Optimal Distributed Submodular Optimization via Sketching

MohammadHossein Bateni
Google
New York, NY
bateni@google.com

Hossein Esfandiari
Harvard University
Cambridge, MA
esfandiari@seas.harvard.edu

Vahab Mirrokni
Google
New York, NY
mirrokni@google.com

ABSTRACT

We present distributed algorithms for several classes of submodular optimization problems such as k-cover, set cover, facility location, and probabilistic coverage. The new algorithms enjoy almost optimal space complexity, optimal approximation guarantees, optimal communication complexity (and run in only four rounds of computation), addressing major shortcomings of prior work. We first present a distributed algorithm for k-cover using only $\tilde{O}(n)$ space per machine, and then extend it to several submodular optimization problems, improving previous results for all the above problems—e.g., our algorithm for facility location problem improves the space of the best known algorithm (Lindgren et al. [20]). Our algorithms are implementable in various distributed frameworks such as MapReduce and RAM models. On the hardness side we demonstrate the limitations of uniform sampling via an information theoretic argument.

Furthermore, we perform an extensive empirical study of our algorithms (implemented in MapReduce) on a variety of datasets. We observe that using sketches 30–600 times smaller than the input, one can solve the coverage maximization problem with quality very close to that of the state-of-the-art single-machine algorithm. Finally, we demonstrate an application of our algorithm in large-scale feature selection.

ACM Reference Format:
MohammadHossein Bateni, Hossein Esfandiari, and Vahab Mirrokni. 2018. Optimal Distributed Submodular Optimization via Sketching. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As important special cases of submodular optimization, maximum k-cover, minimum set cover, and facility location problems are among the most central problems in optimization with a wide range of applications in machine learning, document summarization, and information retrieval; e.g., see [1, 10, 12, 20, 25]. Increasingly, the need for developing distributed algorithms for these problems to handle huge datasets seems inevitable. To address this need, many techniques have been developed for distributed submodular maximization [5, 9, 14, 17, 19, 20, 22, 25, 29]. However many of these results do not take advantage of the special structure of coverage functions, and consequently achieve suboptimal approximation guarantees and/or poor space complexities in terms of the size of the (coverage) instance. In particular, most previous results on submodular maximization either explicitly or implicitly assume *value oracle access* to the submodular function. Such an oracle for coverage functions has the following form: given a subfamily of the (input) family, determine the size of the union of the subsets in the subfamily. Implementing this subroutine is impractical in the presence of large subsets in the family and/or a large ground set. Indeed even communicating entire subsets across machines might be impractical. In this paper, first we aim to address the above issues, and present almost optimal distributed approximation algorithms for coverage optimization problems with *optimal communication* and *space complexity*. Next, we extend our results to several other well-motivated classes of submodular functions, such as facility location and probabilistic coverage among others. Before elaborating on our results, let us describe the coverage optimization problems and the distributed computation models.

Coverage Optimization problems Consider a ground set $\mathcal{E}$ of m elements, and a family $\mathcal{F} \subseteq 2^{\mathcal{E}}$ of n subsets of the elements (i.e., $n = |\mathcal{F}|$ and $m = |\mathcal{E}|$).[1] The *coverage function* C is defined as $C(\mathcal{S}) = |\cup_{U \in \mathcal{S}} U|$ for any subfamily $\mathcal{S} \subseteq \mathcal{F}$ of subsets. Given $k \geq 0$, the goal in k-cover is to pick k sets from $\mathcal{F}$ with the largest union size. Set Cover asks for the minimum number of sets from $\mathcal{F}$ that together cover $\mathcal{E}$ entirely. In this paper, we also study the following variant of this problem, called set cover with λ outliers[2], where the goal is to find the minimum number of sets covering at least a $1 - \lambda$ fraction of the elements $\mathcal{E}$.

MapReduce Model The distributed computation model—e.g., MapReduce [15]—assumes that the data is split across multiple machines. In each round of a distributed algorithm, the data is processed in parallel on all machines: Each machine waits to receive messages sent to it in the previous round, performs its own computation, and finally sends messages to the other machines. The total amount of data a machine processes is called its *load*, which ought to be sublinear in the input size. In fact, two important factors determine a distributed algorithm's performance: (i) the number of rounds of computation, and (ii)

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] There are two separate series of work in this area. We use the convention of the submodular/welfare maximization formulation [4], whereas the hypergraph-based formulation [28] typically uses n, m in the opposite way.
[2] It is sometimes called $(1 - \lambda)$-partial cover in the literature.

the maximum load on any machine. These parameters have been discussed and optimized in previous work [7, 17, 18, 22].

RAM Model Another model for handling large amounts of data is what we call the *RAM model* [2], where the algorithm has random access to any part of the input (say, to the edge lists in the graph) but each lookup takes constant time. For many problems it might be possible to judiciously and adaptively query the data, and solve the problem. Distributed hash tables (such as BigTable) have been proposed and applied in practice [11] to implement algorithms in this model. From a theoretical standpoint, this model is closely related to the communication complexity literature.

Continuous Distributed Monitoring Model In this model, each of η clients observes a separate data stream. At the end of the stream, a server is asked to solve a problem on the union of the data streams via communication with clients. In this model, the goal is to minimize the space and the communication.

Our Contributions In this work, we present distributed algorithms for submodular optimization problems such as k-cover and set cover, addressing several shortcomings of previously studied algorithms and achieving optimal approximation guarantees as well as almost optimal space and communication complexity. To achieve this result, we employ an adaptive sampling (or sketching) technique that can be implemented in a distributed manner. We also rule out effectiveness of various simpler sampling techniques by providing lower bound examples. We extend our algorithm to several problems such as facility location, dominating set, probabilistic coverage, and weighted coverage. Below we summarize our main contributions.

First of all, we focus on a special case of submodular optimization—coverage problems. Here we develop distributed algorithms for k-cover and set cover with λ outliers, that are almost optimal from three perspectives: (i) they achieve optimal approximation guarantees of $1 - 1/e$ and $\log \frac{1}{\lambda}$ for the above two problems, respectively; (ii) they have a memory complexity of $\tilde{O}(n)$ and also $\tilde{O}(n)$ communication complexity; and finally (iii) they run in a few (constant) rounds of computation; see Table 1 for brief comparison of our theoretical results with prior work. We note that the space complexity of our algorithm is independent of the size of the universe of elements and is only a linear function of the number of input sets. This is crucial for tackling coverage instances with very large sets or large total number of elements.

Second, we also rule out effectiveness of uniform sampling as a natural technique [27, e.g.] by providing an information theoretic upper bound on its performance.

Third, we extend our algorithm to several well motivated problems with submodular functions, such as facility location, dominating set, probabilistic coverage, and weighted coverage.

For the facility location problem, our algorithm requires $\tilde{O}(n)$ space, while the best previous algorithm [20] requires $O(nm/k)$ space. Also for dominating set, which has applications to influence maximization in social networks [25], we give the first distributed algorithm that does not need to load all edges of a node onto a single machine (Deferred to the full version). This is crucial for handling graphs with nodes of very high degree. We show that our algorithm can be implemented in MapReduce, RAM and Continuous Distributed Monitoring models. Furthermore, we present extensions of our distributed algorithm to a number of variants of weighted coverage problems (Section 3). Extensions to RAM and continuous distributed monitoring model are presented in Section 4.

Furthermore, we demonstrate the power of our techniques via an extensive empirical study on a variety of applications and publicly available datasets (Section 5). We observe that sketches that are a factor 30–600 smaller than the input suffice for solving k-cover with quality (almost) matching that of the state-of-the-art single-machine algorithm; e.g., for a medium-size dataset, we can obtain 99.6% of the quality of the single-machine STOCHASTIC GREEDY algorithm using only 3% of the input data. Some of the instances we examine in this paper are an order of magnitude larger than the ones studied in prior works [20, 25]. In particular, for facility location we show that our space complexity is significantly smaller than those of [20] (See Figure 5).

Finally, we demonstrate an application of our algorithm in large-scale feature selection by formalizing it as a coverage problem where we aim to choose a subset of features that *cover* as many *pairs of samples* as possible. In doing so, we take advantage of the fact that the space complexity of our algorithm is independent of the number of elements in the instance, and we can solve instances of coverage problem with very large sets.

Further Related Work Although maximum k-cover may be solved using a distributed algorithm for submodular maximization, all the prior work in this area (have to) assume value oracle access to the submodular function, introducing a dependence on the size of the sets in the running time of each round of the algorithms. In this model, Chierichetti et al. [12] present a $1 - \frac{1}{e}$-approximation algorithm for k-cover in polylogarithmic number of rounds of computation, improvable to $O(\log n)$ rounds [9, 19]. Recently randomized core-sets were used to obtain a constant-approximation 2-round algorithm for this problem [14, 22], where the best known approximation factor is 0.54. In other recent work, [24, 26] give a distributed algorithm for submodular cover (a generalization of set cover) in the MapReduce framework, however, their algorithm runs in superconstant number of rounds. Moreover, compared to the result presented here, their algorithm has much larger space complexity when applied to set cover. [27] study a generalization of k-cover for influence maximization using uniform sampling of elements.

Coverage optimization problem has been considered in the streaming setting as well. In a recent work we provide streaming algorithms for coverage optimization problems [8]. There we introduce an abstract sketch and present a sequential implementation of the sketch in the streaming setting. In this paper, we present a nontrivial construction of the sketch with $\tilde{O}(n)$ space using independent machines in a distributed manner. More specifically, while a more direct implementation of the sketch requires $\tilde{O}(n + m)$ space per machine to keep a label for each element, here we use a shared hash function and a randomized load-balancing scheme to improve the space complexity to $\tilde{O}(n)$. Note that going from $\tilde{O}(n + m)$ to $\tilde{O}(n)$ is critical in the empirical performance of the algorithm. The same improvement also enables us to extend our results to the weighted and

Table 1: Comparison of our results to prior work. The first three algorithms work for the more general case of submodular maximization.

Problem	Credit	# rounds	Approximation	Load per machine
k-cover	[19]	$O(\frac{1}{\varepsilon\delta}\log m)$	$1-\frac{1}{e}-\varepsilon$	$O(mkn^{\delta})$
k-cover	[22]	2	0.54	$\max(mk^2, mn/k)$
k-cover	[13]	$\frac{1}{\varepsilon}$	$1-\frac{1}{e}-\varepsilon$	$\frac{\max(mk^2, mn/k)}{\varepsilon}$
facility location	[20, 29]	$O(1)$	$1-\frac{1}{e}-\varepsilon$	$\tilde{O}(nm/k)$
k-cover	Section 2	4	$1-\frac{1}{e}-\varepsilon$	$\tilde{O}(n)$
facility location	Section 3	4	$1-\frac{1}{e}-\varepsilon$	$\tilde{O}(n)$
set cover w. outliers	Section 2	4	$(1+\varepsilon)\log\frac{1}{\lambda}$	$\tilde{O}(n)$
submodular cover	[24]	$\Omega(n^{\frac{1}{6}})$	$\Omega(n^{\frac{1}{6}})$	$\tilde{O}(mn)$
submodular cover	[26]	$O(\frac{\log n \log m}{\epsilon})$	$(1+\varepsilon)\log\frac{1}{\lambda}$	$\tilde{O}(mn)$
dominating set	Full version	4	$(1+\varepsilon)\log\frac{1}{\lambda}$	$\tilde{O}(n)$

Table 2: Comparison of results to prior work for the RAM and continuous distributed monitoring models.

Problem	Credit	Approximation	Runtime
k-cover	[6]	$1-\frac{1}{e}-\varepsilon$	$\tilde{O}(nm)$
k-cover	Section 4	$1-\frac{1}{e}-\varepsilon$	$\tilde{O}(n)$
set cover with outliers	Section 4	$(1+\epsilon)\log\frac{1}{\lambda}$	$\tilde{O}(n)$

probabilistic settings as well as facility location. Moreover, on the hardness side, we show the shortcoming of uniform sampling by providing an (information theoretic) upper bound on its performance. In addition, we show how to apply our result to other settings such as the RAM model and the Distributed Monitoring Model, and other problems such as dominating set. In fact, we only borrow Lemma 2.4 (directly and without proof) from the previous work. All the proofs presented here are new contributions of the current work.

[21] provide another streaming algorithm for set cover problem, with a different technique. Their result is highly based on their iterative algorithm for k-cover. Hence, they do not provide algorithms for other coverage optimization problems. Besides, due to the iterative nature of their algorithm and analysis, it is not clear how to apply this algorithm to distributed settings.

More Notation Coverage problems may also be described via a bipartite graph G, with the two sides corresponding to $\mathcal{F}$ and $\mathcal{E}$, respectively. The edges of G correspond to pairs (S, i) where $i \in S \in \mathcal{F}$. For simplicity, we assume that there is no isolated vertex in $\mathcal{E}$. As is customary, we let $\Gamma(G, V')$ denote the set of neighbors of vertices V' in G. When applied to a bipartite graph G modeling a coverage instance, we can write the coverage function as $C(\mathcal{S}) = |\Gamma(G, \mathcal{S})|$ for any $\mathcal{S} \subseteq \mathcal{F}$.

2 ALGORITHMS FOR k-COVER AND SET COVER

In this section we present distributed algorithms for k-cover and set cover with λ outliers. We aim to develop algorithms that only need $\tilde{O}(n)$ space per machine. As a first attempt, if we want to apply the distributed submodular optimization results to our problems (e.g., DISTGREEDY [25] or composable core-set algorithm [14, 22]), the underlying algorithms would distribute *subsets* across machines. The main issue with such an approach is that sending whole subsets does not scale well for large subsets. A natural way to deal with the issue of large subsets is to subsample elements while sending those sets around, and a natural sampling technique would be *uniform sampling*. We first rule out applicability of such simple sampling schemes for this problem. In particular, we present a hardness example for which the size (i.e., the number of edges) of the instance on each machine has to be $\Omega(nk)$ to obtain a bounded approximation guarantee.

THEOREM 2.1. *Pick arbitrary numbers $n, \beta \geq 1$ and $k \leq n/2$. Let* Alg *be an algorithm that samples elements uniformly at random and reports an arbitrary optimum solution to k-cover on the sampled instance. Unless* Alg *samples more than nk/β^2 edges, its approximation factor is at most $\frac{2}{\beta+1}$.*

PROOF. Consider the following example with k *bonus sets* and $n-k$ normal sets. Moreover, we have βn *special elements* and n normal ones. Each set has edges to all normal elements, and each bonus set has edges to $\beta n/k$ unique bonus elements. Notice that the optimum k-cover solution picks all the k bonus sets, and covers all the $(\beta+1)n$ elements.

Note that each normal element has n edges. Since Alg samples at most nk/β^2 edges, no more than k/β^2 normal elements in expectation make it to the sample. In other words, each element is sampled with probability at most $\frac{k}{\beta^2 n}$. Therefore, Alg

samples at most $\frac{k}{\beta^2 n} \times \beta n = n/\beta$ bonus elements, in expectation.

Indeed, if Alg do not pick any bonus elements corresponding to a bonus set S, in the sampled graph the set S covers the same elements as any normal set does. Thus Alg might pick a normal set instead of S in an arbitrary optimum solution on the sampled graph. Notice that Alg samples at most n/β bonus elements in expectation, which corresponds to no more than n/β distinct bonus sets, in expectation. Hence there is an optimum solution on the sampled graph with n/β bonus sets and $n - n/\beta$ normal sets in expectation. The expected total number of elements in this solution is $n + \frac{n}{\beta} \times \beta n = 2n$. □

This observation suggests that any distributed algorithm should employ a more nuanced sampling (sketching) technique. To this end, we invoke a recent technique of ours [8][3]: if we can develop a distributed algorithm with $\tilde{O}(n)$ space that outputs a sketch (denoted by $H_{\le n}(k, \varepsilon, \delta'')$, or simply $H_{\le n}$) satisfying *three special properties*, we can prove tight approximation guarantees for the following algorithm: solve the problem by running a greedy algorithm on the sketch.

Algorithm 1 Distributed algorithm for k-cover

Input: Input graph G and parameters k, $\varepsilon \in (0, 1]$, δ''.
Output: Solution to the coverage problem.

Let $h : \mathcal{E} \mapsto [0, 1]$ be a uniform, independent hash function.

Round 1: Send the edges of each element to a distinct machine. Let $\tilde{n} = \frac{24n\delta \log(1/\varepsilon) \log n}{(1-\varepsilon)\varepsilon^3}$. For each element v, if $h(v) \le \frac{2\tilde{n}}{m}$, the machine corresponding to v sends $h(v)$ and its degree to machine one; it does nothing otherwise.

Round 2: Machine one iteratively selects elements with the smallest h until the sum of the degrees of the selected vertices reaches $\tilde{n}$. Then it informs the machines corresponding to selected elements.

Round 3: For each selected element v, if the degree of v is less than $\Delta = \frac{n \log(1/\varepsilon)}{\varepsilon k}$, machine v sends all its edges to machine one. Otherwise, it sends Δ arbitrary edges to machine one.

Round 4: Machine one receives the sketch $H_{\le n}$, runs greedy algorithm for k-coverage on it, and output the solution.

Here we develop Algorithm 1, a four-round distributed algorithm[4], and prove the main result of this section, by showing that the output of this algorithm satisfies those three properties with high probability in a distributed setting using only $\tilde{O}(n)$ space. More formally, we prove the following.

THEOREM 2.2. *With probability $1 - \frac{2}{n}$, Algorithm 1 outputs a $(1 - \frac{1}{e} - \epsilon)$-approximate solution to k-cover, and no machine uses more than $\tilde{O}(n)$ space in this algorithm.*

The proof has two ingredients. First of all, we show that the sketch $H_{\le n}$ computed in this algorithm satisfies the following three properties: Given parameters ϵ and δ'', (1) elements are sampled uniformly at random, (2) the degree of each element is upper bounded by $\frac{n \log(1/\varepsilon)}{\varepsilon k}$, and (3) the total number of edges is at least $\frac{24n\delta \log(1/\varepsilon) \log n}{(1-\varepsilon)\varepsilon^3}$, where $\delta = \delta'' \log \log_{\frac{1}{1-\varepsilon}} m$. Secondly, we need to show that the algorithm uses $\tilde{O}(n)$ space per machine. The following lemma summarizes properties of the algorithm that pave the way for the proof of the theorem.

LEMMA 2.3. *Given is a graph $G(\mathcal{F} \cup \mathcal{E}, E)$ along with k, $\varepsilon \in (0, 1]$ and $\delta'' \in (0, 1]$. Then with probability $1 - 1/n^2$,*

- *$H_{\le n}$ has no element with hash value exceeding $\frac{2\tilde{n}}{m}$, and*
- *there are at most $3\tilde{n}$ edges with hash value not exceeding $\frac{2\tilde{n}}{m}$,*

where $\tilde{n} = \frac{24n\delta \log(1/\varepsilon) \log n}{(1-\varepsilon)\varepsilon^3}$.

PROOF. Note that $H_{\le n}$ requires only $\tilde{n}$ edges. Clearly $\tilde{n}$ elements are sufficient to satisfy this. In the rest of the proof we show that, with probability $1 - 1/n$, the number of elements with hash value less than $\frac{2\tilde{n}}{m}$ is within the range $[\tilde{n}, 3\tilde{n}]$. The lower bound together with the fact that H contains at most $\tilde{n}$ elements gives us the first part of the theorem. The upper bound directly proves the second part of the theorem.

For every element $v \in \mathcal{E}$, let X_v be the binary random variable indicating whether $h(v) < \frac{2\tilde{n}}{m}$, and let $X = \sum_{v \in \mathcal{E}} X_v$ denote the number of elements with hash value less than $\frac{2\tilde{n}}{m}$. The Chernoff bound gives

$$\mathbf{Pr}\Big(|X - \mathbb{E}[X]| \ge \frac{1}{2}\mathbb{E}[X]\Big) \le 2\exp\left(-\frac{\frac{1}{4}\mathbb{E}[X]}{3}\right) = 2\exp\Big(-\frac{\mathbb{E}[X]}{12}\Big). \quad (1)$$

Remark that h is a uniform mapping to $[0, 1]$. Thus, $\mathbf{Pr}[h(v) \le \frac{2\tilde{n}}{m}] = \frac{2\tilde{n}}{m}$ for any element v, so we have

$$\mathbb{E}[X] = \sum_{v \in \mathcal{E}} \mathbb{E}[X_v] = \sum_{v \in \mathcal{E}} \frac{2\tilde{n}}{m} = 2\tilde{n}. \quad (2)$$

Putting (1) and (2) together gives us

$$\begin{aligned}\mathbf{Pr}\Big(|X - 2\tilde{n}| \ge \tilde{n}\Big) &\le 2\exp\Big(-\frac{\tilde{n}}{6}\Big) \\ &= 2\exp\Big(-\frac{4n\delta \log(1/\varepsilon) \log n}{(1-\varepsilon)\varepsilon^3}\Big) \\ &\le 2\exp\Big(-2\log n - 1\Big) \\ &< \exp\Big(-2\log n\Big) = \frac{1}{n^2}.\end{aligned}$$

Thus we have $\tilde{n} \le X \le 3\tilde{n}$ with probability $1 - \frac{1}{n^2}$. □

We use the following lemma to prove Theorems 2.2 and 2.5.

LEMMA 2.4 (FROM [8]). *For any $\varepsilon \in (0, 1]$ and any set-cover graph G, there exist sketch-based algorithms that succeed with probability $1 - \frac{1}{n}$ in finding the following.*

(1) *One finds a $(1 - \frac{1}{e} - \varepsilon)$-approximate solution to k-cover on G, working on a sketch with $\tilde{O}(n)$ edges.*
(2) *The other finds a $(1+\varepsilon)\log\frac{1}{\lambda}$ approximate solution to set cover with λ outliers on G. The sketches used altogether have $\tilde{O}(n/\lambda^3) = \tilde{O}_\lambda(n)$ edges.*

Now we are ready to prove Theorem 2.2.

[3] In a recent work [8], we study streaming algorithms for coverage functions and in particular show that, for fixed positive α and δ, an α approximate solution to k-cover on a sketch with the above properties is an $\alpha - \epsilon$ approximate solution on the actual input, with probability $1 - e^{-\delta}$.

[4] Number of rounds were not optimized in the interest of readability.

PROOF. **of Theorem 2.2** We first show that Algorithm 1 uses $\tilde{O}(n)$ space per machine. Next we prove that the algorithm constructs by Round 4 a sketch satisfying the desirable three properties mentioned above. As a result, Lemma 2.4 guarantees that invoking the greedy algorithm in Round 4 produces the promised solution.

The degree of each element is at most n, the number of sets; thus, the space consumption of each machine in the first and third rounds is $\tilde{O}(n)$. In the second round machine number 1 receives $\tilde{O}(1)$ bits from each machine independently with probability $\frac{2\tilde{n}}{m}$. Using the second condition of Lemma 2.3, the number of messages that this machine receives is at most $3\tilde{n}$. Therefore, this machine uses $\tilde{O}(n)$ space. The number of edges machine one receives in the fourth round is at most $\tilde{n}+n = \tilde{O}(n)$.

By the first condition of Lemma 2.3, no element in $H_{\leq n}$ has hash value more than $\frac{2\tilde{n}}{m}$. Thus the machines with no output in the first round do not miss any elements of $H_{\leq n}$. Then the set of elements selected by machine one in round two is the same as in $H_{\leq n}$. Therefore, what machine one receives in the fourth round is $H_{\leq n}$. Discussion at the beginning of the proof finishes the argument. □

While our algorithm for k-cover runs a greedy algorithm on the sketch, our algorithm for set-cover with λ outliers makes logarithmically many guesses on the number of sets in the solution, constructs the $H_{\leq n}$ sketch for each (simultaneously), and solves the problem on each resulting sketch.

THEOREM 2.5. *There exists a four-round distributed algorithm that reports a $(1+\varepsilon)\log\frac{1}{\lambda}$-approximate solution to set cover with λ outliers, with probability $1-\frac{2}{n}$. Moreover, each machine uses $\tilde{O}(n)$ space in the algorithm.*

PROOF. Theorem 2.5 We run $\log_{1+\varepsilon/3} n$ copies of the first three stages of Algorithm 1 (simultaneously) to construct the $\log_{1+\varepsilon/3} n$ different sketches required by Lemma 2.4. The discussion in Theorem 2.2 implies that each copy of the sketch is constructed correctly, with probability $1-\frac{1}{n^2}$. Together with Lemma 2.4 this proves that our algorithm gives a $(1+\varepsilon)\log\frac{1}{\lambda}$-approximate solution to set cover with λ outliers, with probability $1-\frac{1}{n}-\log_{1+\varepsilon/3} n\frac{1}{n^2} > 1-\frac{2}{n}$. □

3 MORE GENERAL SUBMODULAR FUNCTIONS

In this section we extend[5] our results to three classes of submodular functions: In element-weighted k-cover a weight w_v is associated with each element $v \in \mathcal{E}$, and the objective is to maximize the total weight of covered elements. An instance of facility location problem contains a quantity $\alpha_{u,v} \in [0,1]$[6] for each $S \in \mathcal{F}, v \in \mathcal{E}$, denoting that set S covers $\alpha_{S,v}$ fraction of element v. A solution $\mathcal{S} \subseteq \mathcal{F}$ covers $\max_{S\in\mathcal{S}} \alpha_{S,v}$ fraction of element v. Here the objective is to find a solution $\mathcal{S} \subseteq \mathcal{F}$ of size k that maximizes $\sum_{v\in\mathcal{E}} \max_{S\in\mathcal{S}} \alpha_{S,v}$.

[5]Due to space limitations, some proofs are omitted from the body of the paper. All omitted proofs appear in the full version. The full version is available at https://arxiv.org/abs/1612.02327

[6]In general, in facility location, $\alpha_{u,v}$'s are real numbers, but here, we normalize all $\alpha_{u,v}$'s to a number in [0, 1].

Finally in probabilistic k-cover, quantity $\alpha_{S,v} \in [0,1]$ is provided for each pair of $S \in \mathcal{F}$ and $v \in \mathcal{E}$: set S covers element v with probability $\alpha_{S,v}$. A solution $\mathcal{S} \subseteq \mathcal{F}$ covers $1-\prod_{S\in\mathcal{S}}(1-\alpha_{S,v})$ fraction of element v. The objective then is to find a solution $\mathcal{S} \subseteq \mathcal{F}$ of cardinality k that maximizes $\sum_{v\in\mathcal{E}}\left(1-\prod_{S\in\mathcal{S}}(1-\alpha_{S,v})\right)$.

In the first problem, for simplicity we assume that all weights are integers upper-bounded by a number U. Similarly, in the second and the third problems, we assume that $\alpha_{S,v}$ is a factor of $1/U$ for any $v \in S \in \mathcal{F}$.

THEOREM 3.1. *There exists a four-round distributed algorithm that finds a $(1-\frac{1}{e}-\epsilon)$-approximate solution to element-weighted k-cover, with probability $1-\frac{2}{n}$. Moreover, each machine uses $\tilde{O}(n)$ space in this algorithm.*

THEOREM 3.2. *There exists a four-round distributed algorithm that reports a $(1-\frac{1}{e}-\epsilon)$-approximate solution to facility location, with probability $1-\frac{2}{n}$. Moreover, no machine uses more than $\tilde{O}(n)$ space in this algorithm.*

THEOREM 3.3. *There exists a four-round distributed algorithm, using $\tilde{O}(n)$ space per machine, that finds a $(1-\frac{1}{e}-2\epsilon)$-approximate solution to probabilistic k-cover with probability $1-\frac{3}{n}$.*

PROOF. Similarly we transform an instance of probabilistic k-cover to one of k-cover: substitute each element $v \in \mathcal{E}$ with $\zeta = \frac{12(n+1+\log n)U}{\epsilon^2}$ copies of v, and for each set S that contains v we connect S to each copy of v with probability $\alpha_{S,v}$.

We show that with probability $1-\frac{1}{n}$, for all solutions $\mathcal{S} \subseteq \mathcal{F}$, the coverage of $\mathcal{S}$ in the k-cover instance is within a factor $\zeta(1\pm\epsilon/2)$ of that in the original probabilistic k-cover instance. Fix a solution $\mathcal{S}$, and let $\beta_v = 1-\prod_{S\in\mathcal{S}}(1-\alpha_{S,v})$. The Chernoff bounds gives for X, the number of copies of v covered by $\mathcal{S}$, as follows.

$$\begin{aligned}
\mathbf{Pr}\Big(|X-\zeta\beta_v| \geq \zeta\beta_v\epsilon/2\Big) &\leq 2\exp\Big(-\frac{\zeta\beta_v\epsilon^2}{12}\Big)\\
&= 2\exp\Big(-\frac{\frac{12(n+1+\log n)U}{\epsilon^2}\beta_v\epsilon^2}{12}\Big)\\
&\leq 2\exp\Big(-\frac{\frac{12(n+1+\log n)}{\epsilon^2}\epsilon^2}{12}\Big) \qquad \text{since } \beta_v \geq 1/U,\\
&\leq 2\exp\Big(-n+1+\log n\Big) \leq 2^{-n}/n.
\end{aligned}$$

There are 2^n choices for $\mathcal{S}$, hence for all solutions $\mathcal{S} \subseteq \mathcal{F}$, the coverage of $\mathcal{S}$ on the k-cover instance is within the promised interval with probability $1-\frac{1}{n}$.

Since the number of elements in the k-cover instance is at most ζm, the size of the sketch grows logarithmically in U. To sample an element form the k-cover instance uniformly at random, we sample an element from the original probabilistic k-cover instance uniformly at random and connect it to each set $S \in \mathcal{F}$ with probability $\alpha_{S,v}$. □

4 ALGORITHMS FOR OTHER DISTRIBUTED MODELS

In this section we explain how our results apply to the RAM model and continuous distributed monitoring model as well.

Recall that in the RAM model, we have random access to the edge lists, however, each access takes $O(1)$ time.

Algorithm 2 Abstract construction of the sketch

Input: Graph $G(\mathcal{F} \cup \mathcal{E}, E)$ and numbers $k, \varepsilon \in (0, 1], \delta''$.
Output: Sketch $H(V_H, E_H) = H_{\leq n}(k, \varepsilon, \delta'')$.

1: $\delta \leftarrow \delta'' \log \log_{\frac{1}{1-\varepsilon}} m$
2: $h : \mathcal{E} \mapsto [0, 1]$ uniform, independent hash function
3: $V_H \leftarrow \mathcal{F}$ and $E_H \leftarrow \emptyset$ ▷ Initialize
4: **while** $|E_H| < \frac{24n\delta \log(1/\varepsilon) \log n}{(1-\varepsilon)\varepsilon^3}$ **do**
5: $\quad v \leftarrow \arg\min_{v \in \mathcal{E} \setminus V_H} h(v)$
6: $\quad V_H \leftarrow V_H \cup \{v\}$
7: $\quad$ Add $\min(\frac{n \log(1/\varepsilon)}{\varepsilon k}, |\Gamma_G(v)|)$ edges of v to E_H

THEOREM 4.1. *There exists an algorithm that, given random access to the edge lists of coverage instance $G(\mathcal{F} \cup \mathcal{E}, E)$, computes the sketch $H = H_{\leq n}$ in time $\tilde{O}(n)$.*

PROOF. We show how Algorithm 2 can run in the RAM model. Since $|E_H| = \tilde{O}(n)$ at the end, total work done in Line 7 is $\tilde{O}(n)$. In Line 2 we do not need to define the hash function explicitly. When Line 5 seeks the next vertex, it is equivalent to picking a random new vertex. We only need to keep a list of already selected vertices to avoid repetition. □

Once the sketch is constructed we can run a sequential algorithm on the sketch (or sketches) to solve k-cover and set cover with outliers. The proof of the following is almost identical to those of Theorems 2.2 and 2.5 and is omitted.

THEOREM 4.2. *There is an $\tilde{O}(n)$-time, $1 - \frac{1}{e} - \epsilon$-approximation algorithm in the RAM model for k-cover.*

THEOREM 4.3. *There is an $\tilde{O}(n)$-time algorithm in the RAM model that finds a $(1 + \epsilon) \log \frac{1}{\lambda}$-approximate solution for set cover with λ outliers.*

To construct the sketch in the continuous distributed monitoring model, we construct a sketch on each client, using shared random seeds, and send this to the server. On the server we construct the sketch of the union of the received graphs using the same random seeds as the clients. Notice that (1) if we remove an edge that does not exist in the sketch it does not change the sketch, and (2) if an edge exists in the sketch removing other edges does not change the status of this edge. The second observation implies that all of the edges of the sketch of the input graph is sent to the server. The first one indicates that ignoring edges that are not in the sketch does not effect the outcome and hence the outcome of the server is equivalent to the sketch of the whole input. This yields the following two theorems, whose proofs are similar to the above, and are deferred to the full version.

THEOREM 4.4. *There is a $1-\frac{1}{e}-\epsilon$-approximation algorithm in the continuous distributed monitoring model with $\tilde{O}(n)$ space per machine and $\tilde{O}(n)$ communication per machine for k-cover.*

THEOREM 4.5. *There is an algorithm with $\tilde{O}(n)$ space per machine and $\tilde{O}(n)$ communication per machine in the continuous distributed monitoring model that finds a $(1 + \epsilon) \log \frac{1}{\lambda}$-approximate solution for set cover with λ outliers.*

Table 3: General information about our datasets.

Name	Type	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert E\rvert$
livej-3	dominating set	4M	4M	73B
livej-2	dominating set	4M	4M	3.4B
dblp-3	dominating set	320K	320K	330M
dblp-2	dominating set	320K	320K	27M
gutenberg	bag of words	42K	100M	1B
s-gutenberg	bag of words	925	11M	27M
reuters	bag of words	200K	140K	15M
planted-A	planted coverage	10K	10K	1.2M
planted-B	planted coverage	100K	1M	1.2B
planted-C	planted coverage	100K	10M	2.4B
planted-D	planted coverage	101K	10M	1.2B
wiki-main	contribution graph	2.9M	11M	75M
wiki-talk	contribution graph	1.7M	1M	7.3M
news20	feature selection	1.4M	200M	4.3B

5 EMPIRICAL STUDY AND RESULTS

We begin by a brief overview of the datasets and corresponding applications used in our empirical study (see Table 3), and then move to the methodology as well as the experiment results.

We run our empirical study on five types of instances. A summary is presented in Table 3. Dominating set instances include livej-3, livej-2, dblp-3 and dblp-2, where the objective is to cover the nodes via multi-hop neighborhoods. We have two sets of *bag-of-words* instances, where the goal is to cover as many words/bigrams via selecting a few documents: gutenberg and s-gutenberg for books and reuters for news articles. Instances wiki-main and wiki-talk are our contribution graphs where we want to find a set of users who revised many articles. Finally we have some planted set-cover instances that are known to fool the greedy algorithm: planted-A, etc. As for the feature-selection application, we focus on news20 dataset that is discussed in detail in [3].

We remark that, to the best of our knowledge, some of these datasets are an order of magnitude larger than what has been considered in prior work.

5.1 Approach

Recall that the sketch construction consists of two types of prunings for edges and vertices of the graph:

- subsampling the elements, and
- removing edges from large-degree elements.

The theoretical definition of the sketch provides (i) the probability ρ of sampling an element, and (ii) the upper bound σ on their degrees. Though almost tight in theory, in practice one can use smaller values for these two parameters to get desirable solutions. Here we parameterize our algorithm by ρ and σ, and investigate this phenomenon in our experiments.

The STOCHASTICGREEDY algorithm [23] achieves $1-\frac{1}{e}-\varepsilon$ approximation to maximizing monotone submodular functions (hence coverage functions) with $O(n \log(1/\varepsilon))$ calls to the submodular function. This is theoretically the fastest known $1 - \frac{1}{e} - \epsilon$ approximation algorithm for coverage maximization, and is the most efficient in practice for maximizing monotone submodular functions, when the input is very large. Plugged into our MapReduce algorithm, it runs much faster, and loses very little in terms of quality. For smaller instances we compare

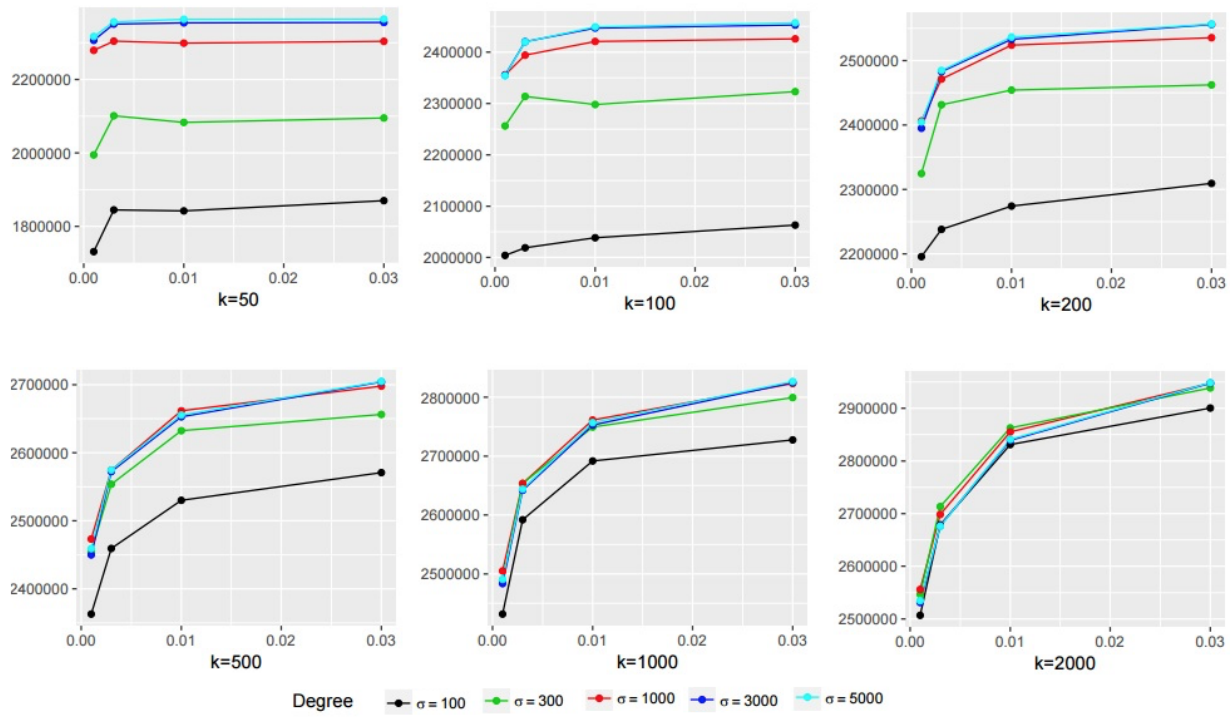

Figure 1: For the dominating-set instance `livej-3`, these plots show the number of covered nodes against the relative size of the sketch with $\rho \in [10^{-3}, 3 \cdot 10^{-2}]$, $\sigma \in [100, 5000]$, and $k \in [10^2, 10^4]$. Curves in one plot correspond to different choices for σ. With large σ, the results of some runs are indistinguishable from the one next to it in the plot, hence invisible.

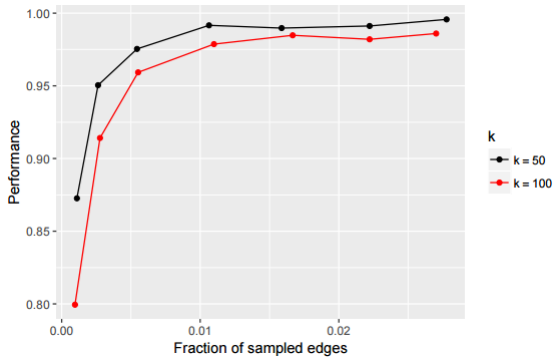

Figure 2: The results for `dblp-3` are shown for $\rho \in [2 \cdot 10^{-3}, 5 \cdot 10^{-2}]$, $\sigma = 100$. We plot our performance relative to StochasticGreedy against the fraction of edges from the input graph retained in our sketch.

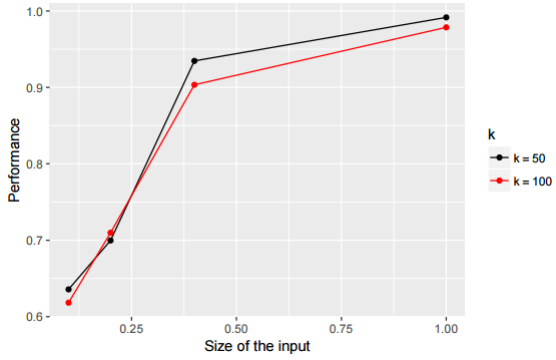

Figure 3: The above are results of running the algorithm on the sampled version of `dblp-3` with $\rho = 0.02$, $\sigma = 100$. The x axis denotes the size of the sampled graph relative to the whole. The y axis shows the quality relative to StochasticGreedy.

our algorithm to StochasticGreedy, but for larger ones we provide convergence numbers to argue that the two should get very similar coverage results.

LiveJournal social network We try different values for ρ, σ, k when running our algorithm on `livej-3`; see Figure 1. For small k, the result improves as σ grows, but increasing ρ has no significant effect. On the other hand, the improvement for larger k comes from increasing ρ while σ is not as important. This observation matches the definition of our sketch, in which the degree bound is decreasing in k and the sampling rate is increasing in k.

DBLP coauthorship network Figure 2 compares results of our algorithm on `dblp-3` (with a range of parameters) to that of StochasticGreedy. Each point in these plots represents the mean of three independent runs. Interestingly, a sketch with merely 3% of the memory footprint of the input graph attains %99.6 of the quality of StochasticGreedy.

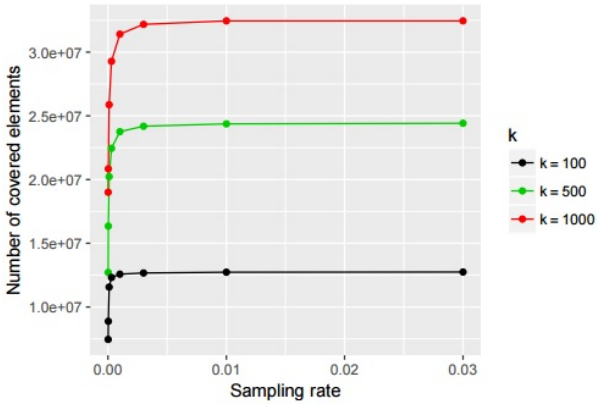

Figure 4: Here we plot the number of covered bigrams against ρ for `gutenberg` with $\rho \in [10^{-5}, 3 \cdot 10^{-2}]$, $\sigma \in [10^2, 10^4]$, and $k \in [10^2, 1000^3]$. The curves corresponding to different values of σ are practically indistinguishable.

Table 4: Results for other datasets.

Instance	Sketch	Quality	Instance	Sketch	Quality
`wiki-main`	0.06%	94.4%	`dblp-2`	1.7%	92%
`wiki-main`	2.4%	99.5%	`dblp-2`	3.1%	96%
`wiki-main`	7.7%	99.9%	`reuters`	1.2%	87%
`wiki-talk`	1.5%	99.2%	`reuters`	3.6%	92%
`planted-A`	8.2%	96%	`reuters`	10%	96%

We run our algorithm on induced subgraphs of `dblp-3` of varying sizes; see Figure 2. Interestingly, the performance of our algorithm improves the larger the sampled graph becomes. In other words, if one finds parameters ρ and σ on a subgraph of the input and applies it to the whole graph, one does not lose much in the performance.

Project Gutenberg dataset We run our algorithm on `gutenberg` with different values for ρ and σ. As shown in Figure 4, the outcome of the algorithm converges quickly. In other words, for $\rho = 0.003$ and $\sigma = 100$, the outcome of StochasticGreedy on our sketch and on the input graph are quite similar, while our sketch is 600 times smaller.

Other datasets Due to space constraints we cannot report detailed results for the other datasets. However, Table 4 shows that for these datasets, a small sketch suffices to get close to the single-machine greedy solution. In fact, these are small enough for the greedy algorithm to run on one machine.

The `livej-2` instance is too big for the single-machine greedy algorithm. Still we can compare our result to what is achievable for a 10% sample of the instance (with about 340 million edges). With a 0.8% footprint we obtain a solution of essentially the same quality. With footprints 0.3%, 0.2%, 0.1% and 0.75%, we lose no more than 1%, 3%, 4% and 9%, respectively.

Except for the smallest, the planted instances are also too big for the greedy algorithm. Nonetheless, looking at the numbers, e.g., for `planted-B`, we notice that the quality of the greedy solution is almost the same for sketches of relative sizes 0.3% and 42%—the latter has about 500 million edges. In particular, sketches of relative size 0.3%, 1% and 10% produce 3%, 2% and 1% error, respectively, compared to the sketch of size 42%. The results are similar for the other two planted instances.

Finally we compare our sketching technique to that in [20]. They present a technique that can be thought of as the second part of our sketching method. We demonstrate in Figure 5 the superiority of our method on two facility location datasets based on the DBLP graph. Our method achieves the same solution quality with much smaller memory footprint (i.e., sketch size).

5.2 Feature-selection Problem

Our algorithm is applicable to the feature-selection problem, which is a first step in many learning-based applications [16], where it is often too expensive to work with the entire matrix or there might be overfitting concerns. Typically a small subset of "representative" features are picked carefully, so as not to affect the overall learning quality. In practice, we gauge the performance of feature selection by *reconstruction error* or *prediction accuracy*; see [3] for details of evaluation criteria.

In order to compare our preliminary results to previous work [3], we model the problem as a maximum k-cover instance by treating columns (i.e., features) as sets and *pairs of rows* (i.e., pairs of sample points) as elements. We say a row *covers* a pair of rows, if that column (feature) is active for both rows (sample points), and seek to pick k columns that *cover as many pairs of rows* as possible.[7]

Table 5 compares our results to prior work. Numbers show prediction accuracy in percentage. For description of the data set and the first four algorithms, see [3]. We note that these algorithms may only run on a 8% sample of the dataset, hence poorer performance compared to the latter two. The fifth column exhibits a distributed version of 2-P (the two-phase optimization): Features are carefully partitioned across many machines by looking at a cut-based objective, and then the two-phase optimization handles each part separately. It is noteworthy that the (distributed) partitioning phase itself takes significant amount of time to run. The last column corresponds to our distributed k-cover algorithm, which is more efficient than the algorithm of the fifth column. The results are similar to that of Part.

Table 5: Results for feature selection on `news20` dataset.

k	Rnd	2-P	DG	PCA	Part	Cover
500	54.9	81.8	80.2	85.8	84.5	86.2
1000	59.2	84.4	82.9	88.6	88.4	89.4
2500	67.6	87.9	85.5	90.6	92.3	91.2

We emphasize that we can run our algorithm on much larger datasets; the evidence of this was provided above where we reported results for `livej-3`, for instance.

6 CONCLUSIONS

In this paper, we present almost optimal distributed algorithms for coverage problems. Our algorithms beat the previous ones in several fronts: e.g., (i) they provably achieve the optimal

[7] We also studied covering rows as opposed to covering pairs of rows, but that approach was not effective.

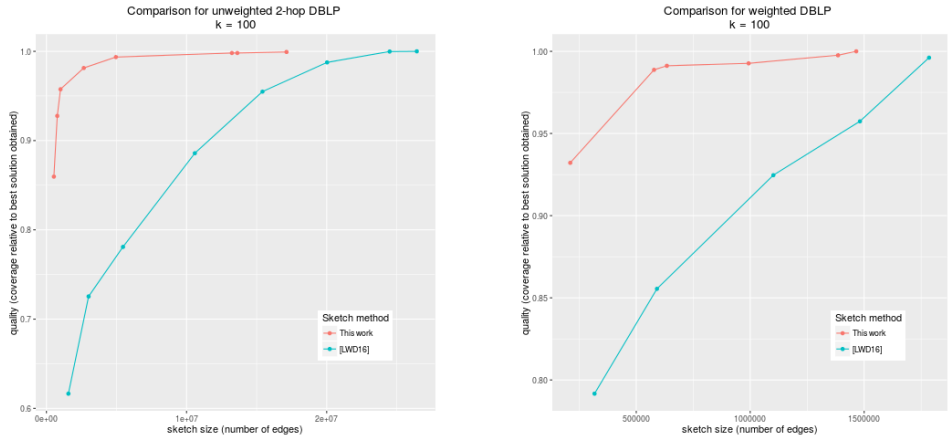

Figure 5: The left plot is for the unweighted `dblp-2` instance, and the one on the right is for the weighted DBLP graph, where the edge weights are the number of common neighbors of two nodes. Two curves in each plot correspond to the "lazy greedy" algorithm run on our sketch or on the sketch introduced in [20]. We see the quality (i.e., coverage) relative to the best solution obtained in terms of the sketch size. Notice that typically half the sketch size suffices for obtaining essentially the same quality.

approximation factors for these problems, (ii) they run in only four rounds of computation (as opposed to logarithmic number of rounds), and (iii) their space complexity is independent of the number of elements in the ground set. Moreover, our algorithms can handle coverage problems with huge subsets (in which even one subset of the input may not fit on a single machine). Our empirical study shows practical superiority of our algorithms. Finally, we identified a new application of our algorithms in feature selection, and presented preliminary results for this application. It would be nice to explore this application in more details in the future.

REFERENCES

[1] Z. Abbassi, V. S. Mirrokni, and M. Thakur. Diversity maximization under matroid constraints. In *KDD*, pages 32–40, 2013.

[2] A. V. Aho and J. E. Hopcroft. *Design & Analysis of Computer Algorithms*. Pearson Education India, 1974.

[3] J. Altschuler, A. Bhaskara, G. Fu, V. S. Mirrokni, A. Rostamizadeh, and M. Zadimoghaddam. Greedy column subset selection: New bounds and distributed algorithms. In *ICML*, pages 2539–2548, 2016.

[4] A. Badanidiyuru, S. Dobzinski, H. Fu, R. Kleinberg, N. Nisan, and T. Roughgarden. Sketching valuation functions. In *SODA*, pages 1025–1035, 2012.

[5] A. Badanidiyuru, B. Mirzasoleiman, A. Karbasi, and A. Krause. Streaming submodular maximization: Massive data summarization on the fly. In *KDD*, 2014.

[6] A. Badanidiyuru and J. Vondrák. Fast algorithms for maximizing submodular functions. In *SODA*, pages 1497–1514, 2014.

[7] M. Bateni, A. Bhashkara, S. Lattanzi, and V. Mirrokni. Mapping core-sets for balanced clustering. In *NIPS*, 2014.

[8] M. Bateni, H. Esfandiari, and V. Mirrokni. Almost optimal streaming algorithms for coverage problems. In *SPAA*, pages 13–23, 2017.

[9] G. E. Blelloch, H. V. Simhadri, and K. Tangwongsan. Parallel and I/O efficient set covering algorithms. In *SPAA*, pages 82–90, 2012.

[10] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *PODS*, pages 155–166, 2012.

[11] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *OSDI*, 2006.

[12] F. Chierichetti, R. Kumar, and A. Tomkins. Max-Cover in Map-Reduce. In *WWW*, pages 231–240, 2010.

[13] R. da Ponte Barbosa, A. Ene, H. L. Nguyen, and J. Ward. A new framework for distributed submodular maximization. *CoRR*, abs/1507.03719, 2015.

[14] R. da Ponte Barbosa, A. Ene, H. L. Nguyen, and J. Ward. The power of randomization: Distributed submodular maximization on massive datasets. In *ICML*, pages 1236–1244, 2015.

[15] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

[16] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[17] P. Indyk, S. Mahabadi, M. Mahdian, and V. Mirrokni. Composable core-sets for diversity and coverage maximization. In *PODS*, 2014.

[18] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *SODA*, pages 938–948, 2010.

[19] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in MapReduce and streaming. In *SPAA*, pages 1–10, 2013.

[20] E. Lindgren, S. Wu, and A. G. Dimakis. Leveraging sparsity for efficient submodular data summarization. In *NIPS*, pages 3414–3422, 2016.

[21] A. McGregor and H. T. Vu. Better streaming algorithms for the maximum coverage problem. In *20th International Conference on Database Theory, ICDT 2017, March 21-24, 2017, Venice, Italy*, pages 22:1–22:18, 2017.

[22] V. S. Mirrokni and M. Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *STOC*, pages 153–162, 2015.

[23] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrák, and A. Krause. Lazier than lazy greedy. In *AAAI*, pages 1812–1818, 2015.

[24] B. Mirzasoleiman, A. Karbasi, A. Badanidiyuru, and A. Krause. Distributed submodular cover: Succinctly summarizing massive data. In *NIPS*, 2015.

[25] B. Mirzasoleiman, A. Karbasi, R. Sarkar, and A. Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *NIPS*, pages 2049–2057, 2013.

[26] B. Mirzasoleiman, M. Zadimoghaddam, and A. Karbasi. Fast distributed submodular cover: Public-private data summarization. In *NIPS*, pages 3594–3602, 2016.

[27] H. T. Nguyen, M. T. Thai, and T. N. Dinh. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *Proceedings of the 2016 International Conference on Management of Data*, pages 695–710. ACM, 2016.

[28] B. Saha and L. Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *SDM*, volume 9, pages 697–708, 2009.

[29] K. Wei, R. K. Iyer, and J. A. Bilmes. Fast multi-stage submodular maximization. In *ICML*, pages 1494–1502, 2014.