability $p = 4\sqrt{L/n}$
2: Partition V randomly into sets $V_1, V_2, \ldots V_T$ to the T machines (one set per machine)
3: Send F to each machine and a central machine C

We prove Theorem 4.1 by first showing that the approximation guarantee is the same as Theorem 3.2.

Algorithm 6 A 2-round distributed algorithm for ARMSM under knapsack constraints.

Input: Parameters d, m, K, estimate τ^* of $f(\mathcal{OPT})$.

Round 1:
1: $F, V_1, \ldots, V_T \leftarrow$ PartitionAndSample(V)
2: **for** each machine M_i (in parallel) **do**
3: $\quad \tau \leftarrow f(\mathcal{OPT})$
4: $\quad$ Let $\mathcal{B}_0$ be the data structure of sets $B_{i,j}$ maintained by AlgMult$_{\emptyset,F}(d, m, K, \tau)$
5: $\quad$ **if** $|\mathcal{B}_0| < L$ **then**
6: $\quad\quad R_i$ be the set of elements $e \in V_i$ that are in $\mathcal{B}$
7: $\quad$ **else**
8: $\quad\quad R_i \leftarrow \emptyset$
9: $\quad$ Send R_i to a central machine C

Round 2 (only on C):
10: Compute $\mathcal{B}_0$ from F as in first round
11: $R \leftarrow \cup_i R_i$
12: **return** $S_\tau \leftarrow$ AlgMult$_{\mathcal{B}_0,R}(d, m, K, \tau)$

Lemma 4.2. *There exists a distributed algorithm that outputs a set Z so that $f(Z)$ has the same approximation guarantee as stated by Theorem 3.2.*

We can also bound the total number of elements sent to the central machine, using a proof similar to [29].

Lemma 4.3. *Let L be an upper bound on the number of elements collected by AlgMult. With probability $1 - e^{-\Omega(L)}$, the number of elements sent to the central machine C is at most $\sqrt{nL}$.*

5 EXPERIMENTS

In this section, we provide empirical evaluation of our algorithms for ARMSM under both single knapsack and multiple knapsack constraints. As no prior work exists in this setting we use the most natural generalizations of standard non-robust algorithms for comparison. We test our most general algorithm AlgMult against such algorithms while measuring the number of elements collected and the quality of the resulting approximation. The aim of our evaluations is to address the following points:

(1) How does AlgMult compare to "robustified" generalizations of other submodular maximization algorithms?
(2) How well does AlgMult perform on real datasets compared to our theoretical worst-case guarantees?
(3) How many elements does AlgMult collect?
(4) Does the performance of AlgMult degrade as the number of elements m removed at the end of the stream increases?

Implementation is available at https://github.com/KDD2019SubmodularKnapsack/KDD2019SubmodularKnapsack.

Robustification. Although there are no existing ARMSM algorithms for knapsack constraints, we propose the following modification to existing algorithms to ensure a fair comparison. Given a submodular maximization algorithm $\mathcal{A}$, we consider its *robustified* version by allowing the algorithm to collect extra elements to obtain its own robust summary. To achieve this, we increase the knapsack capacity by some multiplicative factor, which is selected in such

way that all algorithms collect approximately the same number of elements.

5.1 Baselines

We compare AlgMult to the following algorithms.

Robustified MarginalRatio. This algorithm corresponds to a robustified version of Algorithm 2 from [19], which accepts any element whose marginal density with respect to the stored elements exceeds a certain threshold. Note that while the algorithm is for a single knapsack constraint, it can be trivially extended to multiple knapsack constraints by checking that the thresholding condition holds for all dimensions. This marginal density thresholding algorithm is a natural generalization to knapsack constraints of the streaming algorithm Sieve [4] which gives the best theoretical guarantee under the cardinality constraint.

Robustified offline Greedy. This algorithm builds its summary by iteratively adding to it an element with the largest marginal density. Observe that Greedy is an offline algorithm, which is a more powerful model. However, Greedy is a single knapsack algorithm, so we use it only as a baseline for single knapsack constraints. While there exists a Greedy algorithm [28] under multiple knapsack constraints, it requires $O\left(n^5\right)$ running time, which makes it infeasible on large datasets.

Robustified Multidimensional. This is a robustified version of the streaming algorithm for submodular maximization with multiple knapsack constraints from [40].

5.2 Objectives and Datasets

We evaluate the algorithms on two submodular objective functions:

Dominating set. We use graphs `ego-Facebook` (4K vertices, 81K edges) and `ego-Twitter` (88K vertices, 1.8M edges) from the SNAP database [26]. For a graph $G(V, E)$ and $Z \subseteq V$, we let $f(Z) = \frac{|Z \cup N(Z)|}{|V|}$, where $N(Z)$ is the set of all neighbors of Z. For each knapsack constraint, the cost of each element is selected uniformly at random from the uniform distribution $\mathcal{U}(1, 3)$ and all knapsack constraints are set to 10.

Movie recommendation. Modeling the scenario of movie recommendations we analyze a dataset of movie ratings (in the range $[1, 5]$) assigned by users. For each movie x we use a vector v_x of normalized ratings: if user u did not rate movie x, then set $v_{x,u} = 0$, otherwise set $v_{x,u} = r_{x,u} - r_{avg}$, where r_{avg} denotes the average of all known ratings. Then, the similarity between two movies x_1 and x_2 can be defined as the dot product $\langle v_{x_1}, v_{x_2} \rangle$ of their vectors.

In the case of movie recommendation the goal is to select a representative subset of movies. The domain of our objective is the set of all movies.

For a subset of movies X we consider a parameterized objective function f_X:

$$f_X(Z) = \sum_{x \in X} \max_{z \in Z} \langle v_z, v_x \rangle,$$

where Z is a subset of movies. This captures how representative is Z of the set X. In our experiments, we model the situation of making recommendations to some user so we pick X to be a set of movies rated by the user (we select the user uniformly at random). Hence the maximizer of $f_X(Z)$ corresponds to a subset of movies which represents well user's rated set X.

We use the `ml-20` MovieLens dataset [18], containing 27 278 movies and 20 000 263 ratings. Knapsack constraints model limited demand for movies of a certain type (e.g. not too many action movies, not too many fantasy movies, etc). In the data each movie is labeled by several genres and each knapsack constraint is described by sets of "good" and "bad" genres. Movies with more "good" genres and less "bad" genres have lower cost, allowing the algorithm to choose more such movies. If there are at most t genres describing "good" and "bad" sets then we set the cost of a movie x to be linear in the range $[1, t + 1]$:

$$c(x) = 1 + 0.5 \times (bad(x) - good(x) + t),$$

where $good(x)$ and $bad(x)$ are the numbers of good and bad genres that movie x is labeled with.

For the experiments under one knapsack constraint, we define the good set of movies as $good = \{\text{Comedy}, \text{Horror}\}$, and the bad set of movies as $bad = \{\text{Adventure}, \text{Action}\}$. For experiments under two knapsack constraints, we define the second constraint by an additional set of good movies $good = \{\text{Drama}, \text{Romance}\}$, and an additional set of bad movies $bad = \{\text{Sci-Fi}, \text{Fantasy}\}$. All knapsack constraint bounds are set to 10, limiting the total number of recommended movies.

5.3 Experimental Evaluation and Results

We compare AlgMult against the three baselines described in Section 5.1. First, we obtain robust summaries for AlgMult and for each of the baselines. Second, we adversarially remove elements from these summaries. Finally, we run Offline on the remaining elements in the summaries and compare the values of objective functions on the resulting sets.

Adversarial removals. To ensure a fair comparison, we use the same set of removed elements for all algorithms. This is done by removing the union of sets recommended by all algorithms and then continuing in a recursive fashion if more removals are required.

We define the removal process formally as follows. For an algorithm $\mathcal{A}$, let $S_{\mathcal{A}}$ be the robust summary output by $\mathcal{A}$. We let $R_1 = \cup_{\mathcal{A}} \textsc{Offline}(S_{\mathcal{A}})$, where the union is taken over all four algorithms $\mathcal{A}$ tested. That is, R_1 is the union of the *best* elements selected using AlgMult, Greedy, Multidimensional, and MarginalRatio. This typically already gives a good choice of removals. If more removals are required, we define $R_{k+1} = \cup_{\mathcal{A}} \textsc{Offline}(S_{\mathcal{A}} \setminus \cup_{i=1}^{k} R_i)$. That is, we recursively remove the union of the elements in the optimal sets across all the algorithms and we repeat this process until R_k is empty.

Evaluation. For different numbers of removed elements, we compare the values that are produced by the offline algorithm on robust summaries, i.e. $\textsc{Offline}(S_{\mathcal{A}} \setminus \bigcup_{i=1}^{k} R_i)$ generated by the four algorithms. Since $f(\mathcal{OPT})$ is NP-hard to compute, we compare the performance of each algorithm with upper bounds on $f(\mathcal{OPT})$ to estimate the approximation given by the algorithms. For a single knapsack constraint, the best known upper bound can be computed from Greedy and for multiple knapsack constraints from Multidimensional [40].

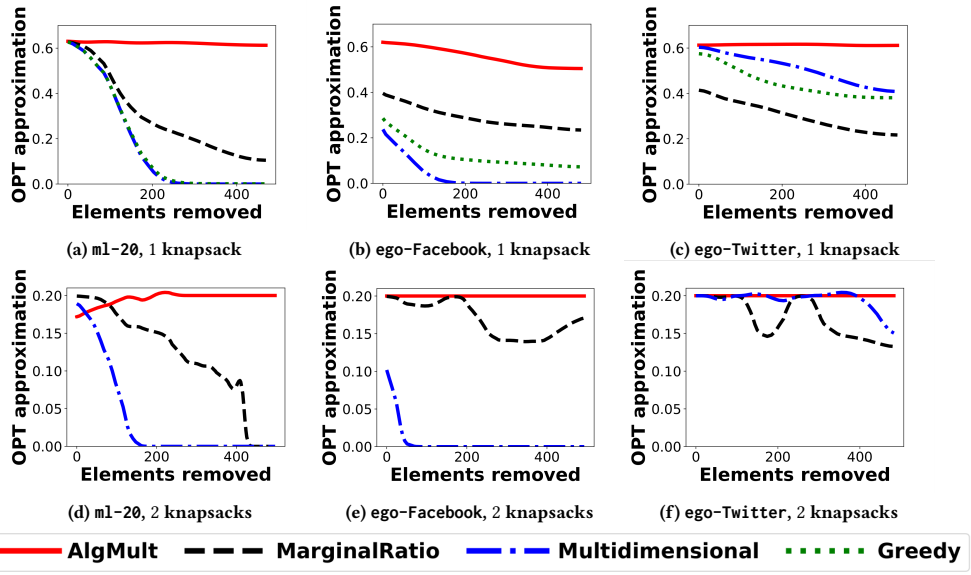

Figure 2: Approximation of $f(\mathcal{OPT})$ for different algorithms (K = 10). In most cases, AlgMult outperforms the other algorithms and achieves the best possible approximation factor ($1 - \frac{1}{e} \approx 0.63$ for one knapsack, 0.2 for two knapsacks).

	ml-20, 1 knapsack	fb, 1 knapsack	twitter, 1 knapsack	ml-20, 2 knapsacks	fb, 2 knapsacks	twitter, 2 knapsacks
AlgMult	641	378	401	1350	2745	4208
MarginalRatio	641	377	402	1350	2745	4209
Multidimensional	87	18	435	72	22	4221
Greedy	647	393	493	-	-	-

Table 1: Sizes of robust summaries produced by the algorithms ($K = 10$).

Results. The results of our experiments are shown in Figure 2. For each algorithm, we plot the ratio of its objective to an upper bound on the optimal solution, which is obtained as previously discussed. Figures 2a, 2b, and 2c show experimental results for 1-knapsack constraints using Greedy as the offline algorithm with approximation factor $1 - e^{-c(s)/K}$, where K is the knapsack constraint and $c(S)$ is the cost of the resulting set. For many instances, $c(S)$ is close to K, so this value is close to $1 - \frac{1}{e} \approx 0.63$. Figures 2d, 2e, and 2f show experimental results for 2-knapsack constraints.

Our evaluations suggest that AlgMult provides the best possible approximation factor for a majority of inputs. Except for the first iterations in Figure 2d, AlgMult outperforms the other algorithms, and achieves roughly the same approximation guarantee as the offline algorithm that knows the items to be removed in advance. In fact, the advantage of AlgMult becomes more noticeable as larger numbers of elements are removed.

Since the baseline algorithms, other than Greedy, require an estimate of $f(\mathcal{OPT})$, we try several such estimations. The non-monotone behavior of the ratio of MarginalRatio to $f(\mathcal{OPT})$ in Figure 2f occurs since MarginalRatio performs better when estimation is close to the true objective. We emphasize the fact that all algorithms, including AlgMult, use the same $f(\mathcal{OPT})$ estimations. It is possible to obtain a more monotone behavior by trying more estimations, but doing so will require collecting more elements.

To evaluate memory consumption, we also report the number of elements collected by each algorithm. These results are presented in Table 1 and show that the algorithms for 2-knapsack constraints collect noticeably more elements than those performing maximization under a single knapsack constraint. The size of the robust summary output by AlgMult does not appear to correlate with the total number of elements, and in the case of `ego-Twitter`, it collects only 5% of the vertices.

Recall that we allow the baseline algorithms to collect extra elements by increasing their knapsack capacity to ensure fair comparison. Hence, almost all the algorithms collect similar numbers of elements for each setup, as shown in Table 1. Note that, however, for some experimental setups Multidimensional collects significantly fewer elements than the other algorithms. This phenomenon persists even if the knapsack capacity is unbounded.

In our empirical evaluations, the number of collected elements did not seem to depend on the number of removed items m. One possible reason for this phenomena is that the algorithms were not executed with small guesses for the optimal objective. As a result when the number of removed elements is large, the optimal objective is below the threshold considered by the algorithm, and therefore more elements are not collected because the threshold is set to be too high. However, it is natural that with sufficiently bad guesses for the optimal objective, any thresholding algorithm will be forced to meaninglessly collect a large number of elements.

6 CONCLUSION

We have given the first streaming and distributed algorithms for adversarially robust monotone submodular maximization subject to single and multiple knapsack constraints. Our algorithms are based on a novel data structure which dynamically allocates new space depending on the elements stored so far and perform well on large scale data sets, even compared to offline algorithms that know in advance which elements will be removed.

For the future work, it is natural to ask whether our framework can be scaled to larger datasets for specific classes of objectives, e.g., is it possible to ensure adversarial robustness with sketching methods for coverage objectives [7]? It would be also interesting to understand the limits on approximation that can be achieved with adversarial robustness and summary size only $\tilde{O}(K+m)$. Finally, an interesting open question is whether it is possible to do adversarially robust *non-monotone* submodular maximization.

REFERENCES

[1] Shipra Agrawal, Mohammad Shadravan, and Cliff Stein. Submodular secretary problem with shortlists. *arXiv preprint arXiv:1809.05082*, 2018.

[2] Nima Anari, Nika Haghtalab, Joseph Naor, Sebastian Pokutta, Mohit Singh, and Alfredo Torrico. Robust submodular maximization: Offline and online algorithms. *CoRR*, abs/1710.04740, 2017.

[3] Dmitrii Avdiukhin, Slobodan Mitrović, Grigory Yaroslavtsev, and Samson Zhou. Adversarially robust submodular maximization under knapsack constraints. *CoRR*, abs/1905.02367, 2019.

[4] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 671–680. ACM, 2014.

[5] Eric Balkanski, Aviad Rubinstein, and Yaron Singer. An exponential speedup in parallel running time for submodular maximization without loss in approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 283–302, 2019.

[6] Eric Balkanski and Yaron Singer. The adaptive complexity of maximizing a submodular function. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 1138–1151, 2018.

[7] MohammadHossein Bateni, Hossein Esfandiari, and Vahab S. Mirrokni. Optimal distributed submodular optimization via sketching. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD*, pages 1138–1147, 2018.

[8] Ilija Bogunovic, Slobodan Mitrović, Jonathan Scarlett, and Volkan Cevher. Robust submodular maximization: A non-uniform partitioning approach. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, pages 508–516, 2017.

[9] Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.

[10] Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: Matchings, matroids, and more. In *Integer Programming and Combinatorial Optimization - 17th International Conference, IPCO. Proceedings*, pages 210–221, 2014.

[11] Chandra Chekuri and Kent Quanrud. Submodular function maximization in parallel via the multilinear relaxation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 303–322, 2019.

[12] Rafael da Ponte Barbosa, Alina Ene, Huy L. Nguyen, and Justin Ward. A new framework for distributed submodular maximization. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS*, pages 645–654, 2016.

[13] Khalid El-Arini and Carlos Guestrin. Beyond keyword search: discovering relevant scientific literature. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 439–447. ACM, 2011.

[14] Alina Ene and Huy L. Nguyen. Submodular maximization with nearly-optimal approximation and adaptivity in nearly-linear time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 274–282, 2019.

[15] Matthew Fahrbach, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Submodular maximization with nearly optimal approximation, adaptivity and query complexity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 255–273, 2019.

[16] Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011.

[17] Ryan Gomes and Andreas Krause. Budgeted nonparametric learning from data streams. In *ICML*, pages 391–398, 2010.

[18] GroupLens. https://grouplens.org/datasets/movielens. MovieLens Datasets.

[19] Chien-Chung Huang, Naonori Kakimura, and Yuichi Yoshida. Streaming algorithms for maximizing monotone submodular functions under a knapsack constraint. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 11:1–11:14, 2017.

[20] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 938–948, 2010.

[21] Ehsan Kazemi, Morteza Zadimoghaddam, and Amin Karbasi. Scalable deletion-robust submodular maximization: Data summarization with privacy and fairness constraints. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, pages 2549–2558, 2018.

[22] Andreas Krause, H Brendan McMahan, Carlos Guestrin, and Anupam Gupta. Robust submodular observation selection. *Journal of Machine Learning Research*, 9(Dec):2761–2801, 2008.

[23] Ariel Kulik, Hadas Shachnai, and Tami Tamir. Approximations for monotone and nonmonotone submodular maximization with knapsack constraints. *Math. Oper. Res.*, 38(4):729–739, 2013.

[24] Ravi Kumar, Benjamin Moseley, Sergei Vassilvitskii, and Andrea Vattani. Fast greedy algorithms in mapreduce and streaming. *ACM Transactions on Parallel Computing (TOPC)*, 2(3):14, 2015.

[25] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429. ACM, 2007.

[26] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[27] Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 510–520. Association for Computational Linguistics, 2011.

[28] Hui Lin and Jeff A. Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings*, pages 912–920, 2010.

[29] Paul Liu and Jan Vondrák. Submodular optimization in the mapreduce model. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019*, pages 18:1–18:10, 2019.

[30] Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Deletion-robust submodular maximization: Data summarization with "the right to be forgotten". In *International Conference on Machine Learning*, pages 2449–2458, 2017.

[31] Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *Advances in Neural Information Processing Systems*, pages 2049–2057, 2013.

[32] Slobodan Mitrović, Ilija Bogunovic, Ashkan Norouzi-Fard, Jakub Tarnawski, and Volkan Cevher. Streaming robust submodular maximization: A partitioned thresholding approach. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems*, pages 4560–4569, 2017.

[33] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978.

[34] Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrović, Amir Zandieh, Aida Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. *arXiv preprint arXiv:1808.01842*, 2018.

[35] James B Orlin, Andreas S Schulz, and Rajan Udwani. Robust monotone submodular function maximization. *Mathematical Programming*, 172(1-2):505–537, 2018.

[36] Ruben Sipos, Adith Swaminathan, Pannaga Shivaswamy, and Thorsten Joachims. Temporal corpus summarization using submodular word coverage. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 754–763. ACM, 2012.

[37] Matthew Staib, Bryan Wilder, and Stefanie Jegelka. Distributionally robust submodular maximization. *CoRR*, abs/1802.05249, 2018.

[38] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004.

[39] Kai Wei, Yuzong Liu, Katrin Kirchhoff, and Jeff Bilmes. Using document summarization techniques for speech data subset selection. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 721–726, 2013.

[40] Qilian Yu, Easton Li Xu, and Shuguang Cui. Streaming algorithms for news and scientific literature recommendation: Monotone submodular maximization with a d -knapsack constraint. *IEEE Access*, 6:53736–53747, 2018.